Gradient Learning in Networks of Smoothly Spiking Neurons with an Additional Penalty Term

Šíma, Jiří
2011

Institute of Computer Science
Academy of Sciences of the Czech Republic

Gradient Learning in Networks of Smoothly Spiking Neurons with an Additional Penalty Term

Jiří Šíma

Technical report No. 1125

November 2011

Pod Vodárenskou věží 2, 182 07 Prague 8, phone: +420 266 053 030, fax: +420 286 585 789,
e-mail:sima@cs.cas.cz

Institute of Computer Science
Academy of Sciences of the Czech Republic

Gradient Learning in Networks of Smoothly Spiking Neurons with an Additional Penalty Term

Jiří Šíma[1]

Technical report No. 1125

November 2011

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 experiments confirmed the smoothness of spike creation/deletion process. Nevertheless, the experiments have also shown that the gradient method get often stuck in local minima corresponding to transient network configurations which emerge from the smooth approximation of discrete events. An additional penalty term introduced in the error function did not help to avoid this side effect, which disqualifies this approach.

Keywords:
spiking neuron, back-propagation, SpikeProp, gradient learning, penalty term

[1]This work was partially supported by projects GA ČR P202/10/1333 and AV0Z10300504.

1 Learning in Networks of Spiking Neurons

Neural networks establish an important class of learning models that are widely applied in practical applications to solving artificial intelligence tasks [6]. The prominent position among neural network models has recently been occupied by networks of spiking (pulse) neurons [5, 9]. As compared to the traditional perceptron networks [12] the networks of spiking neurons represent a biologically more plausible model which has a great potential for processing 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 [10, 19].

Learning in the perceptron networks is usually performed by a gradient descent method, e.g. by using the back-propagation algorithm [15] which explicitly evaluates the gradient of an error function. The same approach has been employed in the *SpikeProp* gradient learning algorithm [2] which learns the desired firing times of output neurons by adapting the weight parameters in the Spike Response Model SRM_0 [5]. Plenty of experiments with *SpikeProp* was carried out which clarified e.g. the role of the parameter initialization and negative weights [11]. The performance of the original algorithm was improved by adding the momentum term [22]. Moreover, the *SpikeProp* was further enhanced with additional learning rules for synaptic delays, thresholds, and time constants [16] which resulted in faster convergence and smaller network sizes for given learning tasks. An essential speedup was achieved by approximating the firing time function using the logistic sigmoid [1]. The *SpikeProp* algorithm was also extended to recurrent network architectures [21].

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 [20]. 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 very discontinuous. Recently, the so-called *ASNA-Prop* has been proposed [17] 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 [4].

In the present report which extends our previous work [18][2] 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 [15], we modify a slightly simplified version of the Spike Response Model SRM_0 by smoothing out the discontinuities along the weight and delay parameter trajectories (Section 2). For this purpose we employ an auxiliary twice differentiable function which is a smooth approximation of the step function. In particular, a new spike arises through a continuous "division" of the succeeding spike into two spikes while the spike disappearance is implemented by a continuous "fusion" with its successor. For our model of smoothly spiking neuron we derive a nontrivial back-propagation-like learning rule for computing the gradient of the error function with respect to both the weight and delay parameters (Section 3) which can be used for supervised learning of desired spike trains in corresponding feedforward networks. In order to take also the temporal dimension into account, we have implemented the chain rule for computing the partial derivatives of the composite error function so that each neuron propagates backwards a variable number of partial derivative terms corresponding to different time instants including the second-order derivative terms. The new gradient learning method can naturally cope with multiple spikes whose number changes in time. Preliminary experiments with the proposed learning algorithm exhibited the smoothness of spike creation/deletion process (Section 5). Nevertheless, the experiments have also shown that the gradient method get often stuck in local minima corresponding to transient network configurations which emerge from the smooth approximation of discrete events. An additional penalty term introduced in the error function (Section 4) did not help to avoid this side effect, which disqualifies this approach.

A related line of study concerns Hebbian learning [3, 8, 13] and self-organization [14] in networks

[2] We have removed the formal intial spikes at zero time and introduce an additional penalty term in the error function.

of spiking neurons. See also paper [7] for a recent review of supervised learning methods in spiking neural networks.

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[3]. 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

$$\int Ax^2(x-\delta)^2\,dx = A\left(\frac{x^5}{5} - 2\delta\,\frac{x^4}{4} + \delta^2\,\frac{x^3}{3}\right) + C \tag{2.1}$$

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:

$$\sigma(\alpha, \beta, \delta\,;\, x) = \begin{cases} \alpha & \text{for } x < 0 \\ (\beta-\alpha)\left(\left(6\frac{x}{\delta} - 15\right)\frac{x}{\delta} + 10\right)\left(\frac{x}{\delta}\right)^3 + \alpha & \text{for } 0 \leq x \leq \delta \\ \beta & \text{for } x > \delta\,. \end{cases} \tag{2.2}$$

We will also need the following partial derivatives of σ:

$$\sigma'(x) = \frac{\partial}{\partial x}\sigma(x) = \begin{cases} \frac{30}{\delta}(\beta-\alpha)\left(\left(\frac{x}{\delta} - 2\right)\frac{x}{\delta} + 1\right)\left(\frac{x}{\delta}\right)^2 & \text{for } 0 \leq x \leq \delta \\ 0 & \text{otherwise,} \end{cases} \tag{2.3}$$

$$\sigma''(x) = \frac{\partial^2}{\partial x^2}\,\sigma(x) = \begin{cases} \frac{60}{\delta^2}(\beta-\alpha)\left(\left(2\frac{x}{\delta} - 3\right)\frac{x}{\delta} + 1\right)\frac{x}{\delta} & \text{for } 0 \leq x \leq \delta \\ 0 & \text{otherwise,} \end{cases} \tag{2.4}$$

$$\frac{\partial}{\partial \alpha}\sigma(x) = \begin{cases} 1 & \text{for } x < 0 \\ 1 - \left(\left(6\frac{x}{\delta} - 15\right)\frac{x}{\delta} + 10\right)\left(\frac{x}{\delta}\right)^3 & \text{for } 0 \leq x \leq \delta \\ 0 & \text{for } x > \delta\,, \end{cases} \tag{2.5}$$

$$\frac{\partial}{\partial \beta}\sigma(x) = \begin{cases} 0 & \text{for } x < 0 \\ \left(\left(6\frac{x}{\delta} - 15\right)\frac{x}{\delta} + 10\right)\left(\frac{x}{\delta}\right)^3 & \text{for } 0 \leq x \leq \delta \\ 1 & \text{for } x > \delta\,. \end{cases} \tag{2.6}$$

In addition, we will use the logistic sigmoid function

$$P(\lambda\,;\, x) = \frac{1}{1 + e^{-\lambda x}} \tag{2.7}$$

[3]For simplicity, we assume a *constant* threshold function (which equals the opposite value of the bias), thus excluding the refractory effects [5] but using the presented techniques one can generalize the model of smoothly spiking neuron and the learning algorithm for a nonconstant threshold function.

(or shortly $P(x)$) with a real gain parameter λ (e.g. $\lambda = 4$) whose derivative is known to be

$$P'(x) = \lambda P(x)(1 - P(x))\,. \tag{2.8}$$

Now we introduce the smooth computational dynamics of the network. Each spiking neuron $j \in V$ in the network may produce a sequence of $p_j \geq 0$ *spikes (firing times)* $0 < t_{j1} < t_{j2} < \cdots < t_{jp_j} < T$ as outlined in Figure 2.1. In addition, define formally $t_{j,p_j+1} = T$. 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*

$$\xi_j(t) = w_{j0} + \sum_{i \in j_{\leftarrow}} w_{ji}\, \varepsilon\,(t - d_{ji} - \tau_i(t - d_{ji})) \tag{2.9}$$

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

$$\big\{0 \leq t \leq T \,|\, \xi_j(t) = 0 \,\&\, \xi_j'(t) > 0\big\} = \big\{t_{j1} < t_{j2} < \cdots < t_{jp_j}\big\} \tag{2.10}$$

as shown in Figure 2.2. Formula (2.9) 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* for all neurons, e.g.

$$\varepsilon(t) = e^{-(t-1)^2} \cdot \sigma_0(t) \tag{2.11}$$

where recall

$$\sigma_0(t) = \sigma(0, 1, \delta_0\,;\, t) \tag{2.12}$$

is defined by (2.2) using parameter δ_0 particularly for the definition of ε. Clearly, ε is a twice differentiable function with a relatively flat spike shape as depicted in Figure 2.3 which reaches its maximum $\varepsilon(1) = 1$ for $t = 1$. Furthermore, $\tau_i : \Re \longrightarrow \Re$ is a smooth approximation of the stair function shown in Figure 2.4 that gives the last firing time t_{is} of neuron i preceding a given time t for $t > t_{i1}$ whereas formally $\tau_i(t) = t_{i1}$ for $t \leq t_{i1}$ (particularly for $p_i = 0$, we get $\tau_i(t) = t_{i1} = T$).

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 $0 < \widetilde{t_{j1}} < \widetilde{t_{j2}} < \cdots < \widetilde{t_{jp_j}} < T = \widetilde{t_{j,p_j+1}}$ by formula

$$\widetilde{t_{js}} = \begin{cases} t_{js} & \text{for } j \in X \\ \sigma\Big(t_{js}, \widetilde{t_{j,s+1}}, \delta\,;\, \delta - \xi_j'(t_{js})\Big) & \text{for } j \in V \setminus X \end{cases} \tag{2.13}$$

using (2.2) where s goes in sequence from p_j down to 1, 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 (2.10). Clearly, $\widetilde{t_{js}} = t_{js}$ for $\xi_j'(t_{js}) \geq \delta$ while $\widetilde{t_{js}} \in (t_{js}, \widetilde{t_{j,s+1}})$ is smoothly decreasing with increasing small $\xi_j'(t_{js}) \in (0, \delta)$ as depicted in Figure 2.5. The purpose of this transformation is to make the creation and deletion of spikes smooth with respect to the weight and delay updates as outlined in Figure 2.6. In particular, by moving along

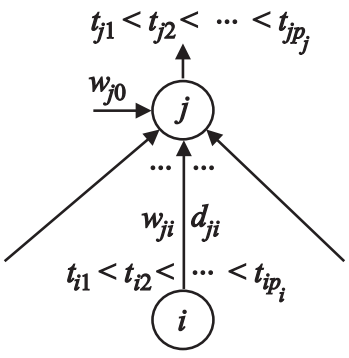

Figure 2.1: A spiking neuron j.

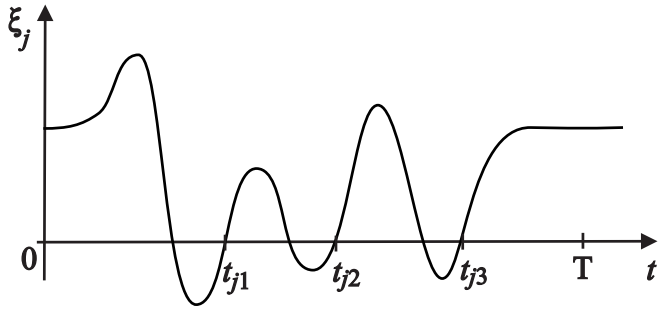

Figure 2.2: The excitation $\xi_j(t)$ of neuron j defining its firing times.

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 derivatives of excitation $\xi'_j(t_{js})$ at the crossing point t_{js} starting at zero as depicted in the zoom square in Figure 2.6. Thus, the new transformed spike $\widetilde{t_{js}}$ arises through a continuous "division" of the succeeding (transformed) spike $\widetilde{t_{j,s+1}}$ into two spikes while the spike disappearance is implemented similarly by a continuous "fusion" with its successor, which is controlled by the first derivative of the excitation. The transformed spikes then define

$$\tau_j(t) = \widetilde{t_{j1}} + \sum_{s=2}^{p_j+1} \left(\widetilde{t_{js}} - \widetilde{t_{j,s-1}}\right) P^c\left(t - \widetilde{t_{js}}\right) \tag{2.14}$$

using the logistic sigmoid (2.7) as shown in Figure 2.7 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. The derivative of j's excitation with respect to time t can be derived from (2.9):

$$\xi'_j(t) = \sum_{i \in j_{\leftarrow}} w_{ji}\, \varepsilon'\left(t - d_{ji} - \tau_i(t - d_{ji})\right)\left(1 - \tau'_i(t - d_{ji})\right) \tag{2.15}$$

where the derivative of response function

$$\varepsilon'(t) = e^{-(t-1)^2} \cdot \left(\sigma'_0(t) - 2(t-1)\sigma_0(t)\right) \tag{2.16}$$

can be evaluated using (2.12) and (2.3) while

$$\tau'_i(t) = \frac{\partial}{\partial t}\tau_i(t) = c\lambda \sum_{s=2}^{p_i+1} \left(\widetilde{t_{is}} - \widetilde{t_{i,s-1}}\right) P^c\left(t - \widetilde{t_{is}}\right)\left(1 - P\left(t - \widetilde{t_{is}}\right)\right) \tag{2.17}$$

is calculated from (2.14) and (2.8).

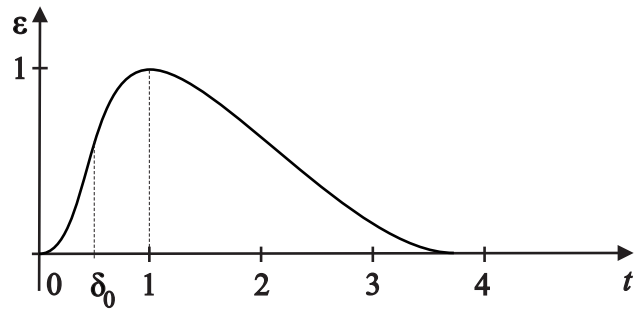

Figure 2.3: The response function $\varepsilon(t)$.

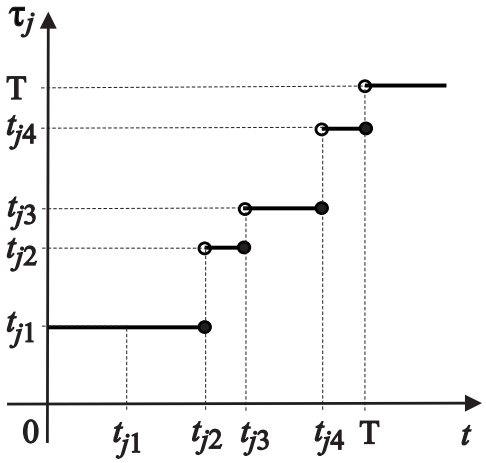

Figure 2.4: The stair function.

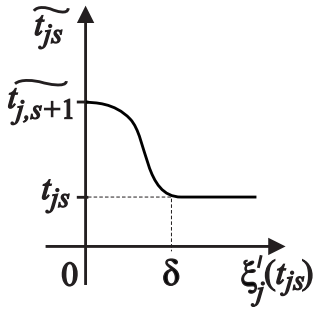

Figure 2.5: The spike transformation.

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 $0 < \varrho_{j1} < \varrho_{j2} < \cdots < \varrho_{jq_j} < T = \varrho_{jq_{j+1}}$ 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*

$$E_1(\mathbf{w}, \mathbf{d}) = \frac{1}{2} \sum_{j \in Y} \left((\tau_j(0) - \varrho_{j1})^2 + \sum_{s=1}^{q_j} (\tau_j(\varrho_{j,s+1}) - \varrho_{js})^2 \right) \tag{3.1}$$

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}, while we need $\tau_j(0) = \varrho_{j1}$ for the first desired spike according to (2.14).

We will derive a back-propagation-like rule for computing the gradient of the error function (3.1) which can then be minimized using a gradient descent method, e.g.

$$w_{ji}^{(t)} = w_{ji}^{(t-1)} - \alpha \frac{\partial E_1}{\partial w_{ji}} \left(\mathbf{w}^{(t-1)} \right) \quad \text{for } i \in j_{\leftarrow} \cup \{0\}, \tag{3.2}$$

$$d_{ji}^{(t)} = d_{ji}^{(t-1)} - \alpha \frac{\partial E_1}{\partial d_{ji}} \left(\mathbf{d}^{(t-1)} \right) \quad \text{for } i \in j_{\leftarrow} \tag{3.3}$$

for every $j \in V$ 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

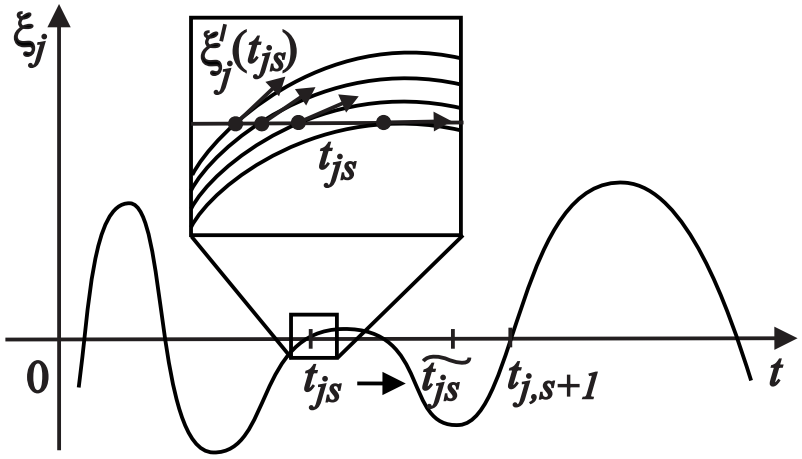

Figure 2.6: The smooth creation and deletion of spikes.

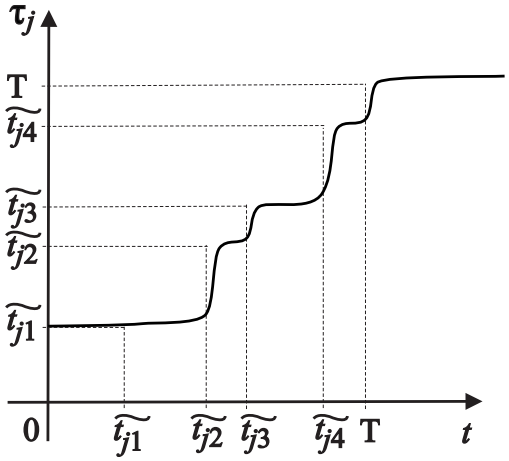

Figure 2.7: The smooth approximation of the stair function.

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 $E_1(\mathbf{w}, \mathbf{d})$ 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

$$\frac{\partial E_1}{\partial w_{ji}} = \sum_{c=1}^{m_j} \left(\pi_{jc} \cdot \frac{\partial}{\partial w_{ji}} \tau_j(u_{jc}) + \pi'_{jc} \cdot \frac{\partial}{\partial w_{ji}} \tau'_j(u_{jc}) \right) \quad \text{for } i \in j_{\leftarrow} \cup \{0\}\,, \tag{3.4}$$

$$\frac{\partial E_1}{\partial d_{ji}} = \sum_{c=1}^{m_j} \left(\pi_{jc} \cdot \frac{\partial}{\partial d_{ji}} \tau_j(u_{jc}) + \pi'_{jc} \cdot \frac{\partial}{\partial d_{ji}} \tau'_j(u_{jc}) \right) \quad \text{for } i \in j_{\leftarrow}\,. \tag{3.5}$$

Notice that the triples $(\pi_{jc_1}, \pi'_{jc_1}, u_{jc_1})$ and $(\pi_{jc_2}, \pi'_{jc_2}, u_{jc_2})$ corresponding to the same time instant $u_{jc_1} = u_{jc_2}$ 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

$$P_j = ((\tau_j(0) - \varrho_{j1}\,,\, 0\,,\, 0)\,;\, (\tau_j(\varrho_{j,s+1}) - \varrho_{js}\,,\, 0\,,\, \varrho_{j,s+1})\,,\, s = 1, \ldots, q_j) \tag{3.6}$$

is obtained directly from (3.1). 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

$$\frac{\partial}{\partial w_{i\ell}}\tau_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}}\tau_j(t) \cdot \frac{\partial \widetilde{t_{js}}}{\partial w_{i\ell}}\,, \tag{3.7}$$

$$\frac{\partial}{\partial w_{i\ell}}\tau_j'(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}}\tau_j'(t) \cdot \frac{\partial \widetilde{t_{js}}}{\partial w_{i\ell}} \tag{3.8}$$

e.g. for some $w_{i\ell}$ at time t where

$$\frac{\partial}{\partial \widetilde{t_{js}}}\tau_j(t) = \begin{cases} P^c\left(t-\widetilde{t_{js}}\right)\left(1-c\lambda\left(\widetilde{t_{js}}-\widetilde{t_{j,s-1}}\right)\left(1-P\left(t-\widetilde{t_{js}}\right)\right)\right) - P^c\left(t-\widetilde{t_{j,s+1}}\right) \\ \qquad\qquad \text{for } s>1\,, \\ 1-P^c\left(t-\widetilde{t_{j2}}\right) \quad \text{for } s=1\,, \end{cases} \tag{3.9}$$

$$\frac{\partial}{\partial \widetilde{t_{js}}}\tau_j'(t) = \begin{cases} c\lambda\left(\left(\left(1-c\lambda\left(\widetilde{t_{js}}-\widetilde{t_{j,s-1}}\right)\left(1-P\left(t-\widetilde{t_{js}}\right)\right)\right)+\lambda\left(\widetilde{t_{js}}-\widetilde{t_{j,s-1}}\right)P\left(t-\widetilde{t_{js}}\right)\right)\right. \\ \quad \left.\times P^c\left(t-\widetilde{t_{js}}\right)\left(1-P\left(t-\widetilde{t_{js}}\right)\right) - P^c\left(t-\widetilde{t_{j,s+1}}\right)\left(1-P\left(t-\widetilde{t_{j,s+1}}\right)\right)\right) \\ \qquad\qquad \text{for } s>1\,, \\ -c\lambda\,P^c\left(t-\widetilde{t_{j2}}\right)\left(1-P\left(t-\widetilde{t_{j2}}\right)\right) \quad \text{for } s=1\,, \end{cases} \tag{3.10}$$

follows from (2.14), (2.17), and (2.8).

Furthermore,

$$\begin{aligned} \frac{\partial \widetilde{t_{js}}}{\partial w_{i\ell}} = \frac{\partial \widetilde{t_{js}}}{\partial \widetilde{t_{j,s+1}}} \cdot \frac{\partial \widetilde{t_{j,s+1}}}{\partial w_{i\ell}} &+ \left(\frac{\partial \widetilde{t_{js}}}{\partial t_{js}} \cdot \frac{\partial t_{js}}{\partial \tau_i} + \frac{\partial \widetilde{t_{js}}}{\partial \xi_j'} \cdot \frac{\partial \xi_j'}{\partial \tau_i}\right) \frac{\partial}{\partial w_{i\ell}}\tau_i(t_{js}-d_{ji}) \\ &+ \frac{\partial \widetilde{t_{js}}}{\partial \xi_j'} \cdot \frac{\partial \xi_j'}{\partial \tau_i'} \cdot \frac{\partial}{\partial w_{i\ell}}\tau_i'(t_{js}-d_{ji}) \end{aligned} \tag{3.11}$$

according to (2.13) and (2.15) where

$$\frac{\partial \widetilde{t_{js}}}{\partial \widetilde{t_{j,s+1}}} = \begin{cases} \frac{\partial}{\partial \beta}\sigma\left(\delta-\xi_j'\left(t_{js}\right)\right) & \text{for } s<p_j\,, \\ 0 & \text{for } s=p_j\,, \end{cases} \tag{3.12}$$

$$\frac{\partial \widetilde{t_{js}}}{\partial t_{js}} = \left(\frac{\partial}{\partial \alpha} - \xi_j''\left(t_{js}\right)\cdot\frac{\partial}{\partial x}\right)\sigma\left(t_{js}, \widetilde{t_{j,s+1}}, \delta\,;\, \delta-\xi_j'(t_{js})\right) \tag{3.13}$$

can be evaluated using (2.6), (2.5), and (2.3), whereas

$$\begin{aligned} \xi_j''(t) = \sum_{i\in j_{\leftarrow}} w_{ji}\Big(&\varepsilon''\left(t-d_{ji}-\tau_i\left(t-d_{ji}\right)\right)\left(1-\tau_i'\left(t-d_{ji}\right)\right)^2 \\ &-\varepsilon'\left(t-d_{ji}-\tau_i\left(t-d_{ji}\right)\right)\tau_i''\left(t-d_{ji}\right)\Big) \end{aligned} \tag{3.14}$$

is derived from (2.15) in which

$$\varepsilon''(t) = e^{-(t-1)^2}\left(\sigma_0''(t) - 4(t-1)\sigma_0'(t) + \left(4(t-1)^2-2\right)\sigma_0(t)\right)\,, \tag{3.15}$$

$$\begin{aligned} \tau_i''(t) = \frac{\partial^2}{\partial t^2}\tau_i(t) = c\lambda^2 \sum_{s=2}^{p_i+1} &\left(\widetilde{t_{is}}-\widetilde{t_{i,s-1}}\right)P^c\left(t-\widetilde{t_{is}}\right)\left(1-P\left(t-\widetilde{t_{is}}\right)\right) \\ &\times\left(c\left(1-P\left(t-\widetilde{t_{is}}\right)\right)-P\left(t-\widetilde{t_{is}}\right)\right) \end{aligned} \tag{3.16}$$

can be calculated from (2.16) and (2.17) using (2.12), (2.3), (2.4), and (2.8). In addition,

$$\frac{\partial \widetilde{t_{js}}}{\partial \xi_j'} = -\sigma'\left(t_{js}, \widetilde{t_{j,s+1}}, \delta\,;\, \delta-\xi_j'(t_{js})\right)\,, \tag{3.17}$$

$$\frac{\partial\,\xi_j'}{\partial\,\tau_i} = -w_{ji}\,\varepsilon''(t_{js}-d_{ji}-\tau_i(t_{js}-d_{ji}))(1-\tau_i'(t_{js}-d_{ji}))\,, \tag{3.18}$$

$$\frac{\partial\,\xi_j'}{\partial\,\tau_i'} = -w_{ji}\,\varepsilon'(t_{js}-d_{ji}-\tau_i(t_{js}-d_{ji})). \tag{3.19}$$

Finally, we calculate the partial derivative $\frac{\partial\,t_{js}}{\partial\,\tau_i}$ for $i \in j_{\leftarrow}$ which also appears in (3.11). According to (2.9) and (2.10), the dependence of t_{js} on τ_i can only be expressed implicitly as $\xi_j(t_{js}) = 0$ which implies the total differential identity

$$\xi_j'(t_{js})\,dt_{js} + \frac{\partial\,\xi_j}{\partial\,w_{i\ell}}\,dw_{i\ell} = 0 \tag{3.20}$$

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,

$$\xi_j'(t_{js}) \cdot \frac{\partial\,t_{js}}{\partial\,w_{i\ell}} = -\frac{\partial\,\xi_j}{\partial\,w_{i\ell}} = -\frac{\partial\,\xi_j}{\partial\,\tau_i} \cdot \frac{\partial\,\tau_i}{\partial\,w_{i\ell}} \tag{3.21}$$

which gives

$$\frac{\partial\,t_{js}}{\partial\,\tau_i} = \frac{\frac{\partial\,t_{js}}{\partial\,w_{i\ell}}}{\frac{\partial\,\tau_i}{\partial\,w_{i\ell}}} = \frac{-\frac{\partial\,\xi_j}{\partial\,\tau_i}}{\xi_j'(t_{js})} = \frac{w_{ji}\,\varepsilon'(t_{js}-d_{ji}-\tau_i(t_{js}-d_{ji}))}{\xi_j'(t_{js})} \tag{3.22}$$

where $\xi_j'(t_{js}) \neq 0$ according to (2.10).

Now, formula (3.11) 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 (3.7) and (3.8) as follows:

$$\begin{aligned}\frac{\partial}{\partial\,w_{i\ell}}\tau_j(t) &= \sum_{s=1}^{p_j} \frac{\partial}{\partial\,\widetilde{t_{js}}}\tau_j(t) \sum_{r=s}^{s+n_{js}} \left(\prod_{q=s}^{r-1} \frac{\partial\,\widetilde{t_{jq}}}{\partial\,\widetilde{t_{j,q+1}}}\right) \left(\left(\frac{\partial\,\widetilde{t_{jr}}}{\partial\,t_{jr}} \cdot \frac{\partial\,t_{jr}}{\partial\,\tau_i} + \frac{\partial\,\widetilde{t_{jr}}}{\partial\,\xi_j'} \cdot \frac{\partial\,\xi_j'}{\partial\,\tau_i}\right) \frac{\partial}{\partial\,w_{i\ell}}\tau_i(t_{jr}-d_{ji}) \right.\\ &\qquad \left. + \frac{\partial\,\widetilde{t_{jr}}}{\partial\,\xi_j'} \cdot \frac{\partial\,\xi_j'}{\partial\,\tau_i'} \cdot \frac{\partial}{\partial\,w_{i\ell}}\tau_i'(t_{jr}-d_{ji})\right)\,, \end{aligned} \tag{3.23}$$

$$\begin{aligned}\frac{\partial}{\partial\,w_{i\ell}}\tau_j'(t) &= \sum_{s=1}^{p_j} \frac{\partial}{\partial\,\widetilde{t_{js}}}\tau_j'(t) \sum_{r=s}^{s+n_{js}} \left(\prod_{q=s}^{r-1} \frac{\partial\,\widetilde{t_{jq}}}{\partial\,\widetilde{t_{j,q+1}}}\right) \left(\left(\frac{\partial\,\widetilde{t_{jr}}}{\partial\,t_{jr}} \cdot \frac{\partial\,t_{jr}}{\partial\,\tau_i} + \frac{\partial\,\widetilde{t_{jr}}}{\partial\,\xi_j'} \cdot \frac{\partial\,\xi_j'}{\partial\,\tau_i}\right) \frac{\partial}{\partial\,w_{i\ell}}\tau_i(t_{jr}-d_{ji}) \right.\\ &\qquad \left. + \frac{\partial\,\widetilde{t_{jr}}}{\partial\,\xi_j'} \cdot \frac{\partial\,\xi_j'}{\partial\,\tau_i'} \cdot \frac{\partial}{\partial\,w_{i\ell}}\tau_i'(t_{jr}-d_{ji})\right) \end{aligned} \tag{3.24}$$

where $0 \leq n_{js} \leq p_j - s$ is defined to be the least index such that

$$\frac{\partial\,\widetilde{t_{j,s+n_{js}}}}{\partial\,\widetilde{t_{j,s+n_{js}+1}}} = 0 \tag{3.25}$$

which exists since at least $\frac{\partial\,\widetilde{t_{j,p_j}}}{\partial\,\widetilde{t_{j,p_j+1}}} = 0$ according to (3.12). Note that the product $\prod_{q=s}^{r-1} \frac{\partial\,\widetilde{t_{jq}}}{\partial\,\widetilde{t_{j,q+1}}}$ in formulas (3.23) and (3.24) equals formally 1 for $r = s$. The summands of formulas (3.23) and (3.24) are used for creating the list P_i for a hidden neuron $i \in V \setminus (X \cup Y)$ so that conditions (3.4) and (3.5) hold for w_{ji} and d_{ji} replaced with $w_{i\ell}$ and $d_{i\ell}$, respectively, 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

$$P_i = \left(f_{jcsr}\left(\frac{\partial\,\widetilde{t_{jr}}}{\partial\,t_{jr}} \cdot \frac{\partial\,t_{jr}}{\partial\,\tau_i} + \frac{\partial\,\widetilde{t_{jr}}}{\partial\,\xi_j'} \cdot \frac{\partial\,\xi_j'}{\partial\,\tau_i}\right)\,,\, f_{jcsr} \cdot \frac{\partial\,\widetilde{t_{jr}}}{\partial\,\xi_j'} \cdot \frac{\partial\,\xi_j'}{\partial\,\tau_i'}\,,\, t_{jr} - d_{ji}\right) \tag{3.26}$$

including factors

$$f_{jcsr} = \left(\pi_{jc} \cdot \frac{\partial}{\partial\,\widetilde{t_{js}}}\tau_j(u_{jc}) + \pi_{jc}' \cdot \frac{\partial}{\partial\,\widetilde{t_{js}}}\tau_j'(u_{jc})\right) \prod_{q=s}^{r-1} \frac{\partial\,\widetilde{t_{jq}}}{\partial\,\widetilde{t_{j,q+1}}} \tag{3.27}$$

for all $j \in i^{\rightarrow}$, $c = 1, \ldots, m_j$, $s = 1, \ldots, p_j$, and $r = s, \ldots, s + n_{js}$, according to the chain rule, where the underlying derivatives can be computed by using formulas (3.9), (3.10), (3.12), (3.13), (3.17)–(3.19), and (3.22).

Thus, the back-propagation algorithm starts with output neurons $j \in Y$ for which the lists P_j are first created by (3.6) 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 (3.26). These lists are then used for computing the gradient of the error function (3.1) according to (3.4) and (3.5) where the derivatives

$$\frac{\partial}{\partial w_{ji}} \tau_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}} \tau_j(t) \sum_{r=s}^{s+n_{js}} \left(\prod_{q=s}^{r-1} \frac{\partial \widetilde{t_{jq}}}{\partial \widetilde{t_{j,q+1}}} \right) \left(\frac{\partial \widetilde{t_{jr}}}{\partial t_{jr}} \cdot \frac{\partial t_{jr}}{\partial w_{ji}} + \frac{\partial \widetilde{t_{jr}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial w_{ji}} \right), \tag{3.28}$$

$$\frac{\partial}{\partial w_{ji}} \tau'_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}} \tau'_j(t) \sum_{r=s}^{s+n_{js}} \left(\prod_{q=s}^{r-1} \frac{\partial \widetilde{t_{jq}}}{\partial \widetilde{t_{j,q+1}}} \right) \left(\frac{\partial \widetilde{t_{jr}}}{\partial t_{jr}} \cdot \frac{\partial t_{jr}}{\partial w_{ji}} + \frac{\partial \widetilde{t_{jr}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial w_{ji}} \right), \tag{3.29}$$

$$\frac{\partial}{\partial d_{ji}} \tau_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}} \tau_j(t) \sum_{r=s}^{s+n_{js}} \left(\prod_{q=s}^{r-1} \frac{\partial \widetilde{t_{jq}}}{\partial \widetilde{t_{j,q+1}}} \right) \left(\frac{\partial \widetilde{t_{jr}}}{\partial t_{jr}} \cdot \frac{\partial t_{jr}}{\partial d_{ji}} + \frac{\partial \widetilde{t_{jr}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial d_{ji}} \right), \tag{3.30}$$

$$\frac{\partial}{\partial d_{ji}} \tau'_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}} \tau'_j(t) \sum_{r=s}^{s+n_{js}} \left(\prod_{q=s}^{r-1} \frac{\partial \widetilde{t_{jq}}}{\partial \widetilde{t_{j,q+1}}} \right) \left(\frac{\partial \widetilde{t_{jr}}}{\partial t_{jr}} \cdot \frac{\partial t_{jr}}{\partial d_{ji}} + \frac{\partial \widetilde{t_{jr}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial d_{ji}} \right) \tag{3.31}$$

are calculated analogously to (3.23) and (3.24). Moreover, the dependencies of t_{jr} on w_{ji} and d_{ji} can again be expressed only implicitly as $\xi_j(t_{jr}) = 0$ according to (2.9) and (2.10) which gives

$$\frac{\partial t_{jr}}{\partial w_{ji}} = -\frac{\frac{\partial \xi_j}{\partial w_{ji}}}{\frac{\partial \xi_j}{\partial t_{jr}}} = \begin{cases} -\frac{1}{\xi'_j(t_{jr})} & \text{for } i = 0 \\ -\frac{\varepsilon(t_{jr} - d_{ji} - \tau_i(t_{jr} - d_{ji}))}{\xi'_j(t_{jr})} & \text{for } i \in j_{\leftarrow}, \end{cases} \tag{3.32}$$

$$\frac{\partial t_{jr}}{\partial d_{ji}} = -\frac{\frac{\partial \xi_j}{\partial d_{ji}}}{\frac{\partial \xi_j}{\partial t_{jr}}} = \frac{w_{ji}\, \varepsilon'\left(t_{jr} - d_{ji} - \tau_i(t_{jr} - d_{ji})\right) \left(1 - \tau'_i(t_{jr} - d_{ji})\right)}{\xi'_j(t_{jr})} \tag{3.33}$$

by the implicit function theorem. In addition,

$$\frac{\partial \xi'_j}{\partial w_{ji}} = \begin{cases} 0 & \text{for } i = 0 \\ \varepsilon'(t_{jr} - d_{ji} - \tau_i(t_{jr} - d_{ji}))(1 - \tau'_i(t_{jr} - d_{ji})) & \text{for } i \in j_{\leftarrow}, \end{cases} \tag{3.34}$$

$$\begin{aligned} \frac{\partial \xi'_j}{\partial d_{ji}} = w_{ji} \Big(& \varepsilon'(t_{jr} - d_{ji} - \tau_i(t_{jr} - d_{ji}))\tau''_i(t_{jr} - d_{ji}) \\ & -\varepsilon''(t_{jr} - d_{ji} - \tau_i(t_{jr} - d_{ji}))(1 - \tau'_i(t_{jr} - d_{ji}))^2 \Big) \end{aligned} \tag{3.35}$$

which completes the calculation of the gradient of the error function.

4 Additional Penalty Term

Experiments with the back-propagation algorithm introduced in Section 3 have shown that the error function (3.1) can reach a local minimum at which transformed firing times $\widetilde{t_{js}}$ are far from corresponding rising or vanishing spikes t_{js}, which occurs when $\xi'(t_{js}) < \delta$ according to (2.13). In this case, the actual sequences of output spikes do not coincide with the desired ones. In order to avoid this pathology, we introduce an additional penalty term E_2 which, being minimized, forces $\xi'(t_{js}) \geq \delta$. In particular, the total error is defined as

$$E(\mathbf{w}, \mathbf{d}) = E_1(\mathbf{w}, \mathbf{d}) + E_2(\mathbf{w}, \mathbf{d}) \tag{4.1}$$

where $E_1(\mathbf{w}, \mathbf{d})$ is the original error function (3.1) and

$$E_2(\mathbf{w}, \mathbf{d}) = -\sum_{j \in Y} \sum_{s=1}^{p_j} \sigma(0, \delta, \delta\,;\, \xi'_j(t_{js})) \tag{4.2}$$

exploits function (2.2). Thus, the gradient of the total error (4.1) which can be used in learning rule (3.2) and (3.3) for E_1 replaced with E, can be expressed as

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E_1}{\partial w_{ji}} + \frac{\partial E_2}{\partial w_{ji}}\,, \qquad \frac{\partial E}{\partial d_{ji}} = \frac{\partial E_1}{\partial d_{ji}} + \frac{\partial E_2}{\partial d_{ji}} \tag{4.3}$$

where the gradient of the original error function (3.1) is computed by the back-propagation algorithm described in Section 3 while the formulas for the partial derivatives of E_2 with respect to weight and delay parameters are derived in the following.

For parameters w_{ji} and d_{ji} associated with an output neuron $j \in Y$, we simply differentiate (4.2):

$$\frac{\partial E_2}{\partial w_{ji}} = -\sum_{s=1}^{p_j} \sigma'(0,\delta,\delta\,;\,\xi_j'(t_{js})) \cdot \left(\frac{\partial \xi_j'}{\partial w_{ji}} + \xi_j''(t_{js}) \cdot \frac{\partial t_{js}}{\partial w_{ji}}\right) \tag{4.4}$$

$$\frac{\partial E_2}{\partial d_{ji}} = -\sum_{s=1}^{p_j} \sigma'(0,\delta,\delta\,;\,\xi_j'(t_{js})) \cdot \left(\frac{\partial \xi_j'}{\partial d_{ji}} + \xi_j''(t_{js}) \cdot \frac{\partial t_{js}}{\partial d_{ji}}\right) \tag{4.5}$$

which can be evaluated using (2.3), (3.34), (3.14), (3.32), (3.35), and (3.33).

The partial derivates of E_2 with respect to $w_{i\ell}$ and $d_{i\ell}$ for hidden neurons $i \in V \setminus (X \cup Y)$, are calculated recursively. First consider neuron i such that $i \in j_{\leftarrow}$ for some output neuron $j \in Y$, for which the underlying derivatives $\frac{\partial E_2}{\partial w_{i\ell}}$ for $\ell \in i_{\leftarrow} \cup \{0\}$ and $\frac{\partial E_2}{\partial d_{i\ell}}$ for $\ell \in i_{\leftarrow}$ are below reduced to $\frac{\partial}{\partial w_{i\ell}}\tau_i(t_{js} - d_{ji})$, $\frac{\partial}{\partial w_{i\ell}}\tau_i'(t_{js} - d_{ji})$ and $\frac{\partial}{\partial d_{i\ell}}\tau_i(t_{js} - d_{ji})$, $\frac{\partial}{\partial d_{i\ell}}\tau_i'(t_{js} - d_{ji})$, respectively, which can be computed using (3.28)–(3.31):

$$\frac{\partial E_2}{\partial w_{i\ell}} = -\sum_{j\in Y}\sum_{s=1}^{p_j} \sigma'(0,\delta,\delta;\xi_j'(t_{js})) \cdot \frac{\partial \xi_j'(t_{js})}{\partial w_{i\ell}}\,, \quad \frac{\partial E_2}{\partial d_{i\ell}} = -\sum_{j\in Y}\sum_{s=1}^{p_j} \sigma'(0,\delta,\delta;\xi_j'(t_{js})) \cdot \frac{\partial \xi_j'(t_{js})}{\partial d_{i\ell}} \tag{4.6}$$

where

$$\frac{\partial \xi_j'(t_{js})}{\partial w_{i\ell}} = \left(\frac{\partial \xi_j'}{\partial \tau_i} + \xi_j''(t_{js}) \cdot \frac{\partial t_{js}}{\partial \tau_i}\right) \frac{\partial}{\partial w_{i\ell}}\tau_i(t_{js} - d_{ji}) + \frac{\partial \xi_j'}{\partial \tau_i'} \cdot \frac{\partial}{\partial w_{i\ell}}\tau_i'(t_{js} - d_{ji})\,, \tag{4.7}$$

$$\frac{\partial \xi_j'(t_{js})}{\partial d_{i\ell}} = \left(\frac{\partial \xi_j'}{\partial \tau_i} + \xi_j''(t_{js}) \cdot \frac{\partial t_{js}}{\partial \tau_i}\right) \frac{\partial}{\partial d_{i\ell}}\tau_i(t_{js} - d_{ji}) + \frac{\partial \xi_j'}{\partial \tau_i'} \cdot \frac{\partial}{\partial d_{i\ell}}\tau_i'(t_{js} - d_{ji})\,. \tag{4.8}$$

By comparing formulas (4.6), (4.7) and (4.8) with (3.4) and (3.5), we can extend list P_i of triples $(\pi_{ic}, \pi_{ic}', u_{ic})$ introduced for back-propagation rule in Section 3 by the following items corresponding to error E_2:

$$\left(-\sigma'(0,\delta,\delta\,;\,\xi_j'(t_{js})) \cdot \left(\frac{\partial \xi_j'}{\partial \tau_i} + \xi_j''(t_{js}) \cdot \frac{\partial t_{js}}{\partial \tau_i}\right)\,,\, -\sigma'(0,\delta,\delta\,;\,\xi_j'(t_{js})) \cdot \frac{\partial \xi_j'}{\partial \tau_i'}\,,\, t_{js} - d_{ji}\right) \tag{4.9}$$

for every $j \in Y \cap i^{\rightarrow}$, $s = 1, \ldots, p_j$ according to the chain rule and (4.3), which can be evaluated using (2.3), (2.15), (3.18), (3.14), (3.22), and (3.19). In this way, the recursive computation of the partial derivatives of E_2 with respect to $w_{i\ell}$ and $d_{i\ell}$ for any hidden neuron $i \in V \setminus (X \cup Y)$, is integrated in the back-propagation algorithm described in Section 3.

5 Experiments and Conclusion

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. The first experimental results confirmed the smoothness of spike creation/deletion. For example, Figure 5.1 from [18] where a slightly different model was used (cf. Footnote 2) 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 (cf. Figure 2.7). Figure 5.1 depicts the process of a spike creation during training when the time instant of a new spike $\widetilde{t_{js}}$ "detaches" from the preceding[4] spike $t_{j,s-1}$ (which,

[4]In this report we use the succeeding spike for the split in contrast to [18].

for 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.

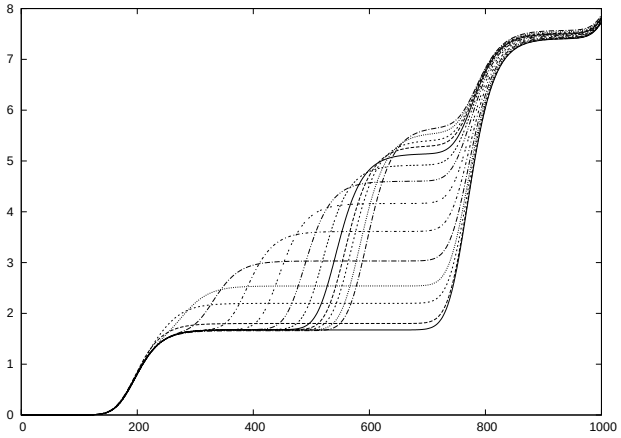

Figure 5.1: Spike creation.

Nevertheless, the experiments have also shown that the gradient method get often stuck in local minima corresponding to transient network configurations which emerge from the smooth approximation of discrete events. In Section 4 we have proposed a solution to this problem which is based on an additional penalty term introduced in the error function. However, this approach did not help to avoid the underlying side effect, which disqualifies our learning algorithm for networks of smoothly spiking neurons.

Acknowledgments

I would like to thank Petr Savický for his idea of expressing the condition on the derivatives by the primitive function (2.1) and of using the total differential identity (3.20), to Petra Vidnerová for her help with computer experiments, and to Eva Pospíšilová for drawing the figures.

Bibliography

[1] Berkovec, K.: Learning in networks of spiking neurons. (in Czech) M.Sc. Thesis, Faculty of Mathematics and Physics, Charles University, Prague, 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] Bohte, S.M., La Poutré, H., Kok, J.N.: Unsupervised clustering with spiking neurons by sparse temporal coding and multi-layer RBF networks. IEEE Transactions on Neural Networks **13** (2) (2002) 426–435

[4] Fiete, I.R., Seung, H.S.: Gradient learning in spiking neural networks by dynamic perturbation of conductances. Physical Review Letters **97**, 048104 (2006)

[5] Gerstner, W., Kistler, W.M.: Spiking Neuron Models: Single Neurons, Populations, Plasticity. Cambridge University Press, Cambridge, UK (2002)

[6] Haykin, S.: Neural Networks: A Comprehensive Foundation. (2nd ed.) Prentice-Hall, Upper Saddle River, NJ (1999)

[7] 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

[8] Kempter, R., Gerstner, W., van Hemmen, J.L.: Hebbian learning and spiking neurons. Physical Review E **59** (4) (1999) 4498–4514

[9] Maass, W., Bishop, C.M. (Eds.): Pulsed Neural Networks. MIT Press, Cambridge, MA (1999)

[10] Maass, W., Schmitt, M.: On the complexity of learning for spiking neurons with temporal coding. Information and Computation **153** (1) (1999) 26–46

[11] Moore, S.C.: Back-propagation in spiking neural networks. M.Sc. Thesis, Department of Computer Science, University of Bath, UK (2002)

[12] Rosenblatt, F.: The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review **65** (6) (1958) 386–408

[13] Ruf, B., Schmitt, M.: Hebbian learning in networks of spiking neurons using temporal coding. Proceedings of the IWANN'97 International Work-Conference on Artificial and Natural Neural Networks, Lanzarote, Canary Islands, Spain, LNCS 1240, Springer Verlag, Berlin (1997) 380–389

[14] Ruf, B., Schmitt, M.: Self-organizing maps of spiking neurons using temporal coding, In Bower, J.M. (ed.): Computational Neuroscience: Trends in Research. Plenum Press, New York (1998) 509–514

[15] Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature **323** (1986) 533–536

[16] 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

[17] 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

[18] Šíma, J.: Gradient learning in networks of smoothly spiking neurons (revised version). Technical Report V-1045, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, Czech Republic (2009)

[19] Šíma, J., Sgall, J.: On the non-learnability of a single spiking neuron. Neural Computation **17** (12) (2005) 2635–2647

[20] Thorpe, S., Delorme, A., Van Rullen, R.: Spike-based strategies for rapid processing. Neural Networks **14** (6-7) (2001) 715–725

[21] Tiňo, P., Mills, A.J.S.: Learning beyond finite memory in recurrent networks of spiking neurons. Neural Computation **18** (3) (2006) 591–613

[22] 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