Abstract: SpikeProp is a supervised learning algorithm for spiking neural networks analogous to back propagation. Like back propagation, it may fail to converge for particular networks, parameters and datasets. However there are several behaviours and additional failure modes unique to SpikeProp which have not been explicitly outlined in the literature. These factors hinder the adoption of SpikeProp for general machine learning use. In this paper we examine the mathematics of SpikeProp in detail and identify the various causes of failure therein. The analysis implies that applying certain constraints on parameters like initial weights can improve the rates of convergence. It also suggests that alternative spike response functions could improve the learning rate and reduce the number of convergence failures. We tested two alternative functions and found these predictions to be true.

Analysis of SpikeProp Convergence with Alternative Spike Response Functions

Vaenthan Thiruvarudchelvan*, James W. Crane*†‡ and Terry Bossomaier*
*Centre for Research in Complex Systems
†School of Biomedical Sciences
Charles Sturt University
Bathurst, Australia
‡The University of Queensland
Queensland Brain Institute
Brisbane, Australia
Email: {vthiru, jcrane, tbossomaier}@csu.edu.au

***Abstract*—SpikeProp is a supervised learning algorithm for spiking neural networks analogous to backpropagation. Like backpropagation, it may fail to converge for particular networks, parameters and datasets. However there are several behaviours and additional failure modes unique to SpikeProp which have not been explicitly outlined in the literature. These factors hinder the adoption of SpikeProp for general machine learning use. In this paper we examine the mathematics of SpikeProp in detail and identify the various causes of failure therein. The analysis implies that applying certain constraints on parameters like initial weights can improve the rates of convergence. It also suggests that alternative spike response functions could improve the learning rate and reduce the number of convergence failures. We tested two alternative functions and found these predictions to be true.**

I. Introduction

Spiking neural networks are a more realistic abstraction of biological networks that have gained popularity over the past decade [1]. They involve computing the time-dependent membrane potential of neurons, which offers greater biological plausibility than the established multilayer perceptron (MLP) [2] model. With greater realism, it is generally held that spiking networks should have much larger capabilities, extending all the way to human cognition. Several groups are attempting to simulate the human brain this way [3], while it has been shown mathematically that spiking neurons have greater representational capacity than MLPs [4].

From a machine learning perspective, just as it took over a decade for the backpropagation algorithm ("backprop") to be popularized for the MLP, it has taken time to develop learning rules for spiking networks. Today several methods have been presented, including SpikeNet [5], SpikeProp [6], ReSuMe [7], Tempotron [8] and others. SpikeProp was among the first general machine learning rules and applied the principle of backpropagation to a feedforward spiking neural network. Instead of backprop adjusting abstract neuron "activations" (representing the average firing rate), SpikeProp adjusts the precise spike time of each neuron, which is allowed to fire only once.

The similarity in methodology of SpikeProp to backprop suggests that it can be used as a straight substitute. In presenting the algorithm, Bohte et al. [6] demonstrated the algorithm on the XOR problem and some common machine learning datasets. A special encoding scheme of receptive fields was used to overcome some limitations of the algorithm.

In earlier work [9], we attempted to use SpikeProp with a real-world dataset in a simplistic manner as permitted by backprop. We encountered a variety of strange behaviours, as will other users of SpikeProp not closely versed in its intricacies. In order to understand some of the phenomena observed, we performed a thorough analysis of the SpikeProp algorithm which we present in this paper.

First we reproduce the mathematics of the standard SpikeProp in Section II, using alternative mathematical notations to avoid some ambiguity in the original paper. This assists in implementation as well as our analysis, but experienced readers may wish to skip this section. Section III describes our analysis of the algorithm, making reference to previous studies. In it we identify the possible benefit of alternative spike response functions. This hypothesis is tested with experiments in Section IV, whose results are examined in the Discussion.

II. SpikeProp Derivation

SpikeProp uses a feedforward network of one input, one hidden and one output layer – H, I and J respectively (figure available in [6]). The algorithm can generalize to more layers, however. Particular neurons from each layer are represented by the lowercase h, i and j. The sets Γ_i and Γ^i refer to the neurons in the immediately preceeding and immediately succeeding layers respectively of a particular neuron i.

Each **connection** between neurons in adjacent layers represents m **subconnections** or "synapses", each with constant incremental delays d_k, $k \in \{1..m\}$. Each subconnection has its own weight variable w_{ij}^k which remains constant at $\hat{w}_{ij}^k$ during a simulation, but is updated by SpikeProp afterwards. We use the circumflex to denote particular values for variables from a given simulation throughout. The variables t_h, t_i and t_j represent the spike times of particular neurons in the respective layers, and $\hat{t}_h$, $\hat{t}_i$ and $\hat{t}_j$ refer to the actual spike times.

The time-varying spike response function or postsynaptic potential (PSP) kernel $\varepsilon(t)$ is defined as an α-function for $t \geq 0$ and zero elsewhere, with τ being the synaptic membrane time-constant (see *alpha* in figure 4):

$$\varepsilon(t) = \frac{t}{\tau} e^{1-\frac{t}{\tau}} \tag{1}$$

The time-varying PSP contribution of a single subconnection from a neuron i to the membrane of a neuron in the next layer is $\varepsilon(t)$ delayed by both its spike time t_i and delay d^k, and zero for $t \leq t_i + d_k$ (see figure 2):

$$y_i^k(t) = \varepsilon(t - t_i - d_k)\big|_{t_i=\hat{t}_i} \tag{2}$$

The time-varying function $x_j(t)$ is the membrane variable of an output neuron, equal to the weighted sum of incoming PSP functions from the previous layer:

$$x_j(t) = \sum_{i\in\Gamma_j} \sum_{k=1}^{m} w_{ij}^k \, y_i^k(t)\big|_{t_i=\hat{t}_i} \tag{3}$$

Hereafter, we omit the braces in $y_i^k(t)$ and $x_j(t)$ for clarity. We use $\hat{x}_j$ to represent the point where the membrane variable first reaches the threshold ϑ, thereby defining t_j. SpikeProp is only concerned with this first spike so subsequent membrane dynamics can be ignored. The error-function we use is half the least mean squares, where T_j is the training target spike time for output neuron j:

$$E = \frac{1}{2} \sum_{j\in J} (t_j - T_j)^2\big|_{t_j=\hat{t}_j} \tag{4}$$

Although this function is always positive and averaged over all outputs, the partial derivatives w.r.t. each weight are not. SpikeProp is a gradient descent method, which adjusts each weight in the direction minimizing E.

A. Derivatives of $y_i^k(t)$

Here we evaluate some derivatives of y_i^k which arise in subsequent calculations. From equation 2:

$$y_i^k = \varepsilon(u) \quad \text{where} \quad u = t - t_i - d_k$$

Using the chain rule, the partial derivative w.r.t. t is:

$$\begin{aligned}\frac{\partial}{\partial t}\left(y_i^k\right) &= \frac{\partial y_i^k}{\partial u}.\frac{\partial u}{\partial t} \\ &= \frac{\partial \varepsilon}{\partial u}\end{aligned} \tag{5}$$

Similarly, the partial derivative w.r.t. t_i is:

$$\begin{aligned}\frac{\partial}{\partial t_i}\left(y_i^k\right) &= \frac{\partial y_i^k}{\partial u}.\frac{\partial u}{\partial t_i} \\ &= -\frac{\partial \varepsilon}{\partial u} \\ &= -\frac{\partial}{\partial t}\left(y_i^k\right)\end{aligned} \tag{6}$$

This result is independent of the particular function of $\varepsilon(t)$. For the standard PSP function in equation 1:

$$\begin{aligned}\frac{d\varepsilon}{du} &= \left(\frac{1}{\tau}\right)\left(e^{1-\frac{u}{\tau}}\right) + \left(\frac{u}{\tau}\right)\left(-\frac{1}{\tau}e^{1-\frac{u}{\tau}}\right) \\ &= \left(\frac{u}{\tau}e^{1-\frac{u}{\tau}}\right)\left(\frac{1}{u} - \frac{1}{\tau}\right) \\ &= \varepsilon(u)\left(\frac{1}{u} - \frac{1}{\tau}\right) \\ \therefore \quad \frac{\partial}{\partial t}\left(y_i^k\right) &= y_i^k\left(\frac{1}{t - t_i - d_k} - \frac{1}{\tau}\right)\end{aligned} \tag{7}$$

Remember from equation 2 that $y_i^k = 0$ for $t \leq t_i + d_k$. Because it is not differentiable at $t = 0$, its time derivative is undefined there. Bohte [6] did not mention this, and we follow Moore [10] in setting $\partial/\partial t\left(y_i^k\right) = 0$ for $t \leq t_i + d_k$.

B. Output Layer

To train the network, we first want to update each weight between the hidden and output layers to minimize the error:

$$\Delta w_{ij}^k = -\eta \left.\frac{\partial E}{\partial w_{ij}^k}\right|_{w_{ij}^k=\hat{w}_{ij}^k} \tag{8}$$

From equation 4, $E = f(t_{j_1}, t_{j_2}, t_{j_3}, ..., t_{j_n})$ so by the multivariate chain rule:

$$\frac{\partial E}{\partial w_{ij}^k} = \sum_{l\in J} \frac{\partial E}{\partial t_l}.\frac{\partial t_l}{\partial w_{ij}^k}$$

Since $\frac{\partial t_l}{\partial w_{ij}^k} = 0$ except for $l = j$, only one term of the sum remains:

$$\left.\frac{\partial E}{\partial w_{ij}^k}\right|_{w_{ij}^k=\hat{w}_{ij}^k} = \left.\frac{\partial E}{\partial t_j}\right|_{t_j=\hat{t}_j} . \left.\frac{\partial t_j}{\partial w_{ij}^k}\right|_{w_{ij}^k=\hat{w}_{ij}^k}$$

Since $t_j = f(x_j)$ (the threshold function), the last derivative can be expanded to give:

$$\left.\frac{\partial E}{\partial w_{ij}^k}\right|_{w_{ij}^k=\hat{w}_{ij}^k} = \underbrace{\left.\frac{\partial E}{\partial t_j}\right|_{t_j=\hat{t}_j}}_{①} \cdot \underbrace{\left.\frac{\partial t_j}{\partial x_j}\right|_{x_j=\hat{x}_j}}_{②} \cdot \underbrace{\left.\frac{\partial x_j}{\partial w_{ij}^k}\right|_{w_{ij}^k=\hat{w}_{ij}^k}}_{③} \tag{9}$$

We evaluate each component term in turn. Substituting equation 4 into the first term and recognizing that all output times are independent of each other, we have:

$$\begin{aligned}① = \left.\frac{\partial E}{\partial t_j}\right|_{t_j=\hat{t}_j} &= \left.\frac{\partial}{\partial t_j}\left(\frac{1}{2}\sum_{l\in J}(t_l - T_l)^2\right)\right|_{t_j=\hat{t}_j} \\ &= \left.\frac{1}{2}\frac{\partial}{\partial t_j}\left((t_j - T_j)^2\right)\right|_{t_j=\hat{t}_j} \\ &= \left.\frac{1}{2}\left(2(t_j - T_j)\right)\right|_{t_j=\hat{t}_j} \\ &= \hat{t}_j - T_j\end{aligned}$$

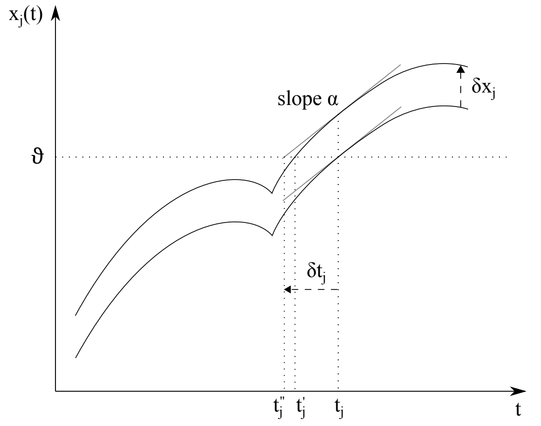


Figure 1. Approximating $\partial t_j/\partial x_j$. As $x_j(t)$ is raised, the threshold is reached earlier, bringing t_j forward to t'_j. Linearly approximating with slope $\alpha = \partial x_j/\partial t$ gives us the spike time t''_j.

To simplify the second term, we make the assumption that time-dependent x_j is linear for an ε-space around t_j (figure 1). As x_j is raised, t_j occurs earlier as a consequence of thresholding. If α is the gradient of x_j at t_j:

$$\begin{aligned} \delta t_j &= -\frac{\delta x_j}{\alpha} \\ \frac{\delta t_j}{\delta x_j} &= -\frac{1}{\alpha} \\ \therefore\quad \frac{\partial t_j}{\partial x_j} &\approx -\frac{1}{\alpha} \end{aligned}$$

Using equation 3 for x_j, the approximation is:

$$\begin{aligned} ② = \left.\frac{\partial t_j}{\partial x_j}\right|_{x_j=\hat{x}_j} &\approx -\frac{1}{\alpha} \\ &= -\frac{1}{\left.\frac{\partial x_j}{\partial t}\right|_{t=\hat{t}_j}} \\ &= -\frac{1}{\sum_{i\in\Gamma_j}\sum_{l=1}^{m} w_{ij}^l.\ \frac{\partial}{\partial t}(y_i^l)\big|_{t=\hat{t}_j,t_i=\hat{t}_i}} \end{aligned}$$

The third term is a straightforward differentiation. Sum terms are zero where $p \neq i$ and $l \neq k$, and y_p^l is constant w.r.t. w_{ij}^k so:

$$\begin{aligned} ③ = \left.\frac{\partial x_j}{\partial w_{ij}^k}\right| &= \left.\frac{\partial}{\partial w_{ij}^k}\left(\sum_{p\in\Gamma_j}\sum_{l=1}^{m} w_{pj}^l.\ y_p^l\big|_{t_p=\hat{t}_p}\right)\right|_{w_{ij}^k=\hat{w}_{ij}^k} \\ &= y_i^k\big|_{t=\hat{t}_j,t_i=\hat{t}_i} \end{aligned}$$

The error E at w_{ij}^k is based on the value of x_j at $t = \hat{t}_j$, which is why the result is evaluated there. Putting the three components together, equation 9 becomes:

$$\left.\frac{\partial E}{\partial w_{ij}^k}\right|_{w_{ij}^k=\hat{w}_{ij}^k} = \frac{(T_j - \hat{t}_j).\ y_i^k\big|_{t=\hat{t}_j,t_i=\hat{t}_i}}{\sum_{i\in\Gamma_j}\sum_{l=1}^{m} w_{ij}^l\ \frac{\partial}{\partial t}(y_i^l)\big|_{t=\hat{t}_j,t_i=\hat{t}_i}} \qquad (10)$$

For convenience, we define the product of terms ① and ②, common to all connections terminating at neuron j, as:

$$\begin{aligned} \delta_j &= \left.\frac{\partial E}{\partial t_j}\right|_{t_j=\hat{t}_j} . \left.\frac{\partial t_j}{\partial x_j}\right|_{x_j=\hat{x}_j} \\ &= \frac{T_j - \hat{t}_j}{\sum_{i\in\Gamma_j}\sum_{l=1}^{m} w_{ij}^l\ \frac{\partial}{\partial t}(y_i^l)\big|_{t=\hat{t}_j,t_i=\hat{t}_i}} \end{aligned} \qquad (11)$$

Equation 8 can now be expressed as:

$$\Delta w_{ij}^k = -\eta.\delta_j.\ y_i^k\big|_{t=\hat{t}_j,t_i=\hat{t}_i} \qquad (12)$$

C. Preceding Layers

The values calculated for δ_j can be used when updating the weights between the input and hidden layers. In fact the process can be iteratively backpropagated to any number of preceding layers this way. The weights are adjusted like in equation 8:

$$\Delta w_{hi}^k = -\eta \left.\frac{\partial E}{\partial w_{hi}^k}\right|_{w_{hi}^k=\hat{w}_{hi}^k} \qquad (13)$$

This expands in a similar fashion to equation 9:

$$\left.\frac{\partial E}{\partial w_{hi}^k}\right|_{w_{hi}^k=\hat{w}_{hi}^k} = \underbrace{\left.\frac{\partial E}{\partial t_i}\right|_{t_i=\hat{t}_i}}_{Ⓐ} . \underbrace{\left.\frac{\partial t_i}{\partial x_i}\right|_{x_i=\hat{x}_i}}_{Ⓑ} . \underbrace{\left.\frac{\partial x_i}{\partial w_{hi}^k}\right|_{w_{hi}^k=\hat{w}_{hi}^k}}_{Ⓒ} \qquad (14)$$

In this case, the first term expands differently to before. From equation 4, $E = f(t_{j_1}, t_{j_2}, t_{j_3}, ..., t_{j_n})$ and multivariate chain rule gives:

$$\left.\frac{\partial E}{\partial t_i}\right|_{t_i=\hat{t}_i} = \sum_{j\in\Gamma^i} \left.\frac{\partial E}{\partial t_j}\right|_{t_j=\hat{t}_j} . \left.\frac{\partial t_j}{\partial t_i}\right|_{t_i=\hat{t}_i}$$

By thresholding $t_j = f(x_j)$, and the last derivative can be expanded by the chain rule, and we see that the first two derivatives are in fact δ_j (equation 11):

$$\begin{aligned} \left.\frac{\partial E}{\partial t_i}\right|_{t_i=\hat{t}_i} &= \sum_{j\in\Gamma^i} \left.\frac{\partial E}{\partial t_j}\right|_{t_j=\hat{t}_j} . \left.\frac{\partial t_j}{\partial x_j}\right|_{x_j=\hat{x}_j} . \left.\frac{\partial x_j}{\partial t_i}\right|_{t_i=\hat{t}_i} \\ &= \sum_{j\in\Gamma^i} \delta_j . \left.\frac{\partial x_j}{\partial t_i}\right|_{t_i=\hat{t}_i} \end{aligned}$$

Substituting in equation 3 for x_j:

$$\left.\frac{\partial E}{\partial t_i}\right|_{t_i=\hat{t}_i} = \sum_{j\in\Gamma^i} \delta_j . \left.\frac{\partial}{\partial t_i}\left(\sum_{p\in\Gamma_j}\sum_{l=1}^{m} w_{pj}^l\ y_p^l\big|_{t_p=\hat{t}_p}\right)\right|_{t_i=\hat{t}_i}$$

Every term of the first sum inside the brackets disappears except when $p = i$, since the contributions of other hidden neurons are independent of t_i:

$$\left.\frac{\partial E}{\partial t_i}\right|_{t_i=\hat{t}_i} = \sum_{j\in\Gamma^i}\delta_j\sum_{l=1}^{m} w_{ij}^l\frac{\partial}{\partial t_i}\left(y_i^l\right)\Big|_{t=\hat{t}_j,t_i=\hat{t}_i}$$

Again the result is evaluated at $t=\hat{t}_j$ because the error E at w_{ij}^k is based on the value of x_j at that point. Equation 6 now transforms the derivative for convenience:

$$\textcircled{A} = \left.\frac{\partial E}{\partial t_i}\right|_{t_i=\hat{t}_i} = -\sum_{j\in\Gamma^i}\delta_j\sum_{l=1}^{m} w_{ij}^l\frac{\partial}{\partial \mathbf{t}}\left(y_i^l\right)\Big|_{t=\hat{t}_j,t_i=\hat{t}_i}$$

The last two terms evaluate as before ($\textcircled{2}$ and $\textcircled{3}$ respectively):

$$\textcircled{B} = \left.\frac{\partial t_i}{\partial x_i}\right|_{x_i=\hat{x}_i} = -\frac{1}{\sum_{h\in\Gamma_i}\sum_{l=1}^{m} w_{hi}^l\ \frac{\partial}{\partial t}(y_h^l)\big|_{t=\hat{t}_i,t_h=\hat{t}_h}}$$

$$\textcircled{C} = \left.\frac{\partial x_i}{\partial w_{hi}^k}\right|_{w_{hi}^k=\hat{w}_{hi}^k} = y_h^k\big|_{t=\hat{t}_i,t_h=\hat{t}_h}$$

Putting the three components together, equation 14 becomes:

$$\left.\frac{\partial E}{\partial w_{hi}^k}\right| = \frac{\sum_{j\in\Gamma^i}\delta_j\sum_{l=1}^{m} w_{ij}^l\ \frac{\partial}{\partial t}\left(y_i^l\right)\big|_{t=\hat{t}_j,t_i=\hat{t}_i}\cdot y_h^k\big|_{t=\hat{t}_i,t_h=\hat{t}_h}}{\sum_{h\in\Gamma_i}\sum_{l=1}^{m} w_{hi}^l\ \frac{\partial}{\partial t}(y_h^l)\big|_{t=\hat{t}_i,t_h=\hat{t}_h}} \tag{15}$$

For convenience, we define the product of terms $\textcircled{A}$ and $\textcircled{B}$, common to all connections terminating at neuron i, as:

$$\begin{aligned}\delta_i &= \left.\frac{\partial E}{\partial t_i}\right|_{t_i=\hat{t}_i}.\left.\frac{\partial t_i}{\partial x_i}\right|_{x_i=\hat{x}_i}\\ &= \frac{\sum_{j\in\Gamma^i}\delta_j\sum_{l=1}^{m} w_{ij}^l\ \frac{\partial}{\partial t}\left(y_i^l\right)\big|_{t=\hat{t}_j,t_i=\hat{t}_i}}{\sum_{h\in\Gamma_i}\sum_{l=1}^{m} w_{hi}^l\ \frac{\partial}{\partial t}(y_h^l)\big|_{t=\hat{t}_i,t_h=\hat{t}_h}}\end{aligned} \tag{16}$$

Equation 13 can now be expressed as:

$$\Delta w_{hi}^k = -\eta.\delta_i.\ y_h^k\big|_{t=\hat{t}_i,t_h=\hat{t}_h} \tag{17}$$

This is essentially the same as equation 12, so we can generalize to:

$$\Delta w_{pq}^k = -\eta.\delta_q.\ y_p^k\big|_{t=\hat{t}_q,t_p=\hat{t}_p} \tag{18}$$

for all p and q in adjacent layers. Based on this derivation, the basic SpikeProp algorithm is presented in algorithm 1.

III. Analysis

At first look, it appears that we can take SpikeProp and substitute it for an MLP, instantly realizing the benefits of spiking networks (Section I). Unfortunately it is not so straightforward, and there are several pitfalls when using SpikeProp. We review some of the issues already presented in the literature and conduct further analysis.

In the last two lines of Algorithm 1, the error function E (equation 4) is used to test whether the network has reached

Algorithm 1: The SpikeProp Algorithm

```
setup network;
initialize weights;
repeat
    foreach training pattern do
        // Set input spike times
        foreach input neuron p do
            set p.spiked, $t_p \leftarrow input_p$;
        end
        // Run network
        $t \leftarrow 0$;
        repeat
            $t \leftarrow t + time\_step$;
            for layer $Q \leftarrow$ input+1 to output do
                foreach neuron q do
                    if q.spiked then continue;
                    $x_q \leftarrow 0$;
                    foreach neuron p in layer (Q-1) do
                        if not p.spiked then continue;
                        foreach subconnection k do
                            $x_q \leftarrow x_q + w_{pq}^k y_p^k(t, t_p)$;
                        end
                    end
                    if $x_q \geq \vartheta$ then
                        set q.spiked, $t_q \leftarrow t$;
                    end
                end
            end
        until $t = max\_steps$ ;
        // Calculate deltas backwards
        for layer $Q \leftarrow$ output to input+1 do
            foreach neuron q do
                calculate $\delta_q$ ;          // Eqs 11, 16
            end
        end
        // Update weights
        for layer $P \leftarrow$ input to output-1 do
            foreach neuron p do
                foreach neuron q in layer (P+1) do
                    foreach subconnection k do
                        $w_{pq}^k \leftarrow w_{pq}^k - \Delta w_{pq}^k$ ; // Eq 18
                    end
                end
            end
        end
        reset network;
    end
    // Calculate total MSE
    totalMSE $\leftarrow 0$
    foreach training pattern do
        run network, calculate MSE ;          // Eq 4
        totalMSE $\leftarrow$ totalMSE + MSE
    end
until totalMSE $\leq$ target_error ;
```

an appropriate level of convergence. We note that it is not mandatory to use E for this purpose. For example, if we wish the output neurons to fire within ϕ ms of the target, we could use the following termination condition instead:

$$\max_{j \in J}(t_j - T_j) \leq \phi \tag{19}$$

SpikeProp will continue to converge as before, since E is implicit in the equations derived above.

A. Time Quantization

The membrane variables of neurons in SpikeProp are calculated at fixed timestep intervals (*time_step* in algorithm 1), meaning that all spike times are quantized to these intervals. Bohte [6] recognized that input and output accuracy is therefore limited by this, and that increased accuracy can be gained by reducing the step size at the cost of greater computation. Alternatively, we propose that widening the time-spread of input and output spike patterns along with a greater synaptic time-constant achieves the same thing, again at greater cost of computation.

Potential users of SpikeProp should remember that pre- and post-processing of input and output data is required, to transform it into and out of spike times. If a receptive field encoding is used like [6] (may be required to avoid the no-spike condition below), this may be significant.

Masaru et al. [11] found that time-stepped evaluation makes the error surface stepped and rough, introducing many minute local minima. This limits the usefulness of higher-order optimisation techniques, which accelerate gradient-descent by approximating the shape of the error surface and jumping straight to the minimum. They are widely used with 2nd generation networks, and several (RProp, QuickProp [12], Levenberg-Marquadt [13]) have been applied to SpikeProp. Lastly, we submit that aliasing introduces further noise where the membrane potential only briefly breaches the threshold during a timestep and the spike is missed. This should be exceedingly rare if the synaptic time-constant is large relative to the timestep.

All these aliasing problems could be eliminated using analogue implementation, or asynchronous event-driven computation [14], the latter offering compute-time benefits as well.

B. Topology and Initial Weights

Sporea et al. [15] clarified the need for a reference neuron. Without an input neuron that always fires at $t = 0$, the network has no way of differentiating input patterns of simultaneous spikes, drastically reducing the number of functions it can learn.

Bohte also introduced inhibitory neurons which produce inverted inhibitory PSPs (IPSPs) at target neurons. Such a neuron is equivalent to multiplying all the outgoing weights of an excitatory neuron by -1. They claimed that using a single inhibitory hidden neuron was necessary for the XOR example [6], but only if the connection's weights had the same polarity. As they pointed out, this is due to the fact that the membrane variable at the target neuron is no longer monotonically increasing. Takase et al. [16] found that such a condition causes surges – large divergences in error between epochs. As training progresses and $x_j(t)$ is raised or lowered, the spike time "jumps" across any dips, producing a surge. (This condition also arises without inhibitory neurons, as a consequence of the decaying spike response and negative weights.) Moore found however, that IPSPs were *not* strictly necessary for convergence. Care must be taken to ensure that the derivative of inhibitory spikes is also inverted when calculating deltas, something that has not been mentioned explicitly before. Oversight of this may be the cause of the conflicting results.

No method for selecting initial weights or thresholds was described by Bohte however. PSPs arriving at a neuron are scaled by their weights, therefore they are related to the spike threshold – too low a threshold and late incoming spikes will be ignored, too high and the neuron will not fire at all. Hence, unlike backprop, weights cannot be initialized haphazardly. Moore [10] examined this problem in more detail, comparing three different weight initialization schemes. His results are reproduced in table I. Method 3 was the fastest converging, at the cost of a high failure rate. The conclusion is that there is a trade-off between convergence speed and failure, which we explore in the next section. He also found that methods 1 and 2 worked with a large learning rate of 1, while method 3 requires smaller learning rates (pp88). Lastly, Moore found that a mixture of positive and negative weights could actually improve performance, and might in fact supplant the need for inhibitory neurons in SpikeProp.

C. Convergence

Like backprop, SpikeProp is susceptible to getting stuck in local minima. Additionally, the SpikeProp algorithm is not well-behaved like backprop and may fail for additional reasons which we term **technical failures**. These are lumped together as "non-convergence" in the literature. To assist our analysis, we reproduce a formulation of backprop [2]:

$$\begin{aligned}
\psi_q &= \sum_{p \in \Gamma_q} w_{pq} \sigma\left(\psi_p\right) \\
\delta_j &= \frac{d\sigma}{dx}\left(\psi_j\right) . \left(\sigma\left(\psi_j\right) - T_j\right) \\
\delta_i &= \frac{d\sigma}{dx}\left(\psi_i\right) . \sum_{j \in \Gamma^i} \delta_j w_{ij} \\
\Delta w_{pq} &= -\eta . \delta_q . \sigma\left(\psi_p\right)
\end{aligned} \tag{20}$$

with ψ_q being the input of neuron q (non-input layers) and $\sigma(\psi_q)$ being its ouput, a function of sigmoid shape with range (0,1) or (-1, 1). The remaining terms have the same conventions as before.

The first thing to note is that in backprop, the output of a neuron $\sigma(\psi_q)$ is always defined. With SpikeProp, it is possible that the membrane variable x_p (equation 3) for neurons in the hidden and output layers never exceeds the

#	Description	Avg. Iter.	Fail%	Tech. Fail%
1	Every weight randomly chosen from [1..10], threshold 50	148.1	1.5	0.5
2	Weights within connection equal, randomly chosen from [1..10], threshold 50	187.1	9.5	5
3	Weights from method 1 normalized by (5×14), threshold 1	121.1	11	0.5

Table I
WEIGHT INITIALIZATION METHODS (PP49) AND RESULTS FROM MOORE [10], EXPERIMENT 7. EACH METHOD WAS USED FOR 200 TRIALS OF XOR NETWORK. AVG. ITERATIONS IS FOR SUCCESSFUL CONVERGENCES, FAIL% = TECH. FAIL% + CONVENTIONAL NON-CONVERGENCE.

threshold to spike, leaving t_p in the algorithm undefined. This was stated by Bohte [6], who avoided it by encoding inputs with sets of neurons effecting receptive fields. However the statement "early spikes are automatically 'more important' than later spikes" is ambiguous (pp23). The no-spike problem is compounded by the effect of inhibitory neurons. We suggest lowering the threshold ϑ or adjusting weights to overcome this.

Next we see that the backprop equations contain no fractions, whereas the denominators of SpikeProp equations 11 and 16 can result in a divide-by-zero error. First ignoring weights, terms of the denominator sums are zero when $\partial/\partial t(y_p^l) = 0$. From section II-A, this is true for $\hat{t}_q \leq \hat{t}_p + d_l$ (i.e. q fired before the delayed spike from p arrived) and at $t = \tau$. The first condition cannot be solely responsible though, since neuron q must have been triggered by at least one predecessor. The zero denominator can also happen through zero weights, or negative terms (where weight or derivative is negative) canceling out positive ones. The same concerns apply to the numerator of equation 16, producing a false local minimum in the error space.

Therefore, it is best to constrain the algorithm to positive weights by clamping negative weights to zero, and selecting a new spike response function with a non-negative derivative. As with inhibitory neurons above, membrane potentials containing downward slopes cause undesirable surge behaviour [16], further supporting a monotonically increasing response function. We note that all common activation functions used with backprop are sigmoid[1], satisfying this criteria.

When a neuron q fires before receiving the spike from subconnection w_{pq}^k, q is essentially independent of it, so there is no sensible way to adjust the weight. This is borne out by the factor y_p^k in equation 18, which sets the weight change to zero (see figure 2). This a second reason why y_p^k must equal zero for $t < 0$ (the intuitive reason being that a spike cannot have had any impact on its target before it arrived). It is this phenomenon manifesting when weights do not change over several epochs, shown in figure 3. A subconnection between neurons is effectively dropped from the network, reducing its power. Furthermore, q may spike progressively later[2] over the course of training, "uncovering" the formerly late spike and causing the error to jump. This is a second cause of surge behaviour to that that described by Takase [16].

[1]Radial basis function (RBF) networks use radial functions like the Gaussian function, however they do not use backpropagation to learn.

[2]Incidentally, this is the reason why epochs tends to take longer to compute as training progresses – more timesteps must elapse for all neurons to fire.

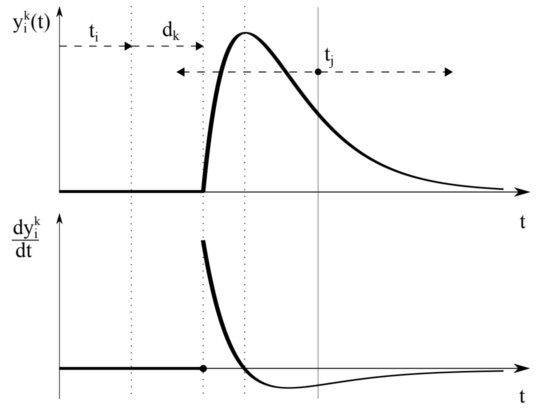
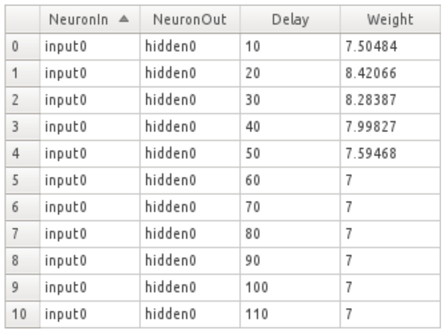


Figure 2. Delayed spike response function $y_i^k(t)$ and its derivative, showing the relationship between t_i, d_k and t_j. The latter can occur sometime before $(t_i + d_k)$, zeroing out terms when deltas are calculated.



	NeuronIn ▲	NeuronOut	Delay	Weight
0	input0	hidden0	10	7.50484
1	input0	hidden0	20	8.42066
2	input0	hidden0	30	8.28387
3	input0	hidden0	40	7.99827
4	input0	hidden0	50	7.59468
5	input0	hidden0	60	7
6	input0	hidden0	70	7
7	input0	hidden0	80	7
8	input0	hidden0	90	7
9	input0	hidden0	100	7
10	input0	hidden0	110	7

Figure 3. A sample of weights after several epochs of training, showing that only the first five in this connection had changed. Delays are in timesteps.

We propose either raising the threshold of neuron q to give subconnection w_{pq}^k a chance to contribute, and/or setting initial weights close to each other (e.g. 1 ± 0.5) to minimise individual connections dominating q.

D. Alternative Spike Response Functions

Notice that the derivation in Section II is agnostic of the particular spike response function $\varepsilon(t)$ used. From the previous section we saw that the delayed spike response function $y(t)$ (equation 2) and its derivative must be zero for $t < 0$. We also saw that a monotonically increasing response function ($\partial y/\partial t \geq 0$) reduces the chances of zero numerators and denominators in the delta equations, corresponding to false local minima or divide-by-zero respectively.

We propose two alternative functions satisfying these criteria, which could improve the convergence rates of SpikeProp:

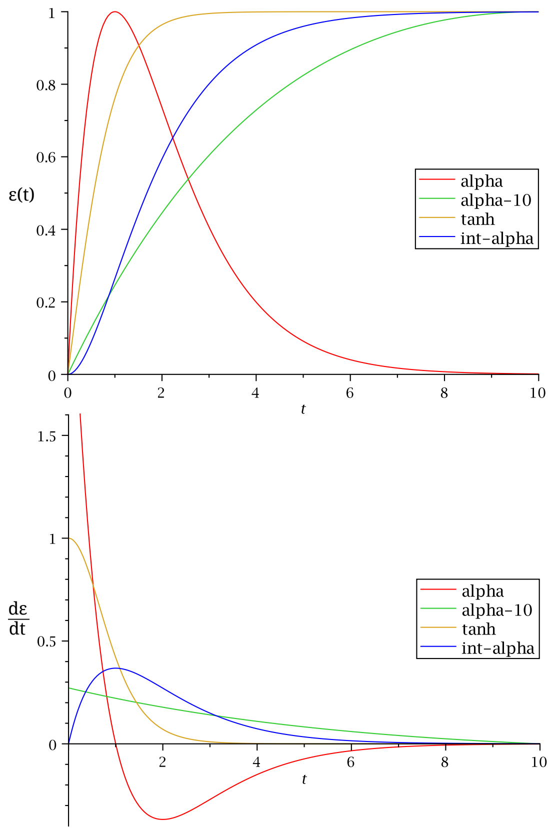


Figure 4. Spike response functions (top) and derivatives (bottom). Alpha plots are the standard alpha function with time-constants of 1 and 10 respectively. Functions tanh and int-alpha have unit time-constants. Note that the derivative of alpha is not asymptotic but reaches a value of e at $t = 0$.

the hyperbolic tangent and the integral of an alpha function which we call int-alpha. We also tried a shifted logistic function, however that is actually equal to the hyperbolic tangent with the time-constant halved. The equations are as follows:

$$\text{tanh:} \quad \varepsilon(t) = \tanh\left(\frac{t}{\tau}\right) \tag{21}$$
$$\frac{d\varepsilon}{dt} = \frac{1}{\tau}\left(1 - \tanh^2\left(\frac{t}{\tau}\right)\right)$$
$$\text{int-alpha:} \quad \varepsilon(t) = 1 - e^{-\frac{t}{\tau}}\left(1 + \frac{t}{\tau}\right) \tag{22}$$
$$\frac{d\varepsilon}{dt} = \frac{t}{\tau^2}e^{-\frac{t}{\tau}}$$

These functions and their derivatives are plotted alongside the standard α-function (equation 1) in figure 4. From the figure, we see that when the time-constant for the standard function is large enough, it is monotonically increasing for the sampling period. Therefore using a large time-constant may be sufficient to escape some of the convergence issues identified.

The int-alpha function is an intriguing possibility. Instead of considering $y(t)$ to be a direct contribution to the somatic membrane potential, we can conceive of ${}^{\partial y}/_{\partial t}$ as a contribution to the *dendritic* membrane voltage. If the dendritic membrane has constant conductance, the current influx is proportional to this voltage. Therefore the integral $y(t)$ is proportional to the electric charge accumulating in the dendrite. The target neuron can be thought of as spiking when the charge accumulated through all its dendrites exceeds a threshold. Interestingly, it is the only function that is differentiable at $t = 0$.

IV. Experiment

To prove whether the alternative functions proposed in the previous section do improve the convergence of SpikeProp, we tested each for 200 trials of the canonical XOR problem with three input neurons (one reference), five hidden neurons (one inhibitory) and one output. Parameters used are listed in table II. The time-constants selected for each alternative spike response function were integers selected to bring their slopes as close as possible to the standard alpha function with a time-constant of 7. The alpha function was also tested with a doubled time constant of 14. Negative weights were permitted because they accelerated convergence here.

V. Results

The results of these simulations are presented in table III. All technical failures were caused by the no-spike condition in this experiment. The first observation is that SpikeProp converged as normal using the two alternative spike response functions. In both cases, convergence failures were almost half that of the standard function, alpha-7. The int-alpha function also tended to converge in 20% fewer epochs. Doubling the time constant of the standard response function, alpha-14, extended the number of epochs required to converge and introduced a large proportion of general convergence failures (some runs were converging very slowly), although technical failures were halved. In terms of execution time, alpha-14 was significantly slower than the others (roughly equivalent), as more timesteps are needed for all neurons to spike.

VI. Discussion

Our basic experiment showed that alternative spike response functions are viable and effective at increasing SpikeProp convergence speed and reducing convergence failure. We note that a) XOR is a very simple problem which did not exhibit zero-denominator failures, masking more potential benefits, and b) no tweaking of parameters was carried out to find the best time-constants (or other parameters) for use with each function. Based on these factors and our experiences with a real-world dataset [9], we anticipate the improvements from using functions like tanh and int-alpha to be greater. Increasing the time-constant of the standard function did reduce failures, at the cost of slower convergence and runtime however.

In Section III we examined in depth the SpikeProp algorithm and identified its sensitivities to several parameters and topologies which can cause unintended behaviour. It helps to understand these issues when configuring a network for a particular application, which is unfortunately still a trial-and-error process.

Parameter	Value	Description
network topology	3,5,1	Input/hidden/output neuron count
η	1	Learning rate
τ	varies	Spike time-constant, see table III
ϑ	50	Spike threshold
ΔT_{in}	6	Input spike time range (ms)
ΔT_{out}	6	Output spike time range (ms)
reference neuron	yes	Additional input with spike time 0
inhibitory neurons	1	Inhibitory hidden neuron count
subconnections	16	Synaptic terminals per connection
weight_init_method	2	Weight init. method (table I)
weight_random_seed	1–200	Random seed for weight init.
negative weights	yes	Permit negative weights
max_steps	5000	Max. steps per simulation
step duration	0.1	Simulation step size (ms)
target_error	2	Target average mean-squared error
max_epochs	1000	Epochs to wait for convergence

Table II
LISTING OF PARAMETERS FOR THE XOR NETWORK USED IN OUR EXPERIMENTS.

$\varepsilon(t)$	τ	Avg. Iter. (Std. Dev.)	Fail%	Tech. Fail%
alpha-7	7	259 (155)	10.5	9.5
alpha-14	14	659 (211)	24	4.5
tanh	3	248 (152)	5.5	5.5
int-alpha	1	194 (120)	6	6

Table III
CONVERGENCE RESULTS FOR DIFFERENT SPIKE RESPONSE FUNCTIONS OVER 200 TRIALS OF XOR NETWORK. COLUMN DESCRIPTIONS ARE THE SAME AS IN TABLE I.

We note however, that Schrauwen et al. [17] have presented an extension to SpikeProp allowing synaptic delays, time constants and thresholds to be adjusted during training. This may help simplify the task of selecting parameters, allowing fewer synapses and faster computation.

VII. Conclusions

In this paper we performed a thorough analysis of the SpikeProp algorithm in order to understand some strange behaviours observed when we first tried to use the algorithm [9]. We were able to explain all of those behaviours, and identified constraints for avoiding some of the pitfalls of SpikeProp.

Through the analysis we identified the possibility of using alternative, monotonically increasing spike response functions to improve convergence speed and success rate. We proposed two alternative functions and tested them against the canonical XOR problem. On that dataset, the new functions were able to halve the convergence failure rate and in one case, accelerate convergence by 20%. These findings validate the use of such spike response functions for improving SpikeProp convergence.

In future work we will apply these new spike response functions to more complex datasets, for which they show promise. We would also like to quantify how parameters like thresholds, negative weights and inhibitory neurons affect SpikeProp, and determine a set of rules for effectively using it.

VIII. Acknowledgements

V.T. was supported by a Ph.D. scholarship from the Center for Research in Complex Systems (CRiCS) at Charles Sturt University.

References

[1] H. Paugam-Moisy, "Spiking neuron networks, a survey," Tech. Rep. IDIAP-RR 06-11, IDIAP Research Institute, February 2006.
[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," tech. rep., Institute for Cognitive Science, University of California San Diego, September 1985.
[3] H. Markram, "The Blue Brain Project," *Nature Reviews Neuroscience*, vol. 7, pp. 153–160, Feb. 2006.
[4] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997.
[5] A. Delorme, J. Gautrais, R. van Rullen, and S. Thorpe, "SpikeNET: A simulator for modeling large networks of integrate and fire neurons," *Neurocomputing*, vol. 26-27, no. 0, pp. 989 – 996, 1999.
[6] S. M. Bohte, J. N. Kok, and H. L. Poutré, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1–4, pp. 17–37, 2002.
[7] F. Ponulak, "ReSuMe - new supervised learning method for spiking neural networks," tech. rep., Poznań: Institute of Control and Information Engineering, Poznań University of Technology., 2005.
[8] R. Gütig and H. Sompolinsky, "The tempotron: a neuron that learns spike timing–based decisions.," *Nature Neuroscience*, vol. 9, no. 3, pp. 420 – 428, 2006.
[9] V. Thiruvarudchelvan and T. Bossomaier, "Towards realtime stance classification by spiking neural network," in *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pp. 1 –8, june 2012.
[10] S. C. Moore, "Backpropagation in spiking neural networks," Master's thesis, University of Bath, 2002.
[11] F. Masaru, T. Haruhiko, K. Hidehiko, and H. Terumine, "Shape of error surfaces in spikeprop," in *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pp. 840 –844, june 2008.
[12] S. McKennoch, D. Liu, and L. Bushnell, "Fast modifications of the spikeprop algorithm," in *Neural Networks, 2006. IJCNN '06. International Joint Conference on*, pp. 3970 –3977, 0-0 2006.
[13] S. Silva and A. Ruano, "Application of the levenberg-marquardt method to the training of spiking neural networks," in *Neural Networks, 2006. IJCNN '06. International Joint Conference on*, pp. 3978 –3982, 0-0 2006.
[14] M. Mattia and P. D. Giudice, "Efficient event-driven simulation of large networks of spiking neurons and dynamical synapses," *Neural Computation*, vol. 12, no. 10, pp. 2305–2329, 2000.
[15] I. Sporea and A. Grüning, "Reference time in spikeprop," in *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pp. 1090 –1092, 31 2011-aug. 5 2011.
[16] H. Takase, M. Fujita, H. Kawanaka, S. Tsuruoka, H. Kita, and T. Hayashi, "Obstacle to training spikeprop networks — cause of surges in training process," in *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, pp. 3062 –3066, june 2009.
[17] B. Schrauwen and J. Van Campenhout, "Extending spikeprop," in *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, vol. 1, pp. 4 vol. (xlvii+3302), july 2004.