Supervised learning with spiking neural networks

Conference Paper · February 2001

DOI: 10.1109/IJCNN.2001.938430 · Source: IEEE Xplore

M. Embrechts
Rensselaer Polytechnic Institute
188 PUBLICATIONS **3,499** CITATIONS

Supervised Learning with Spiking Neural Networks

Jianguo Xin[1] and Mark J. Embrechts[2]
xinj@rpi.edu, Embrem@rpi.edu
Mechanical Engineering, Aeronautical Engineering, & Mechanics[1]
Decision Sciences and Engineering Systems[2]
Rensselaer Polytechnic Institute, 110 8th St., Troy, NY 12180, USA

Abstract

In this paper we derive a supervised learning algorithm for a spiking neural network which encodes information in the timing of spike trains. This algorithm is similar to the classical error back propagation algorithm for sigmoidal neural network but the learning parameter is adaptively changed. The algorithm is applied to the complex nonlinear classification problem and the results show that the spiking neural network is capable of performing nonlinearly separable classification tasks. Several issues concerning the spiking neural network are discussed.

1. Introduction

Since the spiking nature of biological neurons has been studied extensively, the computational power associated with temporal information coding in single spike has also been explored [3,4,6,7,9]. In fact spiking neural networks can simulate arbitrary feed-forward sigmoidal neural network and approximate any continuous function. This has been proven in [10]. And furthermore, theoretically it has been shown that spiking neural networks, where coding information resides in individual spike trains, are computationally more powerful than neural network with sigmoidal activation functions [8].

For practical applications of temporally encoded spiking neural network, in this study we derive an error back propagation based supervised learning algorithm for spiking neural networks. The methodology is somewhat similar to the derivation by Rumelhart et al. [12]. The threshold function is approximated linearly to get around of the discontinuous nature of spiking neurons. The algorithm is tested against the complex nonlinear classification problem using the Iris data set in spiking neural networks. The results show that our algorithm is applicable to practical nonlinear classification tasks.

2. Network Architecture

Figure 1 shows the network architecture. For our purpose, we use the same as in the paper by Natschlager & Ruf [11]. It consists of a feed-forward network of spiking neurons with multiple delayed synaptic terminals. The neurons in the network generate action potentials (spikes) when the internal neuron state variable (termed as 'membrane potential') crosses a threshold. In [5] Gerstner introduced the Spike Response Model (SRM) to describe the relationship between input spikes and the internal state variable. This model can be adapted to reflect the dynamics of different spiking neurons if proper spike response functions are selected.

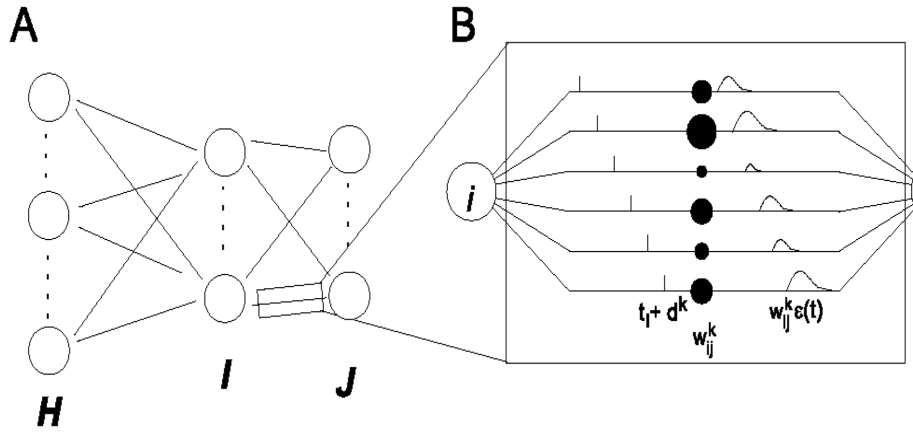

Figure 1. A) Feed-forward spiking neural network B) Connection consisting of multiple delayed synaptic terminals

Consider a neuron j, having a set D_j of immediate pre-synaptic neurons, receives a set of spikes with firing times t_i, $i \in D_j$. It is assumed that any neuron can generate at most one spike during the simulation interval, and discharges when the internal state variable reaches a threshold. The dynamics of the internal state variable $x_j(t)$ are described by the following function:

$$x_j(t) = \sum_{i \in D_j} w_{ij} \, \varepsilon(t - t_i) \qquad (1)$$

Where ε (t) is the spike response function and w_{ij} is the weight describing the synaptic efficacy.

The spike response function ε (t) is given by:

$$\varepsilon(t) = \frac{t}{\tau} e^{1-t/\tau} \qquad (2)$$

Where τ is the membrane potential decay time constant, which determines the rise- and the decay-time of the postsynaptic potential (PSP). Also it is assumed that $\varepsilon(t) = 0$ for $t < 0$.

The individual connection, which is described in the network in [11], consists of a fixed number of m synaptic terminals, where each terminal serves as a sub-connection that is associated with a different delay and weight (cf. figure 1B). The delay d^k of a synaptic terminal k is defined as the difference between the firing time of the pre-synaptic neuron and the time when the post-synaptic potential starts rising (cf. figure 2A, which is in [2]). The un-weighted contribution of a single synaptic terminal to the state variable is introduced in [2] to described a pre-synaptic spike at a synaptic terminal k as a PSP of standard height with delay d^k:

$$y_i^k = \varepsilon(t - t_i - d^k) \qquad (3)$$

Where the time t_i is the firing time of pre-synaptic neuron i, and d^k the delay associated with the synaptic terminal k.

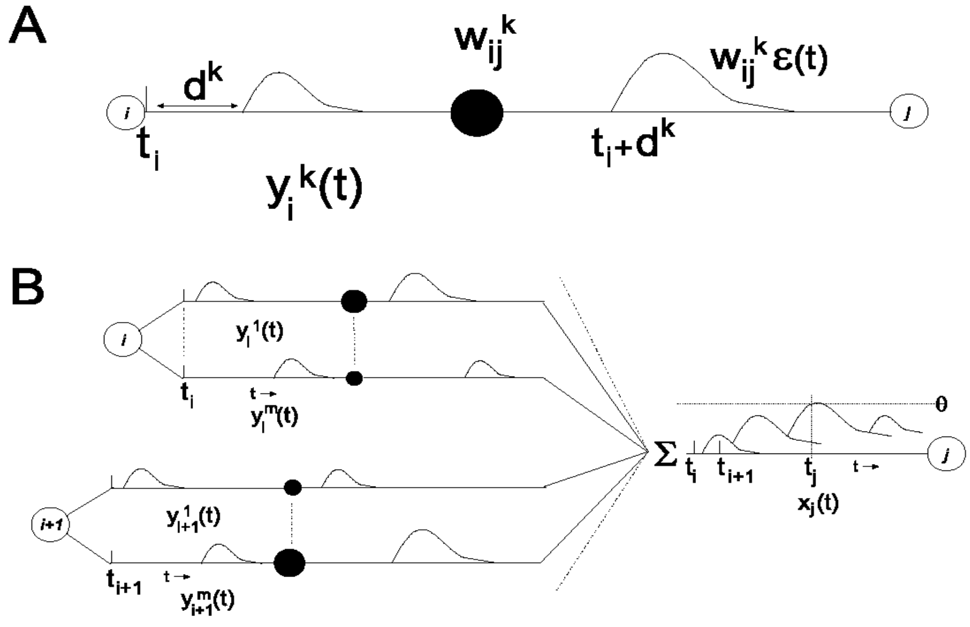

Figure 2. Connections in a feed-forward spiking neuron network: A. Single synaptic terminal: the delayed pre-synaptic potential is weighted by the synaptic efficacy to obtain the post-synaptic potential. B. Two multi-synapse connections: weighted input is summed at the target neuron.

Now consider the multiple synapses per connection. From equation (1) and combining equation (2), the state variable x_j of neuron j receiving input from all neurons i is then described as the weighted sum of the pre-synaptic contributions:

$$x_j(t) = \sum_{i \in D_j} \sum_{k=1}^{m} w_{ij}^k y_i^k(t) \qquad (4)$$

Where w_{ij}^k is the weight associated with synaptic terminal k (cf. figure 2B). The firing time t_j of neuron j is defined as the first time when the state variable crosses the threshold θ : $x_j(t) \geq \theta$. The threshold θ is a constant and fixed for all neurons in the network.

3. Error Back Propagation Algorithm

Consider a fully connected feed-forward neural network with layers identified as *I (Input)*, *H (Hidden)* and *O (Output)*. We will derive an algorithm based on error back propagation. It should be noted that the derived algorithm can be applied to networks with more than one hidden layer as well.

The learning scheme is teacher forcing, that is to learn a set of target firing times, denoted as $\{t_o^t\}$, at the output neurons $o \in O$ for a given set of input patterns $\{P[t_1...t_i]\}$, where $P[t_1...t_i]$ defines a single input pattern described by single spike train for each neuron $i \in I$. The objective function to be optimized is the least squares error function:

$$E = \frac{1}{2} \sum_{o \in O} (t_o^a - t_o^t)^2 \qquad (5)$$

Where $\{t_o^t\}$ and $\{t_o^a\}$ are the target spike train and spiking neural network actual output spike train, respectively.

To minimize the objective function in equation (5), we need to optimize the weight w_{ho} between each separate connection k of the synaptic terminal, e.g. w_{ho}^k. Therefore, for the back propagation rule, the weight correction Δw_{ho}^k needs to be calculated:

$$\Delta w_{ho}^k = -\eta \frac{\partial E}{\partial w_{ho}^k} \qquad (6)$$

Where η is the learning parameter and w_{ho}^k is the weight of connection k from neuron h to neuron o. The derivative on the right hand of equation (6) can be expressed as follows using the chain rule and considering the fact that t_o is a function of x_o, which depends on the weights w_{ho}^k:

$$\frac{\partial E}{\partial w_{ho}^k} = \frac{\partial E}{\partial t_o}(t_o^a)\, \frac{\partial t_o}{\partial w_{ho}^k}(t_o^a) = \frac{\partial E}{\partial t_o}(t_o^a)\, \frac{\partial t_o}{\partial x_o(t)}(t_o^a)\, \frac{\partial x_o(t)}{\partial w_{ho}^k}(t_o^a) \qquad (7)$$

In the above expression we make an assumption just as in [2]: for a small enough region around $t = t_o^a$, the function x_o can be approximated by a linear function of t and thus we express t_o as a function of the threshold post-synaptic input $x_o(t)$ around $t = t_o^a$. In such a small region, the threshold function can be approximated as $\delta t_o(x_o) = -\delta x_o(t_o)/\lambda$. $\frac{\partial t_o}{\partial x_o(t)}$ is the derivative of the inverse function of $x_o(t)$ and λ is the local derivative of $x_o(t)$ with respect to t, e.g. $\lambda = \frac{\partial x_o(t)}{\partial t}(t_o^a)$.

$$\frac{\partial t_o}{\partial x_o(t)}(t_o^a) = \frac{\partial t_o(x_o)}{\partial x_o(t)}\Big|_{x_o=\theta} = \frac{-1}{\lambda} = \frac{-1}{\frac{\partial x_o(t)}{\partial t}(t_o^a)} = \frac{-1}{\sum_{h,l} w_{ho}^l \frac{\partial y_h^l(t)}{\partial t}(t_o^a)} \quad (8)$$

According to the objective function, equation (5), the first factor in the right hand of (7), the derivative of E with respect to t_o, is:

$$\frac{\partial E(t_o^a)}{\partial t_o^a} = t_o^a - t_o^t \quad (9)$$

From expression in equation (4) and apply it for the output layer, we derive:

$$\frac{\partial x_o(t_o^a)}{\partial w_{ho}^k} = \mathrm{y}_\mathrm{h}^\mathrm{k}(\mathrm{t}_\mathrm{o}^\mathrm{a}) \quad (10)$$

Now combine the above results, for equation (6) we have:

$$\Delta w_{ho}^k(t_o^a) = -\eta \frac{y_h^k(t_o^a)\cdot(t_o^t - t_o^a)}{\sum_{h\in D_o}\sum_l w_{ho}^l \frac{\partial y_h^l(t_o^a)}{\partial t_o^a}} \quad (11)$$

For expression simplicity, we define δ_o :

$$\delta_o = \frac{\partial E}{\partial t_o^a}\frac{\partial t_o^a}{\partial x_o(t_o^a)} = \frac{(t_o^t - t_o^a)}{\sum_{h\in D_o}\sum_l w_{ho}^l \frac{\partial y_h^l(t_o^a)}{\partial t_o^a}} \quad (12)$$

Using expression (12), equation (7) can be written as:

$$\frac{\partial E}{\partial w_{ho}^k} = y_h^k(t_o^a)\delta_o \quad (13)$$

And finally we have:

$$\Delta w_{ho}^k = -\eta\, y_h^k(t_o^a)\delta_o \quad (14)$$

Now we consider the error back propagation in the hidden layer. For $h \in H$, the generalized delta error in layer H is defined with actual firing times t_h^a :

Using the above information, we can express part of factor in (7) as follows:

We will not differentiate between the expression $\frac{\partial x_o(t)}{\partial t}(t_o^a)$ and $\frac{\partial x_o(t_o^a)}{\partial t_o^a}$, and the same holds for similar expressions. Clearly the above assumption is related to the learning parameter and only true when it is small. This will be verified in the later section of our study.

$$\delta_h = \frac{\partial E}{\partial t_h^a}\frac{\partial t_h^a}{\partial x_h(t_h^a)} =$$

$$\sum_{j\in D_h}\frac{\partial E}{\partial t_j^a}\frac{\partial t_j^a}{\partial x_j(t_j^a)}\frac{\partial x_j(t_j^a)}{\partial t_h^a}\frac{\partial t_h^a}{\partial x_h(t_h^a)}$$

$$= \sum_{j\in D_h}\delta_j \frac{\partial x_j(t_j^a)}{\partial t_h^a}\frac{\partial t_h^a}{\partial x_h(t_h^a)} \quad (15)$$

where D_h denotes the set of immediate successive neurons in layer O connected to neuron h.

In the above expression we expand the derivative $\frac{\partial E(t_h^a)}{\partial t_h^a}$ in terms of the weighted error contributed to the subsequent layer O. Also the chain rule is used for the case $t = t_h$.

The term $\frac{\partial t_h^a}{\partial x_h(t_h^a)}$ has been calculated in (8), while for $\frac{\partial x_j(t_j^a)}{\partial t_h^a}$:

$$\frac{\partial x_j(t_j^a)}{\partial t_h^a} = \frac{\partial \sum_{l\in H}\sum_k w_{lj}^k y_l^k(t_j^a)}{\partial t_h^a} = \sum_k w_{hj}^k \frac{\partial y_h^k(t_j^a)}{\partial t_h^a} \quad (16)$$

Hence,

$$\delta_h = \frac{\sum_{j\in D_h}\delta_j\{\sum_k w_{hj}^k \frac{\partial y_h^k(t_j^a)}{\partial t_h^a}\}}{\sum_{i\in D_h}\sum_l w_{ih}^l \frac{\partial y_i^l(t_h^a)}{\partial t_h^a}} \quad (17)$$

According to the expression of (14), (15), (16) and (17), the weight update rule for a hidden layer amounts to:

$$\Delta w_{ih}^{k} = -\eta\, y_i^k(t_h^a)\delta_h = -\eta \frac{y_i^k(t_h^a)\sum_j \{\delta_j \sum_k w_{hj}^k \frac{\partial y_h^k(t_j^a)}{\partial t_h^a}\}}{\sum_{i\in D_h}\sum_l w_{ih}^l \frac{\partial y_i^l(t_h^a)}{\partial t_h^a}} \qquad (18)$$

Just as for the classical error back-propagation algorithms, the weight update formula (18) can be generalized to a network with multiple hidden layers. The delta-error at hidden layer *h* is calculated by back propagating the error from the delta-error at higher layer *h + 1*.

4. Momentum and Adaptive Learning Parameter Scheme

In [2] it was reported that no convergence failures of the *SpikeProp* were found for the data sets being studied, but there might exist local minima in some optimization problem, e.g. in the error back propagation algorithm above. To improve convergence and get around of local minimum, a momentum term is added to the weight update procedure, hopefully making the algorithm more robust.

Another issue concerning the error back propagation algorithm is the computation load, or convergence speed. Also convergence rate, which is related to the learning paratemter, was found slow for the *SpikeProp* algorithm in [2], especially for large data sets. Large learning parameter means that the step size is too big for the weight update per iteration and the objective function might not be able to reach its global minima. The optimization procedure might go into the cyclic state. Thus large value of learning parameter usually associates with bad classification results. On the other hand, if the learning parameter is too small, which means we are too conservative in our choice, then it might need much more iterations to reach the global minima, hence the computation time will be longer and the convergence rate is slow. This is the motivation of our adaptive learning parameter scheme – selecting the learning parameter in some optimal sense to speed up the convergence rate. In our algorithm the learning parameter is adjusted dynamically following the *QuickProp* approach. The basic idea is when the weight correction sign is positive, it increases, whereas decreases when the error sign is negative.

So the weight update formula in our algorithm is as follows:

$$w_{ji}^{k+1} = w_{ji}^{k} + \eta_k\, \delta_{ji}^{k} + \alpha\Delta w_{ji}^{k-1} \qquad (19)$$

Where: η_k is the learning parameter per iteration, α is the momentum parameter, and Δw_{ji}^{k-1} is the weight correction in the previous iteration.

5. Encoding Input Variables by Population Coding

For practical classification problems, the input variables are normally in continuous form. So the first problem when we use spiking neural network is how to encode input variables into temporal spike train. Basically there are two schemes. The first is by using increasingly small temporal differences and the second is by distributing the input over multiple neurons. There is a side effect with the first scheme: The increased temporal resolution for the input variables will increase the computational load on the spiking neural network. Another concern with this approach, which has been pointed out in [2], is that it is not biologically plausible. So in our algorithm we adopt the second one - population coding. The basic idea is employing arrays of receptive fields (RF's), which enables the representation of continuous input variables by a pool of neurons with graded and overlapping sensitivity profiles, e.g. Gaussian activation functions. This encoding scheme is biologically more plausible. The highly stimulated neurons are associated with early firing times and less stimulated neurons with later (or no) firing times. Detailed information on population coding is given in [1,2,4,13,14].

6. Experimental Results with Iris Dataset

In this paper we want to concentrate on the Iris dataset and make a thorough study of our algorithm with spiking neural networks against this dataset. The Iris dataset consists of three classes, two of which are not linearly separable, thus provides a good practical problem for our algorithm. The dataset contains 150 data samples with 4 continuous input variables. As in [2], we employ 12 neurons with Gaussian receptive fields to encode each of the 4 input variables, and divide the dataset into training and testing set, each of which has half (75) of the sample numbers.

6.1 Spiking Neural Network Structure

The first thing to be settled is the network structure, namely how many hidden layers and how many neurons in each layer. This is ad hoc and depends on specific problem. For comparison, we also adopted the general one hidden layer structure. The network structures we tried are 50x10x3 and 50x20x3 with static fixed learning parameters 0.05. The results are given below.

Network Structure vs. % of correct classification	Iterations							
	500	1000	1100	1200	1300	1400	1500	1600
50x10x3(1)	94.67	93.33	94.67	94.67	94.67	96.00	97.33	94.67
50x20x3(2)	89.33	94.67	97.33	94.67	94.67	97.33	96.00	94.67

Table 1. Testing classification results vs. different network structure, static learning parameter η = 0.05

The results in table 2 show that the network with more neurons learns comparatively slowly at first. This is due to the more weights to be adjusted. Note that the weights for the 2nd network structure (50x20x3) has doubled compared with the 1st structure (50x10x3). With further iterations, it learns more quickly than the one with less neurons. In our case, classification result has reached the maximum value of 97.33% after 1100 and 1500 iterations, respectively, for the two network structures. This does not necessarily mean the total computation time in the 2nd case is less than the 1st case. In fact, with more weights to be adjusted, for each iteration, the computation load is much heavier in the 2nd structure than the 1st one. This results in more computation time to reach the global minima for the 2nd structure. So in our network, we adopt the 1st smaller compact structure rather than the 2nd one.

6.2 Momentum and Its Effect on the Classification Results

The next comparative study we made is the momentum term in the weight update equation (19).

In our study we found that by adding the momentum term in our algorithm, the classification results for the testing dataset seem to be somewhat oscillating even if the momentum parameter α is small, which means that it deteriorates the classification results.

Another issue concerning the momentum term is it requires more storage when the weight is updated. The previous weight difference has to be saved for the current step. This might be trivial if the data set is small and the computer memory is high enough. But for practical complex nonlinear classification problem where the data set is very large and which entails more weights, the storage will become an important issue when the momentum term is added to the weight adaptation. The computation load for weight update will be higher as well though not as severe as the storage problem. So in our further study, the momentum parameter is set to zero.

6.3 Static Learning Parameter η

Another important parameter to be identified is the learning parameter η. There is no way to decide it theoretically. We can obtain the maximum allowable static value of learning parameter empirically by trial and error. The results of different cases we have tried are given in table 1.

From the results shown in table 1, we can see that with small learning parameter, which is in the range of [0.01, 0.05], the classification results are pretty good – the number of wrongly classified data sample varies between 2 and 5 out of 75 testing data samples for all the iteration 500 till 1600, and the percentage of correct classification is between 93.33% and 97.33%. If the learning parameter is beyond 0.1, the results are rather bad after 500 iterations - the percentage is 33.33% and 34.67%, respectively for the value of 0.1 and 0.5. After further training, the results improve much better – it reaches the maximum percentage 97.33% after 1200 and 1000 iterations for the value of 0.1 and 0.5, respectively. So in our algorithm, we adaptively adjust the learning parameter η and let it varies between 0.001 and 0.05.

The above results also confirm the former assumption in section 3 of this study.

Learning Parameter η	Iteration vs. % of correct classification	500	1000	1100	1200	1300	1400	1500	1600
0.01	Training	97.33	97.33	97.33	97.33	97.33	97.33	97.33	97.33
	Testing	93.33	96.00	97.33	94.67	96.00	97.33	97.33	94.67
0.05	Training	97.33	97.33	96.00	96.00	97.33	96.00	97.33	97.33
	Testing	94.67	93.33	94.67	94.67	94.67	96.00	97.33	94.67
0.1	Training	33.33	76.00	89.33	96.00	96.00	96.00	96.00	92.00
	Testing	33.33	74.67	92.00	97.33	93.33	94.67	97.33	84.00
0.5	Training	36.00	96.00	96.00	94.67	97.33	97.33	97.33	97.33
	Testing	34.67	97.33	96.00	96.00	96.00	97.33	97.33	96.00

Table 2. Classification results vs. static learning parameter η

6.4 Results of the Adaptive Learning Parameter Scheme

In this section, we present the results of our study with adaptive learning parameter scheme. We tested our adaptive algorithm in two cases. For the first case, the initial learning parameter η is 0.005, and the maximum value is limited to 0.05. The classification results are given figure 3.

From figure 1,we can see that the adaptive learning scheme learns much faster for the first 200 iterations, the classification rate reaches 89.33% (32.67% higher) at 100 iterations; one of its local peak, 93.33% (6.67% higher) at 120 iterations. After 450 iterations, both schemes reach their highest percentage 97.33%.

As our second case, the initial learning parameter η is 0.001, and the maximum value is limited to 0.01. The classification results are given in figure 4.

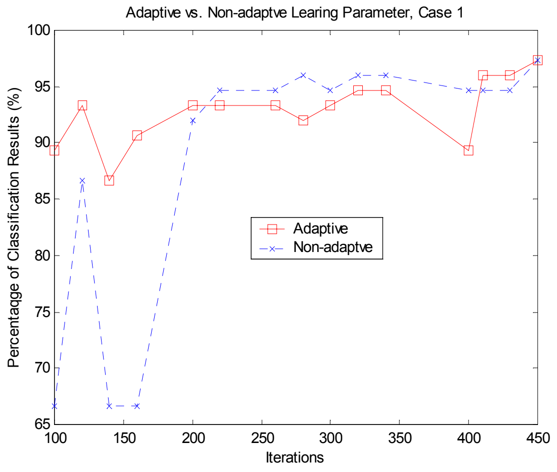

Figure 3. Testing classification results: adaptive vs. non-adaptive learning scheme, case 1.

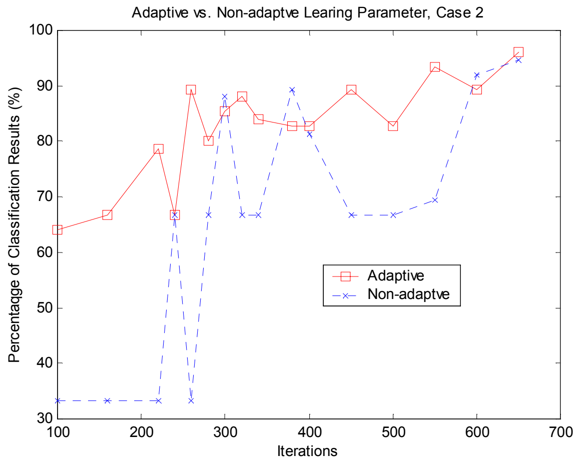

Figure 4. Testing classification results: Adaptive vs. Non-adaptive learning scheme, case 2.

The advantage of the adaptive learning scheme is more pronounced in the second case. From figure 4, we can see that the adaptive learning scheme learns much faster for the first 280 iterations, the classification rate reaches one of its local peak 89.33% (22.67% higher) during this period. For most of the iterations we sampled, the correct classification results are much higher for the adaptive case than the non-adaptive one. After 650 iterations, both schemes reach their highest percentage 96.00% vs. 94.67% (1.33% higher), respectively.

From these two cases, we conclude that the adaptive learning scheme learns faster than the non-adaptive one, and the correct classification percentage is better or comparable to corresponding case. We believe that for more complex nonlinear classification problem with very large data sets, the importance of adaptive learning parameter scheme will be more significant in terms of the computation time and classification rate.

6.5 Comparison Results of Different Learning Algorithm on Iris Dataset

Finally we did some but not thorough comparison between several back propagation algorithms with different neural network structure on the Iris dataset.

First we run the iris dataset against our in-house software - *MetaNeuron* ver. 6.11. This is a thoroughly designed software implementing the back propagation learning algorithm using the sigmoidal function with ANN. By trial and error, the proper network structure for using *MetaNeuron* is 4x10x10x3, e.g. there are two hidden layers. The parameters for learning and momentum are both set to 0.25. The results of running *MetaNeuron* along with the ones in [2] are shown in table 5.

From table 5, we can see that for the Iris dataset, in terms of the iteration numbers to obtain the optimal classification results, our adaptive learning parameter scheme seems to be somewhat more preferable than the rest ones. Regarding the maximum correct classification rate, our scheme is equally as good as *MetaNeuron* and better than the Matlab back propagation "*TRAINLM*" routine and the *SpikeProp* which implements error back propagation algorithm using spiking neural networks.

So the overall performances of our algorithm for the Iris dataset are better than the other three that have been reported or studied.

Iris Dataset					
Algorithm	Inputs	Hidden	Outputs	Iterations	Maximum Classification Rate
SpikeProp	50	10	3	1,000	96.2%
Matlab BP	50	10	3	3,750	95.8%
MetaNeuron	4	10x10	3	1,500	97.33%
QuickProp	50	10	3	450	97.33%

Table 3. Classification result of iris dataset using different back propagation algorithms

We should point out that while testing our algorithm against the Iris dataset, we identified the over-training phenomenon, which is understandable – All error back propagation algorithm has this inherent shortcoming.

7. Conclusions

This study presents an adaptive learning algorithm for spiking neural network based on back propagating the output temporal error. To enhance the robustness of the algorithm when dealing with complex classification problem, a momentum term is considered in the algorithm and its effect on the classification results is studied. The adaptive learning parameter scheme is thoroughly studied against the Iris data set and the results demonstrate that the algorithm for the spiking neural network is successfully capable of solving practical nonlinear complex classification problem.

Acknowledgements

The first author would like to thank Dr. Bohte at the National Research Institute for Mathematics and Computer Science in the Netherlands for the help regarding some contents in the paper, and Dr. Choe in the Department of Computer Sciences, The University of Texas at Austin for the many stimulating discussions about spiking neural networks.

References

[1] P. Baldi and W. Heiligengerg, How sensory maps could enhance resolution through ordered arrangements of broadly tuned receivers, *Biol. Cybern.* 59(1988) 313-318.

[2] S.M. Bohte, J.N. Kok, and H. La Poutre, Error-Backpropagation in Temporally Encoded Networks of Spiking Neurons, in *Proceedings of ESANN'2000*.

[3] Y. Choe, R. Miikkulainen, Self-Organization and Segmentation in a Laterally Connected Orientation Map of Spiking Neurons. *Neurocomputing* volume 21, 139-157, 1998.

[4] A.K. Engel, P. Konig, A.K. Kreiter, T.B. Schillen, and W. Singer, Temporal coding in the visual cortex: new vistas on integration in the nervous system, *Trends Neurosci.* 15(1992) 218-226.

[5] W. Gerstner, Time structure of the activity in neural network models, *Phys Rev E*, 51(1995) 738-758.

[6] W. Gerstner, Spiking neurons, in W. Maass and C. Bishop, editors, *Pulsed Neural Networks* (MIT Press, Cambridge, 1999).

[7] W. Gerstner, R. Kempter, J.L. van Hemmen, and H. Wagner, A neuronal learning rule for sub-millisecond temporal coding, *Nature*, 383(1996) 76-78.

[8] W. Maass, Lower bounds for the computational power of networks of spiking neurons, *Neural Comp.*, 8(1996) 1-40.

[9] W. Maass, Paradigms for computing with spiking neurons, in L. van Hemmen, editor, *Models of Neural Networks*, volume 4, (Spinger Brelin, 1999).

[10] W Maass, Fast sigmoidal networks via spiking neurons, *Neural Comp.*, 9(1997)279-304.

[11] T. Natschlager and B. Ruf, Spatial and temporal pattern analysis via spiking neurons, *Network: Comp. In Neural Syst.*, 9(1998) 319-332.

[12] D.E. Rumelhart, G.E. Hilton, and R.J. Williams, Learning representations by back-propagating errors, Nature, 323(1986) 533-536.

[13] M.N. Shadlen and W.T. Newsome, The variable discharge of cortical neurons: implications for connectivity, computation and information coding, *J. Neuronsci*, 18(1998) 3870-3896.

[14] K.-C. Zhang, I. Ginzburg, B.Ll McNaughton, Interpreting neuronal population activity by reconstruction: Unified framework with application to hippocampal place cells, *J. Neurophys*, 79(1998) 1017-10