**Brain Inspired Cognitive Systems**
**August 29 – September 1, 2004**
**University of Stirling, Scotland, UK**

# SUPERVISED TRAINING OF SPIKING NEURAL NETWORKS WITH WEIGHT LIMITATION CONSTRAINTS

**Q Wu, University of Ulster Magee, Derry, BT48 7JL, Northern Ireland, UK {q.wu@ulster.ac.uk}**

**TM McGinnity, LP Maguire, B Glackin, A Belatreche**
**University of Ulster Magee, Derry, BT48 7JL, Northern Ireland, UK**

## ABSTRACT

There has been much evidence to show that single precise spikes, transfer information among biological neurons. Based on this encoding scheme various spiking neural networks have been proposed to solve computational problems. One such algorithm, a spike time error-backpropagation algorithm for temporally encoded networks of spiking neurons, has been successfully applied to the problem of complex non-liner data classification. There are however, certain features of the algorithm that can be further improved. In this paper, synaptic weight limitation is introduced into the algorithm and a novel solution for the problem raised by non-firing neurons is presented. In addition a square cosine encoder is applied to the input neurons and thus the number of input neurons can be reduced. The improved algorithm becomes more reliable, robust, efficient and reduces the implementation costs. The classical XOR-problem, a function approximation experiment and the Iris benchmark data have been applied to validate the improved algorithm. The experimental results reported show that the modified algorithm produces comparable accuracy in classification with the original approach utilising a smaller spiking neural network.

## 1. INTRODUCTION

Since the integrate-and fire model was first proposed, fire-rate coding has played an important role in the measurement of information transferred among neurons. Recently, there have been many examples presented showing that information is carried by precise spike timing [Gerstner et al, 1996; Riehle et al, 1997; Bi & Poo, 1998; Diesmann et al, 1999; Song et al, 2000, Roberts et al, 2002] and publications by Gerstner have demonstrated transform formulas between pulse codes and firing rates [Gerstner, 1999]. In general, it is acknowledged that by using a temporal information-encoding scheme, a computational spiking neural network can be constructed with a smaller number of spiking neurons than by using a fire-rate scheme. A temporal information-encoding scheme has therefore been applied to many computational spiking neural networks for self-organising or clustering [Choe 1998; Sohn 1999; Lysetskiy 2002]. Bohte et al proposed a spike time error-backpropagation algorithm based on this scheme [Bohte et al, 2002] and demonstrated that spiking neural networks can be applied to solve non-liner classification problems. Spike time error back-propagation (BP) was derived by analogy with a BP algorithm utilised in classical neural networks, however, it was found that the algorithm would not converge if there were no constraints placed on the parameters of the spiking neural network. A crucial issue for the algorithm however, is how to guarantee that all neurons fire at least once, as if a neuron does not fire, the spike-time error cannot be calculated. Modifying several constraints such as the learning rate and the time constant of spiking neurons have been suggested as a means of avoiding this situation however, a solution has not been mentioned in [Bohte et al, 2002] and there are no reports on this issue to date.

In section 2, a general computational spiking neural network is presented, while a brief introduction of the spiking time error-backpropagation algorithm is given. In section 3, the problems with spiking time error-backpropagation algorithm and solutions for improvement of the algorithm are presented. Experimental results and remarks are given in section 4, while conclusion and further study issues are presented in section 5.

## 2. SPIKING NEURAL NETWORK

Spiking Neural Networks (SNNs) are based on spiking neuron models and plasticity synapses. In general a spiking neuron operates by integrating spike responses from presynaptic neurons and generating an output spike when the membrane potential reaches a threshold value. In the approach detailed in this paper, the spiking neuron model utilised was the spike response model [Gerstner, 1999]. The SNN was constructed by interconnecting a number of these neurons using synapses corresponding to different time delays. A computational representation of a spiking neural network based on the spike response model is shown in Figure 1.

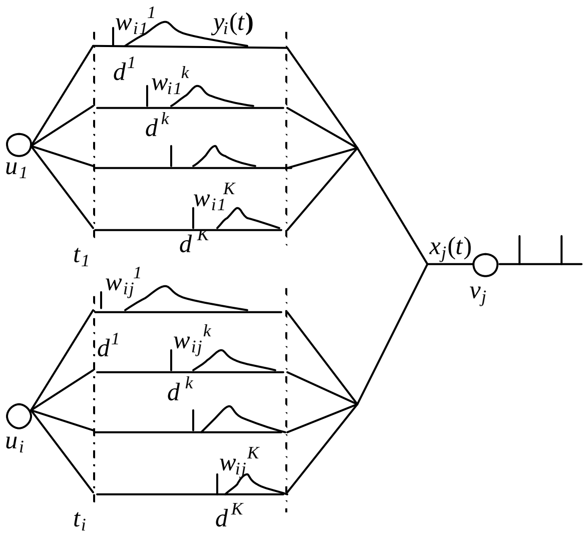


**Figure 1:** *Generic Computational Representation of Spiking Neural Network*

A spike response function [Gerstner, 1999] for each synapse can be written as follows.

$$y_i^k(t) = e(t - t_i^k) = \frac{(t - t_j^k)}{t} e^{1 - \frac{(t - t_i^k)}{t}} \tag{1}$$

Neuron $j$ integrates weighted potential responses caused by spikes from all presynaptic neurons. The membrane potential is written as follows.

$$x_j(t) = \sum_i \sum_k w_{ij}^k e(t - t_i^k) \tag{2}$$

where $t_i^k = t_i + d^k$, and $w_{ij}^k$ is the weight or efficiency of the synapse. When the potential $x_j(t)$ reaches a threshold $q$, the neuron generates an output spike. In [Bohte et el, 2002], the spiking time error is represented by the difference between desired firing time $t_j^d$ and actual firing time $t_j^a$. In a spiking neuron, the firing time is dependent on the membrane potential $x_j(t)$. The transform from firing time error ($t_j^d$ - $t_j^a$) to membrane potential changes of $\Delta x_j(t)$ can be written as

$$\frac{\partial t_j}{\partial x_j(t)}(t_j^a) = \frac{\partial t_{j(x_j)}}{\partial x_j(t)}\bigg|_{x_j = q} = \frac{-1}{\frac{\partial x_j(t)}{\partial t}(t_j^a)} = \frac{-1}{\sum_{i,l} w_{ij}^l \frac{\partial y_j^l(t)}{\partial t}(t_j^a)}$$

where $q$ is the threshold and $y_j^l(t)$ is the potential response from synapse $l$ for neuron $j$. Supposing that neuron $j$ belongs to the output layer and neuron $i$ is presynaptic of neuron $j$, the learning rule for the synaptic weight between the output neuron $j$ and its presynaptic neuron $i$ can be written as follows:

$$\Delta w_{ij}^k(t_j^a) = -h \frac{y_i^k(t_j^a)(t_j^d - t_j^a)}{\sum_{i \in \Gamma_j} \sum_l w_{ij}^k \frac{\partial y_j^l(t_j^a)}{\partial t_j^a}} \tag{3}$$

Following [Bohte et el, 2002], suppose that $h$ represents a neuron in input layer, $i$ a hidden neuron and $j$ an output neuron. The learning rule for the weight between the hidden neurons can be written as follows:

$$\Delta w_{hi}^k = -h\, y_h^k(t_i^a) d_i = -h \frac{y_h^k(t_i^a) \Sigma_j \{ d_j \Sigma_k w_{ij}^k \frac{\partial y_i^k(t_j^a)}{\partial t_i^a} \}}{\Sigma_{n \in \Gamma_i} \Sigma_l w_{ni}^l \frac{\partial y_n^l(t_i^a)}{\partial t_i^a}} \tag{4}$$

where,

$$d_i = \frac{\Sigma_{j \in \Gamma^i} d_j \{ \Sigma_k w_{ij}^k \frac{\partial y_i^k(t_j^a)}{\partial t_i^a} \}}{\Sigma_{h \in \Gamma_i} \Sigma_l w_{hi}^l \frac{\partial y_h^l(t_i^a)}{\partial t_i^a}} \tag{5}$$

and $G_i$ is a set of afferent neurons for neuron $i$. The algorithm based on these rules is called SpikeProp and detailed derivations of the rules can be found in [Bohte *et al*, 2002].

## 3. PROBLEMS AND SOLUTIONS

A problem arises with this approach in that if the rules are applied directly, the algorithm does not consistently converge. It is evident that the learning rule (3) cannot be used if a neuron does not fire because $t_j^a$ cannot be calculated for non-firing neurons. In this paper, a feedback rule as follows is proposed for non-firing neurons.

$$\Delta w_{ij}^k(t_j) = -h \frac{y_i^k(t_j)(J - x_j(t_j))}{x_j(t_j)} \tag{6}$$

However, the problem remains when the number of non-firing neurons becomes large. Inspired by weight limitation of biological neurons [Rossum et al, 2000, Song et al, 2001], weight limitations of $w_{min}$ and $w_{max}$ are set to solve the problem. In order to illustrate this mechanism, a simple spiking neural network is shown in Figure 2 demonstrating the XOR problem.

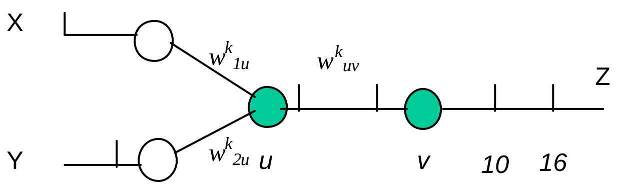


**Figure 2:** *Spiking neural network for XOR problem*

Logic value '1' is assigned an early firing time and logic value '0' is assigned a late firing time. An input spike time of 0ms is associated with logic value ‘1’ while a spike time of 6ms is associated with logic value ‘0’. An output spike at 10 ms represents a logic ‘1’ and an output at 16ms represents a logic ‘0’. The training patterns are shown in Table 1.

| Input | | Output |
|---|---|---|
| X(ms) | Y(ms) | Z(ms) |
| 0 (1) | 0 (1) | 16 (0) |
| 0 (1) | 6 (0) | 10 (1) |
| 6 (0) | 0 (1) | 10 (1) |
| 6 (0) | 6 (0) | 16 (0) |

**Table 1** *Training patterns associations for XOR problem*

In this case, the input encoding time-window is $t_w$= 6ms i.e. input spikes are encoded within 6ms. The maximal network delay is $t_m$= 16ms, i.e. an output should be obtained 16ms after the network started to receive spikes. From figure 2, $u$ represents a neuron in hidden layer while $v$ represents a neuron in output layer. The potential of neuron $v$ is dependent on the output spikes of the hidden layer neurons and can be represented by

$$x_v(t) = \sum_u \sum_k w_{uv}^k e(t - t_u^k)$$

Suppose that neuron $v$ fires at time $t_m$. The membrane potential is

$$x_v(t_m) = \sum_u \sum_k w_{uv}^k e(t_m - t_u^k) = q \tag{7}$$

Let $N$ represent the number of neurons in the hidden layer. Suppose that each neuron in hidden layer fires at same time $t^k$ and the weights from neurons in the hidden layer to neuron $v$ are equal, i.e. $w_v{}^k = w_{1v}{}^k = w_{2v}{}^k = \ldots = w_{Nv}{}^k$. The weights can be calculated as follows

$$w_v^k = \frac{q}{Ne(t_m - t^k)} \tag{8}$$

From these equations it can be seen that if $w_{vu}{}^k$ is not less than $w_v{}^k$ for all synapses from neurons $u \in N$ to neuron $v$, neuron $v$ fires at or earlier than $t_m$. Therefore, $w_v{}^k$ can be regarded as the minimal weight for synapse with delay $t_u{}^k$. By combining Equation (1) and Equation (8), the expression of minimal weight for the synapse with delay $t^k$ can be written as

$$w_v^k(\min) = \frac{qt}{N(t_m - t^k)} e^{\frac{(t_m - t^k)}{t} - 1} \tag{9}$$

By using equation (9) to control the weights during training, the learning algorithm can converge to the desired error. According to equation (9), the initial weights can be set by the following equation.

$$w_{vu}^k(init) = w_v^k(\min) + random\_number \tag{10}$$

In the experiments detailed in this paper, the range of the random number is set to [0,$q$]. By analogy, maximal weights can be derived from the desired earliest output-spike; however, the derivation is ignored because maximal weights are not important for improving the algorithm.

## 4. EXPERIMENT RESULTS

### 4.1. DEMONSTRATION WITH XOR PATTERNS

An algorithm was implemented in software to simulate the spiking neural network shown in Figure 2, with two inputs X and Y, one spiking neuron in the hidden layer and one spiking neuron in the output layer. The following parameters were considered, time constant $t$ =16ms, learning rate=0.005 and threshold=0.5. Therefore, the membrane potential of hidden neuron $u$ is

$$x_u(t) = \sum_h \sum_k w_{hu}^k \frac{(t - t_h^k)}{t} e^{1 - \frac{(t - t_h^k)}{t}} \tag{11}$$

where $h \in \{1,2\}$ is an index of the two inputs, $k \in \{0,1,2,3,\ldots,6\}$ is an index corresponding to a delay time $t^k$ $\in \{$0ms, 1ms, 2ms, 3ms,…,6ms$\}$. The membrane potential of output neuron $v$ can be described as

$$x_v(t) = \sum_{k=0}^{K_u} w_{uv}^k \frac{(t - t_u^k)}{16} e^{1 - \frac{(t - t_u^k)}{16}} \tag{12}$$

where $K_u$ is the index of latest firing time of neuron $u$. Based on these membrane potentials, Equation (6) and the SpikeProp rules are applied to train the network with four XOR patterns. The four output spike-time curves recorded in each training epoch are shown in Figure 3(a). The y-axis corresponds to the output spike time, while the x-axis represents the training epoch number. The blue solid-curve is the output spike time for input pattern (X=0ms, Y=0ms), and the red dot-curve is the output spike time for input pattern (X=6ms, Y=0ms). During a training epoch, the four patterns are applied to update weights once. At epoch 1, output spike at time 10ms was caused by X input spike at 0ms and Y input spike at 0ms. Output spike at time 14ms was caused by X input spike at 6ms and Y input spike at 6ms. When input pattern was X input spike 6ms and Y input spike at 0ms or X input spike 0ms and Y input spike at 6ms, the network fired at 15ms. These initial firing times depend on random initial weights. After training several epochs, the red curve converged on the target output spike time 10ms. The blue curve oscillated but eventually converged on the target firing time 16ms. This curve indicates that the output spike-time for input pattern (X=0ms and Y=0ms) is later than 16ms on multiple occasions during the training time. When the spike time was larger than 20ms, the neuron was regarded as a non-firing neuron in the simulation program and equation (6) was applied for error feedback. If initial weights were in large range, the algorithm could not converge.

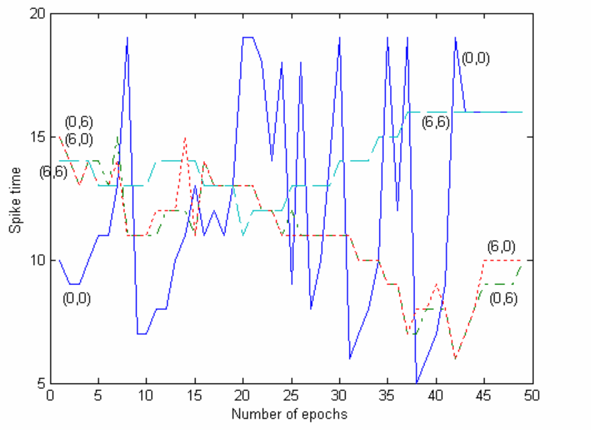
Therefore, weight constraints are proposed to solve this problem. For example, the weights from inputs to neuron $u$ can be calculated by combining Equation (9) and (11). Suppose that the latest firing time for neuron $u$ is $t_{mu}$=12ms. Following Equation (9), the minimal weights from the inputs to neuron $u$ is calculated by

$$w_u^k(\min) = \frac{qt}{N(t_{mu} - t^k)} e^{\frac{(t_{mu} - t^k)}{t} - 1} = \frac{0.5 \times 16}{2(12 - t^k)} e^{\frac{(12 - t^k)}{16} - 1}$$

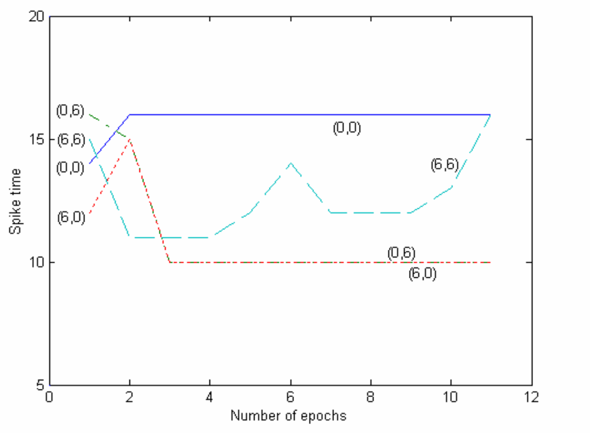
Table 2 shows all values for $w_u{}^k{}_{(\min)}$.

| $k$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $t^k$ | 0ms | 1ms | 2ms | 3ms | 4ms | 5ms | 6ms |
| $w_u^{\,k}{}_{(\min)}$ | 0.26 | 0.27 | 0.28 | 0.29 | 0.31 | 0.33 | 0.36 |

**Table 2:** *Minimal weights from inputs to neuron u*



(a) *Equation (6) applied*



(b) *Equations (9) and (10) applied*

**Figure 3:** *Output Graphs of Training for XOR Patterns*

In order to illustrate how these weights control the neuron spikes, suppose that input X is a spike at 1ms ($k$=1) and input Y is a spike at 6ms ($k$=6). From Table 2, we have $w_{1u}^{1}(\min)=0.27$, $w_{2u}^{6}(\min)=0.36$. Form Equation (11), the membrane potential of neuron $u$ is

$$x_u(t)=(w_{1u}^{1}\frac{(t-1)}{16}e^{1-\frac{(t-1)}{16}}+w_{2u}^{6}\frac{(t-6)}{16}e^{1-\frac{(t-6)}{16}})$$

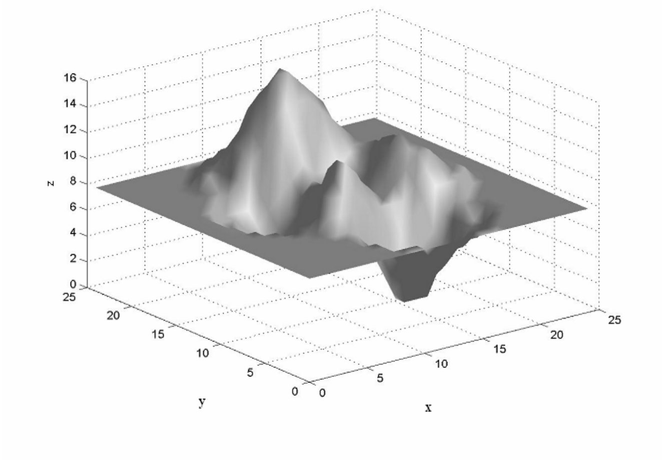
$$x_u(t)=0.27\frac{(t-1)}{16}e^{1-\frac{(t-1)}{16}}+0.36\frac{(t-6)}{16}e^{1-\frac{(t-6)}{16}})$$

when $t$=$t_{um}$=12ms, $x_u(t)$ =0.5059. The value is greater than the threshold value of 0.5.
This implies that neuron $u$ fires earlier than $t_{um}$ =12 ms if the weight $w_{hu}^{\,k}$ is larger than $w_u^{\,k}{}_{(min)}$.

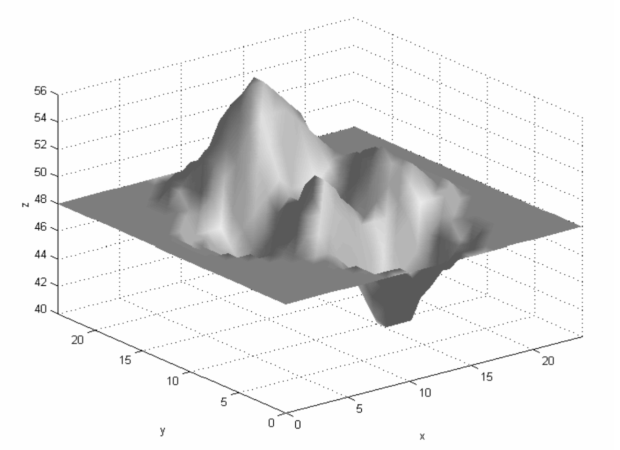
By analogy, a set of minimal weight $w_v^{\,k}{}_{(min)}$ can be obtained and applied to control $w_{uv}^{\,k}$ Under limitation of the weights, the four output spike-time curves recorded in each training epoch were generated corresponding to the four training patterns shown in Figure 3(b). Here it can be seen that the four output spike-times are not later than 16ms. Twelve epochs were needed to converge to the minimal error, which is four times faster than that achieved by the algorithm without weight limitation. If equation (6), (9) and (10) are not used, the training does not always converge to zero error due to the fact that if a neuron does not fire, there is no firing-time-error feedback. In [Bohte et el, 2002], a large number of input encoding neurons are proposed to solve this problem.

## 4.2. SIMULATION OF NON-LINEAR FUNCTION

The network was extended in size from that shown in Figure 2, and was applied to simulate the peaks function from Matlab (see Figure 4(a)). By using the improved algorithm, the network does not require the encoding neurons mentioned in [Bohte et el, 2002].



(a) *24X24 input patterns*



(b) *Neural network output*

**Figure 4:** *Non-liner function simulated by SNN with improved learning algorithm*

The network is constructed in two layers with two input neurons representing $x$ and $y$, 6 spiking neurons in the hidden layer, and one output spiking neuron. In comparison with the network in section 4.1, the time scale has changed from 1ms to ¼ms. The encoding time window $t_m$= 6ms corresponds to $t_m$=24 units of time scale. One unit on the axis in Figure 4 represents ¼ms. Results for the peaks function simulation are shown in Figure 4(b). Figure 5 shows that the error converges to zero at approximately 400 epochs.

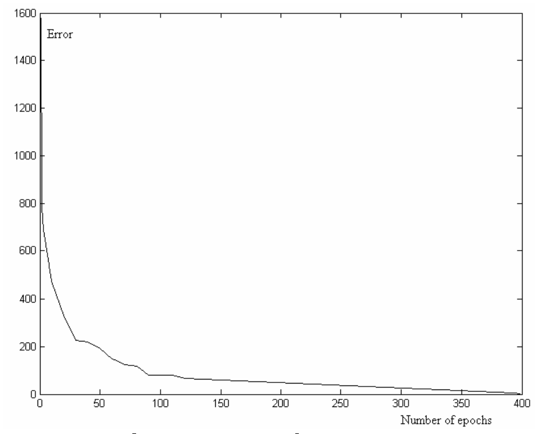


**Figure 5:** *Network output error changes over training time for peaks function simulation*

### 4.3. SIMULATION OF IRIS DATA SET

The Iris Data Set consists of four inputs. The range for input-1 is [4.3, 7.9], input-2 is [2.0, 4.4], input-3 is [1.0, 7.0] while the range for input-4 is [0.1, 2.5]. For example, there are 24 units of time scale for input encoding time window of the SNN in section 4.2. If all input values are converted to [0, 24], some inputs will lose their precision. The authors' solution is to propose a square cosine encoder for the input neurons. Let $x_i$ be an input variable and ($x_{i1}$,$x_{i2}$,...$x_{i\,m}$) be a set of spike times of $m$ input neurons. For input neuron $k$

$$x_{ik} = \text{int}(t_w[\cos^2(p\,\frac{x_i - x_i(\min) - j(k)}{x_i(\max) - x_i(\min)})]^{\frac{1}{e}} \qquad (13)$$

where $x_{ik}$ is an integer corresponding to a spike time, $t_w$ is the encoding time window, $e$ is a constant for adjusting the neuron sensitivity range (0.6 was used in this case), and $j(k)$ is a phase offset for input neuron $k$. In this example, four input neurons were used to deal with input-3 of the Iris data. The precision of the input data is represented by $\gamma$ with $\gamma$ = 0.1, $x_{i(max)}$ =7.0 and $x_{i(min)}$ =1.0. Each neuron is sensitive to ¼ range of the input where the first neuron with $j(1)$ = 0, is sensitive to low and high values (solid curve in Figure 6). Input neuron $k$ for $k$>1 is sensitive to values around $x_{i(min)}$ + ($k$-1)( $x_{i(max)}$ - $x_{i(min)}$) $\gamma/m$, therefore

$$j(k) = \begin{cases} 0 & \text{for } k=1 \\ \frac{1}{m}(k-1)(x_{i(\max)} - x_{i(\min)})g & \text{for } k>1 \end{cases} \qquad (14)$$

The four curves for the input neurons are shown in Figure 6. Input value 3.9 is indicated by the vertical line. Four spike times ($x_{i1}$=0, $x_{i2}$=9, $x_{i3}$=24, $x_{i4}$=6) can be obtained from four points that the vertical line crosses the four curves.

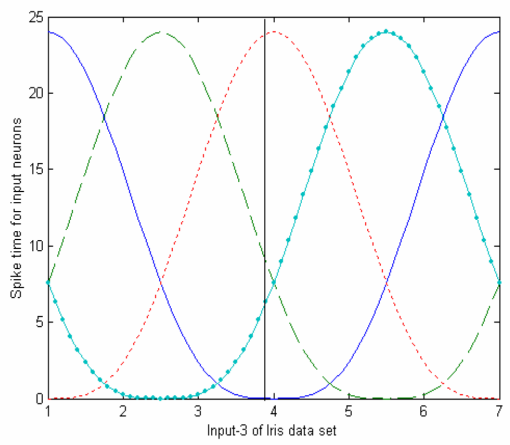


**Figure 6:** *Iris input-3 converted to 4 spike times by square cosine encoder*

By using this encoding scheme, a large input range can be converted to large number of input neurons and a small input range does not need a large number of input neurons. According to the input range, 3 input neurons are assigned to input-1 and one input neuron is assigned to input-2 and input-4 separately. The SNN architecture for the Iris data set is shown in Figure 7.

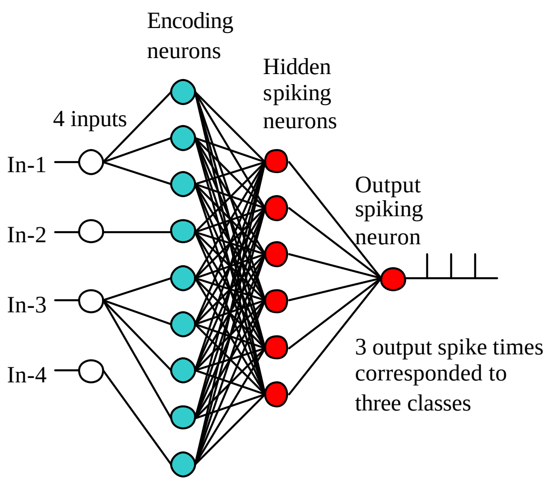


**Figure 7:** *SNN architecture for classification of Iris data*

The Iris data set is divided into two equal sets for cross-validation. One set is applied to train the network where after about 300 epochs using the improved algorithm; the output error of the network converges to zero. After training the network correctly identifies 100 percent of the training set patterns. When the second half of the data is used for testing the accuracy rate reaches 92.0%~98.6%. It was found that the average accuracy rate, found by cross-validating ten times, was 96.6%. A comparison with results reported by [Bohte et el, 2002] is shown in Table 3.

| Algorithm | E | H | O | TI | Train-set | Test-set |
|---|---|---|---|---|---|---|
| SpikeProp | 50 | 10 | 3 | 1000 | 97.4%±0.1 | 96.1%±0.1 |
| MatlabBP | 50 | 10 | 3 | 2.6e10 | 98.2%±0.9 | 95.5%±2.0 |
| MatlabLM | 50 | 10 | 3 | 3,750 | 99.0%±0.1 | 95.7%±0.1 |
| Improved Agorothm | 9 | 6 | 1 | 300 | 100% | 96.6% |

**Table 3:** *Comparison results of Iris data classification*

E: number of encoding neurons
H: number of hidden neurons
O: number of output neurons
TI: number of training iterations

In this section, a square cosine encoder has been introduced, which is similar to the one dimensional receptive field encoding mentioned in [Bohte et el, 2002]. However, utilising the square cosine encoder requires much less input neurons than receptive field encoding. Combining the square cosine encoder and the weight limitation algorithm, a small spiking neural network was trained to solve the Iris data classification problem. It was found that the results were comparable with other approaches.

## 5. CONCLUSIONS

Spiking time error-backpropagation algorithm for temporally encoded networks of spiking neurons is based on the assumption that all neurons fire at least once. The weight limitation approach introduced in this paper ensures that neurons fire at a time that is not later than the maximal network delay and initial weights enable networks to converge to zero error rapidly. Using the square cosine encoder approach makes the input neurons more efficient and results in a lesser number of them being required. Combining these techniques together in the algorithm, a small and efficient SNN can be obtained. The experiments detailed in this paper show comparable results with other approaches. From a hardware engineering perspective, it is of benefit to reduce the implementation costs. In this paper, the SpikeProp algorithm has been improved in that not only does the improved algorithm become more reliable and efficient, but also networks can be designed in smaller size ensuring that implementation costs can be reduced.

## ACKNOWLEDGMENTS

The authors acknowledge the financial and technical contribution of the SenseMaker project (IST-2001-34712) which is funded by the EC under the FET life like perception initiative.

## REFERENCES

[Bi & Poo, 1998] Bi Quo-qing, Poo Mu-ming, “Precise Spike Timing Determines the Direction and Extent of Synaptic Modification in Cultured Hippocampal Neurons”, Neuroscience 18:10464-10472, 1998

[Bohte et al, 2002] S Bohte, JN Kok, HL Poutré, "SpikeProp: Error-Backpropagation for Networks of Spiking Neurons", Neurocomputing, 48(1-4), pp 17-37, 2002

[Choe 1998] Y Choe, R Miikulainen, “Self-organization and segmentation in a laterally connected orientation map of spiking neurons”, Neurocomputing, 21:139-157, 1998

[Diesmann et al, 1999] M Diesmann, MO Gewaltig, and A Aertsen, “Stable propagation of synchronous spiking in cortical neural networks”, Nature, 402:529-533, 1999

[Gerstner, 1999] W Gerstner, “Spiking Neurons”, In Pulsed neural networks/ edited by Wolfgang Maass and Christopher M. Bishop, Cambridge: MIT Press, 1999, pages 1-53.

[Gerstner et al, 1996] W Gerstner, R Kempter, JL Hemmen H Wagner, “A neuronal learning rule for sub-millisecond temporal coding”, Nature, 383:76-78, 1996

[Lysetskiy 2002] M Lysetskiy, A Ozowski and JM Zurada, “Invariant Recognition of Spatio-Temporal Patterns in The Olfactory System Model”, Neural Processing Letters 15:225–234, Kluwer Academic, 2002

[Riehle et al,1997] A Riehle, S Grun, M Diesmann, A Aertsen, “Spike synchronization and rate modulation differentially involved in motor cortical function”, Science 278, 1950-1953, 1997

[Roberts et al, 2002] PD Roberts, CC Bell, "Spike Timing Dependent Synaptic Plasticity in Biological Systems", Biol. Cybern. 87, 392–403, 2002

[Sohn 1999] JW Sohn, BT Zhang, BK Kaang, “Temporal Pattern Recognition Using a Spiking Neural Network with Delays”, Proceedings of the International Joint Conference on Neural Networks (IJCNN'99), vol. 4, pp. 2590-2593, 1999

[Song et al, 2000] S Song, KD Miller, LF Abbott, “Competitive Hebbian Learning Through Spike-Timing Dependent Synaptic Plasticity”, Nature Neuroscience. 3:919-926, 2000

[Song et al, 2001] S Song, LF Abbott, “Column and Map Development and Cortical Re-Mapping Through Spike-Timing Dependent Plasticity”, Neuron 32:1-20, 2001