

# Hebbian Learning with Winner Take All for Spiking Neural Networks

Ankur Gupta\* and Lyle N. Long\*\*

**Learning methods for spiking neural networks are not as well developed as the traditional rate based networks, which widely use the back-propagation learning algorithm. We propose and implement an efficient Hebbian learning method with homeostasis for a network of spiking neurons. Similar to STDP, timing between spikes is used for synaptic modification. Homeostasis ensures that the synaptic weights are bounded and the learning is stable. The winner take all mechanism is also implemented to promote competitive learning among output neurons. We have implemented this method in a C++ object oriented code (called CSpike). We have tested the code on four images of Gabor filters and found bell-shaped tuning curves using 36 test set images of Gabor filters in different orientations. These bell-shaped curves are similar to those experimentally observed in the V1 and MT/V5 area of the mammalian brain.**

## I. INTRODUCTION

Artificial neural networks (ANN) can broadly be classified into three generations. The first generation models consisted of McCulloch and Pitts neurons that restricted the output signals to discrete '0' or '1' values. The second generation models, by using a continuous activation function, allowed the output to take values between '0' and '1'. This made them more suited for analog computations, at the same time, requiring fewer neurons for digital computation than the first generation models [1]. One could think of this analog output between 0 and 1 as normalized firing rates. This is often called a rate coding scheme as it implies some averaging mechanism.

Spiking neural networks belong to the third generation of neural networks and like their biological counterparts use spikes or pulses to represent information flow. Neurological research also shows that the biological neurons store information in the timing of spikes and in the synapses. There have been many studies in the past using spiking neuron models to solve different problems for example spatial and temporal pattern analysis [2], instructing a robot for navigation and grasping tasks [3-5], character recognition [6, 7], and learning visual features [8]. There are also biologically realistic networks or brain simulators ranking medium to high in biological plausibility such as [9-12]. A different and faster approach involves making some simplifications and only using 1 spike per neuron such as [13-15].

Most of the success of the 2<sup>nd</sup> generation neural networks can be attributed to the development of proper training algorithms for them, e.g. the backpropagation algorithm [16, 17]. It is one of the most widely known algorithms for training these networks and is essentially a supervised gradient descent algorithm. In a previous paper [16], we showed how these 2<sup>nd</sup> generation models could be made scalable and run efficiently on massively parallel computers. In that work, we developed an object-oriented, massively-parallel ANN software package SPANN (Scalable Parallel Artificial Neural Network). The software was used to identify character sets consisting of 48 characters and with various levels of resolution and network sizes. The code correctly identified all the characters when adequate training was used in the network. The training of a problem size with 2 billion neuron weights (comparable to rat brain) on an IBM BlueGene/L computer using 1000 dual PowerPC 440 processors required less than 30 minutes.

Due to the spiking nature of the biological neurons, traditional learning algorithms for rate-based networks aren't suitable for them. Also, the learning methods for these spiking networks are not as well developed as the traditional networks. One commonly used unsupervised learning approach for spiking neural networks is called spike time dependent plasticity (STDP) [18-20]. It is a form of competitive Hebbian learning and uses spike timing information to set the synaptic weights. This is based on the experimental evidence that the time delay between pre- and post-synaptic spikes helps determine the strength of the synapse. We have shown previously how these spiking neural networks can be trained efficiently using Hebbian-style unsupervised learning and STDP [7, 21, 22].

The present paper shows how such learning method can be used along with winner take all for competitive learning among neurons. We have implemented this method in an object oriented C++ code (called CSpike), which uses simple and fast neuron models such as leaky integrate and fire (LIF) neuron model, and can be scaled up to network sizes similar to human brain. The paper is organized as follows: In the next section we describe the neuron model and learning method in detail, the third section describes the test problem and network structure used. The fourth and fifth section present results and conclusions respectively.

\*Graduate student, Dept. of Computer Science and Engineering, Email: azg139@psu.edu.

\*\*Distinguished Professor of Aerospace Engineering, Bioengineering, and Mathematics, Email: ln@psu.edu

The Pennsylvania State University, University Park, PA 16802.

## II. NEURON MODEL AND LEARNING

### A. Neuron Model

There are many different models one could use both to model the individual spiking neurons, and also the nonlinear dynamics of the system. Individual neuron models can be categorized according to their biological plausibility and speed. Generally speaking, the more biologically plausible models tend to require more computation time. The Hodgkin-Huxley (HH) [23] model was one of the first detailed neuron models developed, and they received a Nobel prize for this work. It was based on experimental results from squid giant axons. Though the model is quite accurate, it is expensive to compute. Another rather simple model is the leaky integrate and fire (LIF) [24] model, which is much simpler and is computationally cheaper. Izhikevich [25] compares these and many more such models. We [21] have compared HH to LIF and found LIF to be at least 5000 times faster than HH. Even though the LIF model is computationally inexpensive, it can model the spike times very similar to the HH model. This is important as spiking networks often use the timing of spikes for learning. A brief description of the LIF model follows. For a neuron  $i$  in a system of neurons, the voltage would be modeled by:

$$\frac{dv_i}{dt} = \frac{1}{\tau} \left( (I_{input} + I_i)R - v_i \right) \quad (1)$$

Where,  $v$  is voltage,  $\tau=RC$  is the time constant,  $R$  is resistance,  $C$  is capacitance,  $I_{input}$  is a possible input current (usually zero), and  $I_i$  is the current from the synapses. When the membrane potential reaches some threshold value  $v_{th}$ , it produces a spike, and immediately after that the membrane potential is reset to a value  $v_{reset}$ . After this event, the membrane potential recovers to the resting potential value after  $t_{ref}$  time. The current from the synapses can be modeled as:

$$I_i = \alpha \sum_{j=1}^N w_{ij} \sum_{k=1}^M \delta(t - t_{jk}) \quad (2)$$

Where  $N$  is the number of presynaptic neurons and  $M$  is the number of times the  $j^{th}$  presynaptic neuron has fired since the  $i^{th}$  neuron fired. The coefficients  $w_{ij}$  and  $\alpha$ , represent the synapse weights and the increment of current injected per spike, respectively. We have used the above LIF model for this work.

### B. Learning

Hebb [26] postulated that “when an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth or metabolic change takes place in one or both cells such that A’s efficiency, as one of the cells firing B, is increased”. Long term potentiation (LTP) provided strong experimental support for Hebb’s postulate. LTP is the long term increase in synaptic strength and its opposing effect is long term depression (LTD).

Spike time dependent plasticity (STDP) can be viewed as a more quantitative form of Hebbian learning. It emphasizes the importance of causality in synaptic strengthening or

weakening. When the pre-synaptic spikes precede post-synaptic spikes by tens of milliseconds, synaptic efficacy is increased. Whereas, when the post-synaptic spikes precede the pre-synaptic spikes, the synaptic strength decreases. Both in vitro and in vivo experiments have provided evidence for this mechanism [27-30].

Hebbian learning is often implemented as spike-time dependent plasticity (STDP) in spiking neuron codes [20]. Previously, we implemented STDP for character recognition [7] but used the active dendrite dynamic synapse model (ADDS) by [5]. In the present study, we use just the simple LIF model and use a much efficient learning method. The STDP algorithm modifies the synaptic weights using the following algorithm:

$$\Delta w = \begin{cases} A^+ \exp(\Delta t / \tau^+) & \text{if } \Delta t < 0 \\ -A^- \exp(-\Delta t / \tau^-) & \text{if } \Delta t \geq 0 \end{cases} \quad (3)$$

$$w_{new} = \begin{cases} w_{old} + \eta \Delta w (w_{max} - w_{old}) & \text{if } \Delta w \geq 0 \\ w_{old} + \eta \Delta w (w_{old} - w_{min}) & \text{if } \Delta w < 0 \end{cases} \quad (4)$$

Where,  $\Delta t = (t_{pre} - t_{post})$  is the time delay between the presynaptic spike and the postsynaptic spike. If the presynaptic spike occurs before the postsynaptic spike, it probably helped cause the postsynaptic spike, and consequently we should encourage this by increasing the synaptic weight. And if the presynaptic spike occurs after the postsynaptic spike, then we reduce the weight of the synapse since there was no cause and effect in this case. STDP can be used for inhibitory or excitatory neurons.

The above algorithm is not necessarily the optimal learning approach for spiking networks, even though it has worked well on a number of applications. One issue with STDP involves causality. When a post-synaptic neuron fires in a time-marching code (or in real life), it is unknown (at that time) whether one of the presynaptic neurons will fire in the future (and at what time). In the laboratory, current can be injected near individual neurons after they fire, but this is not necessarily an efficient computational algorithm. The above algorithm can be easily implemented for very small systems where memory storage is not an issue, but for millions of neurons and billions of synapses we need extremely efficient algorithms that use minimal computer operations and storage, and minimal gather/scatter operations.

It is also helpful to keep in mind that most of the STDP research is still in its infancy. Most of the STDP experiments are performed on pyramidal cells and excitatory synapses. It is well known that there are a large variety of cell types with different biophysical properties. Also, it is still unclear how STDP operating in timescale of milliseconds can justify most of the behavioral/learning events occurring at much higher time scale. It might be worthwhile to computationally analyze the consequences of STDP in computational simulations with large neural networks. But, here we just

focus on developing and testing a stable learning method that is motivated from STDP.

We have implemented Hebbian learning very efficiently in the JSpike software previously [21]. It was implemented in essentially a homeostatic “event driven” manner. That is, if a neuron fires, then the learning method is called by the neuron that fired. This neuron has ready access to all the presynaptic neurons that connect to it. When the postsynaptic neuron fires (and learning is turned on), it can then loop thru all the presynaptic neurons and compute which ones also fired during the time interval between this and the previous postsynaptic firing. Since the current is reset each time a neuron fires, we simply need to know which presynaptic neurons fired between the last two firings of the postsynaptic firings. These are the neurons that connect to the synapses that are strengthened. Any neuron that has not contributed to the postsynaptic neuron firing has its weight decreased. This approach is spike-time dependent, but it is different than STDP. The weight updates are done using:

$$w = \begin{cases} w_{old} + A^+ e^{(-\Delta t/\tau^+)}, & t_{post2} \leq t_{pre} \leq t_{post1}, \Delta t = t_{post1} - t_{pre} \\ w_{old} - A^- e^{(-\Delta t/\tau^-)}, & t_{pre} \leq t_{post2}, \Delta t = t_{post2} - t_{pre} \end{cases} \quad (5)$$

In the present work, we use this approach and also implement a winner take all mechanism described below.

### C. Homeostasis and Winner take all

The homeostasis mechanism implemented here implies that the net sum of incoming weights associated with a neuron remains the same [22]. This ensures that the synaptic weights are bounded and the learning is stable. The algorithm can be implemented in the following way:

- 1) Let (i,j) be a synapse with  $t_{post2} < t_{pre} < t_{post1}$ , with total p such synapses.  $(i, j) \in A$ . Calculate  $\sum \Delta w^+$
- 2) Let q be no. of synapses with  $t_{pre} < t_{post2}$ .  $(i, j) \in B$ . Calculate q.
- 3) if  $(i, j) \in A$   $w_{new} = w_{old} + \Delta w^+$  else if  $(i, j) \in B$   $\Delta w^- = -\sum (\Delta w^+)/q$ ,  $w_{new} = w_{old} + \Delta w^-$

$$\Delta w^+ = A^+ e^{(-\Delta t/\tau^+)}, \Delta t = t_{post1} - t_{pre} \quad (6)$$

Here,  $t_{pre}$  is the pre-synaptic spike time;  $t_{post1}$  and  $t_{post2}$  are the post-synaptic spike times. We have used  $\tau^+ = 15$  ms,  $A^+ = 0.01$ . The above algorithm assumes that the weights are not saturating to 1 or becoming negative. In order to take care of those cases, synapses are identified beforehand which are going to saturate and instead of incrementing them by  $\Delta w^+$ , they are set to 1 and  $(w_{old} + \Delta w^+ - 1)$  is subtracted from  $\sum \Delta w^+$  to reflect the correct sum. Similarly, the synapses that would become negative are treated the same way.

Winner take all (WTA) is a concept that promotes competitive learning among neurons. After the competition,

only the best neuron will be active for some input and the rest neurons will eventually become inactive for that input. It is worth looking at WTA and other related learning methodologies in light of their generalizabilities and discriminatory capacities. Biologically plausible learning methods can be broadly classified as dense, local, or sparse. Competitive learning such as WTA is a local learning rule as it activates only the unit that fits the input pattern best and suppresses the others through fixed inhibitory connections. This type of ‘grandmother cell’ representation is very limited in the number of discriminable input states it can represent and also is not very generalizable as the winner unit turns on simply when the input is within some Hamming distance from its preferred input. Dense coding can be seen as the other extreme, where a large number of units are active for each input pattern. Thus, it can code a large number of discriminable input states. But then the mapping and learning become more complicated to implement by simple neuron like units [31]. Sparse coding [32, 33] is a tradeoff between these two extremes where the input patterns are represented by the activity in a small proportion of the available units. It is very likely that depending on the requirement, one or a combination of these methods is being used in the brain. The ‘grandmother cell’ type response cannot be totally ruled out. For example ‘grandmother cell’ type representation is found in monkeys for some ‘crucial tasks’ [34].

Reference [35] shows that WTA is quite powerful compared to threshold and sigmoidal gate often used in traditional neural networks. It is shown that any Boolean function can be computed using a single k-WTA unit [35]. This is very interesting, as at least two-layered perceptron circuits are needed to compute complicated functions. They also showed that any continuous function can be approximated by a single soft WTA unit (A soft winner take all operation assumes values depending on the rank of corresponding input in linear order). Another advantage is the fact that approximate WTA computation can be done very fast (linear-size) in analog VLSI chips [36]. Thus, complex feed-forward multi-layered perceptron circuits can be replaced by single competitive WTA stage leading to low power analog VLSI chips [35].

There have been many implementations of winner take all (WTA) computations in recurrent networks in the literature [37, 38] Also there have been many analog VLSI implementations of these circuit [38, 39]. The WTA model we have implemented is influenced by the WTA implementation on recurrent networks by [38]. In their implementation, the neuron that receives spikes with the shortest inter-spike interval is the winner. But it is not clear in their implementation that how starting from random weights, a new neuron can learn a new category.

We implemented a modified version of WTA with Hebbian learning to demonstrate how different neurons can learn different categories. Our algorithm can be described as follows:

- 1) If neuron  $k$  is the winner, it sends an inhibitory pulse  $V_I$  to all other neurons and a self excitatory pulse of  $V_{self}$  to itself. Winner neuron learns the stimulus by Hebbian learning in the next few spikes.
- 2) At the onset of a new stimulus, a neuron not yet fired is given a bias of  $V_B$ . This is important as otherwise the same neuron  $k$  would end up learning the same stimulus. Note that this biasing only happens for initial some spikes when there is a neuron that has still not spiked at all. This procedure can be thought of as a semi-supervised approach in which a teacher is forcing a new category to a yet un-learned neuron.

We are assuming that all membrane potentials are discharged at the onset of a stimulus. This can be achieved, for example, by a decay in the membrane potential.

#### D. Object oriented code

We have followed an object oriented approach (OOP) [40] to build the present code using C++. The OOP principles of polymorphism, encapsulation and inheritance have been followed for easier understanding and future developments. A few classes in the code are the Neuron, Layer, Network, Synapse, and the SynapseMesh class. They are shown in Fig. 1. The Neuron class and the Synapse class are the two basic classes from which other classes are derived. A Network consists of arrays of Layers and a Layer consists of 2-D arrays of neurons. The SynapseMesh class represents the connections and can implement connections of different kinds such as all to all or many to few connections. Fig. 2 shows configuration of a typical network and how synapse meshes are interleaved between layers of neurons. 2-D configuration is chosen so as to process images directly. Another advantage of using C++ is the free availability of the widely used parallel library MPI, which could be used to parallelize the code in future.

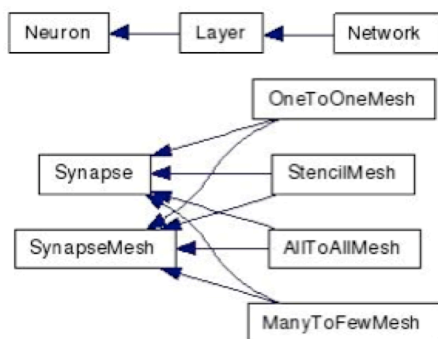


Fig. 1. Various classes in the code. Arrows denote inheritance

Another feature of the code is that it can use chars to store weights instead of floats or doubles. Chars just require 1 byte in contrast to floats or doubles, which requires 32 or 64 bytes usually. This drastically reduces the amount of

memory needed to store the synapses or weights. This is very important as neural codes usually have much more synapses than neurons due to all to all connectivity. The human brain has about  $10^{15}$  synapses, which corresponds to  $10^{15}$  Bytes of storage assuming each synapse requires one byte. The fastest supercomputer currently in the world is IBM Roadrunner [41] with  $10^{14}$  Bytes of memory. The code can convert the synaptic weights on the fly from char to floats or doubles as required by the computation.

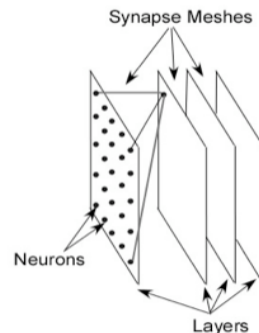


Fig. 2. A typical network showing how synaptic meshes are interleaved between 2-D arrays of neurons.

### III. TEST PROBLEM AND NETWORK STRUCTURE

We use images of 4 Gabor filters oriented at 0, 45, 90, and 135 degrees to train the network. The size of each Gabor images is 25x25 pixels. Thus, there were 625 input neurons, with each neuron connected directly to each pixel. There are 4 output neurons connected using all to all connections. There are also all to all inhibitory lateral connections in the output layer, which encourage competition among neurons.

The grey-scale intensity value of each pixel was scaled between  $I_{min}$  and  $I_{max}$ . Here  $I_{min}$  is the min. current needed to sustain spiking and  $I_{max}$  is the current corresponding to max. firing rate. This scaled current is fed as input to the input neurons. During training, each image is presented for duration of 50 msec cyclically for a total training time of 20 secs. The goal was that each output neuron “learns” a unique Gabor filter image. Once trained, we later test the network with 36 Gabor filters with orientations in step sizes of 5 degrees. The various parameters used in the networks are defined in the appendix.

### IV. RESULTS

Fig. 3 shows the voltage plots of training our spiking network with 4 Gabor filter images at 0, 45, 90, and 135 degree orientations. Each image is presented for duration of 50 msec for a total training time of 20 secs, and learning is done in an unsupervised way. Only the first 0.4 secs of the simulation is shown in the figure. Each output neuron learns to recognize a different Gabor filter and fires more vigorously as it learns to recognize more. Fig. 4 shows the voltage plots of an output neuron before and after training. It learns a particular image by firing earlier and more

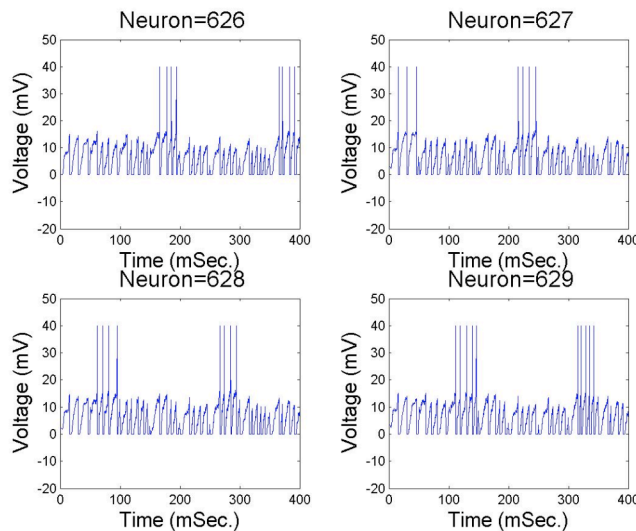


Fig. 3. Voltage plots from the 4 output neurons when presented cyclically with 4 bars of different orientations for every 50 msec. Each neuron learns a bar of different orientation and the inter-spike time decreases as the bar is being learned.

vigorously. Once trained, 36 Gabor filters at step size of 5 degrees were presented to the network. Fig. 5 shows the firing rates of 4 output neurons plotted against filters of different orientations. We observe bell-shaped tuning curves similar to the tuning curves of cells experimentally observed in V1 [42-44] and MT/V5 [45, 46] area.

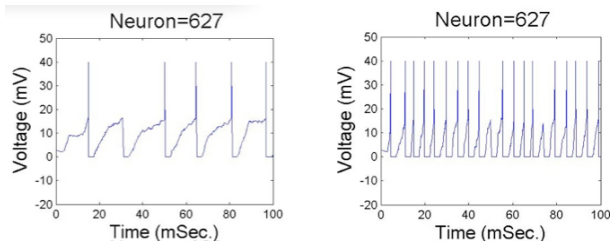


Fig. 4. Voltage of an output neuron before training (left) and after training (right)

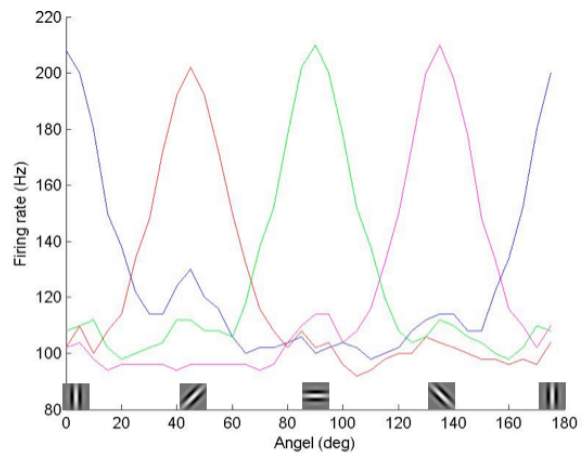


Fig. 5. Shown are the firing rates of 4 output neurons for 36 Gabor filter test images with step size of 5 degrees. Output of each neuron is shown in a different color.

Fig. 6 shows the weights associated with each of the 4 output neurons before and after training. The weights were initialized to random values before training. The weights are represented as intensities, with a weight of 1.0 corresponding to white and 0.0 corresponding to black. We can see that a unique neuron learns a unique Gabor image. But, we would like to emphasize that since the learning is unsupervised; it cannot be determined before training which neuron will learn which image. Fig. 7 shows the variation of error in the weights with number of epochs. An epoch constitutes one pass through all the training images. The error decays exponentially to zero.

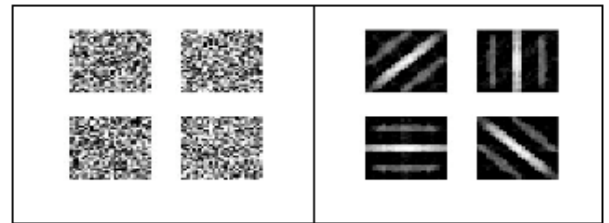


Fig. 6. Weights before training (on left) and after training (on right) for the 4 output neurons. Each output neuron learns to recognize a unique Gabor filter image.

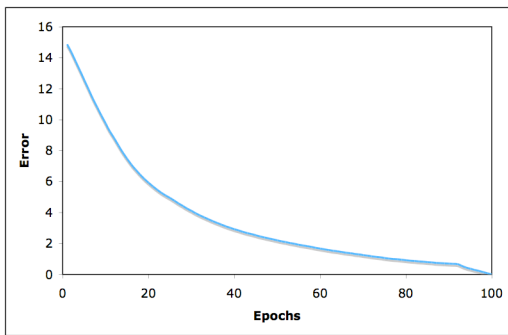


Fig. 7. Variation of error with number of epochs or training passes.

## V. CONCLUSIONS

We propose and implement an efficient and scalable learning method with winner take all for spiking neural networks. The method is implemented in an object oriented C++ code (CSpike). A network is trained with images of Gabor filters at 4 orientations and it is found that each output neuron learns a unique image. Also, upon learning an image, the neuron fires earlier and more vigorously. On testing with 36 Gabor images with different orientations, bell-shaped tuning curves similar to those of cells in the area V1 and MT/V5 of the brain are observed.

One of the advantages of our code is that it uses simple LIF model and can use only 1 byte per synapse. Thus, a large network with billions of synapses could be modeled even on a laptop. Also, we use an object oriented approach, so the code would be easy to understand, modify and extend. We intend to use C++ and MPI and run it on massively parallel supercomputers such as IBM Blue Gene in the future.

## APPENDIX

### RUNNING PARAMETERS

$R=38.3M\Omega$ ,  $C=0.207nF$ ,  $\tau=7.93ms$ ,  $V_{th}=16.4mV$ ,  
 $V_{high}=40mV$ ,  $V_{reset}=0mV$ ,  $\tau^+=15ms$ ,  $A^+=0.01$ ,  $\alpha=0.015$ ,  
 $t_{ref}=2.68mV$ ,  $dt=0.1ms$ .

### ACKNOWLEDGMENT

This research was supported by ONR Grant No. N00014-05-1-0844.

### REFERENCES

- [1] J. Vreeken, "Spiking neural networks, an introduction," Institute for Information and Computing Sciences, Utrecht University Technical Report UU-CS-2003-008, 2002.
- [2] T. Natschläger and B. Ruf, "Spatial and temporal pattern analysis via spiking neurons," *Network: Comput. Neural Syst.*, vol. 9, pp. 319-332, 1998.
- [3] C. Panchev and S. Wermter, "Spike timing dependent synaptic plasticity: from single spikes to spike trains," *Neurocomputing*, vol. 58-60, pp. 365-371, 2004.
- [4] C. Panchev and S. Wermter, "Temporal sequence detection with spiking neurons: Towards recognizing robot language instruction," *Connection Science*, vol. 18, pp. 1-22, 2006.
- [5] C. Panchev, "Spatio-temporal and multimodal processing in a spiking neural mind of a robot," Ph.D. dissertation, University of Sunderland, United Kingdom, May 2005.

- [6] A. R. Baig, "Spatial-temporal artificial neurons applied to online cursive handwritten character recognition," in *European Symposium on Artificial Neural Networks*, Bruges (Belgium), April 2004, pp. 561-566.
- [7] A. Gupta and L. N. Long, "Character recognition using spiking neural networks," in *International Joint Conference on Neural Networks*, August 2007, pp. 53-58.
- [8] T. Masquelier and S. J. Thorpe, "Unsupervised Learning of Visual Features through Spike Timing Dependent Plasticity," *PLoS Comput Biol*, vol. 3, pp. 247-57, 2007.
- [9] BRAIN, Available: <http://brain.unr.edu/ncsDocs/>
- [10] GENESIS, Available: <http://www.genesis-sim.org/GENESIS/>
- [11] NEURON, Available: <http://www.neuron.yale.edu/neuron/>
- [12] R. Brette and e. al, "Simulation of networks of spiking neurons: A review of tools and strategies," *Journal of computational neuroscience*, 2006.
- [13] SPIKENET, Available: <http://www.scen.ucsd.edu/~arno/spikenet/>
- [14] SPIKENET TECH, Available: <http://www.spikenet-technology.com/>
- [15] R. Van Rullen, J. Gautrais, A. Delorme, and S. Thorpe, "Face processing using one spike per neurone," *Biosystems*, vol. 48, pp. 229-239, 1998.
- [16] L. N. Long and A. Gupta, "Scalable massively parallel artificial neural networks," *Journal of Aerospace Computing, Information, and Communication*, vol. 5, 2008.
- [17] P. Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. New York: John Wiley & Sons, 1994.
- [18] B. Berninger and G. Q. Bi, "Synaptic modification in neural circuits: a timely action," *BioEssays*, vol. 24, pp. 212-222, 2002.
- [19] Y. Dan and M. Poo, "Spike time dependent plasticity of neural circuits," *Neuron*, vol. 44, pp. 23-30, 2004.
- [20] S. Song, K. D. Miller, and L. F. Abbott, "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity," vol. 3, pp. 919-926, 2000.
- [21] L. N. Long and A. Gupta, "Biologically-Inspired Spiking Neural Networks with Hebbian Learning for Vision Processing," in *46th AIAA Aerospace Sciences Meeting*, Reno, NV, 2008.
- [22] L. N. Long, "Scalable Biologically Inspired Neural Networks with Spike Time Based Learning," in *Invited Paper, IEEE Symposium on Learning and Adaptive Behavior in Robotic Systems*, Edinburgh Scotland, 2008.
- [23] A. L. Hodgkin and A. F. Huxley, "A quantitative description of ion currents and its applications to conduction and excitation in nerve membranes," *Journal of Physiology*, vol. 117, pp. 500-544, 1952.
- [24] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.
- [25] E. M. Izhikevich, "Which model to use for cortical neurons?," *IEEE transactions on neural networks*, vol. 15, 2004.
- [26] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*. Erlbaum Pub., 1949.
- [27] G.-q. Bi and M.-m. Poo, "Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type," *Journal of Neuroscience*, vol. 18, pp. 10464-72, 1998.
- [28] L. I. Zhang, H. W. Tao, C. E. Holt, W. A. Harris, and M.-m. Poo, "A critical window for cooperation and competition among developing retinotectal synapses," vol. 395, pp. 37-44, 1998.
- [29] W. B. Levy and O. Steward, "Temporal contiguity requirements for long-term associative potentiation/depression in the hippocampus," *Neuroscience*, vol. 8, pp. 791-97, 1983.
- [30] H. Markram, J. Lubke, M. Frotscher, and B. Sakmann, "Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs," *Science*, vol. 275, pp. 213-15, 1997.
- [31] P. Földiák, "Forming sparse representations by local anti-Hebbian learning," *Biol. Cybern.*, vol. 64, pp. 165-170, 1990.
- [32] D. J. Field, *Scale-invariance and self-similar "wavelet" transforms: An analysis of natural scenes and mammalian visual systems*. Oxford, England: Oxford University Press, 1993.
- [33] D. J. Field, "What is the goal of sensory coding?," *Neural Computation*, vol. 6, pp. 559-601, 1994.
- [34] D. I. Perrett, E. T. Rolls, and W. Caan, "Visual neurons responsive to faces in the monkey temporal cortex," *Exp. Brain Research*, vol. 47, pp. 329-342, 1982.

- [35] W. Maass, "Neural Computation with Winner-Take-All as the only Nonlinear Operation," in *Advances in Neural Information Processing Systems*, 1999.
- [36] J. Lazzaro, S. Ryckebusch, M. A. Mahowald, and C. A. Mead, "Winner-take-all networks of O(N) complexity," in *Advances in Neural information Processing Systems*, San Francisco, CA, 1989, pp. 703-771.
- [37] D. Z. Jin and H. S. Seung, "Fast computation with spikes in a recurrent neural network," *Phys Rev E Stat Nonlin Soft Matter Phys*, vol. 65, p. 051922, 2002.
- [38] M. Oster and S. C. Liu, "A Winner-take-all Spiking Network with Spiking Inputs," in *11th IEEE International Conference on Electronics, Circuits and Systems (ICECS)* pp. 203- 206, 2004.
- [39] R. J. Douglas and K. A. C. Martin, *Neuronal Circuits of the Neocortex* vol. 27, 2004.
- [40] D. Yang, *C++ and Object-oriented Numeric Computing for Scientists and Engineers*. New York: Springer, 2001.
- [41] IBM Roadrunner, Available: <http://www.lanl.gov/orgs/hpc/roadrunner>
- [42] G. N. Henry, P. O. Bishop, R. M. Tupper, and B. Dreher, "Orientation specificity and response variability of cells in the striate cortex," *Vision Research*, vol. 13, pp. 1771-1779, 1973.
- [43] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *Journal of Physiology*, vol. 195, pp. 215-243, 1968.
- [44] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurons in the cat's striate cortex," *Journal of Physiology*, vol. 148, pp. 574-591, 1959.
- [45] A. Georgopoulos, J. Kalaska, R. Caminiti, and J. Massey, "On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex," *J. Neurosci.*, vol. 2, pp. 1527-1537, 1982.
- [46] J. H. Maunsell and D. C. Van Essen, "Functional properties of neurons in middle temporal visual area of the macaque monkey. I. Selectivity for stimulus direction, speed, and orientation," *J Neurophysiol*, vol. 49, pp. 1127-1147, 1983.