Reference time in SpikeProp

Ioana Sporea and André Grüning

***Abstract*—Although some studies have been done on the learning algorithm for spiking neural networks SpikeProp, little has been mentioned about the required input bias neuron that sets the reference time start. This paper examines the importance of the reference time in neural networks based on temporal encoding. The findings refute previous assumptions about the reference start time.**

I. Introduction

While the traditional view of artificial neurons consists of representing an analog variable through the firing rate of a neuron [1][2], more recent studies suggests that biological neural systems use the exact time of single action potentials to encode information [3]. These findings have lead to a new way of simulating neural networks based on temporal encoding with single spikes [2]. Investigations of the computational power of spiking neurons have illustrated that realistic mathematical models of neurons can arbitrarily approximate any continuous function. Furthermore, it has been demonstrated that networks of spiking neurons are computationally more powerful than sigmoidal neurons [4].

The present paper explores the significance of the reference time in spiking neural networks trained with the supervised learning rule, SpikeProp, a generalization of back-propagation for spiking neurons [5].

II. SpikeProp

The network architecture consists of a feed forward network of spiking neurons, with each connection between two neurons having a number of *m* synaptic terminals, or sub-connections, with different delays, d^k, between the firing time of the presynaptic neuron and the time the postsynaptic potential starts to rise or to drop if the neuron is excitatory, or respectively inhibitory, with inhibitory neurons generating only negative postsynaptic potentials [5]. The neuron activity is described by the Spike Response Model [6], characterizing the neuron using a single variable, the membrane potential $x(t)$.

The emission of the action potential can be described by a threshold process as follows. An action potential will be fired if the membrane potential $x(t)$ reaches a threshold ϑ at a time $t^{(f)}$:

$$x\left(t^{(f)}\right) = \vartheta \text{ and } \frac{d}{dt}x\left(t^{(f)}\right) > 0$$

A single neuron j receiving input from a set of presynaptic neurons $i \in \Gamma_j$ is described by the variable $x_j(t)$. In this case, the effects of the incoming spikes are summarized and if x_j reaches the threshold ϑ an action potential is triggered. The membrane potential of one neuron *j* is thus described as follows:

$$x_j(t) = \sum_{i\in\Gamma_j}\sum_k w_{ij}^k\, y_i^k(t) = \sum_{i\in\Gamma_j}\sum_k w_{ij}^k\, \epsilon(t - t_i - d^k)$$

where w_{ij}^k is the synaptic efficacy. The sum runs over incoming spikes where Γ_j is the set of all presynaptic neurons of neuron *j*, and t_i is firing time of presynaptic neuron $i \in \Gamma_j$. The term w_{ij}^k represents the weight of the synaptic terminal k, having the delay d^k, between the neurons *j* and *i*.

The learning algorithm considers only the first spike of each neuron, thus the form of the after-potential and any other subsequent spikes are ignored. The above equation together with the threshold criterion defines the simplified spike response model [5].

The spike-response function $\epsilon(t)$ describes a standard postsynaptic potential and has the form:

$$\epsilon(t) = \begin{cases} \frac{t}{\tau}\exp\left(1 - \frac{t}{\tau}\right) & \text{if } t > 0 \\ 0 & \text{if } t \le 0 \end{cases}$$

with τ modelling the membrane potential decay time constant.

The learning method consists of explicitly evaluating the error gradient with respect to the weights of each synaptic terminal. The error-function is defined as the sum of squared difference between the target spike times, t_j^d, and the actual spike times, t_j^a:

$$E = \frac{1}{2}\sum_{j\in J}(t_j^a - t_j^d)^2.$$

The updating rule for each weight is calculated for each synaptic terminal:

$$\Delta w_{ij}^k = -\eta\frac{\partial E}{\partial w_{ij}^k}$$

where η is the learning rate, and w_{ij}^k is the weight of sub-connection k from neuron *i* to neuron *j*. The derivative can be expanded as:

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial t_j^a}\left(t_j^a\right)\frac{\partial t_j^a}{\partial x_j(t)}\left(t_j^a\right)\frac{\partial x_j(t)}{\partial w_{ij}^k}\left(t_j^a\right)$$

where t_j^a is the actual firing time of the postsynaptic neuron *j*. By the implicit function theorem the second term of the above equation is the negative inverse of the derivative $\partial x_j\left(t_j^a\right)\Big/\partial t_j^a$.

Manuscript received February 10, 2011.

I. Sporea and A. Grüning are with the Department of Computing, FEPS, University of Surrey, Guildford, Surrey GU2 7XH (e-mail: {i.nica, a.gruning}@surrey.ac.uk).

In order to keep the notation as in back-propagation, δ_j is defined for each neuron in the output and hidden layers as follows:

$$\delta_j = \frac{\partial E}{\partial t_j}\left(t_j^a\right)\frac{\partial t_j}{\partial x_j(t)}\left(t_j^a\right).$$

For neurons in the output layer, δ_j is:

$$\delta_j = -\frac{t_j^a - t_j^d}{\sum_{i\in\Gamma_j}\sum_k w_{ij}^k \frac{\partial y_i^k\left(t_j^a\right)}{\partial t_j^a}}$$

For neurons in the hidden layer, δ_j is:

$$\delta_j = -\frac{\sum_{i\in\Gamma^j}\delta_i \sum_k w_{ji}^k \frac{\partial y_j^k(t_i^a)}{\partial t_j^a}}{\sum_{i\in\Gamma_j}\sum_k w_{ij}^k \frac{\partial y_i^k(t_j^a)}{\partial t_j^a}}$$

where the set Γ^j represents the all postsynaptic neurons of the neuron j.

After computing all the terms δ_j according to the above equations, the weights are modified as follows:

$$\Delta w_{ij}^k = -\eta\frac{\partial E}{\partial w_{ij}^k} = -\eta y_i^k\left(t_j^a\right)\delta_j,$$

with η the learning rate.

Bohte et al. [5] presented a series of results that demonstrate the learning capabilities of SpikeProp. The SpikeProp algorithm is tested with the classic non-linear XOR problem. The XOR function is encoded in spike-time patterns by associating a 0 with a "late" firing time of 6 ms, and a 1 with an "early" firing time of 0 ms. The output is encoded in the same manner, with a "late" firing time of 16 ms and "early' firing time of 10 ms to represent a 0 and 1, respectively. The input layer of the network consists of an additional third neuron that would designate the reference start time and would fire always at time t = 0. The authors argued that without the additional input neuron the problem would become trivial (Section 4 in [5]). Bohte et al. [5] stated that a network with five hidden neurons, one of which being inhibitory, was able to learn the XOR problem within 250 cycles.

III. Reference Start Time

Although the gradient descent learning method has been tested with various benchmark problems [5], all other studies on SpikeProp ([7], [8], [9], [10], [11], [12]) used the XOR problem or the Iris dataset with identical network structures as in the original paper on SpikeProp [5]. This gives little insight into the capabilities or limitations of a spiking neural network trained with SpikeProp.

One such example is the topology of the spiking neural network designed to solve the XOR problem. It was assumed by Bohte et al. [5] that the network needs an additional ("bias") input neuron to designate the reference start time, otherwise the problem becomes trivial. Indeed, since the patterns consist of the firing times, the absolute time is irrelevant and two of the input patterns would be considered identical without the third input neuron that designates the reference start time – the patterns that have their input neurons fire at the same time.

Consider one such network for the XOR problem, with two input neurons, five hidden neurons one of which is inhibitory, and one output neuron with the same encoding of the patterns used so far by Bohte et al. [5]. For the input pattern $(t_1 = 0, t_2 = 0)$, the membrane potential of a hidden neuron j will be:

$$x_j(t) = \sum_k w_{1j}^k \epsilon(t - t_1 - d^k) + \sum_k w_{2j}^k \epsilon(t - t_2 - d^k)$$

with t_j^a the firing time, $x_j\left(t_j^a\right) = \vartheta$.

For the input pattern $(t_1' = 6, t_2' = 6)$, with $t_i' = t_i + \Delta t$, the membrane potential of the same hidden neuron will be:

$$x_j(s) = \sum_k w_{1j}^k \epsilon(s - t_1' - d^k) + \sum_k w_{2j}^k \epsilon(s - t_2' - d^k).$$

For $s = t + \Delta t$, the above equation becomes:

$$x_j(s) = \sum_k w_{1j}^k \epsilon(t + \Delta t - (t_1 + \Delta t) - d^k) + \sum_k w_{2j}^k \epsilon(t + \Delta t - (t_2 + \Delta t) - d^k)$$

This can be rewritten as follows:

$$x_j(s) = \sum_k w_{1j}^k \epsilon(t - t_0 - d^k) + \sum_k w_{2j}^k \epsilon(t - t_0 - d^k)$$

where $x_j(s) = x_j(t)$ with $s = t + \Delta t$.

Thus at the time $s = t + \Delta t$, the membrane potential has the same shape as the neuron in response to the first pattern at time t. Without the reference start time the membrane potential function $x_j(t)$ has no "knowledge" of absolute time; hence until it receives an input $t > 0$, the neuron is in a passive state. When an input current arrives, the membrane potential shape will shape according to the input, independent of the absolute time value. This conclusion is true for any set of weights w_{ij}^k, and is independent of the spike response function $\epsilon(t)$.

This can be generalized to all neurons in the hidden layer, and to the subsequent layer. To summarize, a spiking neural network of this type will respond to a pattern of time-spikes where all the input neurons fire at the same time t_0, with a spiking time $t_0 + \Delta T$. Since this is true for any set of weights, the problem is independent of the learning algorithm. Hence, by removing the "bias" input neuron, the XOR problem not only fails to become trivial, it becomes impossible to solve.

Consequently, when tested the XOR problem with two input neurons, when presented with input patterns with identical spike times, the network always responds with $t_0 + \Delta T$, where the value of ΔT depended on the set of weights. For all testing conditions described above, the network with only two input neurons was unable to learn the XOR function.

Moreover, when the same network structure was trained to solve the linear AND function, the learning algorithm was incapable to converge because two of its input patterns are

identical without a reference start time. Thus if a multilayer spiking neuron network without a reference time start cannot compute linear functions such as AND, which functions can it still solve?

TABLE I
ALL FUNCTIONS OF TWO VARIABLES

t_0	0 (1)	0 (1)	6 (0)	6 (0)
t_1	0 (1)	6 (0)	0 (1)	6 (0)
f_0	10 (1)	10 (1)	10 (1)	10 (1)
f_1	16 (0)	10 (1)	10 (1)	10 (1)
f_2	10 (1)	16 (0)	10 (1)	10 (1)
f_3	16 (0)	16 (0)	10 (1)	10 (1)
f_4	10 (1)	10 (1)	16 (0)	10 (1)
f_5	16 (0)	10 (1)	16 (0)	10 (1)
f_6	10 (1)	16 (0)	16 (0)	10 (1)
f_7	16 (0)	16 (0)	16 (0)	10 (1)
f_8	10 (1)	10 (1)	10 (1)	16 (0)
f_9	16 (0)	10 (1)	10 (1)	16 (0)
f_{10}	10 (1)	16 (0)	10 (1)	16 (0)
f_{11}	16 (0)	16 (0)	10 (1)	16 (0)
f_{12}	10 (1)	10 (1)	16 (0)	16 (0)
f_{13}	16 (0)	10 (1)	16 (0)	16 (0)
f_{14}	10 (1)	16 (0)	16 (0)	16 (0)
f_{15}	16 (0)	16 (0)	16 (0)	16 (0)

Table 1 shows all functions of two variables as an example. The values in the round brackets are their binary correspondents. A spiking neuron network of this sort can only compute those functions for which the response to the spike time pattern (0, 0) is 10, and the response to the spike time pattern (6, 6) is 16 – these are indeed trivial to compute for a network without a reference time start. There are only four functions, f_8 , f_{10}, f_{12}, f_{14} that can be computed in this way, as opposed to a single perceptron that can compute 14 of these functions with two inputs (see [13] for a graphical and analytical demonstration).

IV. CONCLUSION

Although it was assumed that the XOR problem would become trivial without an input neuron to designate the reference start time [5], simulations and demonstration proved otherwise. Without the bias input neuron, the problem becomes impossible to solve independent of the learning algorithm.

Moreover, linear problems such as the AND function, which can be solve by a single perceptron [14], also become impossible to solve. Although this limitation can be easily solved be adding a bias input neuron, other solutions can be found in different encodings. Such encodings may include multiple spikes, where one could designate the reference start time with an initial spike, without adding another input neuron. This would reduce the number of weights that need to be updated during learning. However, spiking neurons trained with SpikeProp are limited to only one spike. An extension of the SpikeProp learning rule [15] which allows multiple spikes in the input and hidden layer could be used for this purpose, though our simulations (not presented in the present paper) suggest that the algorithm is not stable enough to learn such input-output transformations.

REFERENCES

[1] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Transactions of the Society for Computer Simulation International*, Vol. 14 (4), pp. 1659 – 1671, 1997.

[2] W. Maass, "Fast sigmoidal networks via spiking neurons," *Neural Computation*, Vol. 9, pp. 279-304, 1997.

[3] S.T. Thorpe, M. Imbert, "Biological constrains on connectionist modeling," in, *Connectionism in perspective*, R. Pfeifer, Z. Schreter, F. Fogelman-Soulié , L. Steels, Eds. Elsevier, New York, 1989, pp. 63-92.

[4] W. Maass, "Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons," in *Advances in Neural Information Processing Systems 9*, M.C. Mozer, M.I. Jordan, T. Petsche, Eds. The MIT Press, Cambridge, MA, 1997, pp. 2 11-217.

[5] S. Bohte, J. Kok, H.L. Poutr'e, "Error backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, Vol. 48, pp. 17–37, 2002.

[6] W. Gerstner, "A Framework for Spiking Neuron Models: The Spike Response Model" in: *The Handbook of Biological Physics*, Vol.4 (12), F. Moss, S.Gielen, Eds. Elsevier Science, 2001, pp. 469-516.

[7] J. Xin, M. J. Embrechts "Supervised Learning with Spiking Neuron Networks," in *Proceedings of IEEE International Joint Conference Neural Networks, IJCNN01*, Washington D.C., 2001, pp. 1772–1777.

[8] S.C. Moore, "Back-Propagation in Spiking Neural Networks," M.Sc. thesis, University of Bath, 2002, available at: http://www.simonchristianmoore.co.uk.

[9] B. Schrauwen, J. Van Campenhout, "Improving Spike-Prop: Enhancements to an Error-Backpropagation Rule for Spiking Neural Networks," in *Proceedings of the 15-th ProRISC Workshop, Veldhoven*, the Netherlands, 2004.

[10] S. McKennoch, Dingding Liu, L.G. Bushnell, "Fast modifications of the SpikeProp algorithm," in *Proceedings of IEEE International Joint Conference on Neural Networks, IJCNN06*. Vancouver, Canada, 2006, pp. 3970 – 3977.

[11] M. Fujita, H. Takase, H. Kita, T. Hayashi, "Shape error surfaces in SpikeProp," in *Proceedings of IEEE International Joint Conference Neural Networks, IJCNN08,* Hong Kong, 2008, pp. 840-844.

[12] H. Takase, M. Fujita, H. Kawanaka, S. Tsuruoka, H. Kita, T. Hayashi, "Obstacle to training SpikeProp Networks – Cause of surges in training process –," in *Proceedings of IEEE International Joint Conference Neural Networks, IJCNN09*, Atlanta, USA, 2009, pp. 3062 – 3066.

[13] R. Rojas, "Neural Networks - A Systematic Introduction," Springer-Verlag, Berlin, 1996.

[14] M., Minsky, S. Papert, "Perceptrons: An Introduction to Computational Geometry," MIT Press, Cambridge, MA, 1969.

[15] O. Booij, H. tat Nguyen, "A gradient descent rule for spiking neurons emitting multiple spikes," *Information Processing Letters*, Vol. 95(6), pp. 552 – 558, 2005.