Hindawi Publishing Corporation
Advances in Artificial Neural Systems
Volume 2012, Article ID 713581, 16 pages
doi:10.1155/2012/713581

Research Article

A Radial Basis Function Spike Model for Indirect Learning via Integrate-and-Fire Sampling and Reconstruction Techniques

X. Zhang,[1] G. Foderaro,[1] C. Henriquez,[2] A. M. J. VanDongen,[3] and S. Ferrari[1]

[1] *Laboratory for Intelligent Systems and Control (LISC), Department of Mechanical Engineering and Materials Science, Duke University, Durham, NC 27708, USA*
[2] *Department of Biomedical Engineering and Department of Computer Science, Duke University Durham, NC 27708, USA*
[3] *Program in Neuroscience & Behavioral Disorders, Duke-NUS Graduate Medical School, Singapore, Singapore*

Correspondence should be addressed to S. Ferrari, sferrari@duke.edu

Received 17 February 2012; Accepted 21 May 2012

Academic Editor: Olivier Bastien

This paper presents a deterministic and adaptive spike model derived from radial basis functions and a leaky integrate-and-fire sampler developed for training spiking neural networks without direct weight manipulation. Several algorithms have been proposed for training spiking neural networks through biologically-plausible learning mechanisms, such as spike-timing-dependent synaptic plasticity and Hebbian plasticity. These algorithms typically rely on the ability to update the synaptic strengths, or weights, directly, through a weight update rule in which the weight increment can be decided and implemented based on the training equations. However, in several potential applications of adaptive spiking neural networks, including neuroprosthetic devices and CMOS/memristor nanoscale neuromorphic chips, the weights cannot be manipulated directly and, instead, tend to change over time by virtue of the pre- and postsynaptic neural activity. This paper presents an indirect learning method that induces changes in the synaptic weights by modulating spike-timing-dependent plasticity by means of controlled input spike trains. In place of the weights, the algorithm manipulates the input spike trains used to stimulate the input neurons by determining a sequence of spike timings that minimize a desired objective function and, indirectly, induce the desired synaptic plasticity in the network.

1. Introduction

This paper presents a deterministic and adaptive spike model obtained from radial basis functions (RBFs) and a leaky integrate-and-fire (LIF) sampler for the purpose of training spiking neural networks (SNNs), without directly manipulating the synaptic weights. Spiking neural networks are computational models of biological neurons comprised of systems of differential equations that can reproduce some of the spike patterns and dynamics observed in real neuronal networks [1, 2]. Recently, SNNs have also been shown capable of simulating sigmoidal artificial neural networks (ANNs) and of solving small-dimensional nonlinear function approximation problems through reinforcement learning [3–5]. Like all ANN learning techniques, existing SNN training algorithms rely on the direct manipulation of the synaptic weights [4–9]. In other words, the learning algorithms typically include a weight-update rule by which the synaptic weights are updated over several iterations, based on the reinforcement signal or network performance.

In many potential SNN applications, including neuroprosthetic devices, light-sensitive neuronal networks grown *in vitro*, and CMOS/memristor nanoscale neuromorphic chips [10], the synaptic weights cannot be updated directly by the learning algorithm. In several of these applications, the objective is to stimulate a network of biological or artificial spiking neurons to perform a complex function, such as processing an auditory signal or restoring a cognitive function. In neuroprosthetic medical implants, for example, the artificial device may consist of a microelectrode array or integrated circuit that stimulates biological neurons via spike trains. Therefore, the device is not capable of directly modifying the synaptic efficacies of the biological neurons, as do existing SNN training algorithms, but it is capable of

stimulating a subset of neurons through controlled pulses of electrical current.

As another example, light-sensitive neuronal networks grown *in-vitro* can be similarly stimulated through controlled light patterns that cause selected neurons to fire at precise moments in time, in an attempt to induce plasticity, while their output is being recorded in real time using a multi-electrode array (MEA) [11]. In this case, a digital computer can be used to determine the desired stimulation patterns for an *in-vitro* neuronal network with random connectivity, produced by culturing dissociated cortical neurons derived from embryonic day E18 rat brain [11, 12]. As a result, the cultures may be used to verify biophysical models of the mechanisms by which biological neuronal networks execute the control and storage of information via temporal coding and learn to solve complex tasks over time. In these networks, the actual connectivity and synaptic plasticities are typically unknown and cannot be manipulated directly as required by existing SNN learning algorithms.

This paper presents a novel indirect learning approach and algorithm that assume synaptic weights cannot be updated or manipulated at any time. The learning algorithm induces changes in the SNN weights by modulating spike-timing dependent plasticity (STDP) through controlled input spike trains. The algorithm adapts the input spike trains that are used to stimulate the input neurons of the SNN and, thus, are realizable through controlled pulses of electric voltage or controlled pulses of blue light. The main difficulty to be overcome in indirect learning is that the algorithm aims at adapting pulse signals, such as square waves, in place of continuous-valued weights. While available in closed analytic form, these signals typically are represented by piece-wise continuous, multi-valued (or many-to-one), and nondifferentiable functions that are difficult to adapt or update using optimization or reinforcement-learning algorithms. Furthermore, stimulation patterns typically are generated by spike models that are stochastic, such as the Poisson spike model [5, 13–15]. Thus, even when the spike model is optimized, it does not allow for precise timing of pre- and post-synaptic firings, and as a result, may induce undesirable changes in the synaptic weights.

In this paper, a deterministic spike model that allows for precise timing of neuron firings is obtained using adaptive RBFs to model the characteristics of the spike pattern through a continuous and infinitely differentiable function that also is one to one. The RBF model is combined with an LIF sampling technique originally developed in [16, 17] for the approximate reconstruction of bandlimited functions. It is shown that, by this approach, the spike trains generated by the LIF sampler display the precise characteristics specified by the RBF model. Furthermore, this deterministic spike model can be optimized to modulate STDP through controlled input spike trains that bring about the desired SNN weight change without direct manipulation. The indirect learning approach presented in this paper is applicable both in supervised and unsupervised settings. In fact, when the desired SNN output is unknown, it can be replaced by a reinforcement signal produced by a critic SNN, as shown by the adaptive critic method reviewed in Section 3.

The paper is organized as follows. The model of spiking neural network used to derive and demonstrate the training equations is presented in Section 2. In Section 3, an adaptive critic approach is described to illustrate how the proposed indirect training methodology can be applied using reinforcement learning, for example, to model or control a dynamical plant. The novel spike model and indirect training methodology are presented in Section 4 and demonstrated on a benchmark problem involving a two-node spiking neural network. The generalized form of gradient equations for indirect training is presented in Section 5 and demonstrated through a three-node spiking neural network. These gradient equations show how the methodology can be generalized to any spiking neural network with the characteristics described in the next section.

2. Spiking Neural Network Model

2.1. Models of Neuron and Synapse. Various models of SNNs have been proposed in recent years, motivated by biological studies that have shown complex spike patterns and dynamics to be an essential component of information processing and learning in the brain. The two crucial considerations involved in determining a suitable SNN model are the range of neurocomputational behaviors the model can reproduce, and its computational efficiency [18]. As can be expected, the implementation efficiency typically increases with the number of features and behaviors that can be accurately reproduced [18], such that each model offers a tradeoff between these competing objectives. One of the computational neuron models that is most biophysically accurate is the well-known Hodgkin-Huxley (HH) model [2]. Due to its extremely low computational efficiency, however, using the HH model to simulate large networks of neurons can be computationally prohibitive [18]. Recently, bifurcation studies have been used to reduce the HH dynamics from four to two differential equations, referred to as the Izhikevich model, which are capable of reproducing a wide range of spiking patterns and behaviors with much higher efficiency than the HH model [19].

In [15], the authors proposed an indirect training method based on a Poison spike model and demonstrated it on a network of Izhikevich neurons. In this work it was found that the adaptive critic architecture described in Section 3 could be implemented without modeling the neuron response in closed form. However, the effectiveness of the approach in [15] was limited in that, due to the use of a stochastic Poison model, the Izhikevich SNN could not converge to the optimal control law. Therefore, in this paper, a new deterministic spike model is proposed and implemented by deriving the training equations in closed form, using a leaky integrate-and-fire (LIF) SNN. The LIF is the simplest model of spiking neuron. It has the advantages that it displays the highest computational efficiency and is amenable to mathematical analysis [13, 14].

The LIF membrane potential, $v(t)$, is governed by

$$C_m \frac{dv(t)}{dt} = I_{\text{leak}}(t) + I_s(t) + I_{\text{inj}}(t), \tag{1}$$

where C_m is the membrane capacitance, $I_{\text{leak}}(t)$ is the current due to the leak of the membrane, $I_s(t)$ is the synaptic input to the neuron, and $I_{\text{inj}}(t)$ is the current injected to the neuron [20]. The leak current is defined as follow:

$$I_{\text{leak}}(t) = -\frac{C_m}{\tau_m}[v(t) - V_0], \tag{2}$$

where V_0 is the resting potential and τ_m is the passive-membrane time constant [20]. τ_m is related to the capacitance, C_m, and the membrane resistance, R_m, of the membrane potential by $\tau_m = R_m C_m$.

The response of the membrane potential is obtained by solving the differential equation (1) analytically for $v(t)$, such that

$$v(t) = V_0 + e^{-t/\tau_m} \int_{t_0}^{t} \frac{I_{\text{inj}}(\rho)}{C_m} e^{\rho/\tau_m} d\rho, \tag{3}$$

where ρ is a dummy variable used for integration and t_0 is the time at which the membrane potential equals V_0 [20]. Whenever the membrane potential, $v(t)$, reaches a prescribed threshold value, V_{th}, the neuron will fire. At this point, we have considered an isolated neuron that is stimulated by an external current, $I_{\text{inj}}(t)$. When the LIF neuron (1) is part of a larger network, the input current, referred to as the *synaptic current*, is generated by the activity of presynaptic neurons.

Let the synaptic current, denoted by $I_s(t)$, be modeled as the sum of the currents of *excitatory* presynaptic neurons and of *inhibitory* presynaptic neurons,

$$I_s(t) = C_m \sum_{k=1}^{N_E} a_{E,k} S_{E,k}(t) + C_m \sum_{k=1}^{N_I} a_{I,k} S_{I,k}(t), \tag{4}$$

where the subscript E denotes inputs from *excitatory* neurons, while the subscript I denotes inputs from *inhibitory* neurons. The amplitudes, $a_{E,k} > 0$ and $a_{I,k} < 0$, represent the change in potential due to a single synaptic event and depend on the weight of the synapse. N_E and N_I are the numbers of excitatory and inhibitory current synapses, respectively. $S_{E,k}$ and $S_{I,k}$ describe the excitatory and inhibitory synaptic inputs as a series of input spikes to each synapse. The synaptic inputs are modeled as a sum of instantaneous impulse functions,

$$S_{E,k}(t) = \sum_{t_{E,k}} \delta(t - t_{E,k}), \tag{5}$$

$$S_{I,k}(t) = \sum_{t_{I,k}} \delta(t - t_{I,k}), \tag{6}$$

where $t_{E,k}$ and $t_{I,k}$ are the firing times of the presynaptic neurons and δ represents the Dirac delta function [21].

In addition to the above deterministic properties, there are prevalent stochastic qualities in biological neural networks caused by effects such as thermal noise and random variations in neurotransmitters. For simplicity, these effects are assumed negligible in this paper. However, the reader is referred to [15] for a technique that can be used to incorporate these effects in the above SNN model.

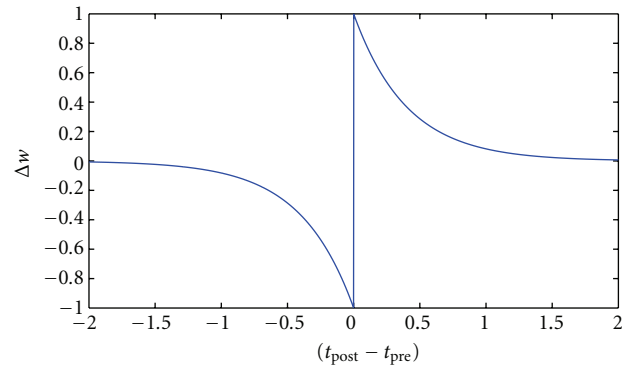


Figure 1: STDP term as a function of the time delay between the last spike of postsynaptic neuron and presynaptic neuron.

2.2. Model of Spike-Timing-Dependent Plasticity (STDP). A persistent learning mechanism known as spike-timing-dependent plasticity, recently observed in biological neuronal networks, is used in this paper to model synaptic plasticity in the LIF SNN. Synaptic plasticity refers to the mechanism by which the synaptic efficacies or *strengths* between neurons are modified over time, typically as a result of the neuronal activity. These changes are known to be driven in part by the correlated activity of adjacently connected neurons. The directions and magnitudes of the changes are dependent on the relative timings of the presynaptic spike arrivals and postsynaptic firings. For simplicity, in this paper, all changes in synaptic strengths are assumed to occur solely as a result of the spike-timing-dependent plasticity mechanism. The approach presented in [5] can be used to also incorporate a model of the Hebbian plasticity.

Spike-timing-dependent plasticity (STDP) is known to modify the synaptic strengths according to the relative timing of the output and input action potentials, or spikes, of a particular neuron. If the presynaptic neuron fires shortly before the postsynaptic neuron, the strength of the connection will be increased. In contrast, if the presynaptic neuron fires after the postsynaptic neuron, the strength of the connection will be decreased, as illustrated in Figure 1. Two constants τ_+ and τ_- determine the ranges of the presynaptic to postsynaptic interspike intervals over which synaptic strengthening and weakening occur. Let the synaptic efficacy or strength be referred to as *weight* and denoted by w. A is the maximum amplitude of the change of weight due to a pair of spikes. t_{pre} and t_{post} denote firing times of the presynaptic neuron and the postsynaptic neuron, respectively. Then, for each set of neighboring spikes, the weight adjustment is given by,

$$\Delta w = A e^{[(t_{\text{pre}} - t_{\text{post}})\tau]}, \tag{7}$$

such that the connection weight increases when the postsynaptic spike follows the presynaptic spike and the weight decreases when the opposite occurs. The amplitude of the adjustment Δw lessens as the time between the spikes becomes larger, as is illustrated in Figure 1.

Different methods can be used to identify the spikes that give rise to the STDP mechanism. In this paper, the *nearest-spike STDP model* [22] is adopted, by which, for every spike of the presynaptic neuron, its nearest postsynaptic spike is used to calculate the timing difference in (7), regardless of whether it takes place before or after the presynaptic firing. Then, from (7), the weight change due to one of the spikes of the ith neuron is modeled by the rule

$$\Delta w_i(\Delta t_i) = \begin{cases} A_+ e^{\Delta t_i/\tau_+} & \text{if } \Delta t_i \le 0 \\ A_- e^{-\Delta t_i/\tau_-} & \text{if } \Delta t_i > 0, \end{cases} \tag{8}$$

where

$$\Delta t_i = t_{1,i} - \underset{t_{2,k}}{\operatorname{argmin}} \left| t_{1,i} - t_{2,k} \right|. \tag{9}$$

Δt_i is the firing time difference between the two neurons, $t_{1,i}$ is the firing time of the presynaptic neuron, and $t_{2,i}$ is the firing time of the postsynaptic neuron. The constants used in this paper are $\tau_+ = \tau_- = 20\,\text{ms}$, $A_+ = 0.006$, and $A_- = 1.5A_+$, where A_+ and A_- determine the maximum amounts of synaptic modification that occur when Δt is approximately zero [21].

From (8), the firing time of the postsynaptic neuron is chosen such that the absolute firing time difference between the presynaptic neuron and the postsynaptic neuron, $|t_{1,i} - t_{2,k}|$, is minimized. Therefore, the postsynaptic spike $t_{2,k}$ that is nearest to the presynaptic firing time $t_{1,i}$ is used to calculate the firing time difference Δt_i. The value of $t_{2,k}$ that minimizes $|t_{1,i} - t_{2,k}|$ can be found by comparing the firing time difference of the two neurons. In order to obtain an indirect learning method that does not rely on the direct manipulation of the synaptic weights, in this paper, it is assumed that the weights can only be modified according to the STDP rule (8). Also, the training algorithm cannot specify any of the terms in (8) directly but can only induce the firing times in (8) by stimulating the input neurons, for example, through controlled pulses of electric voltage or blue light.

3. Adaptive Critic Architecture for Indirect Reinforcement Learning

Many approaches have been proposed to train SNNs by modifying the synaptic weights by means of reward signals or by update rules inspired by reward-driven Hebbian learning and modulation of STDP [5, 8]. However, these methods are not applicable to approaches involving biological neuronal networks (e.g., neuronal cultures grown *in vitro*), because biological synaptic efficacies cannot be manipulated directly by the training algorithm. Similarly, in nanoscale neuromorphic chips, the CMOS/memristor synaptic weights cannot be manipulated directly but may be modified via controlled programming voltages that induce a mechanism analogous to STDP, as demonstrated in [10]. This paper presents an approach for training an SNN through controlled input pulse signals (e.g., voltages or blue light) that are generated by a new deterministic and adaptive spike model presented in Section 4.1.

The derivation of the training equations used to adapt the proposed spike model and subsequent pulse signals are demonstrated in Section 4.2 using a two-node LIF SNN. It is shown that the synaptic weight can be updated by the proposed indirect training method and driven precisely to a desired value, by minimizing a function of the synaptic weight error. It follows that, using a simple chain rule, the same training equations can be used to minimize a function of the decoded SNN output error, as is typically required by supervised training. In this section, we show how the indirect training method can be used for reinforcement learning, using a critic SNN to modulate the STDP mechanisms in an action SNN with synaptic weights that cannot be manipulated directly (e.g., a biological neuronal network).

Let NN_a represent an action SNN of LIF neurons, exhibiting STDP and modeled as described in Section 2. It is assumed that the weights of the action SNN cannot be manipulated directly, and the only controls available to the learning algorithm are the training signals (Figure 2), comprised of programming voltages that can be delivered using a square wave or the Rademacher function (Section 4.1). Now, let NN_c denote a critic SNN of feedforward fully connected HH neurons. In this architecture, schematized in Figure 2, NN_a is treated as a biological network and NN_c is treated as an artificial network implemented on a computer or integrated circuit. Thus, the synaptic weights of NN_c, defined as w_{ij}, are directly adjustable, while the synaptic efficacies of NN_a can only be modified through a simulated STDP mechanism which is modulated by the input spikes from NN_c. The algorithm presented in this section trains the network by changing the values of w_{ij} inside NN_c, which are assumed to be bounded by a positive constant $w_{\max}$ such that $-w_{\max} \le w_{ij} \le w_{\max}$, for all i, j.

In this paper, spike frequencies are used to code continuous signals, and the leaky integrator

$$\hat{u} = \alpha \sum_{t_k \in S_i(T)} e^{\beta(t_k - t)} H(t - t_k) - \gamma \tag{10}$$

is used to decode spike trains and convert them into continuous signals as required according to the architecture in Figure 2 Where, α, β, and γ are user-specified constants, $H(\cdot)$ is the Heaviside function, and $S_i(T)$ is the set of spiking times of the output neuron. One advantage of the leaky integrator decoder is its effectiveness of filtering inevitable noise during the flight control without influencing the speed of matching the continuous value with the target function.

3.1. Adaptive Critic Recurrence Relations. Typically, adaptive critic algorithms are used to update the actor and critic weights by computing a synaptic weight increment through an optimization-based algorithm, such as backpropagation ([23], page 359). Therefore a new approach is required in order to apply adaptive critics to SNNs in which changes in the synaptic weights can occur only as a result of pre- and postsynaptic neuronal activity. The training approach presented in Section 4 can be combined with the policy and value-iteration procedures described in this section to supervise stimulation patterns in the action SNN such

that the synaptic strengths, and subsequently their dynamic mappings, are adapted according to the STDP rule in (8).

Suppose the action SNN is being trained to control or model a dynamical system which obeys a nonlinear differential equation of the form

$$\dot{x}(t) = f[x(t), u(t), t], \tag{11}$$

where $x \in X \subset \mathbb{R}^n$ and $u \in U \subset \mathbb{R}^m$ denote the dynamical system state and control inputs, respectively, and $\dot{x}$ denotes the derivative of x with respect to time. The model of the dynamical system (11) may be unknown or imperfect and may be improved upon over time by a system identification (ID) algorithm. The macroscopic behavioral goals of the plant can be expressed by the value function or *cost-to-go*

$$V^{\pi}[x(t_k)] = \sum_{t_k=t}^{t_f-1} \mathcal{L}[x(t_k), u(t_k), x(t_k+1)], \tag{12}$$

where $u(t_k) = \pi[x(t_k)]$ is the unknown control law or dynamic mapping to be learned by the action SNN, NN_a. Then, the cost-to-go in (12) can be used to represent the future performance of the action network and dynamical system, as is accrued from the present, t_k, up to the final time, t_f, if subject to the present control law $\pi[\cdot]$. The Lagrangian $\mathcal{L}[\cdot]$ represents instantaneous behavioral goals as a function of x and u.

Since the dynamic mapping $\pi[\cdot]$ must be adapted over time through the learning algorithm and an accurate plant model may not be available, the cost-to-go typically is unknown and is learned by the critic, NN_c. Then, by Bellman's principle of optimality [24], at any time t_k the cost-to-go can be minimized online with respect to the present control law, based on the known value of $x(t_k)$ and predictions of $x(t_k + 1)$. The value of $x(t_k)$ is assumed to be fully observable from the dynamical system at any present time t_k, and $x(t_k + 1)$ is predicted or estimated from the approximation of the system's dynamics (11), based on the present values of the state and the control. In [25], Howard showed that if iterative approximations of the control law and optimal cost-to-go, denoted by π_ℓ and V_ℓ, respectively, are modified by a policy-improvement routine and a value-determination operation, respectively, they eventually converge to their optimal counterparts π^* and V^*.

The policy-improvement routine states that, given a cost-to-go function $V(\cdot, \pi_\ell)$ corresponding to a control law π_ℓ, an improved control law $\pi_{\ell+1}$ can be obtained as follows:

$$\pi_{\ell+1}[x(t_k)] = \arg\min_{u \in U} \{\mathcal{L}[x(t_k), u(t_k), x(t_k+1)] + V[x(t_{k+1}, \pi_\ell)]\}, \tag{13}$$

such that $V[x(t_k), \pi_{\ell+1}] \leq V[x(t_k), \pi_\ell]$, for any $x(t_k)$. Furthermore, the sequence of functions $C = \{\pi_\ell \mid \ell = 0, 1, 2, \ldots\}$ converges to the optimal control law π^*. The value-determination operation states that given a control law $\pi(\cdot)$, the cost-to-go can be updated according to the following rule:

$$V_{\ell+1}[x(t_{k+1}), \pi] = \mathcal{L}[x(t_k), u(t_k), x(t_k+1)] + V_{\text{ell}}[x(t_{k+1}, \pi)], \tag{14}$$

such that the sequence of functions $V = \{V_\ell \mid \ell = 0, 1, 2, \ldots\}$ converges to V^*.

Then, at every iteration cycle ℓ of the adaptive critic algorithm, the above policy and value-iteration procedures can be used to in tandem to supervise training of the actor and critic SNN; that is,

$$u(t_k) \longleftarrow NN_a[\ell, x(t_k)] \approx \pi_\ell[x(t_k)], \tag{15}$$

$$V^{\pi}[x(t_k)] \longleftarrow NN_c[\ell, x(t_k), \pi] \approx V_\ell[x(t_k), \pi], \tag{16}$$

respectively where $\ell = Mk$ and M is a positive constant chosen based on the desired frequency of the updates. It can be seen that, at $\ell + 1$, the desired mappings for NN_a and NN_c, provided by (13) and (14), respectively, are based on the actor and critic outputs at ℓ, and on $x(t_{k+1})$, which can be estimated from an available dynamical system model (11). To accomplish (15) and (16), two error metrics defined as a function of the actual (decoded) actor and critic outputs and of the desired actor and critic outputs (13) and (14) must be minimized by the chosen training algorithm, as explained in the following subsection.

3.2. Action and Critic Network Training Algorithm. The action and critic networks implemented in Figure 2 differ in that while the critic network can be trained by a conventional algorithm that manipulates the synaptic weights directly (e.g., see [5, 15]), the action network must be trained by inducing weight changes indirectly through STDP (8), such that its (decoded) output $\hat{u}$ will closely match u in (15), for all $x \in X$. Since the weights within NN_a cannot be adjusted directly, they are manipulated with training signals from NN_c. Connections are made between q pairs of action/critic neurons which serve as outputs in NN_c and inputs for NN_a (Figure 2). To provide feedback signals to the critic, connections are also created between r pairs of neurons from NN_a to NN_c. The information flow through the networks is illustrated in Figure 2. Since it is not known what training signals will produce the desired results in NN_a, the critic must also be trained to match V^{π} in (16) for all $x \in X$ and $u \in U$.

Both updates (15) and (16) can be formulated as follows. Let SNN$[\cdot]$ denote an action or critic network comprised of multiple spiking neurons and STDP synapses. Then, as shown in (15) and (16), SNN$[\cdot]$ must represent a desired mapping between two vectors $\xi \in \mathbb{R}^p$ and $y \in \mathbb{R}^r$:

$$y(t_k) = g_\ell[\xi(t_k)] \longleftarrow \text{SNN}[\xi(t_k), W_\ell(t_k)] \triangleq \hat{y}(t_k). \tag{17}$$

The desired mapping $g_\ell : \mathbb{R}^p \to \mathbb{R}^r$ can be assumed known and stationary during every iteration cycle ℓ and is updated by the policy/value-iteration routine. Let $W_\ell(t_k) = \{w_{ij}(t_k)\}$ denote a matrix containing the SNN synaptic weights at time t_k. Then, for proper values of W_ℓ, if the SNN is provided

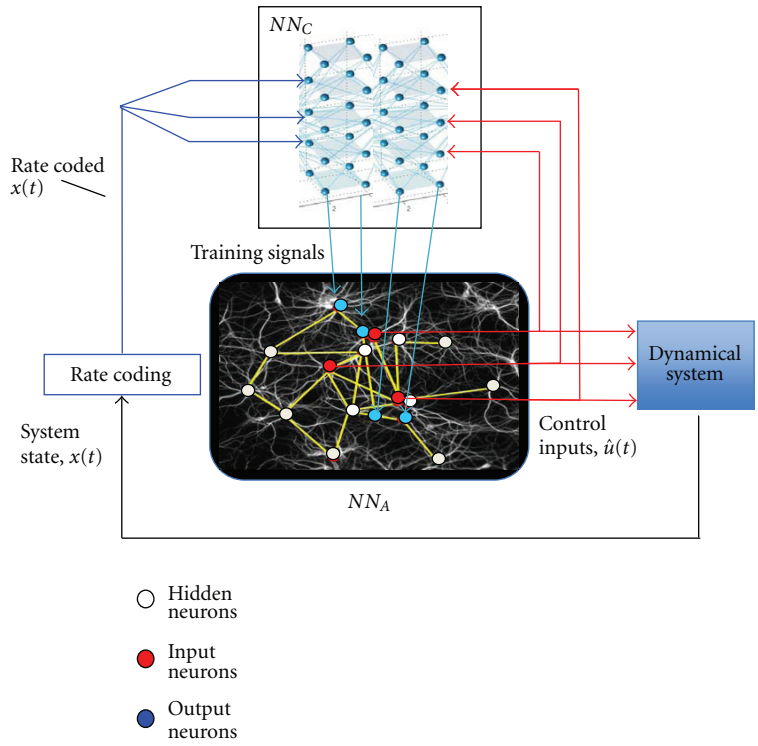


Figure 2: Adaptive critic architecture comprised of a critic network, NN_c, and, an action network, NN_a, to control a dynamical system.

with an observation of the input, ξ^l, encoded as n spike train sequences $X_i^l = \{\hat{t}_{i,\kappa} : \kappa = 1,\ldots,N_i^l\}$, $i = 1,\ldots,n$, over a time interval $[t_k, t_{k+\tau}]$ with the firing times at $\hat{t}_{i,\kappa}$, the decoded output of the SNN must match the output $y^l = g_\ell[\xi^l]$, where since $\tau \ll (t_{k+1} - t_k)$, X_i^l can be used to encode instantaneous values of ξ^l at any t_k. Then, the chosen SNN training algorithm must modify the synaptic weights such that a figure of merit representing distance is optimized at the output space of the mapping in (17).

For the critic network, NN_c, the synaptic weights are adjusted manually, using the reward-modulated Hebbian approach presented in [5, 15]. Because the target function y is known for all values of ξ, the SNN output error $(y - \hat{y})$ is also known and can be used as a feedback to the critic in the form of an imitated chemical reward, $r(t)$, that decays over time with time constant τ_c. The critic reward is modeled as

$$r(t) = \frac{[b(\hat{y}, y) + r(t - \Delta t)]e^{-(t-\hat{t}_i)}}{\tau_c}, \tag{18}$$

where for the critic $y \in \mathbb{R}$. Therefore, the error function can be defined as $b(\hat{y}, y) = \text{sgn}(y - \hat{y})$, where $\text{sgn}(\cdot)$ is the signum function. Thus, the value of $b(\cdot)$ is positive when the critic's output is too low, and it is negative when the critic's output is too high.

For the action network, NN_a, the synaptic weights cannot be manipulated directly; therefore the distance between y and the (decoded) SNN's output $\hat{y}$ is minimized with respect to the parameters of the RBF spike model described in the next section. From (17), we identify (tag) two sets of neurons referred to as input and output neurons (Figure 2), where each neuron in the set provides the response for one element of ξ and y, respectively, in the form of a spike train. It is assumed that the set of input neurons can be induced to fire on command with very high precision over a training time interval $[t_k, t_{k+\tau}]$, with $\tau < T \ll (t_{k+1} - t_k)$. The input neurons' firings could be implemented in practice by using local programming voltages with controlled pulse width and height that are easily realizable both in CMOS/memristor chips [10] and in neuronal networks grown *in vitro* [26]. In order to induce STDP in a manner that will improve the SNN representation of the mapping in (17), the programming voltages are delivered based on an optimized spiking sequence $S_i^\iota = \{\hat{t}_{i,\kappa} : \kappa = 1,\ldots,N_i^\iota\}$, $i = 1,\ldots,q$, during the ιth time interval of the training algorithm, $[t_\iota, t_{\iota+1}]$.

Existing spike models [13, 14] cannot be used to generate S_i^ι because they are stochastic and, as such, they do not allow for precise timing of pre- and postsynaptic firings. As a result, they may induce undesirable changes in the synaptic weights by virtue of the STDP rule (7), illustrated in Figure 1, which shows that a small difference in the arrival time of a pre- and postsynaptic spike can obtain the opposite effect in terms of the weight change Δw. Beside allowing for precise timing of neuron firings, the new deterministic spike model presented in the next section can be updated by the training algorithm, by optimizing a corresponding continuous signal modeled by a superposition of Gaussian radial basis functions (RBFs), which is characterized by a set of adjustable parameters $P = \{w_k, c_k, \beta_k \mid k = 1,\ldots,N\}$, where w_k is the height the RBF

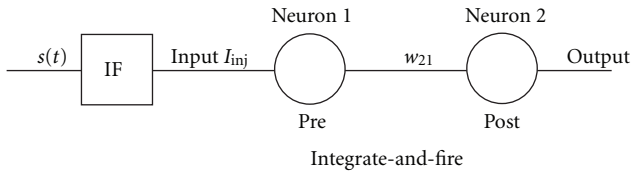


Figure 3: Model of two-node LIF spiking neural network.

pulse, c_k is the center of RBF pulse, β_k is the width of the RBF pulse, and N is the number of RBF pulses. As shown in the next section, the continuous RBF signal can be integrated against a suitable averaging function in a leaky integrator and then compared to a positive threshold, by means of a leaky integrate-and-fire (LIF) sampler. Subsequently, a precise pulse function with desired widths and intensity can be generated at a sequence of time instants, $\hat{t}_{i,\kappa}$, during the interval $[t_\iota, t_{\iota+1}]$, that correspond to the centers of the RBF signal specified by P. Thus, a set of optimal RBF parameters P^* used to generate S_i^ι can be determined by minimizing a measure of the distance between the (decoded) SNN's output $\hat{y}$ and the desired output y, computed by the policy-improvement routine (13).

In the next section, the derivation of gradient equations that can be used to minimize the SNN's output error $(y - \hat{y})$ with respect to P is illustrated by means of a two-node LIF neural network, with STDP modeled as shown in Section 2. Let $P = \{p_{lk}\}$ denote a matrix of RBF parameters to be adjusted by the training algorithm. It can be easily shown that the minimization of an error function $E(y - \hat{y})$ with respect to the elements of P when y is a known constant can be accomplished using the gradients $\partial\hat{y}/\partial p_{lk}$, or some function thereof, based on the chosen unconstrained minimization algorithm [27]. As shown in (17), for a given input ξ^l the only SNN variables to be optimized are the synaptic weights w_{ij}. From the STDP rule (7), the weights are a function of the parameters p_{lk}, which determine the input spike sequence (or training signal) to the SNN (Figure 3). By virtue of the chain rule of differentiation, it follows that

$$\frac{\partial\hat{y}}{\partial p_{lk}} = \frac{\partial\hat{y}}{\partial w_{ij}}\frac{\partial w_{ij}}{\partial p_{lk}}. \tag{19}$$

Since $\partial\hat{y}/\partial w_{ij}$ can be computed from the SNN model (Section 2), it follows that if a training algorithm can successfully modify the synaptic weights w_{ij} using the proposed spike model, it also can successfully modify $\hat{y}$, simply by redefining the error function. In other words, if a training algorithm can successfully train the synaptic weights of an SNN using the gradient $\partial w_{ij}/\partial p_{lk}$, it follows that it can also train the SNN using the gradient $\partial\hat{y}/\partial p_{lk}$, since the component $\partial\hat{y}/\partial w_{ij}$ is known and given by the SNN equations (1)–(5). Based on this property, the novel spike model and indirect training algorithm are illustrated by training the synaptic weight of a two-node LIF SNN to meet a desired value w^*, without direct manipulation.

4. Indirect Training Methodology

Consider the two-node LIF SNN schematized in Figure 3, modeled using the approach described in Section 2. $s(t)$ represents the input given to the LIF sampler, and $S(t)$ denotes the corresponding LIF sampler's output. Based on the approach presented in Section 3, the SNN synaptic strength w_{21}, representing the synaptic efficacy for a presynaptic neuron (labeled by $i = 1$) and a postsynaptic neuron (labeled by $i = 2$) cannot be modified directly by the training algorithm but can only change as a result of the STDP mechanism described in Section 2. In place of controlling w_{21}, the goal of the indirect training algorithm is to determine a spike train that can be used to stimulate the input neuron ($i = 1$) using I_{inj}, thereby inducing it to spike, such that the synaptic weight w_{21} changes from an initial (random) value to a desired value w^*.

For this purpose, we introduce a continuous spike model comprised of a superposition of Gaussian radial basis functions (RBFs):

$$s(t) = \sum_{k=1}^{N} w_k \exp\left[-\beta_k(\|t - c_k\|)^2\right], \tag{20}$$

where w_k determines the height of the kth RBF, which will decide the constant current input I_{inj}. β_k determines the width of the kth RBF and c_k determines the center of the kth RBF, where $k = 1, \ldots, N$ and N is the number of RBFs. Thus, the set of RBF adjustable parameters is $P = \{w_k, c_k, \beta_k \mid k = 1, \ldots, N\}$. A spike train can be obtained from the continuous spike model (20) by processing s using a suitable LIF sampler that outputs a square pulse function $S(t)$ with the same heights, widths, and centers, as the RBF spike model (20).

In this two-node SNN model, there is no synaptic current sent to the first neuron because there are no presynaptic neurons connected to it. Rather, the input for the first neuron is the square pulse function provided by the output of the LIF sampler. Therefore, during one square pulse, the input, $I_{inj}(t)$, to neuron $i = 1$ can be viewed as a constant, which for our case is h, the height of the RBF. For a fixed threshold and a constant input, the time it takes for the first neuron to fire (or a spike to be generated) is

$$T = -\tau_m \ln\left[1 - \frac{\theta}{hR_m}\right] \quad (hR_m > \theta), \tag{21}$$

where θ is the potential difference between the spike generating threshold, V_{th}, and the resting potential, V_0; that is, $\theta = V_{th} - V_0$. h is the height of input square pulse, R_m is the resistance of the membrane, and τ_m is the passive-membrane time constant [20]. Thus, from (21), the value of T can be controlled by adjusting h. We allow the first neuron to fire near the end of a square pulse by inputting a spike sequence generated by an RBF model with a suitable width and height. Then, the firing times of the first neuron can be easily adjusted by altering the centers of the RBF.

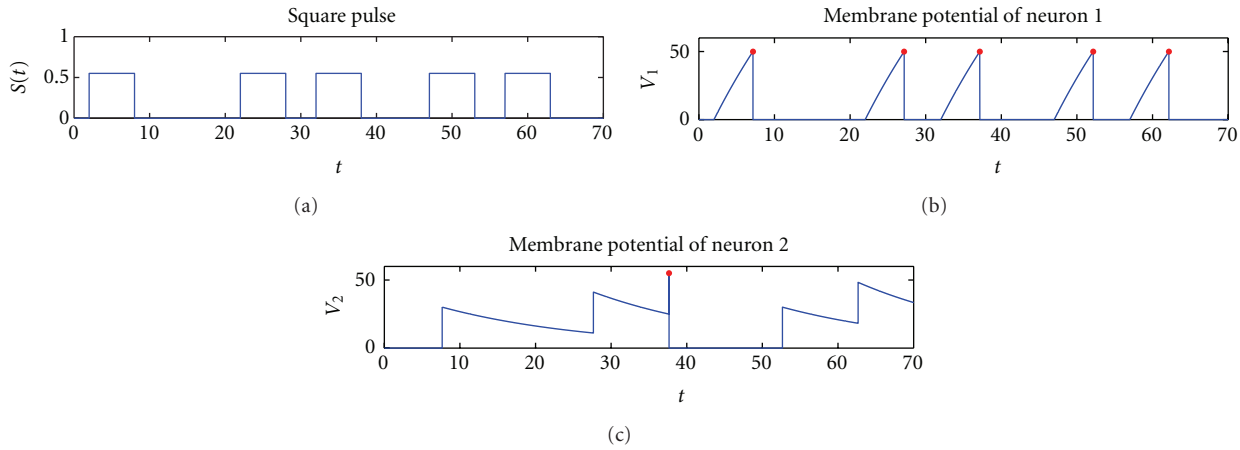


FIGURE 4: Membrane potential of the two neurons in the SNN in Figure 3.

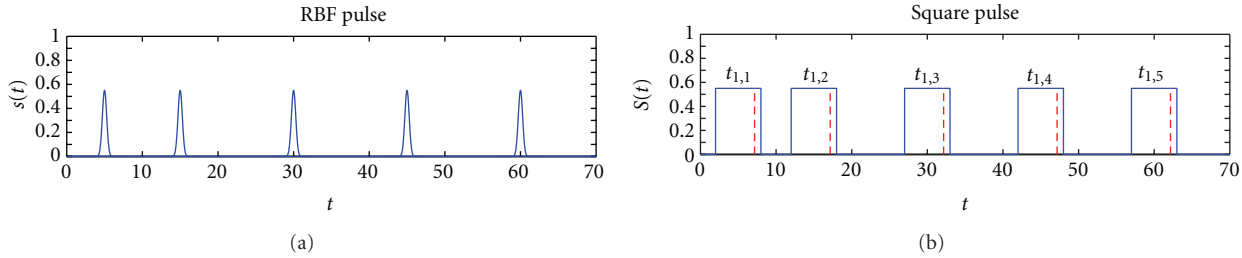


FIGURE 5: Deterministic spike model signals.

In this simple example, it is assumed that the synapse is of the excitatory type. Therefore, (4) can be written as,

$$I_s(t) = C_m a_E \sum_{k=k_0}^{k_t} \delta(t - t_{1,k} - \tau_d), \tag{22}$$

where τ_d is a known constant that represents the conduction delay of the synaptic current from neuron $i = 1$ and $t_{1,k}$ are the firing times of the first neuron. k_0 is the index of the spike of the first neuron, which occurs after the previous spike of the second neuron, and k_t is the index of the spike of the first neuron, which provokes the firing time of the second neuron. The only input to the second neuron is the synaptic current defined in (22). Therefore, substituting (22) and (2) into (1) results in the following equation for the membrane potential,

$$C_m \frac{dv(t)}{dt} = -\frac{C_m}{\tau_m}[v(t) - V_0] + C_m a_E \sum_{k=k_0}^{k_t} \delta(t - t_{1,k} - \tau_d). \tag{23}$$

Solving (23) for the membrane potential of the second neuron provides the response of neuron $i = 2$,

$$v(t) = V_0 + \sum_{k=k_0}^{k_t} a_E \exp\left[-\frac{t - t_{1,k} - \tau_d}{\tau_m}\right] H(t - t_{1,k} - \tau_d), \tag{24}$$

where $H(\cdot)$ is the Heaviside function. Whenever the membrane potential in (24) exceeds the threshold V_{th}, the second neuron fires, and the membrane potential $v(t)$ is then set instantly equal to V_0. It follows that the firing times of the second neuron $t_{2,j}$ can be written as a function of the firing times of the first neuron,

$$t_{2,j} = (t_{1,k_t} + \tau_d) H\left\{\sum_{k=k_0}^{k_t} a_E \exp\left[-\frac{t_{2,j} - t_{1,k} - \tau_d}{\tau_m}\right] \times H\left(t_{2,j} - t_{1,k} - \tau_d\right) - (V_{\text{th}} - V_0)\right\}, \tag{25}$$

where j is the index of the firing times. In addition, the membrane potentials of the first neuron and the second neuron can be calculated using (24) and (21).

An example of the membrane potential time history for the two neurons is plotted in Figure 4, where the square pulse function S used to stimulate the first neuron is plotted along with the neurons' membrane potentials v_1 and v_2. The red dots denote the firing times and potentials for the two neurons. It can be seen from Figure 4 that, with this input, the first neuron fires five times and the second neuron fires one time. In this SNN, the input to the second neuron is given only by the synaptic current caused by the firing of the first neuron. It can be seen that only when the second and third firing times of the first neuron are very close, the membrane potential of the second neuron increases over the threshold, causing it to fire, whereas the fourth and fifth

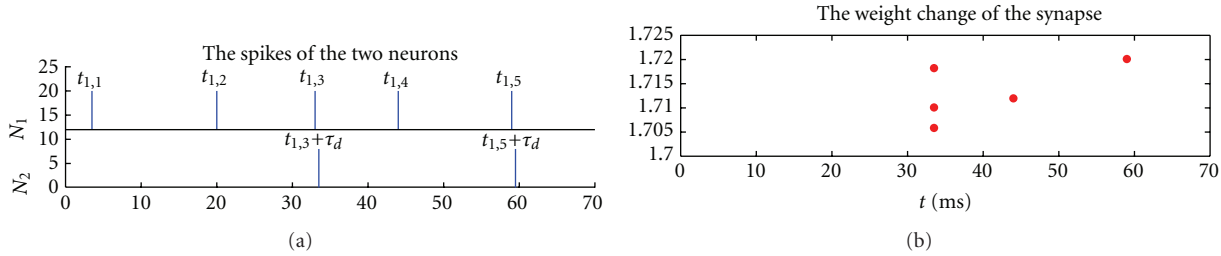


Figure 6: Optimized input spike train and indirect weight changes brought about by STDP.

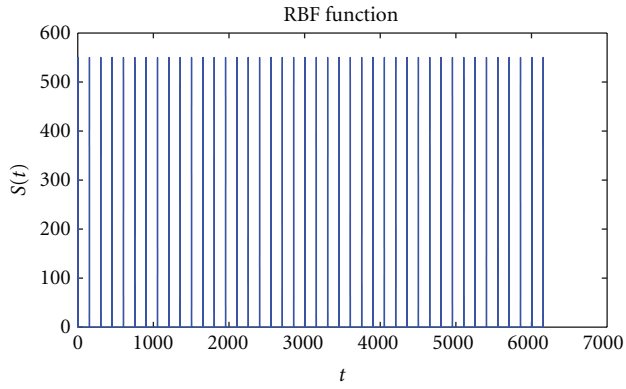


Figure 7: Optimized RBF spike model for the SNN in Figure 3.

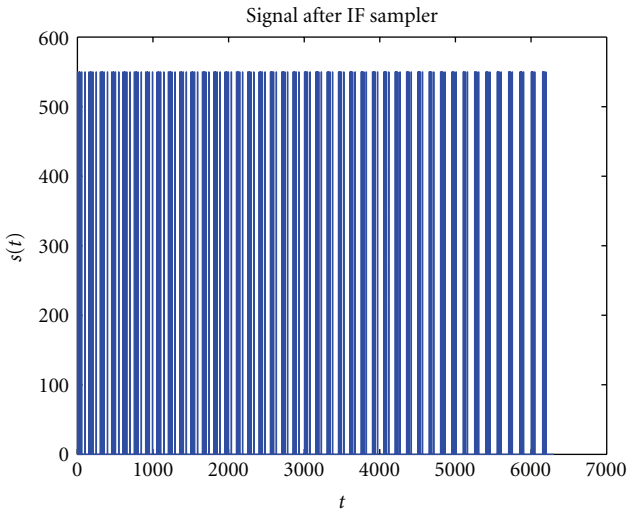


Figure 8: Square wave obtained by the LIF sampler, for the RBF spike model in Figure 7.

firing times of the first neuron are too far apart to cause firing of the second neuron.

4.1. Deterministic Spike Model. The deterministic spike model consists of a continuous RBF model in the form (20) combined with an LIF sampler that converts the RBF into a square wave function. The LIF sampler developed in [28] for the approximate reconstruction of bandlimited functions is adopted, which converts any continuous signal $f(t)$ into a square wave function by integrating it against an averaging function $u_{k,z}(t)$. The integrated result is compared to a positive threshold and a negative threshold, such that when either one of the two thresholds is reached, a pulse is generated at time $t_k = z$. The value of the integrator is then reset and the process repeats. In this paper, the averaging function $u_{k,z}(t)$ is chosen to be the exponential $e^{\alpha(t-z)}X_{[t_k,z]}$, where X_I is the characteristic function of I and $\alpha > 0$ is a constant that models the leakage of the integrator, as due to practical implementations [28]. Then, the LIF sampler firing condition that generates the square wave (or sequence of pulses) is

$$\pm\theta = \int_{t_k}^{t_{k+1}} f(t)e^{\alpha(t-t_{k+1})}dt \triangleq \langle f, u_k\rangle, \tag{26}$$

where t_k is the time instant of the sample, t_{k+1} is the next time instant of the sample, u_k is the averaging function, θ is the threshold value, and α is the leakage of the integrator. The output of the LIF sampler can be expressed in terms of the time instants at which the integral reaches the threshold, $\{t_0, \ldots, t_n\}$, and by the samples $q_1, \ldots, q_n$, defined as

$$q_j \triangleq \int_{t_j}^{t_{j-1}} f(x)e^{\alpha(x-t_j)}dx, \quad \text{for } 1 \le j \le n. \tag{27}$$

From (26), it follows that $|q_j| = \theta$. By adjusting the parameters of the LIF sampler, it is possible to convert the RBF signal in (20) to a square wave comprised of pulses with the same width, height, and centers as the RBFs in (20). Thus, by using the RBF spike model in (26) with suitable widths and heights, it is possible to induce the first neuron to fire shortly before the end of each pulse of the square-pulse function (also considering the refractory period of the neuron). For simplicity, in this example, it is assumed that the heights w_k and widths β_k are known positive constants of equal magnitudes for all k. Then, the centers of the RBF comprise the set of adjustable parameters, $P = \{c_k \mid k = 1, \ldots, N\}$, to be optimized. The same approach can be easily extended to a case where all of the RBF parameters are adapted.

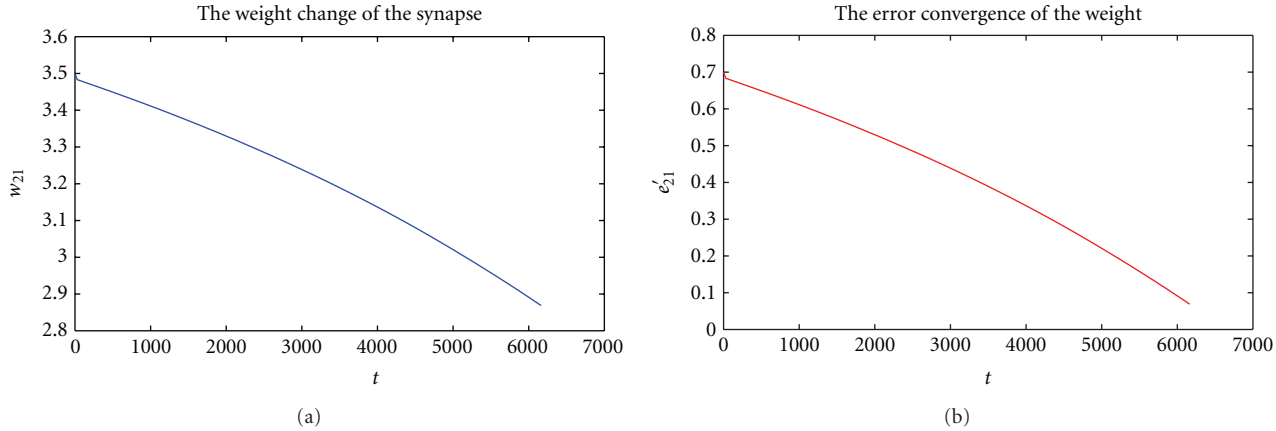


Figure 9: Indirect weight changes brought about by STDP and corresponding deviation from $w^* = 2.8$, for the SNN in Figure 3 stimulated using the input spike train in Figure 7.

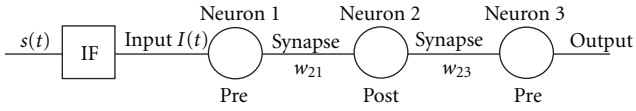


Figure 10: Model of three-node LIF spiking neural network.

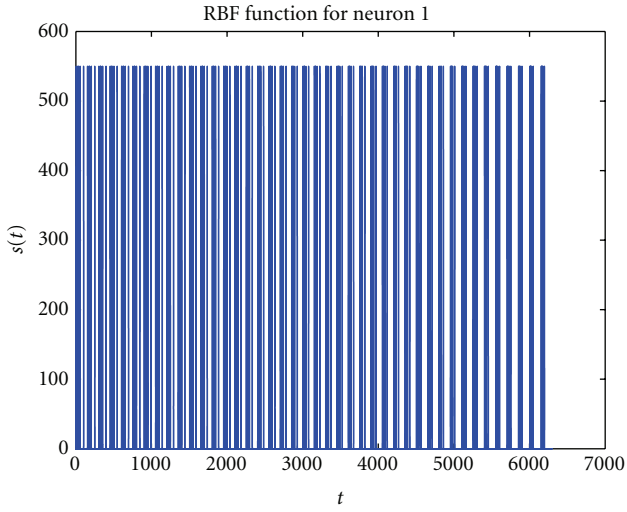


Figure 11: Optimized RBF spike model for the SNN in Figure 10.

It can be shown using the LIF SNN model in Section 2 that, under the aforementioned assumptions, the firing times of the first neuron satisfy the constraints,

$$t_{1,k} \leq c_k + \frac{\beta}{2},$$
$$t_{1,k} + \Delta^{\text{abs}} \geq c_k + \frac{\beta}{2}, \tag{28}$$

where Δ^{abs} is an absolute refractory time of the neuron. Then, the firing times of the first neuron can be written as a function of the RBF centers, c_k. By design, the first neuron fires after the same time interval, relative to each square pulse

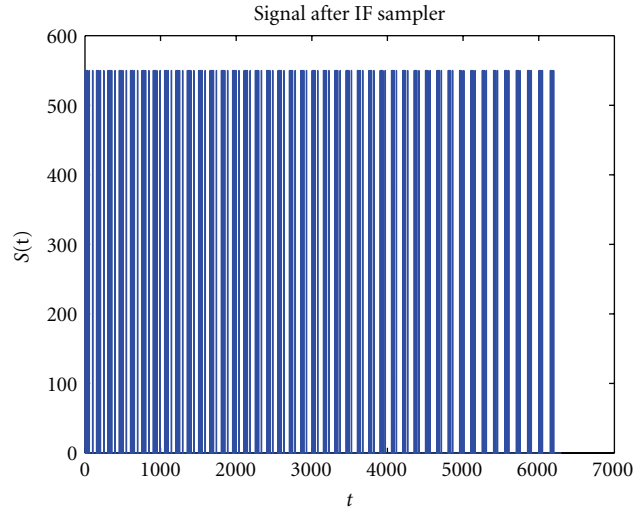


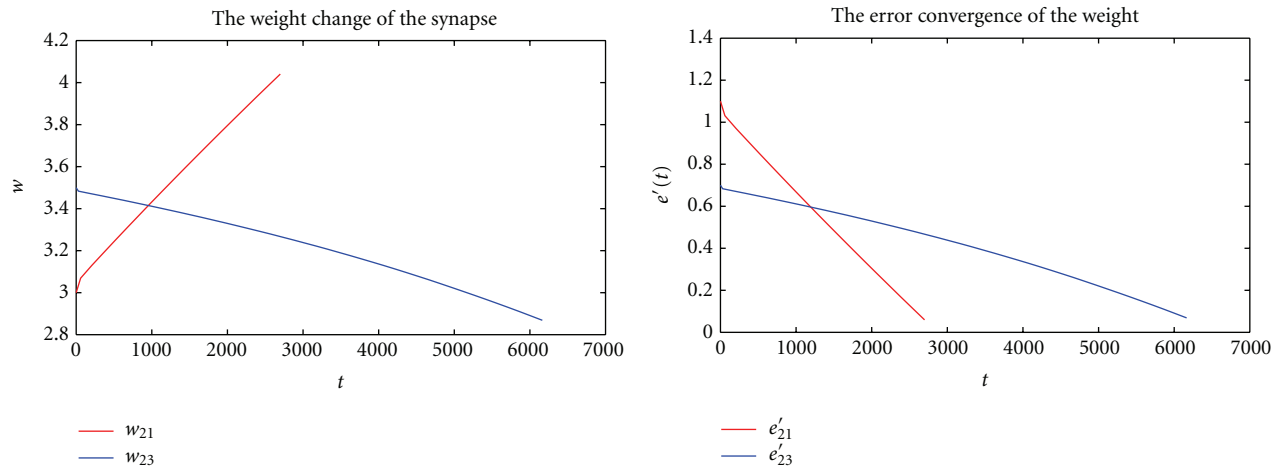
Figure 12: Square wave obtained from the LIF sampler, using the optimized RBF spike model in Figure 11.

(Figure 4), due to the chosen RBF heights and widths. Then, the firing times of the first neuron can be written as,

$$t_{1,k} = c_k - \frac{\beta}{2} + T, \tag{29}$$

where $t_{1,k}$ denotes the firing times of the first neuron, c_k are the centers of the RBF, β is the constant width of the RBF, and T is given by (21).

The RBF spike model is demonstrated in Figure 5, where an example of RBF output obtained from (20) is plotted and, after being fed to the LIF sampler, produces a corresponding square wave which can be used as controlled pulse. It can be seen that the centers of the RBFs precisely determine the times at which a pulse occurs in the square wave and that the square wave maintains the desired constant width and magnitude for all k. The next subsection illustrates how, by adapting the continuous RBF spike model in (20), it is



(a) (b)

FIGURE 13: Indirect weight changes brought about by STDP and corresponding error functions, for the SNN in Figure 10 stimulated using the input spike train in Figure 12.

possible to minimize a desired error function and train the synaptic weight of the SNN without directly manipulating it.

4.2. Indirect Training Equations. As explained in Section 2.2, it is assumed that the synaptic weight w_{21} can only be modified by controlling the activity of the SNN input neuron(s), with $i = 1$, and that it obeys the nearest-spike STDP model in (8). Over time, the synaptic weight can change repeatedly, and therefore its final value can be written as the sum of all incremental changes that have occurred over the time interval $[t_l, t_{l+1}]$,

$$w_{21}(t) = \prod_{i=1}^{M} w_0(1 + \Delta w_i), \tag{30}$$

where, for the two-neuron SNN in Figure 3, $\Delta w_1, \ldots, \Delta w_M$ are due to M pairs of pre- and postsynaptic spikes that occur at any time $t \in [t_l, t_{l+1}]$. Every weight increment Δw_i is induced via STDP and, thus, obeys equation (8). Since the firing times in (8) depend on the centers of the RBF input through (21)–(29), it follows that every weight increment Δw_i is a function of the RBF centers. In this example, the constraints (28) can be written as,

$$\frac{\beta}{2} < c_1, \qquad c_N + \frac{\beta}{2} < t_f, \qquad c_k + \beta < c_{k+1}, \quad \text{for } k = 1, \ldots, N, \tag{31}$$

where, from Figure 5, $N = 5$.

A training-error function is defined in terms of the desired synaptic weight w^* and the actual value of the synaptic weight $w_{21}(t)$. While different forms of error function may be used, the chosen form determines the complexity of the derivation of the training gradients. It was found that the most convenient form of training-error function can be derived from the ratio of the actual weight over the desired weight,

$$e(t) = \frac{w_{21}(t)}{w^*}, \tag{32}$$

where $w_{21}(t)$ is the weight value at time $t \in [t_l, t_{l+1}]$, obtained from all previous spikes. As explained in Section 3, w^* can be viewed as the weight that leads to desired output $\hat{y}$ for a given input ξ. It can be seen that when w_{21} is equal to w^*, e is equal to one. Plugging (30) into (36), the weight ratio can be rewritten as,

$$e(t) = \frac{1}{w^*} \prod_{i=1}^{M} w_0(1 + \Delta w_i), \tag{33}$$

and the error between w_{21} and w^* can be minimized by minimizing the natural logarithm of the ratio (33):

$$E(t) \triangleq \ln[e(t)] = \ln(w_0) + \ln(1 + \Delta w_1) + \cdots + \ln(1 + \Delta w_M) - \ln(w^*). \tag{34}$$

As is typical of all optimization problems, minimizing a quadratic form presents several advantages [29]. Therefore, at any time $t \in [t_l, t_{l+1}]$, the indirect training algorithm seeks to minimize the quadratic training-error function:

$$J(t) \triangleq E(t)^2 = \{\ln[e(t)]\}^2 = \{\ln(w_0) + \ln(1 + \Delta w_1) + \cdots + \ln(1 + \Delta w_M) - \ln(w^*)\}^2. \tag{35}$$

Then, indirect training can be formulated as an unconstrained optimization problem in which J is to be minimized with respect to the RBF centers, or $P = \{c_k : c_k \in [t_l, t_{l+1}]\}$. Any gradient-based numerical optimization algorithm can be utilized for this purpose. The analytical form of the gradient $\partial J/\partial c_k$ depends on the form of the spike patterns. The following subsection derives this gradient analytically for one example of spike patterns and demonstrates that, using this gradient, the synaptic weight can be trained indirectly to meet the desired value w^* exactly. The same results were derived and demonstrated numerically for all other possible

spike patterns, but they are omitted here for brevity. Another representation of error $e'(t)$ is defined below to make the convergence of error more understandable in figures,

$$e'(t) = |w_{21} - w^*|. \tag{36}$$

where, $|\cdot|$ denotes the absolute value.

4.3. Derivation of Gradient Equations for Indirect Training. Consider the case in which $M = 5$, $w^* = 1.72$, and $2a_E > V_{\text{th}} - V_0 > a_E$, which results in a two-node SNN (Figure 3) in which neuron $i = 1$ must fire at least two times in order for neuron $i = 2$ to fire. An example of such a spike pattern is shown in Figure 6, where N_1 and N_2 denote spike trains of neuron $i = 1$ and $i = 2$, respectively. In this case, the first neuron fires five times, and the second neuron fires two times. The firing time of the first neuron is controlled by the RBF spike model described in Section 4.1. In Figure 6, $t_{i,k}$ denotes the kth firing time of the ith neuron with $k \in \mathcal{I}_i$, where $\mathcal{I}_i = \{k = 1, \ldots, 5\}$ is an index set for the firing times and i is the index labeling the neurons. In this example, the second neuron fires after $t_{1,3}$ and $t_{1,5}$, because $t_{1,2}$, $t_{1,3}$ and $t_{1,4}$, $t_{1,5}$ are close enough to cause the membrane potential of the second neuron, v_2, to exceed the threshold.

Initially, the synaptic weight is equal to 1.7. The weight change is discontinuous due to the discrete property of spikes. As shown in Figure 6, the weight increases three times at $t_{1,3} + \tau_d$, decreases at $t_{1,4}$, and increases again at $t_{1,5} + \tau_d$. For this example, the synaptic weight changes in five increments:

$$\begin{aligned}
\Delta w_1 &= A_+ \exp\left(\frac{t_{1,1}}{\tau_+}\right) \exp\left(\frac{-t_{1,3}}{\tau_+}\right) \exp\left(\frac{-\tau_d}{\tau_+}\right) \\
\Delta w_2 &= A_+ \exp\left(\frac{t_{1,2}}{\tau_+}\right) \exp\left(\frac{-t_{1,3}}{\tau_+}\right) \exp\left(\frac{-\tau_d}{\tau_+}\right) \\
\Delta w_3 &= A_+ \exp\left(\frac{-\tau_d}{\tau_+}\right) \\
\Delta w_4 &= -A_- \exp\left(\frac{t_{1,3}}{\tau_-}\right) \exp\left(\frac{-t_{1,4}}{\tau_-}\right) \exp\left(\frac{\tau_d}{\tau_-}\right) \\
\Delta w_5 &= A_+ \exp\left(\frac{-\tau_d}{\tau_+}\right).
\end{aligned} \tag{37}$$

When the equations above are substituted in (35), the training-error function can be written as

$$\begin{aligned}
J = \Bigg\{ &\ln(w_0) + \ln\left[1 + A_+ \exp\left(\frac{t_{1,1}}{\tau_+}\right) \exp\left(\frac{-t_{1,3}}{\tau_+}\right) \exp\left(\frac{-\tau_d}{\tau_+}\right)\right] \\
&+ \ln\left[1 + A_+ \exp\left(\frac{t_{1,2}}{\tau_+}\right) \exp\left(\frac{-t_{1,3}}{\tau_+}\right) \exp\left(\frac{-\tau_d}{\tau_+}\right)\right] \\
&+ \ln\left[1 + A_+ \exp\left(\frac{-\tau_d}{\tau_+}\right)\right] \\
&+ \ln\left[1 - A_- \exp\left(\frac{t_{1,3}}{\tau_-}\right) \exp\left(\frac{-t_{1,4}}{\tau_-}\right) \exp\left(\frac{\tau_d}{\tau_-}\right)\right] \\
&+ \ln\left[1 + A_+ \exp\left(\frac{-\tau_d}{\tau_+}\right)\right] - \ln(w^*)\Bigg\}^2.
\end{aligned} \tag{38}$$

Then, the gradients of J with respect to the RBF centers are given by

$$\begin{aligned}
\frac{\partial J}{\partial c_1} &= \frac{\partial E^2}{\partial c_1} = \frac{\partial E^2}{\partial t_{1,1}} = 2E\left[\frac{\Delta w_1}{\tau_+(1 + \Delta w_1)}\right] \\
\frac{\partial J}{\partial c_2} &= \frac{\partial (E^2)}{\partial c_2} = \frac{\partial E^2}{\partial t_{1,2}} = 2E\left[\frac{\Delta w_2}{\tau_+(1 + \Delta w_2)}\right] \\
\frac{\partial J}{\partial c_3} &= \frac{\partial E^2}{\partial c_3} = \frac{\partial E^2}{\partial t_{1,3}} \\
&= 2E\left[\frac{-\Delta w_1}{\tau_+(1 + \Delta w_1)} + \frac{-\Delta w_2}{\tau_+(1 + \Delta w_2)} + \frac{\Delta w_4}{\tau_-(1 + \Delta w_4)}\right] \\
\frac{\partial J}{\partial c_4} &= \frac{\partial E^2}{\partial c_4} = \frac{\partial E^2}{\partial t_{1,4}} = 2E\left[\frac{-\Delta w_4}{\tau_-(1 + \Delta w_4)}\right] \\
\frac{\partial J}{\partial c_5} &= \frac{\partial E^2}{\partial c_5} = \frac{\partial E^2}{\partial t_{1,5}} = 0.
\end{aligned} \tag{39}$$

Using the above gradients, the optimal values of $c_1, \ldots, c_5$ can be obtained by minimizing J using a gradient-based numerical algorithm such as Newton's method.

An example of indirect learning algorithm implementation is shown in Figure 6, where the RBF-adjustable parameters (which, in this case, coincide with the firing times of neuron $i = 1$) are optimized to induce a change in the synaptic weight w_{21} from an initial value of 1.704, to a desired value $w^* = 1.72$. Another example, for which the gradient equations are omitted for brevity, is shown in Figures 7–9. In this case, the desired weight value $w^* = 2.8$ is far from the initial weight $w_{21}(t_t) = 3.5$, and thus $N = 43$ spikes are required to train the SNN. The optimal RBF input and corresponding controlled pulse used to stimulate neuron $i = 1$ are shown in Figures 7 and 8, respectively. The time history of the weight w_{21} is plotted in Figure 9(a), along with the corresponding deviation from w^* plotted in Figure 9(b).

5. Generalized Form of Gradient Equations for Indirect Training

A more general form of the gradient equations can be found by rewriting the response of the membrane potential for neuron $i = 2$ in (24), as follows,

$$v(t) = V_0 + \sum_{k=a}^{b} a_E \exp\left[-\frac{t - t_{1,k} - \tau_d}{\tau_m}\right] H(t - t_{1,k} - \tau_d), \tag{40}$$

where $t_{1,k}$ denotes the kth firing time of neuron $i = 1$. Then, the gradients of the training objective function (35) with respect to the centers of the RBF spike model for $M = 5$ are given by the equations in the Appendix, which are obtained in terms of the two functions,

$$\begin{aligned}
g_+\left(t_{1,i}, t_{1,j}\right) &= A_+ \exp\left(\frac{t_{1,i}}{\tau_+}\right) \exp\left(\frac{-t_{1,j}}{\tau_+}\right) \exp\left(\frac{-\tau_d}{\tau_+}\right), \\
g_-\left(t_{1,i}, t_{1,j}\right) &= -A_- \exp\left(\frac{t_{1,i}}{\tau_-}\right) \exp\left(\frac{-t_{1,j}}{\tau_-}\right) \exp\left(\frac{\tau_d}{\tau_-}\right),
\end{aligned} \tag{41}$$

where $t_{1,i}$, $t_{1,j}$ are two distinct firing times of neuron $i = 1$.

The methodology presented in Section 4 can also be extended to larger SNNs, although in this case it may be more convenient to compute the gradients of the objective function numerically. As an example, consider the three-neuron SNN in Figure 10, modeled by the approach in Section 2 and with two synaptic weights w_{21} and w_{23} that each obey the STDP mechanism in (8). Suppose that the desired values of the synaptic weights are $w_{21}^* = 4.1$ and $w_{23}^* = 2.8$. Using the indirect learning method presented in this paper, and the gradient provided in the Appendix, a training-error function formulated in terms of the deviations of $w_{21}(t)$ and $w_{23}(t)$ from w_{21}^* and w_{23}^*, respectively, can be minimized with respect to the parameters (centers) P of the RBF spike model (20).

Once the optimal RBF spike model, plotted in Figure 11, is fed to the LIF sampler, the controlled pulse plotted in Figure 12 is obtained and implemented via I_{inj}. The controlled pulse is thus used to stimulate neuron $i = 1$ at precise instants in time that corresponds to centers of the optimal RBF spike model. By this approach, $w_{21}(t)$ can be made to converge to the desired value w_{21}^*. The same method is implemented to stimulate neuron $i = 2$ to make the value of $w_{23}(t)$ converge to w_{23}^*. The time histories of the weight values $w_{21}(t)$ and $w_{23}(t)$, obtained by the indirect training algorithm are plotted in Figure 13(a). As is also shown by the corresponding training errors, plotted in Figure 13(b), the SNN weights over time converge to the desired values, that is $w_{21}^* = 4.1$ and $w_{23}^* = 2.8$. These results demonstrate that, even for larger SNNs, the indirect training method presented in this paper is capable of modifying synaptic weights until they meet their desired values, without direct manipulation. Since this indirect training algorithm only relies on modulating the activity of the input neurons via controlled input spike trains, it also has the potential of being realizable *in vitro* and *in silico* to train biological neuronal networks and CMOS/memristor nanoscale chips, respectively.

6. Summary and Conclusion

Recently, several algorithms have been proposed for training spiking neural networks through biologically plausible learning mechanisms, such as spike-timing-dependent synaptic plasticity. These algorithms, however, rely on being able to modify the synaptic weights directly or STDP. In other words, they minimize a desired objective function with respect to weight increments that are assumed to be controllable and, thus, are decided by a weight update rule, and then are implemented directly by the training algorithm. In several potential applications of spiking neural networks, synaptic weights cannot be manipulated directly but change over time by virtue of pre- and postsynaptic neural activity. In these applications, the activity of selected input neurons can be controlled via programming voltages or pulses of blue light that induce precise spiking of the input neurons, at precise moments in time.

This paper presents an indirect learning method that induces changes in the synaptic weights by modulating spike-timing-dependent plasticity using controlled input spike trains, in lieu of the weight increments. The key difficulty to be overcome is that indirect learning seeks to adapt a pulse signal, such as a square wave or Rademacher function, in place of continuous-valued weights. Pulse signals that can be delivered to stimulate input neurons and cause them to spike, such as blue light patterns or programming voltages, are represented by piece-wise continuous, multi-valued (or many-to-one), and nondifferentiable functions that are not well suited to numerical optimization. Furthermore, stimulation patterns typically are generated by spike models that are stochastic, such as the Poisson spike model. Therefore, even when the spike model is optimized, it does not allow for precise timing of pre- and post-synaptic firings, and as a result, may induce undesirable changes in the synaptic weights. This paper presents a deterministic and adaptive spike model derived from radial basis functions and a leaky integrate-and-fire sampler. This spike model can be easily optimized to determine the sequence of spike timings that minimizes a desired objective function and, then, used to stimulate input neurons. The results demonstrate that this methodology is capable of inducing the desired synaptic plasticity in the network and modify synaptic weights to meet their desired values only by virtue of controlled input spike trains that are realizable both in biological neuronal networks and in CMOS/memristor nanoscale chips.

Appendix

The gradients of the training objective function can be derived as follows. The membrane potential of neuron 2 is denoted by $v_{i.j}(t)$, where the membrane potential can only be affected by presynaptic spikes that occur within the time interval $[t_{1.i}, t_{1,j}]$.

$$\frac{\partial E^2}{\partial c_1} = \begin{cases} 2E\left[\dfrac{g_+(t_{1,1}, t_{1,3})}{\tau_+(1 + g_+(t_{1,1}, t_{1,3}))}\right] & \text{if } v(t_{1,3} + \tau_d)_{1,3} > V_{\text{th}} \\ 2E\left[\dfrac{g_+(t_{1,1}, t_{1,2})}{\tau_+(1 + g_+(t_{1,1}, t_{1,2}))}\right] & \text{if } v(t_{1,2} + \tau_d)_{1,2} > V_{\text{th}} \\ 2E\left[\dfrac{g_+(t_{1,1}, t_{1,4})}{\tau_+(1 + g_+(t_{1,1}, t_{1,4}))}\right] & \text{if } v(t_{1,4} + \tau_d)_{1,4} > V_{\text{th}} \\ 2E\left[\dfrac{g_+(t_{1,1}, t_{1,5})}{\tau_+(1 + g_+(t_{1,1}, t_{1,5}))}\right] & \text{if } v(t_{1,5} + \tau_d)_{1,5} > V_{\text{th}}. \end{cases}$$

$$\frac{\partial E^2}{\partial c_2} = \begin{cases} 2E\left[\dfrac{g_+(t_{1,2},t_{1,3})}{\tau_+(1+g_+(t_{1,2},t_{1,3}))}\right] & \text{if } v(t_{1,3}+\tau_d)_{1,3} > V_{\text{th}} \\ 2E\left[\dfrac{-g_+(t_{1,1},t_{1,2})}{\tau_+(1+g_+(t_{1,1},t_{1,2}))}\right] & \text{if } v(t_{1,2}+\tau_d)_{1,2} > V_{\text{th}},\ v(t_{1,5}+\tau_d)_{3,5} > V_{\text{th}} \\ & c_3 > \dfrac{c_5+c_2+2\tau_d}{2} \\ 2E\left[\dfrac{-g_+(t_{1,1},t_{1,2})}{\tau_+(1+g_+(t_{1,1},t_{1,2}))} + \dfrac{g_-(t_{1,2},t_{1,3})}{\tau_-(1+g_-(t_{1,2},t_{1,3}))}\right] & \text{if } v(t_{1,2}+\tau_d)_{1,2} > V_{\text{th}},\ v(t_{1,4}+\tau_d)_{3,4} > V_{\text{th}}, \\ & c_3 < \dfrac{c_4+c_2+2\tau_d}{2} \text{ or } v(t_{1,2}+\tau_d)_{1,2} > V_{\text{th}}, \\ & v(t_{1,5}+\tau_d)_{3,5} > V_{\text{th}},\ c_3 < \dfrac{c_5+c_2+2\tau_d}{2} \\ 2E\left[\dfrac{-g_+(t_{1,1},t_{1,2})}{\tau_+(1+g_+(t_{1,1},t_{1,2}))} + \dfrac{g_-(t_{1,2},t_{1,3})}{\tau_-(1+g_-(t_{1,2},t_{1,3}))}\right. & \\ \left. + \dfrac{g_-(t_{1,2},t_{1,4})}{\tau_-(1+g_-(t_{1,2},t_{1,4}))} + \dfrac{g_-(t_{1,2},t_{1,5})}{\tau_-(1+g_-(t_{1,2},t_{1,5}))}\right] & \text{if } v(t_{1,2}+\tau_d)_{1,2} > V_{\text{th}},\ v(t_{1,5}+\tau_d)_{3,5} < V_{\text{th}} \\ 2E\left[\dfrac{g_+(t_{1,2},t_{1,4})}{\tau_+(1+g_+(t_{1,2},t_{1,4}))}\right] & \text{if } v(t_{1,4}+\tau_d)_{1,4} > V_{\text{th}}. \end{cases}$$

$$\frac{\partial E^2}{\partial c_3} = \begin{cases} 2E\left[\dfrac{-g_+(t_{1,1},t_{1,3})}{\tau_+(1+g_+(t_{1,1},t_{1,3}))} + \dfrac{-g_+(t_{1,2},t_{1,3})}{\tau_+(1+g_+(t_{1,2},t_{1,3}\ ))}\right. & \\ \left. + \dfrac{g_-(t_{1,3},t_{1,4})}{\tau_-(1+g_-(t_{1,3},t_{1,4}\ ))}\right] & \text{if } v(t_{1,3}+\tau_d)_{1,3} > V_{\text{th}}, \\ & v(t_{1,5}+\tau_d)_{4,5} > V_{\text{th}},\ c_3 > 2c_4 - c_5 - 2\tau_d \\ 2E\left[\dfrac{-g_+(t_{1,1},t_{1,3})}{\tau_+(1+g_+(t_{1,1},t_{1,3}))} + \dfrac{-g_+(t_{1,2},t_{1,3})}{\tau_+(1+g_+(t_{1,2},t_{1,3}))}\right] & \text{if } v(t_{1,3}+\tau_d)_{1,3} > V_{\text{th}},\ v(t_{1,5}+\tau_d)_{4,5} > V_{\text{th}}, \\ & c_3 < 2c_4 - c_5 - 2\tau_d \\ 2E\left[\dfrac{g_+(t_{1,3},t_{1,5})}{\tau_+(1+g_+(t_{1,3},t_{1,5}))}\right] & \text{if } v(t_{1,2}+\tau_d)_{1,2} > V_{\text{th}},\ v(t_{1,5}+\tau_d)_{3,5} > V_{\text{th}}, \\ & c_3 > \dfrac{c_5+c_2+2\tau_d}{2} \quad \text{or} \\ & v(t_{1,5}+\tau_d)_{1,5} > V_{\text{th}} \\ 2E\left[\dfrac{-g_-(t_{1,2},t_{1,3})}{\tau_-(1+g_-(t_{1,2},t_{1,3}))}\right] & \text{if } v(t_{1,2}+\tau_d)_{1,2} > V_{\text{th}},\ v(t_{1,5}+\tau_d)_{3,5} > V_{\text{th}}, \\ & c_3 < \dfrac{c_5+c_2+2\tau_d}{2} \quad \text{or} \\ & v(t_{1,2}+\tau_d)_{1,2} > V_{\text{th}},\ v(t_{1,4}+\tau_d)_{3,4} > V_{\text{th}}, \\ & c_3 < \dfrac{c_4+c_2+2\tau_d}{2} \text{ or} \\ & v(t_{1,2}+\tau_d)_{1,2} > V_{\text{th}},\ v(t_{1,5}+\tau_d)_{3,5} < V_{\text{th}} \\ 2E\left[\dfrac{g_+(t_{1,3},t_{1,4})}{\tau_+(1+g_+(t_{1,3},t_{1,4}))}\right] & \text{if } v(t_{1,2}+\tau_d)_{1,2} > V_{\text{th}},\ v(t_{1,4}+\tau_d)_{3,4} > V_{\text{th}}, \\ & c_3 > \dfrac{c_4+c_2+2\tau_d}{2} \text{ or} \\ & v(t_{1,4}+\tau_d)_{1,4} > V_{\text{th}} \\ 2E\left[\dfrac{-g_+(t_{1,1},t_{1,3})}{\tau_+(1+g_+(t_{1,1},t_{1,3}))} + \dfrac{-g_+(t_{1,2},t_{1,3})}{\tau_+(1+g_+(t_{1,2},t_{1,3}))}\right. & \\ \left. + \dfrac{g_-(t_{1,3},t_{1,4})}{\tau_-(1+g_-(t_{1,3},t_{1,4}))} + \dfrac{g_-(t_{1,3},t_{1,5})}{\tau_-(1+g_-(t_{1,3},t_{1,5}))}\right] & \text{if } v(t_{1,3}+\tau_d)_{1,3} > V_{\text{th}},\ v(t_{1,5}+\tau_d)_{4,5} < V_{\text{th}}. \end{cases}$$

$$\frac{\partial E^2}{\partial c_4} = \begin{cases} 2E\left[\dfrac{-g_-(t_{1,3},t_{1,4})}{\tau_-(1+g_-(t_{1,3},t_{1,4}))}\right] & \text{if } \nu(t_{1,3}+\tau_d)_{1,3} > V_{\text{th}},\ \nu(t_{1,5}+\tau_d)_{4,5} > V_{\text{th}}, \\ & c_4 < \dfrac{c_3+c_5+2\tau_d}{2} \text{ or} \\ & \nu(t_{1,3}+\tau_d)_{1,3} > V_{\text{th}},\ \nu(t_{1,5}+\tau_d)_{4,5} < V_{\text{th}} \\ 2E\left[\dfrac{g_+(t_{1,4},t_{1,5})}{\tau_+(1+g_+(t_{1,4},t_{1,5}))}\right] & \text{if } \nu(t_{1,3}+\tau_d)_{1,3} > V_{\text{th}},\ \nu(t_{1,5}+\tau_d)_{4,5} > V_{\text{th}}, \\ & c_4 > \dfrac{c_3+c_5+2\tau_d}{2} \text{ or} \\ & \nu(t_{1,2}+\tau_d)_{1,2} > V_{\text{th}},\ \nu(t_{1,5}+\tau_d)_{3,5} > V_{\text{th}}, \\ & c_3 > \dfrac{c_2+c_5+2\tau_d}{2} \text{ or} \\ & \nu(t_{1,2}+\tau_d)_{1,2} > V_{\text{th}},\ \nu(t_{1,5}+\tau_d)_{3,5} > V_{\text{th}}, \\ & c_3 < \dfrac{c_2+c_5+2\tau_d}{2} \\ 2E\left[\dfrac{-g_+(t_{1,3},t_{1,4})}{\tau_+(1+g_+(t_{1,3},t_{1,4}))} + \dfrac{g_-(t_{1,4},t_{1,5})}{\tau_-(1+g_-(t_{1,4},t_{1,5}))}\right] & \text{if } \nu(t_{1,2}+\tau_d)_{1,2} > V_{\text{th}}, \\ & \nu(t_{1,4}+\tau_d)_{3,4} > V_{\text{th}},\ c_4 < 2c_3 - c_2 - 2\tau_d \\ 2E\left[\dfrac{g_-(t_{1,4},t_{1,5})}{\tau_-(1+g_-(t_{1,4},t_{1,5}))}\right] & \text{if } \nu(t_{1,2}+\tau_d)_{1,2} > V_{\text{th}},\ \nu(t_{1,4}+\tau_d)_{3,4} > V_{\text{th}}, \\ & c_4 > 2c_3 - c_2 - 2\tau_d \\ 2E\left[\dfrac{-g_-(t_{1,2},t_{1,4})}{\tau_-(1+g_-(t_{1,2},t_{1,4}))}\right] & \text{if } \nu(t_{1,2}+\tau_d)_{1,2} > V_{\text{th}},\ \nu(t_{1,5}+\tau_d)_{3,5} < V_{\text{th}} \\ 2E\left[\dfrac{-g_+(t_{1,1},t_{1,4})}{\tau_+(1+g_+(t_{1,1},t_{1,4}))} + \dfrac{-g_+(t_{1,2},t_{1,4})}{\tau_+(1+g_+(t_{1,2},t_{1,4}))}\right. & \\ \quad \left. + \dfrac{-g_+(t_{1,3},t_{1,4})}{\tau_+(1+g_+(t_{1,3},t_{1,4}))} + \dfrac{g_-(t_{1,4},t_{1,5})}{\tau_-(1+g_-(t_{1,4},t_{1,5}))}\right] & \text{if } \nu(t_{1,4}+\tau_d,1,4) > V_{\text{th}} \\ 2E\left[\dfrac{g_+(t_{1,4},t_{1,5})}{\tau_+(1+g_+(t_{1,4},t_{1,5}))}\right] & \text{if } \nu(t_{1,5}+\tau_d,1,5) > V_{\text{th}}. \end{cases}$$

$$\frac{\partial E^2}{\partial c_5} = \begin{cases} 0 & \text{if } \nu(t_{1,3}+\tau_d)_{1,3} > V_{\text{th}}, \\ & \nu(t_{1,5}+\tau_d)_{4,5} > V_{\text{th}},\ c_5 > 2c_4 - c_3 - 2\tau_d \\ 2E\left[\dfrac{-g_+(t_{1,4},t_{1,5})}{\tau_+(1+g_+(t_{1,4},t_{1,5}))}\right] & \text{if } \nu(t_{1,3}+\tau_d)_{1,3} > V_{\text{th}}, \\ & \nu(t_{1,5}+\tau_d)_{4,5} > V_{\text{th}},\ c_5 < 2c_4 - c_3 - 2\tau_d \\ 2E\left[\dfrac{-g_+(t_{1,3},t_{1,5})}{\tau_+(1+g_+(t_{1,3},t_{1,5}))} + \dfrac{-g_+(t_{1,4},t_{1,5})}{\tau_+(1+g_+(t_{1,4},t_{1,5}))}\right] & \text{if } \nu(t_{1,2}+\tau_d)_{1,2} > V_{\text{th}}, \\ & \nu(t_{1,5}+\tau_d)_{3,5} > V_{\text{th}},\ c_5 < 2c_3 - c_2 - 2\tau_d \\ 2E\left[\dfrac{-g_+(t_{1,4},t_{1,5})}{\tau_+(1+g_+(t_{1,4},t_{1,5}))}\right] & \text{if } \nu(t_{1,2}+\tau_d)_{1,2} > V_{\text{th}}, \\ & \nu(t_{1,5}+\tau_d)_{3,5} > V_{\text{th}}, c_5 > 2c_3 - c_2 - 2\tau_d \\ 2E\left[\dfrac{-g_-(t_{1,4},t_{1,5})}{\tau_-(1+g_-(t_{1,4},t_{1,5}))}\right] & \text{if } \nu(t_{1,2}+\tau_d)_{1,2} > V_{\text{th}},\ \nu(t_{1,4}+\tau_d)_{3,4} > V_{\text{th}} \\ 2E\left[\dfrac{-g_-(t_{1,2},t_{1,5})}{\tau_-(1+g_-(t_{1,2},t_{1,5}))}\right] & \text{if } \nu(t_{1,2}+\tau_d)_{1,2} > V_{\text{th}},\ \nu(t_{1,5}+\tau_d)_{3,5} < V_{\text{th}} \\ 2E\left[\dfrac{-g_-(t_{1,3},t_{1,5})}{\tau_-(1+g_-(t_{1,3},t_{1,5}))}\right] & \text{if } \nu(t_{1,3}+\tau_d)_{1,3} > V_{\text{th}},\ \nu(t_{1,5}+\tau_d)_{4,5} < V_{\text{th}} \\ 2E\left[\dfrac{-g_-(t_{1,4},t_{1,5})}{\tau_-(1+g_-(t_{1,4},t_{1,5}))}\right] & \text{if } \nu(t_{1,4}+\tau_d)_{1,4} > V_{\text{th}} \\ 2E\left[\dfrac{-g_+(t_{1,1},t_{1,5})}{\tau_+(1+g_+(t_{1,1},t_{1,5}))} + \dfrac{-g_+(t_{1,2},t_{1,5})}{\tau_+(1+g_+(t_{1,2},t_{1,5}))}\right. & \\ \quad \left. + \dfrac{-g_+(t_{1,3},t_{1,5})}{\tau_+(1+g_+(t_{1,3},t_{1,5}))} + \dfrac{-g_+(t_{1,4},t_{1,5})}{\tau_+(1+g_+(t_{1,4},t_{1,5}))}\right] & \text{if } \nu(t_{1,5}+\tau_d)_{1,5} > V_{\text{th}}. \end{cases}$$

(A.1)

Acknowledgment

This work was supported by the National Science Foundation, under ECCS Grant 0925407.

References

[1] J. J. B. Jack, D. Nobel, and R. Tsien, *Electric Current Flow in Excitable Cells*, Oxford University Press, Oxford, UK, 1st edition, 1975.

[2] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of Physiology*, vol. 117, no. 4, pp. 500–544, 1952.

[3] W. Maass, "Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons," *Advances in Neural Information Processing Systems*, vol. 9, pp. 211–217, 1997.

[4] C. M. A. Pennartz, "Reinforcement learning by Hebbian synapses with adaptive thresholds," *Neuroscience*, vol. 81, no. 2, pp. 303–319, 1997.

[5] S. Ferrari, B. Mehta, G. Di Muro, A. M. J. VanDongen, and C. Henriquez, "Biologically realizable reward-modulated hebbian training for spiking neural networks," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '08)*, pp. 1780–1786, Hong Kong, June 2008.

[6] R. Legenstein, C. Naeger, and W. Maass, "What can a neuron learn with spike-timing-dependent plasticity?" *Neural Computation*, vol. 17, no. 11, pp. 2337–2382, 2005.

[7] J. P. Pfister, T. Toyoizumi, D. Barber, and W. Gerstner, "Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning," *Neural Computation*, vol. 18, no. 6, pp. 1318–1348, 2006.

[8] R. V. Florian, "Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity," *Neural Computation*, vol. 19, no. 6, pp. 1468–1502, 2007.

[9] S. G. Wysoski, L. Benuskova, and N. Kasabov, "Adaptive learning procedure for a network of spiking neurons and visual pattern recognition," in *Proceedings of the 8th International Conference on Advanced Concepts for Intelligent Vision Systems (ACIVS '06)*, vol. 4179 of *Lecture Notes in Computer Science*, pp. 1133–1142, Antwerp, Belgium, September 2006.

[10] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano Letters*, vol. 10, no. 4, pp. 1297–1301, 2010.

[11] A. M. VanDongen, "Vandongen laboratory," http://www.vandongen-lab.com/.

[12] T. J. Van De Ven, H. M. A. VanDongen, and A. M. J. VanDongen, "The nonkinase phorbol ester receptor α1-chimerin binds the NMDA receptor NR2A subunit and regulates dendritic spine density," *Journal of Neuroscience*, vol. 25, no. 41, pp. 9488–9496, 2005.

[13] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*, MIT Press, Cambridge, Mass, USA, 2001.

[14] W. Gerstner and W. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*, Cambridge University Press, Cambridge, UK, 2006.

[15] G. Foderaro, C. Henriquez, and S. Ferrari, "Indirect training of a spiking neural network for flight control via spike-timing-dependent synaptic plasticity," in *Proceedings of the 49th IEEE Conference on Decision and Control (CDC '10)*, pp. 911–917, Atlanta, Ga, USA, December 2010.

[16] A. Aldroubi and K. Gröchenig, "Nonuniform sampling and reconstruction in shift-invariant spaces," *SIAM Review*, vol. 43, no. 4, pp. 585–620, 2001.

[17] A. A. Lazar and L. T. Toth, "A toeplitz formulation of a real-time algorithm for time decoding machines," in *Proceedings of the Telecommunication Systems, Modeling and Analysis Conference*, 2003.

[18] E. M. Izhikevich, "Which model to use for cortical spiking neurons?" *IEEE Transactions on Neural Networks*, vol. 15, no. 5, pp. 1063–1070, 2004.

[19] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[20] A. N. Burkitt, "A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input," *Biological Cybernetics*, vol. 95, no. 1, pp. 1–19, 2006.

[21] S. Song, K. D. Miller, and L. F. Abbott, "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity," *Nature Neuroscience*, vol. 3, no. 9, pp. 919–926, 2000.

[22] P. Sjostrom, G. Turrigiano, and S. Nelson, "Rate, timing, and cooperativity jointly determine cortical synaptic plasticity," *Neuron*, vol. 32, no. 6, pp. 1149–1164, 2001.

[23] S. Ferrari and R. Stengel, "Model-based adaptive critic designs," in *Learning and Approximate Dynamic Programming*, J. Si, A. Barto, and W. Powell, Eds., John Wiley & Sons, 2004.

[24] R. E. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, NJ, USA, 1957.

[25] R. Howard, *Dynamic Programming and Markov Processes*, MIT Press, Cambridge, Mass, USA, 1960.

[26] A. M. J. VanDongen, J. Codina, J. Olate et al., "Newly identified brain potassium channels gated by the guanine nucleotide binding protein G(o)," *Science*, vol. 242, no. 4884, pp. 1433–1437, 1988.

[27] J. E. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, SIAM, Englewood Cliffs, NJ, USA, 1996.

[28] H. G. Feichtinger, J. C. Príncipe, J. L. Romero, A. Singh Alvarado, and G. A. Velasco, "Approximate reconstruction of bandlimited functions for the integrate and fire sampler," *Advances in Computational Mathematics*, vol. 36, no. 1, pp. 67–78, 2012.

[29] R. F. Stengel, *Optimal Control and Estimation*, Dover Publications, Inc., 1986.