Supervised Synaptic Weight Adaptation for a Spiking Neuron

Bryan A. Davis, Deniz Erdogmus, Yadunandana N. Rao, Jose C. Principe

Electrical Engineering Department, University of Florida, Gainesville, FL 32611, USA

***Abstract*- A novel algorithm named Spike-LMS is described that adapts the synaptic weights of an artificial spiking neuron to produce a desired response. The derivation of Spike-LMS follows from the derivation of the Least-Mean Squares (LMS) algorithm used in adaptive filter theory. Spike-LMS works directly in the domain of spike trains, and therefore makes no assumptions about any particular neural encoding method. This algorithm is able to identify the synaptic weights of a spiking neuron given the pre-synaptic and post-synaptic spike trains.[1]**

Introduction

Spiking neural networks have received a good deal of attention in the past few years. A key difficultly in applying them to engineering applications is that application data is not often described by spike trains. Many methods of encoding continuous-valued, discrete-time data into spike trains have been proposed, usually taking the form of rate encoding or temporal encoding. These two encoding methods have problems. Rate encoding regards the individual arrival times as unimportant, and thus underutilizes the information capacity of the spike train. Using only rates is known to be insufficient for many time-sensitive processing tasks, unless the output of a large number of neurons is considered in aggregation (population encoding). Alternatively, temporal encoding methods invariably require some form of synchronization, and the resulting firing patterns are not consistent with many biological firing patterns [1].

When using any formally described encoding method, it is possible to develop supervised learning rules for training a spiking neuron by finding the local minima of a cost function. The cost function should be defined in terms of the error of the pre-encoded data, and the local minima can be found using gradient descent. However, this method will only ensure that the pre-encoded error will be minimized. If a rate code is used, there is no guarantee that the output spike train will match the desired spike train; only their rates will match. If instead a temporal code is used then a synchronization signal must be incorporated into the model to provide the relative timing information [1]. An example of supervised learning with synchronized inputs is described in [2].

The Spike-LMS learning method proposed in this paper does not suffer from these ailments. The algorithm assumes no form of encoding and thus works directly with the spike trains. It is difficult to define a performance criterion in terms of only spike trains (an attempt is made in the derivation described here, but it is not a true performance criterion). Such a performance measure is necessary in order to use gradient techniques typically used in artificial neural network models. Instead of attempting to optimize some measure performance, we formulate the supervised learning problem as a system identification problem. We then describe an algorithm that adapts a neural model to produce the same output as some unknown neural model. If we are able to identify the parameters of the unknown system accurately, then output of the two systems should be nearly identical.

To perform system identification, we create an adaptive model identical to the given model as shown in Fig. 1. We then adapt the weights so that the outputs of the two models converge. In this figure, the variables describing the unknown fixed-weight neuron are identified by a superscript $^{(*)}$. All inputs and the output are spike trains.

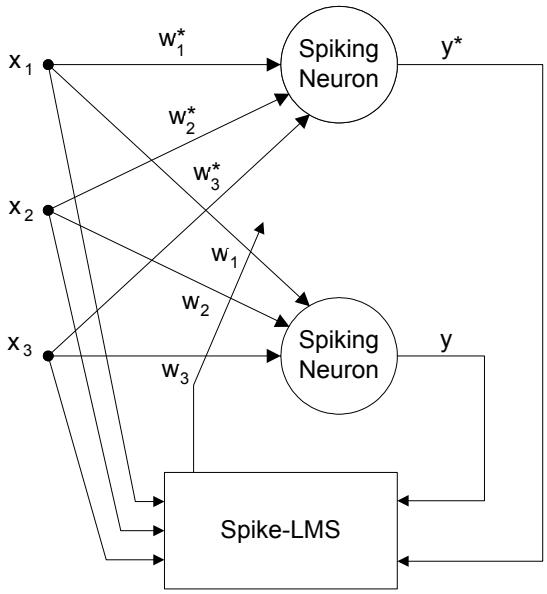

Fig. 1. Identification of the synaptic weights of a spiking neuron: $x_1 - x_3$ are the input spike trains, y is the adaptive output spike train, y* is the desired spike train, $w_1 - w_3$ are the adaptive synaptic weights and $w_1^* - w_3^*$ are the desired synaptic weights.

Spike Response Model of a Neuron

We are using the Spike Response Model (SRM) of a spiking neuron. This model is shown to be a generalization of the Integrate-and-Fire model used throughout the literature. This section is provided to give the reader a basic understanding of the model, and to introduce quantities used

This work was partially supported by NSF EIA 0135946.

in subsequent sections in this paper. A more complete treatise of the SRM can be found in [3].

In the SRM, the neuron receives input spike trains of unit Dirac impulses from a number of other neurons and emits a similar output spike train. We define the set of firing times of the i^{th} input as X_i, and the set of firing times of the output as Y, where $Y = f\left(X_1,...,X_n\right)$ and n is the number of inputs.

The neurons are connected to each other by synapses. The synapses behave as linear filters of the input spike with an impulse response given by $w_i\varepsilon(t)$, where w_i is the connection strength. For a realistic neuron, $\varepsilon(t)$ must be causal and must approach 0 at $t = \infty$. The responses of all the synapses are added together to form the cell potential $v(t)$. Thus,

$$v(t) = \sum_{i=1}^{n} \sum_{t_i^{(x)} \in X_i} w_i \varepsilon\left(t - t_i^{(x)}\right) \tag{1}$$

When the cell potential reaches a threshold $\gamma(t)$ at time $t^{(y)}$, the neuron fires and the threshold is increased by $\eta(t - t^{(y)})$. For a realistic neuron, $\eta(t)$ must be causal and must approach 0 at $t = \infty$. Using a time-varying threshold to describe the SRM is referred to as the *dynamic threshold model* [3]. In general, the threshold is given by

$$\gamma(t) = \gamma_0 + \sum_{t^{(y)} \in Y} \eta\left(t - t^{(y)}\right) \tag{2}$$

where γ_0 is the initial threshold. A typical example how the cell potential changes through time is shown in Fig. 2.

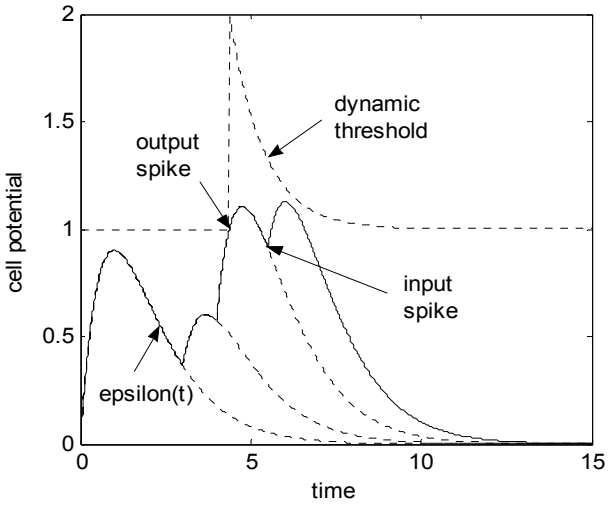

Fig 2. Plot of the cell potential (solid line) and the dynamic threshold (dashed line on top) versus time for the SRM model. Each "hump" is the additive effect of the synaptic input spike response on the cell potential.

SPIKE-LMS ALGORITHM

Following the derivation shown in [4] for the LMS algorithm, we define the cost function as the difference between the desired response and the output of the system. When the system output is a spike train, this cost function has many undesirable properties. For example, if we shift the ideal output spike train y(t) = d(t) by any $\varepsilon \neq 0$ to y(t – ε) = d(t), then the cost function jumps from zero to the maximum possible cost such that the time average of the two signals is the same. If the desired signal is non-periodic, then in effect there are only two values for the time average of the cost: 0 (when y(t) = d(t)) and twice the time average of the desired response squared. This cost function is not desirable for gradient descent because the gradient is zero everywhere except at point discontinuities where it is infinite. Nevertheless, we can use this definition to motivate an understanding of the Spike-LMS algorithm. Let the cost be

$$J = \left(d - y\right)^2. \tag{3}$$

Then

$$\frac{\partial J}{\partial w_i} = -2\left(d - y\right)\frac{\partial y}{\partial v}\frac{\partial v}{\partial w_i}, \tag{4}$$

and from definition of v, we have

$$\frac{\partial v}{\partial w_i} = \sum_{\forall s \in S_i} \varepsilon\left(t - t_s\right). \tag{5}$$

The other partial derivative in (2), $\partial y/\partial v$, is not easily defined because of the threshold firing behavior of y. However, we know that that the true expression of $\partial J/\partial w_i$ is not useful, and the other terms in this expression are rather simple, therefore $\partial y/\partial v$ must be the cause of the undesirable properties of $\partial J/\partial w_i$. Increasing v will generally shorten the time of the next firing of y, in effect "increasing" y, thus $\partial y/\partial v$ should be positive. We will replace $\partial y/\partial v$ with 1—our only justification being that the resulting algorithm works well. With this substitution, the gradient descent expression becomes:

$$\frac{\partial J}{\partial w_i} \approx -2\left(d - y\right)\sum_{\forall s \in S_i} \varepsilon\left(t - t_s\right) \tag{6}$$

The desired and output spike trains, d and y are trains of impulses, so $\partial J/\partial w_i$ can be expressed as:

$$\frac{\partial J}{\partial w_i} \approx \begin{cases} 0, & d(t) = y(t) \\ 2\sum_{\forall s \in S_i} \varepsilon\left(t - t_s\right), & y(t) = \delta(0) \\ -2\sum_{\forall s \in S_i} \varepsilon\left(t - t_s\right), & d(t) = \delta(0) \end{cases} \tag{7}$$

Finally, if we adapt the weights according to the steepest descent rule, we get the following algorithm we call the Spike-LMS algorithm (ξ is the step-size parameter):

1. When the output y fires update w_i so that
$$w_i \leftarrow w_i - \xi \sum_{\forall s \in S_i} \varepsilon\left(t - t_s\right) \tag{8}$$
2. When the desired response d fires, update w_i so that
$$w_i \leftarrow w_i + \xi \sum_{\forall s \in S_i} \varepsilon\left(t - t_s\right) \tag{9}$$

Note that if both d and y fire at the same time, the two updates cancel each other, which is what we expect.

Alternatively, the Spike-LMS algorithm can be thought out as a combination of Hebbian and anti-Hebbian learning. Hebbian learning is used between the pre-synaptic input spike trains $x_1 \ldots x_N$, and the desired spike train d. Anti-Hebbian learning is used between $x_1 \ldots x_N$, and the output spike train y. This is the same concept used in Widrow's LMS algorithm for linear adaptive systems [4]. In that algorithm, the weight update for each input is proportional to the product of the input and the error between the output and desired response:

$$\begin{aligned} \Delta w_i &= \eta x_i (d - y) \\ &= \eta x_i d - \eta x_i y \end{aligned} \tag{10}$$

The first term is a Hebbian-like update between the input and the desired response and the second term is an anti-Hebbian update between the input and the output [5].

Simulation Results

In the preceding section, we described an algorithm for updating the synaptic weights of a spiking neuron to perform supervised learning and system identification. In this section, we will show some simulations. To demonstrate the system, we use generated Poisson spike trains with a constant rate. Past work in neuroscience shows that spike trains produced by the brain can be roughly modeled as being Poisson distributed point processes [6].

To simulate the SRM neuron we chose $\varepsilon(t)$ to be the response of a linear second-order filter with a double real pole, i.e.

$$\varepsilon(t) = \frac{t}{\tau} \exp\left(-\frac{t}{\tau}\right). \tag{11}$$

We chose $\eta(t)$ to be a decaying exponential with the same time constant as $\varepsilon(t)$, i.e.

$$\eta(t) = \exp\left(-\frac{t}{\tau}\right), \tag{12}$$

and we chose $\gamma_0 = 1$.

These choices make the SRM behave equivalently to the second-order Integrate-and-Fire model. Additionally, these choices were easily implemented using an event driven simulator to model them. Using a discrete-time simulation would require much more computation and/or introduce quantization artifacts, depending on the resolution of the time slicing.

The simulated neural architecture was setup as shown in Fig. 1 with two neurons identical in everything except the synaptic weights. The weights for the neuron that produces the desired response were fixed, and the weights for the neuron that produces the output were varied. The inputs were applied to both spike trains simultaneously. After each spike produced by the desired neuron, the weights are increased according to (9), and after each spike produced by the output neuron, the weights are decreased according to (8). Gaussian jitter (σ = 1ms) was added to the desired spike train to test the system for noise robustness.

In the experiment shown in Figs. 3-5, we examine the performance of the Spike-LMS algorithm when used with Poisson input spike trains. The Poisson spike trains emitted spikes at an average rate of 100 Hz, 50 Hz and 25 Hz for inputs 1, 2 and 3, respectively. The weights of the synaptic connections for the neuron producing the desired response were fixed at 0.4, 0.2 and 0.5. Figs. 3 and 4 show the weight tracks of the LMS algorithm as learning takes place. The weight tracks take some time to converge, especially for the second input, which has the slowest firing rate.

The filter parameter τ is larger in Fig. 4 that in Fig. 3, which causes the convergence rate to be slower. In fact, the weight convergence time will be large if τ is too large or too small. As τ gets larger, the output fires more often because the input spikes have a better chance of adding up above the threshold before the spike response decays, and the rate of weight updates will increase proportionally. If τ is too small, then very few output spikes will be produced, thus increasing the convergence time. The reason for this is that for each output spike, there are more contributions from a larger number of input spikes. Thus, it is likely that an input spike regardless of synaptic weight will be adjusted up and down by the algorithm on desired spikes and output spikes, respectively. These noisy take longer to converge, even though there are more of them. On the other hand, if τ is much smaller than the spike rate, only spikes that occur very close in time will increase the cell potential past the threshold, and weights will take a long time to converge.

Fig. 5 shows the last twelve output spikes and the associated desired response for this experiment with τ = 10ms. This shows that once the weights are trained the output matches the desired response very closely.

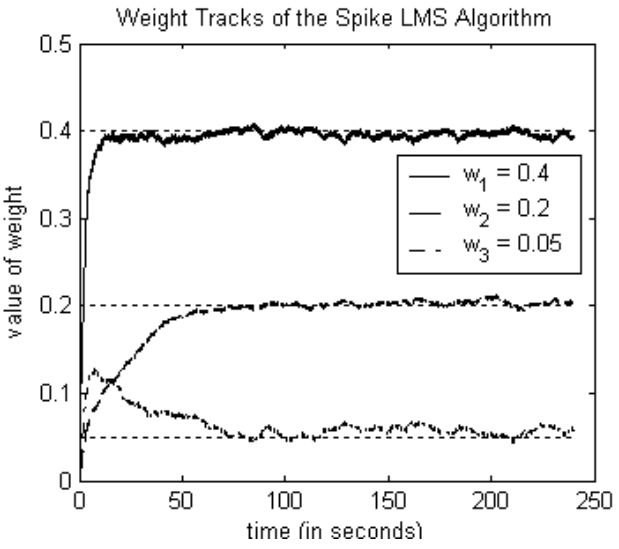

Fig 3. Weight tracks of the Spike LMS algorithm with τ = 10ms. Average spike rates are 100 Hz, 25 Hz and 50 Hz for inputs 1, 2 and 3, respectively. The step size η is 0.002.

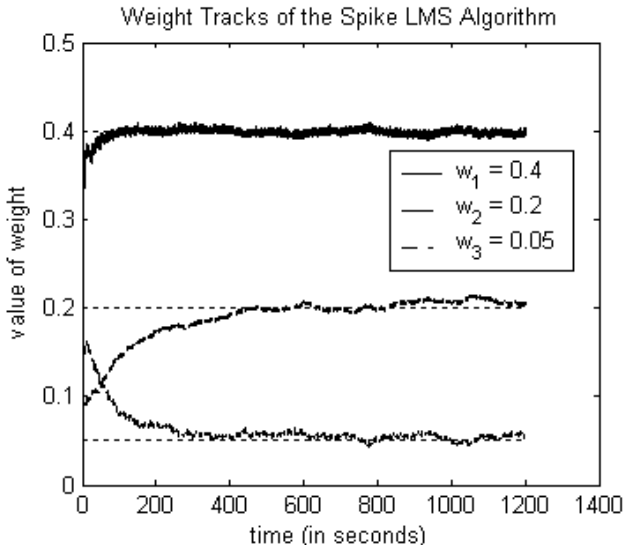

Fig 4. Weight tracks of the Spike LMS algorithm with τ = 50ms. Average spike rates are 100 Hz, 25 Hz and 50 Hz for inputs 1, 2 and 3, respectively. The step size ξ is 0.002.

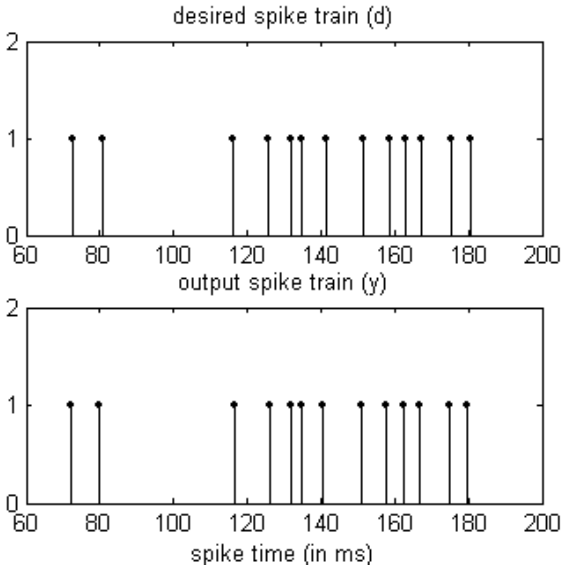

Fig. 5. Plot of the desired spike train (d) and the output spike train (y) after weight convergence of the last 12 spikes. Notice that the spike trains are almost identical. Here, τ = 10ms. The average absolute error between these sets of spikes is 0.4ms.

Conclusion

We have shown that the Spike-LMS algorithm is able to identify the synaptic weights of an SRM neuron given its inputs and outputs. This weight adaptation mechanism is not dependent on any neural encoding scheme applied to the spike trains, and assumes very little about the nature of the spike trains themselves, in contrast to previously proposed supervised learning methods. The algorithm presented here is simple, yet robust. Adaptive step-size techniques used in many other artificial neural network algorithms can also be easily applied to the Spike-LMS to improve convergence and the misadjustment of the adapted weight values.

We know from biology and from the power of human intelligence that spike trains are an extremely powerful and robust method of representing real-world data. Unfortunately, we still do not understand how this data is encoded into spike trains; it is likely that the number of encoding methods used in biology is large and each method is tailored to a specific adaptation. The power of the algorithm presented here is that it does not require knowledge of the encoding method, and should work wherever the SRM model is applicable.

The utility of this algorithm probably will not lie with its ability to learn static data by converting it into spike trains using some encoding scheme, although this is certainly possible. Rather it can be used as a tool to explore the possibilities of learning when using spiking neurons.

A specific application of this idea is in modeling small biological neural networks. If the spike trains of the biological network can be probed, this algorithm can be used to model biology by finding the synaptic weights. The weights can then be used in an SRM artificial neural network that mimics biology.

References

[1] Gerstner, W. "Spiking Neurons", in *Pulsed Neural Networks*, Maass, W. & Bishop, C.M. (Eds.). MIT Press, Mass. 2002.

[2] Bohte, S.M., Kok, J.N. & La Poutré, H, "Error-Backpropagation in Temporally Encoded Networks of Spiking Neurons", Neurocomputing, November 2002, 48(1-4), pp 17-37.

[3] Gerstner, W., "A Framework for Spiking Neuron Models: The Spike Response Model", in *The Handbook of Biological Physics*, Gielen, S. et al. (Eds.) Elsevier, Amsterdam. 1999.

[4] Haykin, S., *Adaptive Filter Theory*, fourth edition. Prentice Hall, NJ, 2002.

[5] Principe, J.C., Euliano, N.R. & Lefebvre, C.W., *Neural and Adaptive Systems: Fundamentals through Simulations*. John Wiley & Sons, Inc. 2000.

[6] Rieke, F. (1997) *Spikes: Exploring the Neural Code*. MIT Press, Mass.