LETTER Communicated by Sander M. Bohte

Spike-Timing Error Backpropagation in Theta Neuron Networks

Sam McKennoch
Samuel.McKennoch@loria.fr
Thomas Voegtlin
voegtlin@loria.fr
INRIA, Campus Scientifique, F-54506 Vandoevre-Les-Nancy, France

Linda Bushnell
lb2@u.washington.edu
Department of Electrical Engineering, University of Washington, Seattle, WA 98195, U.S.A.

The main contribution of this letter is the derivation of a steepest gradient descent learning rule for a multilayer network of theta neurons, a one-dimensional nonlinear neuron model. Central to our model is the assumption that the intrinsic neuron dynamics are sufficient to achieve consistent time coding, with no need to involve the precise shape of postsynaptic currents; this assumption departs from other related models such as SpikeProp and Tempotron learning. Our results clearly show that it is possible to perform complex computations by applying supervised learning techniques to the spike times and time response properties of nonlinear integrate and fire neurons. Networks trained with our multilayer training rule are shown to have similar generalization abilities for spike latency pattern classification as Tempotron learning. The rule is also able to train networks to perform complex regression tasks that neither SpikeProp or Tempotron learning appears to be capable of.

1 Introduction

Recent neuroscience research has shown that substantial information about stimuli is contained in the timing of action potentials. Examples include echolocation in bats, changes in insect flight patterns, visual stimuli recognition, and monkey reaction time in which it appears that only a small number of spikes is involved in the computation (Thorpe, Fize, & Marlot, 1996; Rieke, Warland, Steveninck, & Bialek, 1999). Other experiments have shown that spike timing is important for hyperacuity, in which the signal-to-noise ratio of stimuli and output spikes allows a greater resolution than at first seems possible. For example, weakly electric fish can respond to

signal changes from other fish on the order of 100 nanoseconds (Carr, Heiligenberg, & Rose, 1986).

Computer simulations have shown that spiking neurons have computational advantages over rate-coded neurons (Maass, 1996b), are able to approximate complicated mathematical functions (Iannella & Kindermann, 2005), and can generate complex periodic spike patterns (Memmesheimer & Timme, 2006). Due to the temporal coding inherent to spiking neuron networks, they are especially useful for tasks with a temporal component, such as speech recognition (Alnajjar & Murase, 2005) or the control of artificial neuroprosthetics (Popović & Sinkjr, 2000).

Previous attempts at training spiking neural networks have used a linear summation of postsynaptic potentials (PSP) at the soma, combined with gradient-based supervised learning rules (Bohte, 2003; Gütig & Sompolinsky, 2006) or more biologically realistic spike-timing-dependent plasticity learning rules (Legenstein, Naeger, & Maass, 2005). For example, SpikeProp (Bohte, 2003) works by linearizing the voltage at output spike times in order to prevent discontinuities. A number of extensions to SpikeProp have appeared over the years, including networks of spiking neurons that produce multiple output spikes (Booij & Nguyen, 2005) and reducing training times (McKennoch, Lui, & Bushnell, 2006; Schrauwen & Van Campenhout, 2004; Xin & Embrechts, 2001). In the case of the Tempotron algorithm (Gütig & Sompolinsky, 2006), the learning rule attempts to minimize the amount by which the maximum neuron potential for incorrectly classified patterns deviates from the firing threshold. Tempotrons have been demonstrated on binary classification tasks, which are only a small subset of typical learning problems. Classification is determined by either the presence or absence of an output spike, a spike coding that is not readily extendable to nonbinary classification tasks. According to Gütig and Sompolinsky (2006), the Tempotron is also expected to have degraded performance at classification tasks where the input spike dynamics exist at multiple scales.

In these models, the shape of the PSP is of critical importance to the training algorithm (in Tempotron, the time of maximum neuron potential is directly dependent on the PSP shape; in Spikeprop, the PSP shape determines when the threshold will be crossed). This shape is dictated by a simple first-order linear differential equation along with a voltage reset rule for when the spiking threshold is crossed, which together are known as the linear integrate-and-fire (LIF) model.

A clear advantage of first-order dynamics is that it makes the derivation of a learning rule simpler, because the contribution of a spike does not depend on the internal state of the postsynaptic neuron. The postsynaptic and somatic membrane potentials are written as explicit functions of time, thus abstracting away from internal neural dynamics. Hence, the potential at the soma is a weighted sum of PSPs, and the contribution of a PSP to the sum is not dependent on the internal state of the neuron when this PSP arrives.

However, this approach moves the computational burden from the soma to the synapses. Indeed, the PSPs computed at these synapses must be made to interact with each other in the precise way required for desired output spike times. Bohte, Kok, and La Poutré (2000) require the use of multiple delay connections between each neuron pair in order to allow different interactions between the weighted PSPs. For the XOR problem, SpikeProp requires about 16 connections between each neuron pair, each with a different delay. These extra parameters add complexity to the learning problem. More crucial, the interaction between PSPs must take place on a small timescale, limited by the size of the rising segment of the PSPs (Maass, 1996a). The difficulties faced by LIF-based networks suggest that something important has been lost in the simplification. Some authors have argued that the LIF model should be avoided at all cost (Dayan & Abbott, 2001; Florian, 2003; Izhikevich, 2004), because assumptions such as a linear approximation for the overall membrane current and a simplified action potential description are too simple to reflect real neuron dynamics.

In order to increase biological realism, the LIF model used by SpikeProp and Tempotron learning, also called the simplified spike response model (SRM_0; Gerstner & Kistler, 2002), can be made into the full SRM by adding dependencies on the time since the last output spike was produced into the response kernels (Gerstner & Kistler, 2002). As was the case with SRM_0, the full SRM does not have internal state variables. Instead, voltage is still an explicit function of time, which in principle would make it possible to extend the approach used in SpikeProp or Tempotron. However, the mathematical analysis is much more difficult in that case, and a learning rule adapted to the SRM has yet to be derived.

Another way of matching more closely the neural responses is to introduce more realistic differential equations in the description of the neuronal dynamics (Feng, 2001; Izhikevich, 2004). Perhaps the most realistic conductance-based dynamical neuron model is the Hodgkin-Huxley (HH) model. Parameters in this model were derived empirically in part, and thus exhibit good biophysical grounding. The model consists of four interdependent differential equations that model the membrane potential and gating variables. Given that there are four state variables, the dynamics of the HH model are not entirely amenable to analysis. A common simplification is to assume that certain gating variables have very fast time constants and thus can be approximated by their steady-state values. This simplification reduces the model to a membrane potential variable and a recovery variable that can then be analyzed using phase plane analysis. Two-dimensional biophysically grounded models include the FitzHugh-Nagumo and Morris-Lecar models (Gerstner & Kistler, 2002). The Izhikevich simple model of choice is also two-dimensional and is designed to capture a full range of neuron dynamics at the expense of reduced biophysical grounding (Izhikevich, 2006). Even with the reduction to two state variables, the neuron model is complex enough to make

certain types of analysis very difficult, if not prohibitive. However, rather than return to the LIF or SRM_0 models, Izhikevich (2004) argues that quadratic integrate-and-fire (QIF) neurons should be used instead; they are nearly as computationally efficient but have additional important dynamic properties, such as spike latency, bistability of resting and tonic spiking modes, and activity-dependent thresholding. QIF neurons also have a frequency response that matches biological observations better than the LIF neuron (Brunel & Latham, 2003). By a simple transformation, the QIF can be mapped to the theta neuron model, a nonlinear phase model.

Nonlinear models such as the QIF have the important property that their response to an incoming synaptic current is not constant across time but changes with the internal state of the neuron. Voegtlin (2007) proposed that time coding should be adapted to this property. In this proposal, spikes arriving at different times will trigger different responses, and this precisely attributes different meaning to different spike times. Thus, the dynamic response properties of neurons are essential and must be present in the neuron model. This cannot be achieved with the LIF model. However, the relatively simple (one-dimensional) theta model has nonlinearities that allow modeling variable response properties. The cyclic nature of the theta neuron phase model also allows a continuous reset (as opposed to the quadratic model), thus allowing derivative-based calculations without any sort of linearization assumptions. This model was chosen for the derivation of a gradient-based learning rule (Voegtlin, 2007). The learning rule was later extended to a robotic application where the activity-dependent thresholding property is used as a noise filter (McKennoch, Sundaradevan, & Bushnell, 2007). However, the method by Voegtlin (2007) was developed for a single layer of neurons only, which is not sufficient to approximate complex functions. In addition, it made certain assumptions about the gradient that prevent training in many cases.

In this letter, we develop a multilayer gradient descent learning rule that is based on the theta neuron model dynamics and can perform the same kind of tasks as SpikeProp or Tempotron. In our model, computations rely not on the shape of postsynaptic currents but on the intrinsic neuron dynamics. Thus, in the derivation of our learning rule, we greatly simplify the shape of synaptic currents. We are not claiming that synaptic time constants do not play a role in natural systems. For example, synaptic time constants have an effect on the period of oscillations in coupled networks (Marinazzo, Kappen, & Gielen, 2007; Börgers & Kopell, 2003). However, our hypothesis is that they are not essential to time coding.

Essential for this learning rule is the development of an analytical event-driven framework for simulating theta neurons. Event-driven methods use the exact or analytical solution to the neuron's dynamics. The state trajectory of spiking neurons is determined by the intrinsic neuron dynamics and the synaptic inputs. When the synaptic input spikes are relatively sparse,

it is more efficient to view the neuron as a hybrid system having both time-driven state changes as driven by the intrinsic dynamics and asynchronous event-driven state changes as driven by synaptic inputs (Cassandras & Lafortune, 2006). If an analytic solution exists for the time-driven state changes, we need only separately calculate the event-driven changes of state when an input is received and then use the analytical solutions to the dynamics of the neuron to project the state to the next input spike time.

Additionally, many of the more complex neuron models do not have a closed-form analytic solution for the state variables; thus, numerical integration is unavoidable. Numerical integration is sensitive to an artificially chosen time step. If the time step is too large, the firing results may be inaccurate. This inaccuracy is increased as the magnitude of the synaptic efficiency parameters increases. If the time step is too small, simulations will take an unreasonable amount of time to complete. Event-driven methods have been developed for the LIF neuron (Mattia & Giudice, 2000) and more recently for the QIF neuron as well (Tonnelier, Belmabrouk, & Martinez, 2007). The development of this event-driven framework and the accompanying learning rule constitute the novel work in this letter and together are used to perform complex computations using only the intrinsic dynamics of nonlinear neuron models.

The letter is organized as follows. In section 2, we present the theta neuron model and its properties. In section 3, we derive a learning rule that can be applied to both the output and hidden layer of a multilayer theta neuron network. Section 4 describes our event-driven simulation framework. In section 5, we verify the new learning rule through a number of simulation experiments. We include an analysis of learning rule performance using simple inverter and delayer data sets and replicate the spike latency pattern classification experiment performed with Tempotron learning in Gütig and Sompolinsky (2006). Next, we train a multilayer theta neuron network to learn a number of standard machine learning classification and regression problems. Comparisons are made to similar results produced using SpikeProp. Experiments are also performed to demonstrate the sensitivity to three important network parameters. Section 6 gives conclusions and future work.

2 Theta Neuron Model

In this section, we present the theta neuron model, the neuron model around which our learning rule is derived. The theta neuron model is a canonical model, meaning that it is useful for capturing and generalizing the relevant dynamics of a class of models. More specifically, the model is canonical in the sense that any type I neuron close to its bifurcation point (of type saddle node on invariant circle (SNIC) since this is a type I neuron), can be mapped to the theta neuron model. In type I neurons, the output spike frequency increases smoothly from zero as the input current is increased. This type

of neuron model is well suited for modeling cortical excitatory pyramidal neurons (Izhikevich, 2006).

As with all other canonical models, the theta neuron model is formed from a piece-wise continuous change of variables from other neuron models. In the case of the QIF neuron model, if we perform the following change of variables from potential, u, to phase, θ,

$$u(t) = \tan\left(\frac{\theta(t)}{2}\right), \tag{2.1}$$

we arrive at the canonical theta neuron model (Ermentrout & Kopell, 1986). The theta neuron model produces an output spike when the phase passes π. This change of variables indicates that we are considering the case where the quadratic model spike magnitude goes to infinity (in finite time), corresponding to a phase of π. The phase evolves toward π in the absence of input spikes once the spiking threshold (θ_{FP}^{+}) has been passed.

The theta neuron model has been used in a number of experiments, such as synaptically generating traveling waves (Osan & Ermentrout, 2001) and developing spike train statistics in the presence of white gaussian noise (Lindner, Longtin, & Bulsara, 2003). Theta neurons represent an excellent trade-off between model complexity and analytical tractability. Being a one-dimensional model, they cannot model complex neuron dynamics such as bursting; however, their dynamics can be viewed quite simply on a phase circle rather than resorting to full phase plane analysis for 2D models. The quadratic and the theta neuron model are among the simplest models that are excitable, which means that for small perturbations from the neuron's resting equilibrium, these perturbations may become greatly amplified before the neuron returns to rest. Spikes may be cancelled here as well. Recovery is implicit in the cyclical nature of the theta neuron model, eliminating any need for a separate recovery state variable.

The trajectory of the phase in a theta neuron is described by

$$\tau\frac{d\theta}{dt} = (1-\cos\theta) + \alpha I(t)(1+\cos\theta), \tag{2.2}$$

where τ is the neuron phase time constant, α is a scaling constant, and $I(t)$ is the input current that drives the dynamics. As was done in Lim and Kim (2007), τ is set to 1 ms to create a correspondence between this model and real spikes that have widths on the order of milliseconds. As previously stated, we focus on using the neuron dynamics to support complex calculations. Because we are assuming that synaptic time constants are nonessential, we are able to treat our synaptic current inputs as instantaneous shock inputs modeled by Dirac delta functions. In this way, equation 2.2 remains the only equation of state that we need to keep track of.

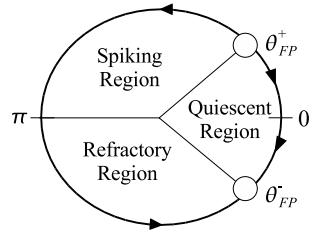


Figure 1: Theta neuron phase circle. The theta neuron phase circle is shown for the case where the baseline current $I_o < 0$. When the phase passes π, the neuron produces an output spike. In the spiking region, the neuron will fire after some period of time. In the quiescent region, the phase will decay back to θ^-_{FP} unless enough excitatory input spikes push the phase into the spiking region. In the refractory region, the phase is recovering from a previous output spike, thereby making it more difficult to produce another output spike immediately. This type of refractoriness is relative (rather than absolute in which immediately producing another output spike is impossible).

Synaptic currents depolarize or hyperpolarize the membrane potential, and thereby increase or decrease the phase, depending on the sign of the synaptic efficiency. The total input current, $I(t)$, is defined as the sum of a baseline current, I_o, plus the sum of J synaptic impulse inputs at times t_j with size modulated by weights w_j:

$$I(t) = I_o + \sum_{j=1}^{J} w_j \delta\left(t - t_j\right). \tag{2.3}$$

When $I_o > 0$, a limit cycle forms, which causes spikes to fire periodically as a function of I_o. As I_o approaches zero, the frequency also approaches zero, which is, in effect, the definition of type I neural excitability. When $I_o < 0$, two fixed points can be found by setting equation 2.2 equal to zero and solving for θ. One fixed point is a saddle point (θ^+_{FP}), and the other is a stable fixed point (θ^-_{FP}). The neuron phase (which directly maps to the membrane potential) moves around its phase circle toward the stable fixed point and away from the saddle point. When the phase crosses π, the neuron is said to have fired. In Figure 1, the unforced response of the neuron is plotted as a phase circle and is broken up into three operating regions for the case where $I_o < 0$. The neuron phase right before the jth input spike arrives is defined as θ^-_j, while the phase right after this input spike is defined as θ^+_j. Because synaptic currents are modeled as impulses,

$$\theta^+_j = 2\,\mathrm{atan}\left(\alpha w_j + \tan\left(\frac{\theta^-_j}{2}\right)\right), \tag{2.4}$$

which follows from equations 2.1, 2.2, and 2.3. As in Voegtlin (2007), the remaining time until the next output spike as a function of the neuron phase is

$$F(t) = \int_{\theta(t)}^{\pi} \frac{d\theta}{(1-\cos\theta) + \alpha I(t)(1+\cos\theta)} = F\big|_{\theta(t)}^{\pi}.$$

If there are input spikes, F is discontinuous and can be broken up into continuous pieces,

$$F(t_i) = \sum_{j=i}^{J-1} \int_{\theta_j^+}^{\theta_{j+1}^-} \frac{d\theta}{(1-\cos\theta) + \alpha I_o(1+\cos\theta)} = \sum_{j=i}^{J-1} F\big|_{\theta_j^+}^{\theta_{j+1}^-},$$

where $\theta_J^- = \pi$. The input spike times are ordered; thus, $t_2 > t_1$, and so forth.

Theta neurons exhibit the interesting properties of spike latency and activity-dependent thresholding, both of which do not exist in the LIF model. Spike latency refers to the separation of the time between when the phase exceeds the spiking threshold ($\theta > \theta_{FP}^+$) and when an output spike is generated ($\theta = \pi$). This separation allows further incoming input spikes to modulate or even cancel the output spike. Activity-dependent thresholding means that a spike's effect on the phase depends on the internal neuron state. This state is determined by the magnitude and timings of input spikes already received, as well as the current spike, as shown in Figure 2.

A strong benefit of activity-dependent thresholding in our model is the long timescale over which computations can take place. In the SpikeProp model, the length of the rising segment of the PSP limits the range of computation. For example, in a two-input data set, for the SpikeProp model, if the inputs are spaced farther apart in time than a few synaptic time constants, then the effects of the input spikes will be independent of one another, greatly reducing the computational possibilities. In contrast, our practical range is the section of the response curve where the response is not constant, which, as shown in Figure 2, is on the order of tens of milliseconds. Unlike the SpikeProp and Tempotron models, these long timescales also allow computations where input spike dynamics exist at multiple scales.

In addition, activity-dependent thresholding eliminates the need for delays. In the SpikeProp learning methodology, each synapse must be expanded to a group of synapses, each with a different delay, such that there is an appropriate range of delays specific to the training data set. These delays may also be trained using the method in Schrauwen and Van Campenhout (2004). Delays are needed to cause the increases in potential from each input spike to interact in the way needed to produce the desired output spike times because an input spike of a given magnitude will change the neuron potential by an equal amount regardless of when the input spike occurs.

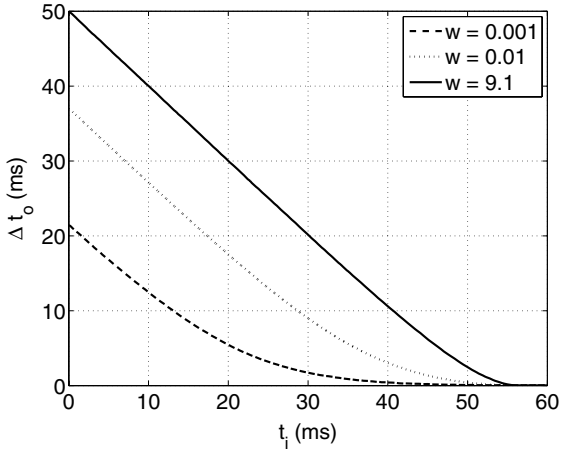


Figure 2: Theta neuron response curves. The curves show the change in output firing time (Δt_o) when there is a single input spike (at t_i) compared to no input spikes at all. The baseline current, $I_o = -0.005$. Curves are shown for different positive values of synaptic weights. In all cases, the firing time is more affected when the input spike occurs earlier. Given that the initial phase used here is 0.0001 above θ^+_{FP}, if no input spikes occurred, the neuron would fire at a baseline firing time, $t_{BL} = 56.2$ ms. Of importance is the time range over which the response is not constant, meaning the range over which computation is possible. Depending on the weight, this time range is nearly the entire span of time from 0 ms until t_{BL}.

Recall that with the original SpikeProp, 16 separate connections between neuron pairs were needed, each with different delays, in order to learn the simple binary XOR problem. In the theta model, the effect of the delay is incorporated directly into the intrinsic neuron dynamics, thus eliminating the need for delay training or multiple static delay connections.

3 Theta Neuron BackProp Training Rule

This section presents the development of a steepest-descent error backpropagation learning rule for theta neuron networks. The method described refines our recently developed gradient-based method for training the output firing times of a single layer of theta neurons (Voegtlin, 2007). The previous method applied only to a single layer of theta neurons and also made certain simplifications in the error gradient calculation that under many conditions make it a nonideal approximator to the exact error gradient. The learning rule derived applies to neurons that receive and produce one spike per synapse, although the training rule could be generalized for multispike domains.

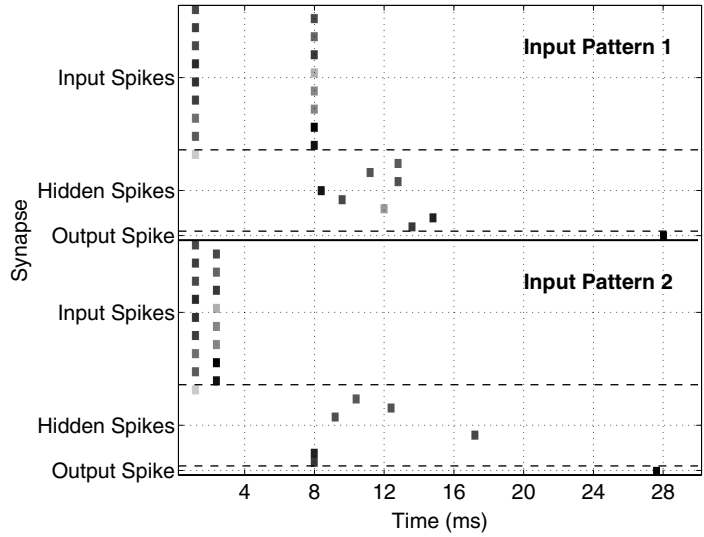


Figure 3: Multilayer computation. This raster plot shows the spikes being propagated by all the neurons in a network trained for calculating two different input patterns of the cosine function. There is a reference input at 1 ms and two inputs at 8 ms and 2.23 ms, respectively, representing cosine input values. Each gray-scale pixel represents a spike on a specific synaptic connection. There are two input spikes, but because there are eight hidden neurons (not all of which fire in pattern 2), the spikes get modulated differently coming into each hidden neuron. The gray-scale pixel coloring refers to the synaptic weight, with darker colors being more excitatory and lighter colors being more inhibitory.

The network topology is a fully connected multilayer feedforward network. We use the following notation convention: $\mathcal{I}$, $\mathcal{H}$, and $\mathcal{O}$, the set of all input, hidden, and output layer neuron indices, respectively. The sizes of these layers are $|\mathcal{I}| = M$, $|\mathcal{H}| = N$, and $|\mathcal{O}| = P$. The indices j and k are used as generic neuron indices. The indices m, n, and p are used for referring to neurons on the input, hidden, and output layers, respectively. The desired firing time of an output neuron is $\bar{t}_p$, while the actual firing time is t_p. Neurons in a layer are indexed by the order of the output spikes that they produce—hence, for the hidden neuron, which produces the first output spike, $n = 1$. This choice of notation may cause the indexing to change from one input pattern to the next, but this change is not problematic. Figure 3 shows a spatiotemporal raster plot of the spikes propagating through a network trained to calculate the cosine function (see section 5).

Our learning rule is based on the steepest gradient-descent method. Steepest gradient-descent learning algorithms are often the basis of more complex learning rules—RProp and QuickProp, for example (Reed & Marks, 1998). The learning rule developed here is shown to be structurally

similar to the classic Error BackProp used in rate-coded neurons but is adapted to the nonlinear spiking theta neuron model. For example, because the effect of an input spike is voltage dependent, the calculation of the error backpropagation terms becomes recursive. The derivatives needed for gradient descent are based on event-driven simulation techniques, which are developed in section 4. This technique involving the analytical solution of the remaining time function is much more efficient than time-based techniques when the discontinuous changes to the neuron phase are relatively sparse.[1]

3.1 Output Layer Weight Training. The following derivation describes how weights affiliated with connections between hidden and output neurons should be adjusted in order to achieve desired firing times. The error for a single input pattern is defined as a sum of squared errors (SSE):

$$E = \frac{1}{2}\sum (t_p - \bar{t}_p)^2. \tag{3.1}$$

As with any other gradient steepest descent learning rule, the weights are incrementally adjusted in the opposite direction of the error gradient. In batch learning, all input patterns are presented before the weights are changed (by the summation of the changes required by each input pattern), while for online learning, weights are changed after the presentation of each input pattern. In either case, one epoch is defined as the presentation of all input patterns once. Following the notation of traditional error backpropagation in rate-coded neural networks, the gradient calculation may be broken up into terms found by moving forward through the network (y) and terms found by backpropagating the error back through the network (δ). For the output layer,

$$\Delta w_{np} = -\eta_w \frac{\partial E}{\partial w_{np}} = -\eta_w \frac{\partial E}{\partial t_p}\frac{\partial t_p}{\partial w_{np}} = -\eta_w \delta_p y_{np}, \tag{3.2}$$

where η_w is the synaptic efficiency learning rate. The gradient term $\partial E / \partial w_{np}$ is found by taking derivatives on the event-driven equations found in section 4. The functional equivalents are reproduced here:

$$\begin{aligned} E &= f_1(t_p, \bar{t}_p), & t_p &= f_2(t_{\hat{N}}, \theta_{\hat{N}}^+, \alpha I_o), \\ \theta_j^+ &= f_3(\theta_j^-, w_j, \alpha), & \theta_j^- &= f_4(\theta_{j-1}^+, t_j - t_{j-1}, \alpha I_o). \end{aligned} \tag{3.3}$$

[1]The gradient equations derived in this section were verified by comparing to a numerically calculated gradient over different parameter combinations. In all cases, the difference between the numerical and calculated gradient was negligible, and as the numerical step size decreased, the difference decreased as well.

Let $\hat{N}$ be the index for the final input spike before an output spike is received. Input spikes that occur after the output spike have no effect. Expanding equation 3.2,

$$\Delta w_{np} = \frac{\partial E}{\partial t_p}\frac{\partial t_p}{\partial \theta^+_{\hat{N}p}}\frac{\partial \theta^+_{\hat{N}p}}{\partial \theta^-_{\hat{N}p}}\frac{\partial \theta^-_{\hat{N}p}}{\partial \theta^+_{(\hat{N}-1)p}}\cdots\frac{\partial \theta^-_{(n+1)p}}{\partial \theta^+_{np}}\frac{\partial \theta^+_{np}}{\partial w_{np}}.$$

This chain rule expansion relates a weight's effect on the final error by stepping through all the functional dependencies. The input spike from neuron n will alter the effect that all subsequent spikes (from neurons $n+1$ through $\hat{N}$) have on the phase of neuron p and therefore the output spike time, t_p, as well. The first and second terms are easily calculated:

$$\frac{\partial E}{\partial t_p} = t_p - \bar{t}_p, \quad \frac{\partial t_p}{\partial \theta^+_{\hat{N}p}} = \frac{-1}{(1-\cos\theta^+_{\hat{N}p}) + \alpha I_o(1+\cos\theta^+_{\hat{N}p})}.$$

The last term and terms of the type $\partial\theta^+_{jp}/\partial\theta^-_{jp}$ are calculated by differentiating equation 2.4:

$$\frac{\partial \theta^+_{np}}{\partial w_{np}} = \alpha(1+\cos\theta^+_{np}), \quad \frac{\partial \theta^+_{jp}}{\partial \theta^-_{jp}} = \frac{(1+\cos\theta^+_{jp})}{(1+\cos\theta^-_{jp})}.$$

Terms of the type $\partial\theta^-_{jp}/\partial\theta^+_{(j-1)p}$ come from taking the derivative of f_4 in equation 3.3 and applying a number of standard identities from trigonometry:

$$\frac{\partial \theta^-_{jp}}{\partial \theta^+_{(j-1)p}} = \frac{(1-\cos\theta^-_{jp}) + \alpha I_o(1+\cos\theta^-_{jp})}{\big(1-\cos\theta^+_{(j-1)p}\big) + \alpha I_o\big(1+\cos\theta^+_{(j-1)p}\big)}.$$

Combining the previous two equations,

$$\frac{\partial \theta^+_{jp}}{\partial \theta^+_{(j-1)p}} = \frac{\big(\tan^2\big(\frac{\theta^-_{jp}}{2}\big) + \alpha I_o\big)\big(1+\cos\theta^+_{jp}\big)}{\big(1-\cos\theta^+_{(j-1)p}\big) + \alpha I_o\big(1+\cos\theta^+_{(j-1)p}\big)} \tag{3.4}$$

Putting the results together reduces to the following learning rule:

Output-Layer Learning Rule

$$\begin{aligned}\Delta w_{np} &= -\eta_w \delta_p y_{np} \\ \delta_p &= (t_p - \bar{t}_p) \\ y_{np} &= \frac{-\alpha(1+\cos\theta^+_{np})}{(1-\cos\theta^+_{\hat{N}p}) + \alpha I_o(1+\cos\theta^+_{\hat{N}p})}\prod_{j=n+1}^{\hat{N}}\frac{\partial\theta^+_{jp}}{\partial\theta^+_{(j-1)p}}.\end{aligned} \tag{3.5}$$

3.2 Hidden-Layer Weight Training. A portion of the analysis of the hidden layer is similar to the output-layer analysis. First consider

$$\Delta w_{mn} = -\eta_w \frac{\partial E}{\partial w_{mn}} = -\eta_w \frac{\partial E}{\partial t_n} \frac{\partial t_n}{\partial w_{mn}} = -\eta_w \delta_n y_{mn}.$$

The analysis for the last term, $\partial t_n / \partial w_{mn}$, is identical to that from the output layer, arriving at

$$\frac{\partial \theta^+_{jn}}{\partial \theta^+_{(j-1)n}} = \frac{\left(\tan^2\left(\frac{\theta^-_{jn}}{2}\right) + \alpha I_o\right)\left(1 + \cos\theta^+_{jn}\right)}{\left(1 - \cos\theta^+_{(j-1)n}\right) + \alpha I_o\left(1 + \cos\theta^+_{(j-1)n}\right)}$$

$$y_{mn} = \frac{-\alpha\left(1 + \cos\theta^+_{mn}\right)}{\left(1 - \cos\theta^+_{\hat{M}n}\right) + \alpha I_o\left(1 + \cos\theta^+_{\hat{M}n}\right)} \prod_{j=m+1}^{\hat{M}} \frac{\partial \theta^+_{jn}}{\partial \theta^+_{(j-1)n}}.$$

Now back to the first term,

$$\delta_n = \frac{\partial E}{\partial t_n} = \sum_{t_p > t_n} \frac{\partial E}{\partial t_p} \frac{\partial t_p}{\partial t_n}$$

The second term in this summation is the most complicated term in this derivation. By taking into account the functional dependencies described in equation 3.3,

$$\frac{\partial t_p}{\partial t_n} = \frac{\partial t_p}{\partial \theta^+_{\hat{N}p}} \frac{\partial \theta^+_{\hat{N}p}}{\partial \theta^-_{\hat{N}p}} \left(\prod_{k=\hat{N}}^{n+2} \frac{\partial \theta^-_{kp}}{\partial \theta^+_{(k-1)p}} \frac{\partial \theta^+_{(k-1)p}}{\partial \theta^-_{(k-1)p}} \right) \frac{d\theta^-_{(n+1)p}}{dt_n}. \tag{3.6}$$

Further expanding the last term,

$$\frac{d\theta^-_{(n+1)p}}{dt_n} = \left(\frac{\partial \theta^-_{(n+1)p}}{\partial t_n} + \frac{\partial \theta^-_{(n+1)p}}{\partial \theta^+_{np}} \frac{\partial \theta^+_{np}}{\partial \theta^-_{np}} \frac{\partial \theta^-_{np}}{\partial t_n} \right).$$

There are two terms inside the parentheses because $\theta^-_{(n+1)p}$ is a function of the change in time since the last input spike, so t_n appears twice. Much of equation 3.6 has already been calculated by y_{np}. Thus, to avoid redundant calculations, it is helpful to rewrite this equation in terms y_{np} and the remainder, γ_{np},

$$\frac{\partial t_p}{\partial t_n} = y_{np} \gamma_{np},$$

where

$$\gamma_{np} = \frac{d\theta^-_{(n+1)p}}{dt_n} \Bigg/ \left(\frac{\partial \theta^-_{(n+1)p}}{\partial \theta^+_{np}} \frac{\partial \theta^+_{np}}{\partial w_{np}} \right).$$

Putting the results together reduces to the following learning rule:

Hidden-Layer Learning Rule

$$\begin{aligned}
\Delta w_{mn} &= -\eta_w \delta_n y_{mn} \\
\delta_n &= \sum_{t_p > t_n} \delta_p y_{np} \gamma_{np} \\
y_{mn} &= \frac{-\alpha(1+\cos\theta^+_{mn})}{(1-\cos\theta^+_{\hat{M}n}) + \alpha I_o (1+\cos\theta^+_{\hat{M}n})} \prod_{j=m+1}^{\hat{M}} \frac{\partial\theta^+_{jn}}{\partial\theta^+_{(j-1)n}} \\
\gamma_{np} &= -w_{np}\left(\alpha w_{np} + 2\tan\left(\frac{\theta^-_{np}}{2}\right)\right).
\end{aligned} \tag{3.7}$$

With this result, along with the output-layer learning rule in section 3.1, training of multilayer theta neuron networks is now possible. These equations also apply regardless of whether the baseline current is positive or negative.

3.3 Analogy to Rate-Coded Neural Networks. The learning rule results developed in this section are analogous to the backprop algorithm derivation for rate-coded models. The variable y is calculated by moving forward through the network, and δ is calculated by then propagating the errors back through the network. The variable δ is specific to a neuron regardless of which input we are examining. In rate-coded models, y is also specific, but for our system, the interaction between the transfer function and input timings does not allow this effect to be separated out, and so y is specifically a measure of the effect of the weight between neuron j and neuron k on the output spike time of neuron k.

Recall that y_{jk} indicates the value of y between neurons j and k, where j and k index the neuron firing order in two different layers. As j increases (meaning we are examining neurons whose output spike times occur later and later), y_{jk} will tend toward zero. Generally each subsequent input has less and less of an effect on the phase. The last parts of equations 3.5 and 3.7 multiply these effects on the phase together, forcing y_{jk} toward zero. This is similar to sigmoidal saturation in terms of rate-coded models. The variable y_{jk} will actually reach zero when enough input spikes have preceded it, so that subsequent input spikes arrive after neuron k has already fired. Since the input arrives after the output, it has no effect. Interestingly, y_{jk} may also be viewed recursively:

$$y_{(j+1)k} = \frac{\left(\alpha w_{jk} + \tan\left(\frac{\theta^-_{jk}}{2}\right)\right)^2 + \alpha I_o}{\tan^2\left(\frac{\theta^-_{(j+1)k}}{2}\right) + \alpha I_o} y_{jk}.$$

Recursiveness makes intuitive sense here since the theta neuron incorporates the activity-dependent threshold property that states that the effect of each input spike is based on the previous input history.

4 Event-Driven Simulation

In this section, a collection of equations is derived that enable the dynamics of the theta neuron to be simulated using an event-driven (rather than numerical) methodology. The existence of these event-driven equations makes possible the derivation of exact expressions for gradient terms without resorting to linearization or other simplifying assumptions as was presented in section 3. The derivation is based on the theta neuron's remaining time function, which describes the amount of time remaining until an output spike is produced as a function of input current and phase. This function was previously shown to have an analytical solution in Ermentrout (1996). However, the authors of that work were more interested in aspects of neuron synchronization and thus did not develop the analytic solution into one that handles input spikes, removes imaginary numbers from the calculation, or accounts for both positive and negative baseline currents. The method employed here is analogous to the event-driven simulation method for impulse inputs in QIF neurons derived in Tonnelier et al. (2007). Because our method here is applied to theta neurons rather than quadratic neurons, it has the potential to handle multiple input and output spikes in a more mathematically concise way. The postfiring phase reset in quadratic neurons is handled by a rule, outside of the differential equation that describes the neuron dynamics. However, in theta neurons, because of the cyclic nature of trigonometric functions, after passing π, the value of the next presynaptic input phase automatically is reduced by 2π, placing it in the refractory region and ready to handle more input spikes.

The phase of the theta neuron evolves smoothly according to equation 2.2 except when input spikes occur; thus, we would like to calculate the neuron phase at an input spike time as a function of previous spike times and phase at those spike times. The integrated remaining time function for when the baseline current is negative is

$$F = -\frac{1}{\beta}\,\mathrm{atanh}\left(\frac{\tan\left(\frac{\theta}{2}\right)}{\beta}\right),$$

where $\beta = \sqrt{|\alpha I_o|}$, a constant that appears often in our calculations. From this equation, we can derive the relationship between the phase and time for a baseline trajectory. The index of the jth connection to neuron k is j,

given that the inputs are sorted by their firing times:

$$t_j = t_{(j-1)} + F(\theta_{jk}^-) - F\left(\theta_{(j-1)k}^+\right)$$
$$\theta_{jk}^- = 2\,\mathrm{atan}\left[\beta \tanh\left(-\beta\left(t_j - t_{(j-1)} + F\left(\theta_{(j-1)k}^+\right)\right)\right)\right].$$

Using a number of trigonometric identities as well as equation 2.4, we have the following two equations that can be used to evolve the neuron phase according to received input spikes:

$$\theta_{jk}^- = 2\,\mathrm{atan}\left[\frac{\beta\left(c_{1_{jk}} + c_{2_{jk}}\right)}{1 + c_{1_{jk}} c_{2_{jk}}}\right], \quad \theta_{jk}^+ = 2\,\mathrm{atan}\left[\alpha w_{jk} + \frac{\beta\left(c_{1_{jk}} + c_{2_{jk}}\right)}{1 + c_{1_{jk}} c_{2_{jk}}}\right],$$

where $c_{1_{jk}} = \tan(\theta_{(j-1)k}^+/2)/\beta$ and $c_{2_{jk}} = \tanh\left(-\beta(t_j - t_{(j-1)})\right)$. If the neuron fires during normal movement around the phase trajectory (as opposed to on an input spike), then the output spike time (t_k) after the last input spike, t_N, has occurred is

$$t_k = t_N + \frac{1}{\beta}\,\mathrm{atanh}\left(\frac{1}{c_{1_{(N+1)k}}}\right). \tag{4.1}$$

The baseline firing time, t_{BL}, that is, the neuron firing time in the absence of input spikes, can be determined from equation 4.1 under the assumption that the neuron phase is in the spiking region,

$$t_{BL} = \frac{1}{\beta}\,\mathrm{atanh}\left(\frac{\beta}{\tan\left(\frac{\theta_0}{2}\right)}\right), \tag{4.2}$$

where θ_0 is the initial phase. Throughout this letter, we set θ_0 to a small amount (0.0001) above the positive fixed point. Time zero then is defined at the point when all the theta neurons in the network have their phase set to θ_0 and are left to evolve according to their dynamics. Additionally, each neuron is assigned a reference input, which is defined to come at a fixed time before all the other input pattern spikes. This reference input effectively allows all the neurons to have different initial phases at this reference time (since the weights connecting the neurons to the reference inputs will vary on training). When I_o is positive, the equations are slightly modified:

$$F = \frac{1}{\beta}\,\mathrm{atan}\left(\frac{\tan\left(\frac{\theta}{2}\right)}{\beta}\right)$$
$$\theta_{jk}^- = 2\,\mathrm{atan}\left[\beta \tan\left(\beta\left(t_j - t_{(j-1)} + F\left(\theta_{(j-1)k}^+\right)\right)\right)\right].$$

Again, using a number of trigonometric identities, we have,

$$\theta_{jk}^{-} = 2\,\mathrm{atan}\left[\frac{\beta\left(c_{1_{jk}} + c_{2_{jk}}\right)}{1 - c_{1_{jk}}c_{2_{jk}}}\right], \quad \theta_{jk}^{+} = 2\,\mathrm{atan}\left[\alpha w_{jk} + \frac{\beta\left(c_{1_{jk}} + c_{2_{jk}}\right)}{1 - c_{1_{jk}}c_{2_{jk}}}\right].$$

The equation for $c_{1_{jk}}$ is unchanged from when $I_o < 0$, but now $c_{2_{jk}} = \tan(\beta(t_j - t_{(j-1)}))$. The output spike time after the last input spike, t_N, has occurred is

$$t_k = t_N + \frac{1}{\beta}\,\mathrm{atan}\left(\frac{1}{c_{1_{(N+1)k}}}\right).$$

Using a simple data set that emulates logical inversion as a test case (see section 5.1 for more details on the data set) and a theta neuron network with $I_o = -0.005$, an initial phase just above the positive fixed point and five hidden neurons, the event-driven simulation method takes about 240 ms for each output spike time calculation, while the numerical integration method takes about 5 seconds. The event-driven calculation has almost a factor of 50 improvement in speed. This speed improvement holds under most testing conditions and starts to gradually decrease only when the number of hidden neurons is very large or the number of input spikes is large. Thus, we can conclude that event-driven simulation provides a valuable speed benefit under most normal operating conditions.

5 Experiments

In this section, we perform a series of experiments that use an in-house-built Matlab graphical user interface capable of simulating and training theta neuron networks (it is available by contacting the authors). These experiments use the training rules derived in section 3 to examine different properties of theta neuron training and compare the results to other relevant experiments. The first experiment is a toy problem involving a single neuron with a single input in order to demonstrate the basic properties of the learning rule. The second experiment involves a biologically significant problem of learning to classify spike latency patterns and is compared to results obtained using Tempotron learning.

Next, a number of more computationally intensive multilayer machine learning problems are discussed. Results in these cases are compared with SpikeProp and other neuron-based learning methods. One regression problem we learn is the cosine function. Because cosine is one of the basis functions for the Fourier series, being able to learn cosine is a strong indicator of universal function approximation ability. Put another way, if we can learn a cosine of arbitrary frequency and phase, then networks trained in this way could be placed side by side with the addition of another layer to

add the cosine outputs together into the approximated function. The addition of cosine terms requires that we be able to train a network to perform linear addition, which was done successfully in other experiments that we ran and are not shown here.

Finally, we examine the sensitivity of theta neuron networks to network parameters. Overall, our experiments show that the theta neuron networks trained with our rule perform as well as or better than other comparable spiking neuron training techniques and also show how training and network parameters can be chosen. For efficiency purposes, all of the experiments use event-driven simulation for determining the impact each input spike has on each neuron's phase (see section 4). The scaling parameter, α, is set to 1 throughout.

5.1 Simple Inverter and Delayer Training. In this section, we seek to design an experiment that visually demonstrates our learning rule. The simplest possible network is a single neuron with one input in addition to a reference input. Recall that the reference input receives a spike at a fixed time regardless of the input pattern, in much the same way that the bias input works in rate-coded networks. This simple network has two parameters: the weight associated with the reference input and the weight associated with the variable input. For a given input pattern, it is possible to calculate the error in between the actual and desired output spike times for different values of these two parameters. Because we are using only two parameters, the error surface may be viewed as a three-dimensional plot. Using this error surface, we can examine the trajectories of the weight parameters as they are trained. Ideally, the learning method should minimize the error, so the trajectories will move from the initial weight point to a value of the weights at which there is a minimum on the error surface.

Two such simple problems are the simple inverter and delayer problems. The reference input time in each case is 1 ms, and the possible input times are 3 ms and 6 ms. For the inverter, the desired output times are 30 ms and 20 ms, respectively, while for the delayer, the desired output times are 20 ms and 30 ms. For the inverter, the final trained weights that achieve this output spike time mapping are such that the reference input weight is negative, forcing the phase into the quiescent region between the two fixed points. The second weight is positive, which brings the phase back above the positive fixed point, ensuring that the neuron will eventually fire. Thus, we are able to test that our learning rule remains valid when the neuron phase is temporarily in the quiescent region.

If the theta neuron fires during the normal phase trajectory, the output spike time is determined exclusively by the final input spike time and the final post-input spike phase according to equation 4.1. The method proposed by Voegtlin (2007) derived an approximation to the error gradient based on only the input spike to which the weight is associated. This method, however, creates a discontinuity in the final gradient at θ_{np}^{+}, a discontinuity

Table 1: Inverter and Delayer Training Results.

Method	Initial Weights		Training Epochs: Inverter	Training Epochs: Delayer
Method by Voegtlin (2007)	0.01	0.01	668	DNC
	0.01	0.02	DNC	DNC
	−0.01	0.02	DNC	DNC
	0.02	0.01	524	DNC
	0.02	−0.01	11	DNC
Proposed method	0.01	0.01	684	114
	0.01	0.02	819	20
	−0.01	0.02	992	312
	0.02	0.01	585	197
	0.02	−0.01	8	290

Notes: A single neuron with one reference input was batch trained using the inverter or delayer data set. The stopping criterion was an MSE of 0.05 or 2500 epochs, whichever came first. η_w = 2e-7 and $I_o = -0.005$. *DNC* = does not converge, and indicates that either the output neuron stopped firing or that 2500 epochs were reached and the error had become oscillatory. If the data set did not converge, smaller learning rates were attempted to verify that the reason for not converging was not because of too high a learning rate.

that is not present in the exact gradient. Table 1 compares training results on the inverter and delayer using different initial weight values for the approximate method in Voegtlin (2007) and the method described in section 3, which finds the exact gradient analytically. Training with the approximate gradient method often fails to converge for the parameters shown. In cases where convergence does take place, convergence times are similar to the method proposed here.

The error surface for both the inverter and the delayer contains a downhill trough with a shallow slope on one side and a steep slope on the other, above which the neuron does not fire. In Figure 4, the MSE trajectory for the calculation using the exact gradient method in this letter is identical to the numerically calculated exact gradient, while the gradient approximation method from Voegtlin (2007) does not converge to the global minimum. Successful training of the inverter data set has verified that training is possible in the quiescent region. This result is expected since it is the same differential equation that describes the neuron behavior for all regions.

5.2 Spike Latency Pattern Classification. In this section, tests on a single neuron are continued, but with a much more realistic test to determine the theta neuron's spatiotemporal classification and generalization abilities. In this case, the input is a vector. For each input pattern, a spike time for each synaptic connection is chosen at random and assigned one of two class memberships. The learning of these classes is referred to as spike latency pattern classification. Spike latency codes have been shown

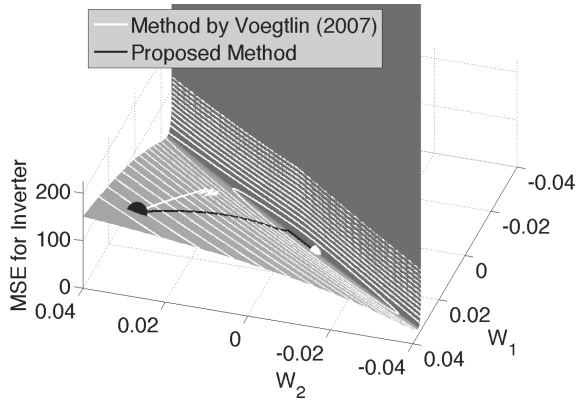


Figure 4: Inverter error surface. Two MSE trajectories are shown using a single neuron with one input plus the reference input. The white trajectory was calculated using the approximate gradient calculation method from Voegtlin (2007), while the black trajectory was calculated using the learning method proposed here. The trajectory using the previous rule exhibits erratic behavior due to an artificial discontinuity that the gradient approximation rule produces, while the method described in this work follows the numerically calculated gradient trajectory exactly and minimizes the error as desired. The MSE in the top right of the graph is undefined since the neuron did not produce any output spikes. The initial weights are represented by the black circle, and the global minimum is represented by the white circle. White contour lines are provided as well.

to play an important role in both retinal ganglion cells (Thorpe, Delorme, & Van Rullen, 2001) and the olfactory system (Hopfield, 1995). As in Gütig and Sompolinsky (2006), the load on the system, L, is defined as the number of input patterns divided by the number of input synapses, M. Here we train a single neuron with $M = 500$ and $L = 1$ to have 0% classification error. Classification error (CE) is the percentage of input patterns that are classified incorrectly.

Different output spike times are assigned to represent different classes. For an input pattern to be classified correctly, it must produce an output spike that is closer to its class's output spike than to any other class's output spike. In Gütig and Sompolinsky (2006), the presence or absence of an output spike determined the classification. This encoding scheme could be approximated by the theta neuron. In theory, if the learning rate was arbitrarily small and the floating-point precision of the computer in use arbitrarily large, then the phase could be moved arbitrarily close to the positive fixed point, and thus produce an output firing time of any positive real number, even approaching infinity. More practically, however, choosing very small learning rates makes learning proceed much slower

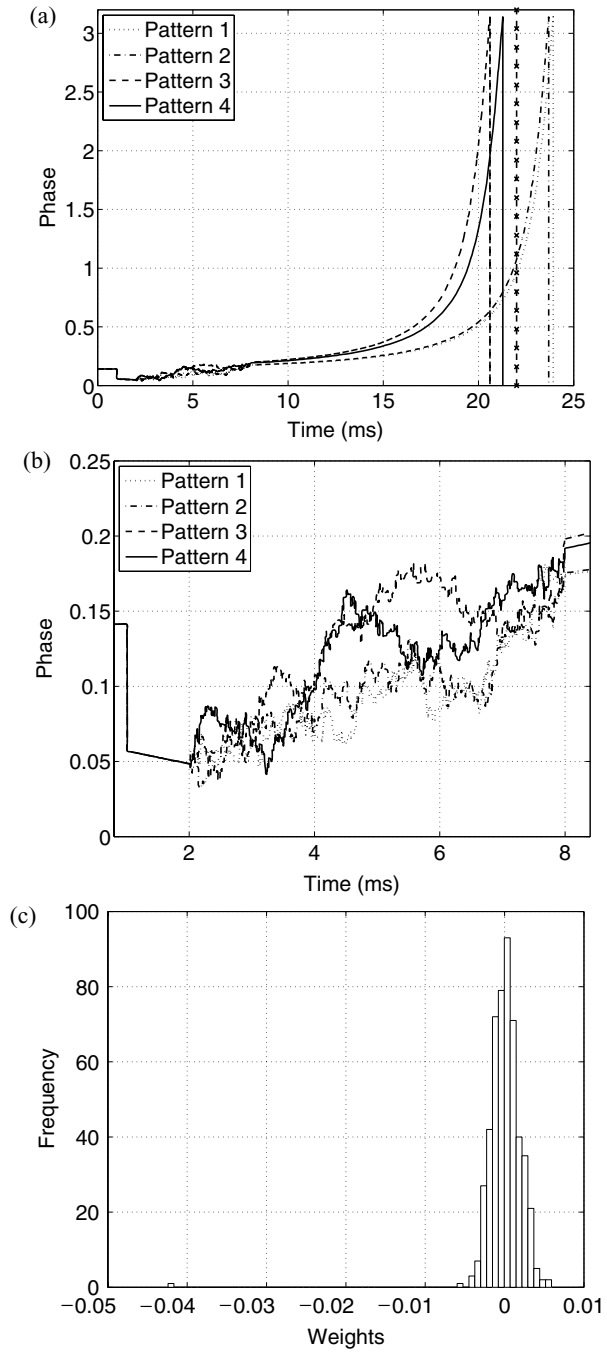
than is necessary, and moving the phase arbitrarily close to the positive fixed point increases the likelihood that the phase will get pushed to the other side of the positive fixed point, causing the output spike to be canceled. Thus, we instead chose to use specific finite desired output spike times for the two classes of input patterns.

The phase of some example patterns as well as the learned weight distribution is shown in Figure 5. When this experiment was performed with a Tempotron, we looked at both training time and generalization error to measure success. It is difficult to compare training times in our case because of the nonsmoothness of the error surface, which requires modulation of the learning rate to obtain the fastest theta neuron network training time possible. Empirical modifications of the learning rate during experiments not included in this letter have verified this statement and suggest that this method would benefit from more complex gradient-descent techniques such as momentum or the Levenberg-Marquardt algorithm. However, as with all other experiments in this section, we use only a constant learning rate in order to provide the simplest comparison possible.

Although it is difficult to compare training times, once training is complete (CE = 0%), generalization error can more easily be compared. Generalization error for this data set is determined by looking at jitter. Jitter is some amount of gaussian noise added to the times of each input spike pattern in order to observe the effect on classification. Classifying correctly in the presence of noise indicates that the network has generalized the input pattern since it is able to classify correctly with similar inputs to the trained ones. Normally generalization is determined by examining the network response to testing data that have not been previously used for training. For the spike latency data set, there is no explicit underlying function from which to generate the testing data since the training has been created in a random way so we rely on jitter to examine generalization instead.

In Figure 6 we compare the jitter results for spike latency classification with that of the Tempotron. Our input spike time range is 2 ms to 8 ms, while our output spike times are 20 ms and 24 ms for the two pattern classes. The amount of added jitter was normalized over the input spike range in order to produce a fair comparison between our method and Tempotron learning. The results show that our proposed learning rule can reproduce the Tempotron training results and with better generalization error. Our model can handle almost two orders of magnitude more noise while staying within generalization errors of 10%. One possible explanation for this result is that the activity-dependent thresholding property of theta neurons is acting to suppress the jitter similar to what was done in McKennoch et al. (2007).

We also examined the case where training was allowed to continue past the point where 0% classification error was achieved—in this case, 7% longer. For low values of jitter, this network exhibited lower generalization error than the previous theta neuron result. However, this network


(a)
Pattern 1
Pattern 2
Pattern 3
Pattern 4
Phase
Time (ms)
(b)
Pattern 1
Pattern 2
Pattern 3
Pattern 4
Phase
Time (ms)
(c)
Frequency
Weights

also showed some initial signs of memorization rather than generalization, meaning that it was starting to become overtrained. This balancing of memorization and generalization is common to standard rate-coded neural networks and is to be expected in theta neurons as well.

In other spike latency experiments that were run with 5 or 10 input synapses rather than 500, the load was varied in order to determine the critical storage capacity (α_c). The critical storage capacity is the maximum load at which the network can still learn all the input patterns' classes (Gütig & Sompolinsky, 2006; Li & Harris, 2004). Our estimated value is $\alpha_c = 2$ for the theta neuron, equivalent to single-layer perceptrons and slightly less than that for the Tempotron (where $\alpha_c \approx 3$). A different choice of parameters perhaps might increase the theta neuron α_c (see, e.g., the effect of the output spike interval in the following section).

5.3 Machine Learning Classification Problems. Three classification problems or data sets were tested using theta neuron networks: the binary XOR problem, the Fisher Iris data set, and the Wisconsin Breast Cancer data set. These experiments are of increasing complexity. The XOR problem is very simple and used to examine the choice of the number of hidden neurons as well as the desired output spike times. The other two experiments are more complex, with the results being compared to other related neuron-based learning methods. In all cases, satisfactory training results were achieved. Online learning was used rather than batch learning, as it proved much more successful at avoiding local minima. MSE was measured

---

Figure 5: Spike latency performance. The phase and weight distribution of a single theta neuron is shown in response to four spike latency pattern. There are 500 input synapses. Input spikes arrive between 2 ms and 8 ms. Patterns 1 and 2 belong to class 1 and have a desired output spike time of 20 ms, while patterns 3 and 4 belong to class 2 and have a desired output spike time of 24 ms. $I_o = -0.005$, the initial weight value (w_{ini}) is 0.01 and $\eta_w = 4.25\text{e-}11$. (a) Complete phase plot. The output spikes for all patterns shown fall on the desired side of the class decision line (the vertical line marked with x). (b) Area of phase modulation. The period of time during which input spikes arrive is shown. The reference spike occurs at 1 ms. Class 1 patterns spend more time above the positive fixed point, while class 2 patterns spend more time below the positive fixed point, thus causing them to produce later output spikes as desired. (c) Learned weight distribution. The weights form a normal distribution around zero with a slight skew toward positive weights. The slight positive skew ensures that output spikes will not be canceled. The single outlier is the weight associated with the reference input. As the number of inputs increases, the magnitude of the learned weights decreases as each input spike needs to affect the output spike time less and less.

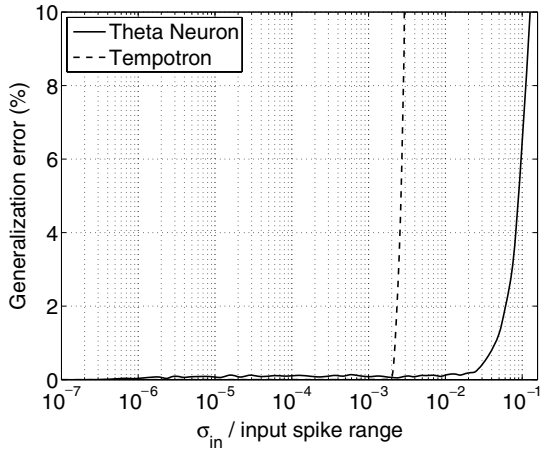


Figure 6: Jitter comparison to tempotron. A single neuron is trained with the method proposed here and by the method in Gütig and Sompolinsky (2006). The number of synapses and input patterns is 500 ($\alpha = 1$). $I_o = -0.005$ and $w_{ini} = 0.01$. For the theta neuron, training is stopped when the classification error becomes zero. The theta neuron is shown to be much less sensitive to higher levels of jitter, indicating better generalizability than the Tempotron. There is one pattern near the decision boundary (because the classification error has just reached 0%) that is especially sensitive to low levels of jitter. In fact, this one pattern is the only one that gets misclassified until the normalized jitter reaches about $7 \cdot 10^{-3}$. Training the network 7% longer after 0% classification error has been reached delays the first appearance of an error from a normalized jitter of $3.5 \cdot 10^{-7}$ to $1 \cdot 10^{-3}$.

relative to the encoded desired output spike times and the actual output spike times. A small amount of normally distributed noise was added to the initial weight value, w_{ini}, on each synapse in order to avoid local minima created by symmetry. Experimental parameters, including the spike time values that input and output spikes are mapped to, are given in Table 2. These parameters were largely chosen empirically. However, the discussion that follows in this section as well as sections 5.4 and 5.5 develops a methodology for choosing spike mappings and other parameters.

Our first machine learning experiment was performed on the simple binary XOR data set. XOR is a simple but important problem in machine learning in part because the data pattern is not linearly separable. In the experiments performed, all four data patterns are correctly classified when the MSE is at about 2, but further training allows better separability and tolerance to noise. The number of hidden neurons, N, was varied in order to determine its optimal value. This best value of N is tied to the data set complexity. Figure 7 shows the computation time and training epochs as N

Table 2: Machine Learning Classification Experiment Overview: Parameters for the Machine Learning Classification Experiments.

	XOR	Fisher Iris	Wisconsin Breast Cancer
Inputs	0, 1	[0, 79]	[1, 10]
Input spikes	3 ms, 6 ms	[2 ms, 8 ms]	[2 ms, 8 ms]
Output spikes	20 ms, 30 ms	[20 ms, 30 ms]	[18 ms, 28 ms]
Outputs	0, 1	0, 1, 2	2, 4
Learning rate	1e-6	1e-6	7e-8
Baseline current	-0.005	-0.005	-0.005
Initial weight	0.01	0.01	0.01
Number hidden neurons	5	8	8
Online versus batch	Online	Online	Online

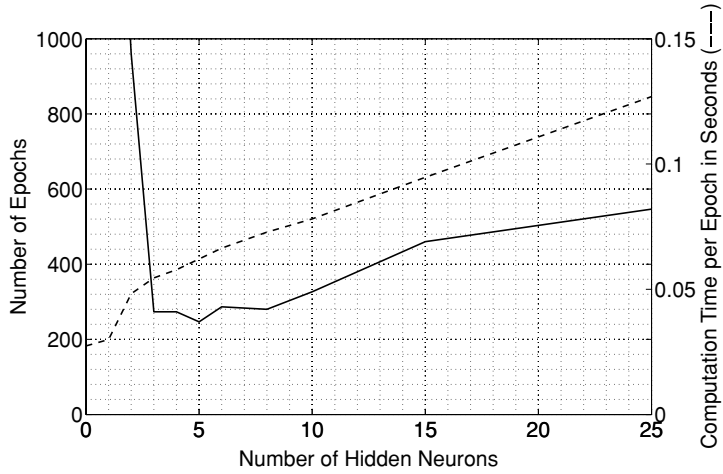


Figure 7: Hidden neuron analysis. This plot shows the effects of varying the number of hidden neurons, N, on the XOR problem. The data set is trained to 0% classification error with $\eta_w =$ 1e-6, $I_o = -0.005$, and $w_{ini} = 0.01$. As N increases, the computation time per epoch (the actual CPU time taken to perform the necessary calculations) also increases, as expected, since the number of parameters is increasing. The number of epochs needed for training has a minimum at $N = 5$, where there are neither too many nor not enough trainable parameters to perform the XOR computation.

is varied. To minimize the number of training epochs, we set $N = 5$. Using the parameters in Table 2, XOR is trained to an MSE of 0.05 in 2524 epochs and a classification error of zero in 240 epochs.

For the XOR problem, we also examined the effect of translating and dilating the desired output spike times in order to examine its effect and also determine if there is an optimal spike interval. The results are summarized

Table 3: Effect of Output Spike Times on XOR Training.

Translation		Dilation	
Desired Output Spike Times	Epochs	Desired Output Spike Times	Epochs
6.1 16.1	140, $\eta_w = 8e\text{-}5$	20, 22	200, $\eta_w = 1e\text{-}5$
10, 20	210, $\eta_w = 3e\text{-}5$	20, 26	200, $\eta_w = 6e\text{-}6$
15, 25	240, $\eta_w = 5e\text{-}6$	20, 28	540, $\eta_w = 4e\text{-}6$
20 30	240, $\eta_w = 1e\text{-}6$	20 30	240, $\eta_w = 1e\text{-}6$
25, 35	310, $\eta_w = 3e\text{-}7$	20, 32	240, $\eta_w = 1e\text{-}6$
30, 40	1780, $\eta_w = 3e\text{-}8$	20, 34	300, $\eta_w = 5e\text{-}7$
35, 45	2900, $\eta_w = 7e\text{-}9$	20, 40	360, $\eta_w = 2e\text{-}7$
40, 50	>30,000, $\eta_w = 2e\text{-}10$	20, 45	1400, $\eta_w = 3e\text{-}8$

Notes: In this comparison of training results between different output spike times for the XOR problem, a multilayer theta neuron network with five hidden neurons, $I_o = -0.005$ and $w_{ini} = 0.01$, was used in training to a classification error of 0%. The highest learning rate possible such that the system remained stable was used in each case. To force the output spike times to occur later, it is necessary to push the phase closer and closer to the positive fixed point. As a result, a smaller learning rate is required to ensure that the neurons do not stop firing altogether.

in Table 3. For translation of the output spike interval, earlier values result in quicker training times up to the input spike interval (3–6 ms here). Later output spike times require a smaller learning rate (and thus slower training) since the phase is getting pushed closer to the positive fixed point to achieve the slower time constant. For dilation, there is a dynamic equilibrium with later spike times harder to train, but more widely separable classes easier to classify. Once the upper class becomes late enough, then the second effect dominates, and training time begins to increase. Thus, the spike interval can be chosen without too much care as long as the upper output spike time is kept as small as possible. Regarding the learned weight distribution, as was the case with the Tempotron experiment, the learned weights have a normal distribution with a slight skew to positive weights. The spread of the distribution decreases as the output spike range increases, where smaller, more precise weights are needed to realize the desired input-output relationship (and therefore a smaller learning rate as well). Although this discussion was specifically applied to XOR, a classification problem, the same logic should apply to regression problems, in the sense that they are like a classification problem with an infinite number of classes over a fixed range. Indeed, in section 5.4, we show how the choice of the spike interval relates to the dynamics of the neuron through its response curve.

Table 4 contains classification results on the Fisher Iris and Wisconsin Breast Cancer data sets for the proposed learning rule in comparison to other neuron-based methods. The Fisher Iris data set consists of four iris attributes such as petal length and a single output that indicates which of

Table 4: Classification Results (%).

Learning Method	Network Size	Epochs	Classification	
			Train	Test
Fisher Iris data set				
SpikeProp	$50 \times 10 \times 3$	1000	97.4%	96.1%
Dynamic synapse SNN	$4 \times 10 \times 1$	Unknown	96	97.3
BP A	$50 \times 10 \times 3$	2.6e6	98.2	95.5
BP B	$4 \times 8 \times 1$	1e5	98.0	90.0
Theta neuron BP	$4 \times 8 \times 1$	1080	100	98.0
Wisconsin Breast Cancer data set				
SpikeProp	$64 \times 15 \times 2$	1500	97.6	97.0
Dynamic synapse SNN	$9 \times 6 \times 1$	Unknown	97.2	97.3
BP A	$64 \times 15 \times 2$	9.2e6	98.1	96.3
BP B	$9 \times 8 \times 1$	1e5	97.2	99.0
Theta neuron BP	$9 \times 8 \times 1$	3130	98.3	99.0

Notes: In this comparison of classification results between different training methods, $I_o = -0.005$, $w_{ini} = 0.01$, and $\eta_w =$ 1e-6 and 7e-8 for the Fisher Iris and Wisconsin Breast Cancer data sets, respectively. Results for SpikeProp and rate-coded model error backpropagation A (BP A) are taken from (Bohte et al. (2000), and results for dynamic synapse SNN are taken from Belatreche, Maguire, and McGinnity (2006). The network size used in BP A is inflated to match that of SpikeProp. The network size in BP B is reduced to the network size of the theta neuron network and thus has the same number of trainable weights. Default Matlab parameters for backpropagation and normalized inputs and outputs were used for training with BP B.

three iris types has been measured. The output classes for the Fisher data set include both linearly and nonlinearly separable data. There are 150 data patterns: we used 100 for training data and the remaining 50 for testing data. The Wisconsin Breast Cancer data set was originally developed by William Wolberg to study fine needle aspiration cytological diagnosis. This data set contains 699 training patterns, each of which has nine attributes (such as cell nucleus area and symmetry), and a single output that determines whether the tumor was malignant or benign (Wolberg & Mangasarian, 1990). We used 599 patterns for our training data and the remaining 100 for our testing data. Both data sets were obtained from the UCI Machine Learning Repository (Asuncion & Newman, 2007). For both data sets, our learning method classifies much better than the methods we compare it to, even achieving perfect classification for the Fisher Iris data set. These data sets are very popular in the machine learning community. For example, a recent paper on Bayesian classifiers achieved 94.87% training accuracy (Kotsiantis & Pintelas, 2005) with the Fisher Iris data set and one on support vector machines achieved a 99.33% training accuracy on the same data set (Zhong & Fukushima, 2007). However, the fairest comparison for our results is to other neuron-based learning methods, especially spiking ones.

5.4 Machine Learning Regression Problems. Regression problems are well suited to neural networks due to the networks' ability to generalize functions by interpolating between training data points. In order to achieve a very small MSE, the training performed in this section temporarily pushed the neuron phase into the refractory region below both fixed points. Additionally, some of the hidden neurons occasionally failed to fire when presented with certain training data patterns. If the hidden neuron fails to fire for all training patterns, then it is effectively pruned from the network and will have no impact on the output spike time. However, if the hidden neuron fires for some training patterns and not others, an additional degree of freedom is produced, effectively creating a variable-size hidden layer. One use of this variable-size hidden layer was shown in Figure 3. The network is able to create similar outputs for the cosine function in response to widely separated inputs by having nonfiring hidden neurons (in the second pattern).

As was the case with the classification problems, the problem sets are not presented in terms of spike times; thus, a mapping must be performed to transform inputs and outputs to and from spike times. In the previous section, we performed this mapping empirically. We now use a linear mapping between the data set input or output space and the range of input or output spikes. The mapping ranges are determined by the neuron dynamics as revealed in the response curve. Recall that the response curve is a plot that demonstrates the effect of an input spike on the output spike time. For the case of negative baseline currents, if the first input spike arrives very early, it has the most effect. For example, for a typical weight connection, an input spike arriving at 0 ms will alter the output spike time by 30 ms from the baseline firing time, while an input spike arriving later, at 20 ms, will alter the output spike time only by 5 ms. In these cases, the baseline current is chosen such that the baseline firing time, t_{BL}, is about 56 ms. From this observation, it is reasonable to choose an input spike range of 0% to 20% of t_{BL}. A similar analysis can be performed for output spike times resulting in a range of 20% to 50% of t_{BL} being useful for the desired range of output spike times.

The first regression problem we trained on a theta neuron network is the cosine function (see Table 5). While training the cosine function, if batch training is used with a low learning rate, 5e-7 or less, the network becomes stuck in a local minimum where the cosine function is approximated by a best: fit line—a horizontal line passing through the origin. Using online training, the network is able to recover from this local minimum (as occurs frequently with rate-coded networks as well). Once through this local minimum, learning proceeds rapidly for half the cosine, and then more slowly for the other half. When eight hidden neurons were used, $\eta_w = 2\text{e-}6$, $I_o = -0.005$, and $w_{ini} = 0.01$, the cosine is trained to an MSE of 0.05 in 300 epochs. The number of training patterns was 60, while the number of testing patterns was 10. In both cases, patterns were generated by randomly

Table 5: Machine Learning Regression Experiment Overview: Parameters and Training Results for the Machine Learning Regression Experiments.

	Cosine	Sexton5
Inputs	$[0, 2\pi]$	$[-100, 100]$
Input spikes	[2 ms, 8 ms]	[2 ms, 8 ms]
Output spikes	[20 ms, 28 ms]	[20 ms, 28 ms]
Outputs	$[-1, 1]$	[9.9e-5 9.9e5]
Learning rate	2e-6	4e-6
Baseline current	-0.005	-0.008
Initial weight	0.01	0.01
Number hidden Neurons	8	8
Online versus batch	Online	Online
Epochs to MSE of 0.05	300	545

choosing a number in the input range of 0 to 2π and simply taking the cosine of the input to generate the desired output. The MSE for the test points (data patterns not directly trained on) is highly proximate to that of the training points. This low MSE indicates that the network is interpolating between training patterns in such a way that the general cosine function is being learned, as opposed to the case where there might be nonsmooth erratic behavior between training points.

A more difficult regression problem is the Sexton 5 function ($y = x^3 - x^2$), which has a flat section around $x = 0$. As with cosine, a best-fit line local minimum exists. Once through this local minimum, half of the function trains rapidly, while the other half trains much more gradually. Figure 8 shows the regression results and MSE results. Using eight hidden neurons, $\eta_w =$ 4e-6, $I_o = -0.008$, and $w_{ini} = 0.01$, Sexton 5 is trained to an MSE of 0.05 in 545 epochs and an MSE of 0.01 in 936 epochs. Again, the MSE for the testing data indicates good generalizability.

5.5 Parameter Sensitivity. In previous sections, we examined the importance of choosing appropriate input and output spike mappings relative to the neuron dynamics. We continue in this section to examine three other parameters that are also important to network training: the baseline current (I_o), initial weight value (w_{ini}), and the learning rate (η_w). In order to better quantify the relative sensitivity to each parameter, we train a reduced spike latency pattern classification data set and a delay regression data set. These data sets are simple and quick to train and yet complex enough to be indicative of the effects of the parameters under study. In reality, there are complex interactions between these parameters; thus, these sensitivity experiments should be viewed with appropriate caution. The networks we used all had three hidden neurons and unless otherwise specified, $I_o = -0.005$, $w_{ini} = 0.01$, and $\eta_w =$ 3e-7. The reduced spike latency pattern classification problem had a stopping criterion of 0%

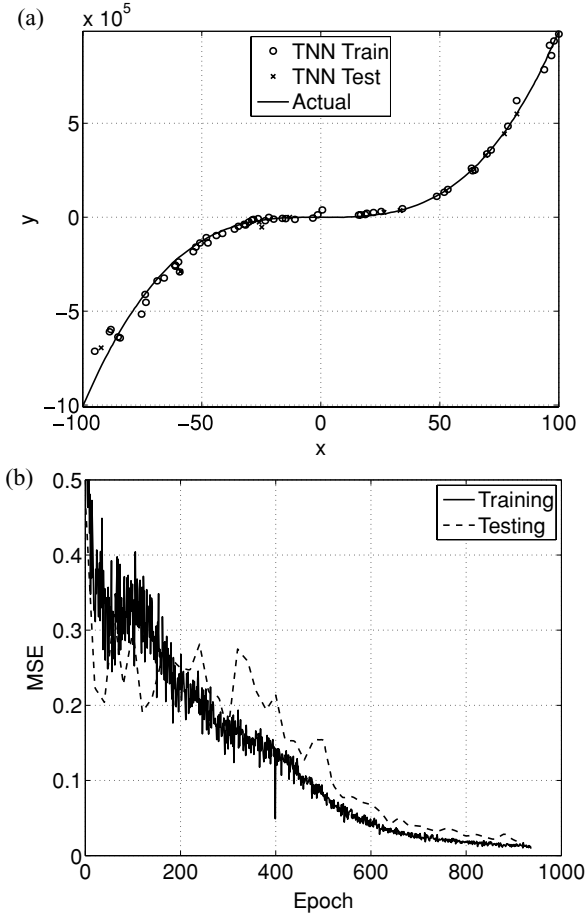


Figure 8: Sexton 5 training. $I_o = -0.008$, $\eta_w =$ 4e-6, and $w_{ini} = 0.01$. (a) Sexton 5 regression results. Both the testing and training patterns are in high agreement with the Sexton 5 function. (b) Sexton 5 MSE results. Testing data were generated every 10 epochs.

classification error. The number of input patterns was 10, and the load was 1. The output spike times for the two pattern classes were 20 ms and 24 ms (see section 5.2 for more details). For the delay regression problem, input spike times were trained to map linearly to the output spike range of 16 ms to 28 ms. The training and testing set sizes were 30 and 10 input patterns, respectively. The stopping criterion was an MSE of less than 0.1. In both cases, input spikes were generated from 2 ms to 8 ms.

Recall that when $I_o > 0$, the neuron undergoes tonic spiking at some frequency determined by I_o. When $I_o < 0$, a resting equilibrium appears,

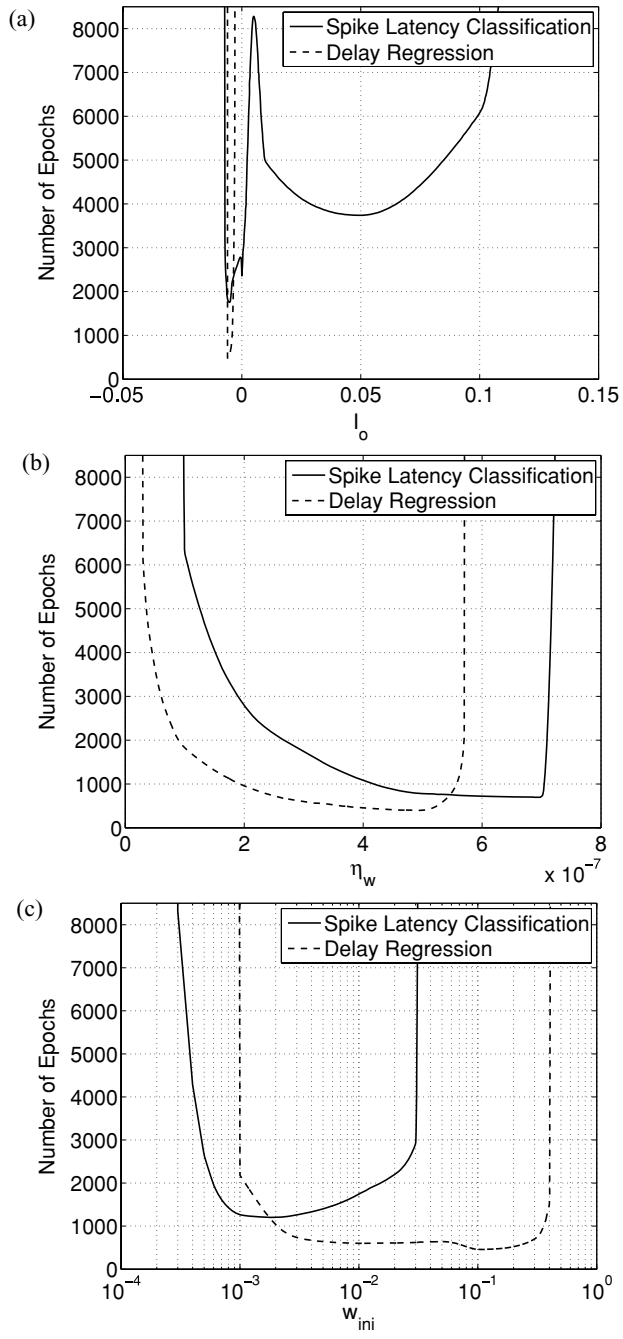
and the dynamics of the neuron become richer as the phase may evolve in either direction depending on the region it is in. In all previous experiments, we have taken advantage of these richer dynamics by using neurons with exclusively negative values of I_o. As part of our examination of I_o, we vary this parameter through a range of positive and negative values. The magnitude is kept relatively small because the theta neuron model is valid only when the neuron is near bifurcation, meaning that I_o is close to zero.

Figure 9a shows the training results as I_o is varied over a region where I_o is close to zero and the number of epochs required for training is less than about 8500. The minimum for both data sets occurs when I_o is approximately -0.0055, which corresponds to $t_{BL} \approx 54$ ms based on equation 4.2. For the regression data set, the acceptable values of I_o are in a very narrow range around this minimum. The spike latency data set has a wider range of acceptable I_o values, including some values where I_o is positive. As I_o approaches zero, the phase ceases changing over time unless an input spike is received. Thus, there are three minima on the plot: one corresponding to an optimal value of I_o being negative, one positive, and one where I_o is zero, meaning that the dynamics are determined by spikes only. It is likely that the demand on the network is higher for the regression data set, and therefore the neuron dynamics when $I_o > 0$ are not sufficient enough to allow the network to be trained.

As η_w is increased from near zero, the number of training epochs decreases exponentially, as shown in Figure 9b. At some point, the learning rate becomes high enough that the weights make too large a jump and the output neuron may stop firing. The closer that the learning rate can be moved to this point, the faster the network will train.

Finally, if w_{ini} is chosen on the correct order of magnitude as the final trained weights, then training may proceed much more quickly. As a good heuristic, consider choosing w_{ini} such that the initial output spike time will be in the middle of the output spike range for an average input pattern. To accomplish this task, consider the inputs as a single input occurring at a mean input time that passes into a single hidden neuron with synaptic weight $w_{ini}N$. Likewise, group the output neurons together with a single weight $w_{ini}P$. Training can then be performed on this simple multilayer two-neuron, single-output network. A different w_{ini} can be assigned to each layer of the final network, or an average can be taken of the hidden and output layer w_{ini} to be used on the entire final network. Note that as the number of input synapses increases, the size of the weight should decrease to preserve a constant initial output spike time. For the delay network, w_{ini} for the hidden weights is found to be 0.0053, and for the spike latency 10-pattern classification data set, an initial w_{ini} of 0.0053 is found. Both values agree approximately with the center of the range of acceptable values for w_{ini} in Figure 9c.

From the experiments to date, including some not published in this work, a few observations on training theta neuron networks can be made. As with rate-coded networks, the error initially drops off rapidly and then


(a)
Spike Latency Classification
Delay Regression
Number of Epochs
8000
7000
6000
5000
4000
3000
2000
1000
0
−0.05
0
0.05
0.1
0.15
I_o
(b)
Spike Latency Classification
Delay Regression
Number of Epochs
0
2
4
6
8
η_w
x 10^{-7}
(c)
Spike Latency Classification
Delay Regression
Number of Epochs
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
w_{ini}

slowly approaches a minimum on the error surface. The shape of the MSE versus epochs curve is a function of gradient steepest descent training, so a similar MSE curve to that obtained using rate-coded networks is to be expected. Also similar is the fact that online training helps prevent the network from getting stuck in local minimums (such as best-fit lines for regression problems). A problem specific to spiking neurons is spike loss. As was discussed in section 5.4, in the hidden layer, spike loss can be beneficial by creating a variable-sized hidden layer, but in the output layer, spike loss can prevent training from converging.

6 Conclusion and Future Work

We have demonstrated that the dynamic properties of neurons are sufficient to sustain universal type computations without the need to rely on the precise shape of PSPs. Although this result does not mean that the dynamics of synapses have no effect on time-based computations, our work demonstrates that computations can be carried out without relying on this level of description, and our comparison to other models suggests that it is more efficient to do so. Computations were performed by networks of theta neurons trained using a learning rule based on only the intrinsic neuron dynamics, with highly simplified synaptic currents. In this way, we have defined a time code that is adapted to the natural dynamics of neurons and potentially uses less neural hardware for calculations because of the synaptic simplification and lack of multiple delay connections (Lengyel, Kwag, Paulsen, & Dayan, 2005). The theta neuron model dynamics also allows longer computational timescales as determined by the neuron response curve rather than the much quicker synaptic dynamics.

Our learning rule takes advantage of both the nonlinearity and the simplicity of the theta neuron model. Using analytical expressions for the neural

Figure 9: Network parameter training sensitivity. Unless otherwise specified, $I_o = -0.005$, $\eta_w =$ 3e-7, and $w_{ini} = 0.01$. As the parameters are decreased, training times increase gradually. However, when parameters are increased, neurons may stop firing during training in a nonrecoverable way. For the spike latency data classification data set, the demands on the neuron dynamics are simple enough that training is possible when $I_o > 0$ and when $I_o = 0$ corresponds to dynamics determined only by input spikes. (a) Baseline current (I_o) versus training time. The optimal I_o for this set of parameters is -0.006 for the delay regression data set and -0.005 for the spike latency pattern data set. (b) Learning rate (η_w) versus training time. The optimal η_w for this set of parameters is 5e-7 for the delay regression data set and 7e-7 for the spike latency pattern data set. (c) Initial weight (w_{ini}) versus training time. The optimal w_{ini} for this set of parameters is 0.1 for the delay regression data set and 0.002 for the spike latency pattern data set.

dynamics, we derived relatively simple expressions for the error gradient. In our simulation experiments, we showed that the performance of theta neuron networks compared favorably to other spiking neuron learning methods when given appropriate training parameters. Our learning rule demonstrated much higher generalization in a spike latency pattern classification task than did the Tempotron model. Training on typical machine learning problems produced classification results that in most cases exceed the performance of SpikeProp or rate-coded models trained with standard BackProp. Our work has demonstrated that networks of theta neuron are capable of performing interesting and complex calculations using only the intrinsic neuron dynamics and are able to overcome the limitations of comparable learning techniques.

In further research, theta neuron networks could be adapted to unsupervised learning, different types of network structures, recovering nonspiking neurons, delay learning, and learning rate heuristics. The gradient calculation could be expanded to use a natural gradient approach to account for interdependencies between synaptic weights (Amari, 1998). Similar gradient-based learning could be applied to Izhikevich's simple neuron model, which is two-dimensional but capable of modeling a broader range of neuron behaviors (Izhikevich, 2006). It is not clear that an analytic solution exists for the error gradient in this model, but a slower numerical solution for the error gradient could also be used for training. We are currently pursuing expanding the learning rule to accommodate more than one input or output spike per synapse. In this way, complex input and output spike train mappings could be achieved, opening up exciting applications in performing computations on natural signals from electrode recordings.

Acknowledgments

This work is supported in part by the U.S. National Science Foundation Grant No. ECS-0322618.

References

Alnajjar, F., & Murase, K. (2005). Self-organization of spiking neural network generating autonomous behavior in a real mobile robot. In *Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)* (pp. 1134–1139). Washington, DC: IEEE Computer Society.

Amari, S. I. (1998). Natural gradient works efficiently in learning. *Neural Computation, 10*(2), 251–276.

Asuncion, A., & Newman, D. (2007). *UCI machine learning repository.* Available online at http://www.ics.uci.edu/∼mlearn/MLRepository.html.

Belatreche, A., Maguire, L. P., & McGinnity, M. (2006). Advances in design and application of spiking neural networks. *Soft Computing, 11*(3), 239–248.

Bohte, S. M. (2003). *Spiking neural networks*. Unpublished doctoral dissertation, Centre for Mathematics and Computer Science, Amsterdam.

Bohte, S., Kok, J., & La Poutré, H. (2000). Spike-prop: Error-backprogation in multilayer networks of spiking neurons. In M. Verleysen (Ed.), *Proceedings of the European Symposium on Artificial Neural Networks ESANN'2000* (pp. 419–425). Bruges, Belgium: D facto.

Booij, O., & Nguyen, H. T. (2005). A gradient descent rule for spiking neurons emitting multiple spikes. *Information Processing Letters, 95*(6), 552–558.

Börgers, C., & Kopell, N. (2003). Synchronization in networks of excitatory and inhibitory neurons with sparse, random connectivity. *Neural Computation, 15*(3), 509–538.

Brunel, N., & Latham, P. E. (2003). Firing rate of the noisy quadratic integrate-and-fire neuron. *Neural Computation, 15*(10), 2281–2306.

Carr, C. E., Heiligenberg, W., & Rose, G. J. (1986). A time-comparison circuit in the electric fish midbrain. I. behavior and physiology. *Journal of Neuroscience, 6*, 107–119.

Cassandras, C. G., & Lafortune, S. (2006). *Introduction to discrete event systems*. New York: Springer-Verlag.

Dayan, P., & Abbott, L. F. (2001). *Theoretical neuroscience: Computational and mathematical modeling of neural systems*. Cambridge, MA: MIT Press.

Ermentrout, B. (1996). Type I membranes, phase resetting curves, and synchrony. *Neural Computation, 8*(5), 979–1001.

Ermentrout, G. B., & Kopell, N. (1986). Parabolic bursting in an excitable system coupled with a slow oscillation. *SIAM Journal on Applied Mathematics, 46*(2), 233–253.

Feng, J. (2001). Is the integrate-and-fire model good enough? A review. *Neural Networks, 14*(6-7), 955–975.

Florian, R. V. (2003). *Biologically inspired neural networks for the control of embodied agents* (Tech. Rep.). Cluj-Napoca, Romania: Center for Cognitive and Neural Studies.

Gerstner, W., & Kistler, W. (2002). *Spiking neuron models: An introduction*. Cambridge: Cambridge University Press.

Gütig, R., & Sompolinsky, H. (2006). The tempotron: A neuron that learns spike timing-based decisions. *Nature Neuroscience, 9*(3), 420–428.

Hopfield, J. J. (1995). Pattern recognition computation using action potential timing for stimulus representation. *Nature, 376*(6535), 33–36.

Iannella, N., & Kindermann, L. (2005). Finding iterative roots with a spiking neural network. *Information Processing Letters, 95*(6), 545–551.

Izhikevich, E. M. (2004). Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks, 15*(5), 1063–1070.

Izhikevich, E. M. (2006). *Dynamical systems in neuroscience: The geometry of excitability and bursting*. Cambridge, MA: MIT Press.

Kotsiantis, S. B., & Pintelas, P. E. (2005). Logitboost of simple Bayesian classifier. *Informatica (Slovenia), 29*(1), 53–60.

Legenstein, R., Naeger, C., & Maass, W. (2005). What can a neuron learn with spike-timing-dependent plasticity? *Neural Computation, 17*(11), 2337–2382.

Lengyel, M., Kwag, J., Paulsen, O., & Dayan, P. (2005). Matching storage and recall: Hippocampal spike timing-dependent plasticity and phase response curves. *Nature Neuroscience, 8*(12), 1677–1683.

Li, Y., & Harris, J. G. (2004). A spiking recurrent neural network. In *IEEE Computer Society Annual Symposium on VLSI* (pp. 321–322). Washington, DC: IEEE Computer Society.

Lim, W., & Kim, S.-Y. (2007). Coherence resonance in coupled theta neurons. *Journal of the Korean Physical Society, 50*, 219–223.

Lindner, B., Longtin, A., & Bulsara, A. (2003). Analytic expressions for rate and CV of a type I neuron driven by white gaussian noise. *Neural Computation, 15*(8), 1761–1788.

Maass, W. (1996a). Lower bounds for the computational power of networks of spiking neurons. *Neural Computation, 8*(1), 1–40.

Maass, W. (1996b). Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons. In M. C. Mozer, M. I. Jordan, & T. Petsche (Eds.), *Advances in Neural Information Processing Systems, 9* (pp. 211–217). Cambridge, MA: MIT Press.

Marinazzo, D., Kappen, H. J., & Gielen, S. C. A. M. (2007). Input-driven oscillations in networks with excitatory and inhibitory neurons with dynamic synapses. *Neural Computation, 19*(7), 1739–1765.

Mattia, M., & Giudice, P. D. (2000). Efficient event-driven simulation of large networks of spiking neurons and dynamical synapses. *Neural Computation, 12*(10), 2305–2329.

McKennoch, S., Lui, D., & Bushnell, L. (2006). Fast modifications of the spikeprop algorithm. In *Proceedings of the IEEE world congress on computational intelligence (WCCI).* Piscataway, NJ: IEEE Press.

McKennoch, S., Sundaradevan, P., & Bushnell, L. G. (2007). Theta neuron networks: Robustness to noise in embedded applications. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN).* Piscataway, NJ: IEEE Press.

Memmesheimer, R. M., & Timme, M. (2006). Designing the dynamics of spiking neural networks. *Physical Review Letters, 97*(18), 188101-1–188101-4.

Osan, R., & Ermentrout, B. (2001). Two dimensional synaptically generated traveling waves in a theta-neuron neural network. *Neurocomputing, 38–40*, 789–795.

Popović, D., & Sinkjr, T. (2000). *Control of movement for the physically disabled: Control for rehabilitation technology*. New York: Springer.

Reed, R. D., & Marks, R. J. (1998). *Neural smithing: Supervised learning in feedforward artificial neural networks*. Cambridge, MA: MIT Press.

Rieke, F., Warland, D., Steveninck, R. de Ruyter van, & Bialek, W. (1999). *Spikes: Exploring the neural code*. Cambridge, MA: MIT Press.

Schrauwen, B., & Van Campenhout, J. (2004). Extending spikeprop. In J. V. Campenhout (Ed.), *Proceedings of the International Joint Conference on Neural Networks (IJCNN)* (pp. 471–476). Piscataway, NJ: IEEE Press.

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

Thorpe, S., Fize, D., & Marlot, C. (1996). Speed of processing in the human visual system. *Nature, 381*(6582), 520–522.

Tonnelier, A., Belmabrouk, H., & Martinez, D. (2007). Event-driven simulations of nonlinear integrate-and-fire neurons. *Neural Computation, 19*(12), 3226–3238.

Voegtlin, T. (2007). Temporal coding using the response properties of spiking neurons. In B. Schölkopf, J. Platt, & T. Hoffman (Eds.), *Advances in neural information processing systems, 19* (pp. 1457–1464). Cambridge, MA: MIT Press.

Wolberg, W. H., & Mangasarian, O. (1990). Multisurface method of pattern separation for medical diagnosis applied to breast cytology. In *Proceedings of the National Academy of Sciences* (vol. 87, pp. 9193–9196). Washington, DC: National Academy of Sciences.

Xin, J., & Embrechts, M. J. (2001). Supervised learning with spiking neural networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)* (vol. 3, pp. 1772–1777). Piscataway, NJ: IEEE Press.

Zhong, P., & Fukushima, M. (2007). Regularized nonsmooth Newton method for multi-class support vector machines. *Optimization Methods and Software, 22*, 225–236.

Received September 14, 2007; accepted April 14, 2008.