International Journal of Engineering & Technology, 7 (4.29) (2018) 173-177

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper



Spike Response Function Weight and Delay Updating Strategy Using Delay Rules

Abdullah H. Almasri[1*], Shahnorbanun Sahran[2], Eiad Yafi[3]

[1]*Faculty of Business Technology and Accounting, Unitar International University, 47301, Petaling Jaya, Selangor Darul Ehsan, Malaysia*

[2]*Faculty of Information Science and Technology, Center for Artificial Intelligence, Universiti Kebangsaan Malaysia, 43600, Bangi, Selangor Darul Ehsan, Malaysia*

[3]*University Kuala Lumpur, Malaysian Institute of Information Technology, 50250, Kuala Lumpur, Malaysia*

**Corresponding author E-mail: abdullah@unitar.my*

Abstract

Spike Response Function (SRF) plays an important role in the temporal coding Spiking Neural Network (SNN) as it has a significant role to determine when the neuron should fire. This paper studies the important role of the SRF in the SNN learning stability. It proposes a novel method to find out the rules to update delay for each class to make SRF stable, and then using these rules to update delay and weight simultaneously at the SNN learning rule. This method updates the delay depending on the local result to make SRF stable. The main issue of this paper is to put forward the idea that weight and delay parameters could and need to be updated simultaneously to make both SRF and SNN stable during the learning process. The delay rules strategy which have been found could be used for pattern recognition application which use SNN. The limitation of this work is that; getting the updating delay rules depends on a sample data from each class and the way of selecting the rules.

Keywords: *Spiking Neural Network; Spike Response Function; Weight; Delay; Classification; Pattern Recognition.*

1. Introduction

Spiking Neural Network (SNN) is the third generation of neural networks; it is computationally more powerful than the previous two generations of neural network models [1]. Experimental evidence has accumulated during the last few years, something that indicates that many biological neural systems use the timing of single action potentials ("spikes") to encode information [2]. These experimental results from neurobiology have led to the investigation of the third generation of neural network models which employ spiking neurons (or "integrate-and-fire neurons") as computational units [1]. Maass [1] has analyzed the computational power of networks of spiking neurons with regard to temporal coding with single spikes. It turns out that this computational model has at least the same computational power as neural nets of the first two generations with similar size. These mathematical models for spiking neurons do not provide a complete description of the extremely complex computational function of a biological neuron. Rather, like the computational units of the previous two generations of neural network models, these are simplified models that.

focus on just a few aspects of biological neurons. However, in comparison with the previous two models, they are substantially more realistic. In particular, they describe much better the actual output of a biological neuron. Hence this allows the researcher to investigate on a theoretical level the possibilities of using time as a resource for a computation and communication [1].

However, a mathematically rigorous analysis of the computational power of networks of spiking neurons has so far been missing [1]. Maass [1] believes that such analysis will be helpful in understanding the organization of computations in complex biological neural systems. Spiking neuron networks have turned out to be very powerful [1], but there is still not much known about possible learning and higher computational mechanisms [3].

The remaining body of this paper consists of four sections. Section (2) shows the related works. Section (3) shows the importance of Spike Response Function stability. Section (4) explains the method of finding the delay rules to update delay and weight simultaneously. Finally, the discussion is given in section (5).

2. Related Works

The algorithms which have been proposed by [4-16] alters the weight for learning, while [3,7,12,17] alters the delay. Awadalla & Abdellatif Sadek [18] method depends on four steps, each one depends on the previous. Their steps are: synaptic weights update, synaptic delay update, synaptic time constant update and neuron threshold update. Their algorithm encounters a high computational and time cost due to their strategy.

Updating weight and delay simultaneously was an important issue which appeared in the literature as a missing solution for it [3-20], to the best of the author's knowledge none of them has found a way to do so.

3. Spike Response Function Stability

The spike response model (SRM) is a general leaky-integrate-and-fire model. The leaky-integrate-and-fire model describes the biophysical mechanisms of the neuron mostly by means of its mem-

Copyright © 2018 Authors. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

brane potential [21]. Furthermore, this model gives great importance to the time lap taken from the last firing event. The model describes the state of a neuron j at time t by the state variable $\mu_j(t)$.

Suppose that the spike response function $\varepsilon(st)$ which describes the internal state of neuron [21] is given by the equation (1) and described in Figure 1:

$$\varepsilon(st) = \frac{st}{a} * e^{1-\frac{st}{a}};$$
$$st = t - (input + delay)$$
$$\text{where } a > 0 \tag{1}$$

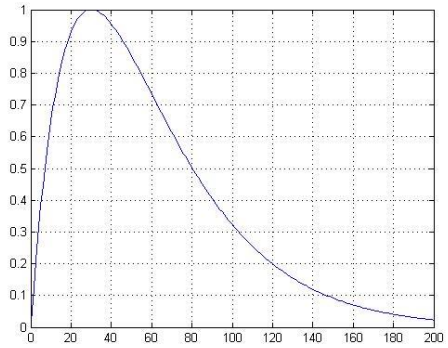


Fig. 1: Spike Response Function

For a spiking neuron j the potential at time t is defined by the equation (2):

$$u_j(t) = \sum_{i \in \Gamma_j}^{n} w_{ji} * \varepsilon_{ji}(t - t_i - d_{ji}) \tag{2}$$

By looking at equation (2), SRF and weight plays a main role in determining the output value which determines which neuron will fire. The *st* parameter which passes to the function needs to be studied in such a way to make it stable during the learning process. Three parameters pass through $\varepsilon(st)$ which are (t, input and delay) as appeared in equation (1). The first two are stable which means that the delay parameter which passed as a parameter in the SRF plays a hiding role in the learning and needs to be studied to make it stable during learning process.

After presenting the input pattern to the SNN, the neuron which has first been fired is found; the weight will be updated using learning algorithm technique to update weight whether to decrease it or to increase it depends on some condition.

Now, the question is: if the network feeds by the same inputs after updating the weights, will the same winner neuron fire again first; especially if it was in the correct class?

To clarify this idea, this scenario is studied: suppose that the neuron which has first been fired belongs to the correct class, which means if the network feeds with the same inputs after updating the weight using the learning rules used, the question is: will the same neuron fire again first which is in the correct class? There is no guarantee that if the network feeds with the same input after updating the weight using learning rules, the same winner neuron in the correct class will be fired again first; and there is no SNN learning algorithm guaranteeing so.

However, there is something interesting in the SNN which needs to be studied well, that is, updating the weight will normally let the internal state of the neuron i.e. SRF $\varepsilon(st)$ change, and that will lead the winning time of the neuron change dramatically, which means that the winning time changed in unexpected and unpredictable manner because of $\varepsilon(st)$ behaviour.

Therefore, the objective of this paper is to study how the $\varepsilon(st)$ could be stable after updating the weight, verify the role of $\varepsilon(st)$ in the learning process, and reach to the stage to update delay and weight simultaneously at the learning rule. To do so, the rules of updating the delay parameter need to be obtained, then discover the relation between the class type and the behaviour of $\varepsilon(st)$, and put up the rules for updating the delay at the learning rule. So by this way, the weight and delay parameters would be updated simultaneously at the learning rule and that is the main objective in this paper. With this finding, more research is needed on updating weight and delay simultaneously in SNN learning algorithm.

4. Identifying Delay Rules Proposed Method

For easy analysis and implementation, a simple method has been used, n records have randomly been selected from each class type as a sample data, a tracking number for updating delay rule is used for each class type to select the most frequent updating rule repeated during the learning process for each class type.

If the neuron fires at $st_{winnertime} = st_{wt}$, when the network feeds with the same input after updating the weight, the neuron could fire at $(st_{wt}, st_{wt} - st_{step} \text{ or } st_{wt} + st_{step}) \xrightarrow{st_{step}=1}$ $(st_{wt}, st_{wt} - 1 \text{ or } st_{wt} + 1)$, one step back and forward st_{wt} is used here to choose the delay rule for ease of implementation, as after updating the weight and feed the SNN with the same record, the neuron is expected to not fire far away from the previous and subsequent stage, and it is almost enough to show that the delay needs to be updated to make SRF stable. Delay rule is the rule to update delay at the learning rule through finding out the maximum times $(st_{wt}, st_{wt} - 1 \text{ or } st_{wt} + 1)$ appears using n records selected for each class.

4.1. Steps to make SRF and though SNN learning stable

Five steps to make SRF and though SNN learning stable are presented in Figure 2, and discussed in detail as follows:

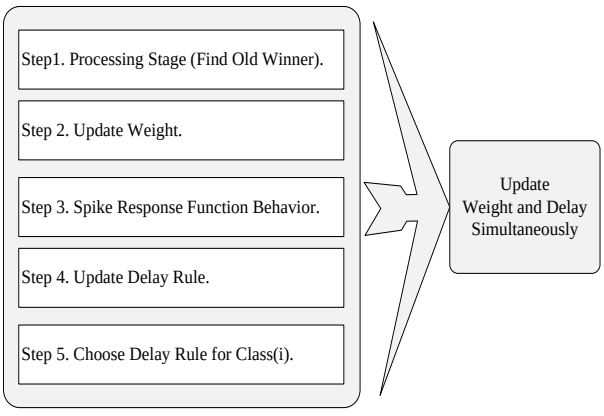


Fig. 2: General site-specific proposed to make SRF stable

4.1.1. Processing stage (finding old winner $winner_{old}$)

After selecting n records randomly from each class as a sample data, one by one record feeds to the SNN to find out the winning neuron during the learning cycle for Class i; $Input \in [tc_{\min}, tc_{\max}]$, $Delay \in [d_{\min}, d_{\max}]$. Where $d_{\min}$ and $d_{\max}$ refer to the minimum and maximum value of delay, $tc_{\min}$ and $tc_{\max}$ refer to the minimum and maximum value of the temporal coding assigned experimentally.

After finding the winning neuron, the spike time of the winning neuron st_{wt} is kept as a reference for future use. The pseudo code for this stage is in Figure 3.

Step 1:
Present a training input pattern from Class i to the SNN.
Step 2:
FOR each t DO
Update the synapse potential.
Update the output.
WHILE $t < timewindow$
Step 3: Find winner (call it $winner_{old}$).

Fig. 3: Pseudo code for processing stage

4.1.2. Update weight

Updating the weights is performed by using a learning algorithm technique for a classification task. There is no specific learning algorithm that has been selected from the literature to apply this technique, as the objective is to discuss the importance of SRF and the proposed technique in order to determine the delay rules that will aid in reaching the point in which the SNN learning algorithm is able to simultaneously update the weight and delay values.

4.1.3. Spike response function behavior

After updating the weights, the SNN again feeds back with the same record, which has been used before to study the $\varepsilon(st)$ behavior whether it is static or dynamic through seeing whether the same neuron, which has been won before $winner_{old}$, wins again $winner_{new}$. Here are two cases that need to be addressed: The first one is that the $winner_{old}$ is in the correct class; and the second case is if the $winner_{old}$ is in the incorrect class, the pseudo code for the two cases are in Figure 4.

In the first case, two cases need to be studied: the first one is if the $winner_{new}$ is the same as $winner_{old}$, here $\varepsilon(st)$ is stable; so it just needs to update the delay rule track number that no change on the delay rule has been done in this case and then return to the learning algorithm with new input data. The second one is if the $winner_{new}$ is not the same as the $winner_{old}$; here $\varepsilon(st)$ needs to be stabilized. The winning neuron needs to be found at $st_{wt}-1$ and $st_{wt}+1$ in one step back and one step forward only; for ease of analysis and implementation.

So, three cases need to be addressed and studied: the first one is finding the winner at $st_{wt}-1$, if $winner_{old}$ is the same as the $winner_{new}$, that is, the $\varepsilon(st)$ needs to be stabilized by updating the delay rule, and then updating the track number whose rule needs to be updated. The second is to find the winner at $st_{wt}+1$, if the $winner_{old}$ is same as the $winner_{new}$, that is, the $\varepsilon(st)$ needs to be stabilized by updating the delay rule, and then updating track number whose rule needs to be updated. The third case is if the winning neuron at $st_{wt}-1$ and $st_{wt}+1$ is not the same as the $winner_{old}$, that is, if the winning neuron could be at $st_{wt}-n$ or $st_{wt}+n$ where $n=2,3,...,st_{\max}$, nothing is to be done in this case as mentioned earlier in this paper that one step back and one step forward has been taken for ease of analysis and implementation.

In the second case, two cases need to be studied: the first one is if the $winner_{new}$ is same as $winner_{old}$; here $\varepsilon(st)$ needs to be stabilized. The winning neuron needs to be found at $st_{wt}-1$ and $st_{wt}+1$ in only one step back and one step forward for ease of analysis and implementation as follows; three cases need to be addressed and studied: the first one is to find the winner at $st_{wt}-1$, if the $winner_{new}$ is in the correct class, that is, the $\varepsilon(st)$ needs to be stabilized by updating the delay, so it needs to update the delay rule track. The second one is to find the winner at $st_{wt}+1$, if the $winner_{new}$ is in the correct class, that is, the $\varepsilon(st)$ needs to be stabilized by updating the delay, so it needs to update the delay rule track. The third one, if the winning neuron is at $st_{wt}-1$ and $st_{wt}+1$ not in the correct class, that is, if the winning neuron could be at $st_{wt}-n$ or $st_{wt}+n$ where $n=2,3,...,st_{\max}$, in this case nothing is to be done in this method. The second case is if the $winner_{new}$ is not the same as the $winner_{old}$: If $winner_{new}$ is in the correct class, so it just needs to update the delay rule track, or else return to the learning algorithm with new input data

A. If $winner_{old}$ is in the correct class:
a. $winner_{old} = winner_{new}$
Go to update delay rule (section 4.4).
b. $winner_{old} \neq winner_{new}$ ($\varepsilon(st)$ dynamic)
Case (1):
Update the synapse potential at st_{wt} *-1*
Update the output at st_{wt} *-1*
Find winner at st_{wt} *-1*
If $winner_{old} = winner_{new}$ Go to update delay rule (section 4.4).
Case (2):
Update the synapse potential at st_{wt} *+1*
Update the output at st_{wt} *+1*
Find winner at st_{wt} *+1*
If $winner_{old} = winner_{new}$ Go to update delay rule (section 4.4).
Case (3):
If neither Case (1) nor Case (2) then return to the learning algorithm with new input data from Class *i*.

B. If $winner_{old}$ is in the incorrect class:
a. $winner_{old} = winner_{new}$
Case (1):
Update the synapse potential at st_{wt} *-1*
Update the output at st_{wt} *-1*
Find winner at st_{wt} *-1*
If $winner_{new}$ is in the correct class Go to update delay rule (section 4.4).
Case (2):
Update the synapse potential at st_{wt} *+1*
Update the output at st_{wt} *+1*
Find winner at st_{wt} *+1*
If $winner_{new}$ is in the correct class Go update delay rule (section 4.4).
Case (3):
If neither Case (1) nor Case (2) then return to the learning algorithm with new input data from Class *i*.
b. $winner_{old} \neq winner_{new}$
If $winner_{new}$ is in the correct class go to update delay rule (section 4.4), else return to the learning algorithm with new input data from Class *i*.

Fig. 4: Pseudo code proposed for SRF behavior

.

4.1.4. Update delay rule

To stabilize the SRF, only one of three delay rules will be updated; these are $dly_{new_1} = dly_{old}$, $dly_{new_2} = dly_{old} - 1$ or $dly_{new_3} = dly_{old} + 1$ where dly_{new_1}, dly_{new_2} and dly_{new_3} represent the track number for how many times the delay rule repeated during the learning cycle; and here three cases will be studied as shown in Figure 5.

Step1: Tracking delay rules parameter: $\begin{cases} dly_{new_1} = dly_{old} \\ dly_{new_2} = dly_{old} - 1 \\ dly_{new_3} = dly_{old} + 1 \end{cases}$

Step 2:
IF winner fires at
st_{wt} $\{dly_{new_1}$ ++
ELSE IF winner fires at
st_{wt} - 1 $\begin{cases} \text{if } st_{wt} > \varepsilon(st_{wt})_{max};\ \varepsilon(st_{wt}) & \Rightarrow dly_{new_2} ++ \\ \text{if } st_{wt} \le \varepsilon(st_{wt})_{max};\ \varepsilon(st_{wt}) & \Rightarrow dly_{new_3} ++ \end{cases}$
ELSE IF winner fires at
$st_{wt} + 1$ $\begin{cases} \text{if } st_{wt} > \varepsilon(st_{wt})_{max};\ \varepsilon(st_{wt}) & \Rightarrow dly_{new_3} ++ \\ \text{if } st_{wt} \le \varepsilon(st_{wt})_{max};\ \varepsilon(st_{wt}) & \Rightarrow dly_{new_2} ++ \end{cases}$

Step 3: Then return to the learning algorithm with new input data from Class *i*.

Fig. 5: Pseudo code proposed for updating delay rule

The first case is if the $winner_{new}$ neuron fires at st_{wt} , the delay rule dly_{new_1} will remain the same. The second case is if the $winner_{new}$ neuron fires at $st_{wt} - 1$, the delay will be updated as follows; if the st_{wt} is greater than $\varepsilon(st)_{\max}$, the delay will be decreased one step, or else it will be increased one step. The third case is if the $winner_{new}$ neuron fires at $st_{wt} + 1$, the delay will be updated as follows; if the st_{wt} is greater than $\varepsilon(st)_{\max}$, the delay will increase one step, or else will decrease one step. The delay change range would be $[dly - n * st_{step}, dly + n * st_{step}]$.

4.1.5. Choosing delay rule for class i

Finally, choosing the delay rule for each *Class i* is carried out using equation (3) as follows:

$$Class(i) = SelectRule(Max(dly_{new1}, dly_{new2}, dly_{new3})) \quad (3)$$

The maximum value within dly_{new_1}, dly_{new_2} and dly_{new_3} is selected and employed in the learning rule to update the delay for class i. Next, it returns to the processing stage with another class to determine its delay rule. In other words, after the learning cycle is complete for all the records that have been selected for class *i* (e.g., if $n = 10$ records have been selected from each class, and three delay rules could be applied (dly_{new_1}, dly_{new_2} or dly_{new_3})); if the delay rule dly_{new_1} is repeated *3* times, *2* times for rule dly_{new_2}, and *5* times for rule dly_{new_3}; Rule dly_{new_3} will be selected to reflect the behavior of $\varepsilon(st)$ for class i, as the track number is the maximum. (i.e.) for class i, the delay rule will be dly_{new_3}. Further investigation is required to select n to tackle the situation where the repeated times for two rules are equal.

4.2. Update weight and delay simultaneously

After finding out all delay rule for each class, these rules will be used for updating weight and delay simultaneously at the learning rule. The learning rule will be as shown in Figure 6.

A. Update weight (as used in the learning algorithm rule for classification).
B. Update delay.
IF the winner belongs to:
Case 1: Apply the reflecting updating delay rule for class 1.
Case 2: Apply the reflecting updating delay rule for class 2.
…
Case N_{class} : Apply the reflecting updating delay rule for class N_{class}
END IF.

Fig. 6: The basic steps proposed of updating weight and delay

Updating delay does not depend on the state if the winner in the correct or incorrect class. The delay rule is founded for both cases when the neuron in the correct or incorrect class (Refer to 4.3). So the updating delay rules will be as in step 2.

5. Discussion

Two main issues at this paper have been studied. Firstly, the hiding role of SRF during the learning process and the need to make it stable during the learning process. Secondly, the need to update weight and delay simultaneously at the main learning rule to make SNN learning rule stable.
This paper proves logically that SRF plays an important role in the temporal coding SNN as it has a significant role to determine when the neuron should fire. An SNN learning stability is an important issue that needs to be improved by finding out the parameters which play an important role during the learning process without taking care of assigning those parameters carefully, as the role of those parameters is still a going debate.

6. Conclusion

SRF stability guides this research study to come to the point where weight and delay would be updated simultaneously at the learning rule to make SNN learning stable as much as possible, which is the main objective of this paper.

References

[1] Maass, W., Networks of spiking neurons: the third generation of neural network models, Neural Networks, Vol 10, No. 9, pp. 1659-1671, 1997
[2] Hopfield, J., Pattern recognition computation using action potential timing for stimulus representation, Nature, Vol 376, No. 6535, pp. 33-36, 1995.
[3] Natschläger, T. & B. Ruf 1998. Spatial and temporal pattern analysis via spiking neurons. Network: Computation in Neural Systems 9(3): 319-332.
[4] Bohte, S., H. La Poutré & J. Kok 2002. Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer RBF networks. IEEE Transactions on Neural Networks 13(2): 426-435.
[5] Bohte, S. M., J. N. Kok & H. La Poutré 2002. Error-backpropagation in temporally encoded networks of spiking neurons. Neurocomputing 48(1–4): 17-37.
[6] Bohte, S. M., J. N. Kok & H. La Poutre 2000. Unsupervised classification of complex clusters in networks of spiking neurons. International Joint Conference on Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS. 3 pp. 279-284 vol.3.

[7] Charles, E. Y. A. 2006. Supervised and Unsupervised Weight and Delay Adaptation earning in temporal Coding Spiking Neural Networks. Thesis Doctor of Philosophy Cardiff, Cardiff.
[8] Ghosh-Dastidar, S. & H. Adeli 2007. Improved spiking neural networks for EEG classification and epilepsy and seizure detection. Integrated Computer-Aided Engineering 14(3): 187-212.
[9] Ghosh-Dastidar, S. & H. Adeli 2009. A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection. Neural Networks 22(10): 1419-1431.
[10] Grüning, A. & I. Sporea 2012. Supervised Learning of Logical Operations in Layered Spiking Neural Networks with Spike Train Encoding. Neural Processing Letters 36(2): 117-134.
[11] Jianguo, X. & M. J. Embrechts 2001. Supervised learning with spiking neural networks. International Joint Conference on Neural Networks, 2001. Proceedings. IJCNN '01. pp. 1772-1777 vol.3.
[12] Ruf, B. & M. Schmitt 1998. Self-organization of spiking neurons using action potential timing. IEEE Transactions on Neural Networks 9(3): 575-578.
[13] Sahran, S. 2007. Application of Spiking Neural Networks and the Bees Algorithm to Control Chart Pattern Recognition.Thesis Doctor of Philosophy Cardiff, Cardiff.
[14] Sporea, I. & A. Grüning 2012. Supervised Learning in Multilayer Spiking Neural Networks. Neural Computation 25(2): 473-509.
[15] Zhang, C.-w. & H.-j. Liu 2009. A New Supervised Spiking Neural Network. Second International Conference on Intelligent Computation Technology and Automation, 2009. ICICTA '09. 1 pp. 23-26.
[16] Kasabov ,N.K.(2014).NeuCube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data. Neural Networks , 52,62–76.
[17] Tao, X. & H. Michel 2004. Data clustering via spiking neural networks through spike timing-dependent plasticity. 1 pp. 168–173.
[18] Awadalla, M. H. A. & M. Abdellatif Sadek 2012. Spiking neural network-based control chart pattern recognition. Alexandria Engineering Journal 51(1): 27-35.
[19] QingXiang, W., D. Bell, Q. Guilin & C. Jianyong 2006. Knowledge Representation and Learning Mechanism Based on Networks of Spiking Neurons. Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on. 4 pp. 2796-2801.
[20] Abdullah H. Almasri and Shahnorbanun Sahran, 2014. Time Window, Spike Time and Threshold Boundary for Spiking Neural Network Applications. Journal of Applied Sciences, 14: 317-324.
[21] Wolfgang Maass & C. M. Bishop. 2001. Pulsed Neural Networks Ed. 1st Edition. Cambridge, Massachusetts: The MIT Press.