Long, Lyle N., "Scalable Biologically Inspired Neural Networks with Spike Time Based Learning," Invited Paper, IEEE Symposium on Learning and Adaptive Behavior in Robotic Systems, Edinburgh, Scotland, Aug. 6-8, 2008.

Scalable Biologically Inspired Neural Networks with Spike Time Based Learning

Lyle N. Long

The Pennsylvania State University
229 Hammond Building
University Park, PA 16802
LNL@PSU.EDU

and

The California Institute of Technology
200 Beckman Behavioral Biology Building
Pasadena, CA 91025
LNL@CALTECH.EDU

Abstract

This paper describes the software and algorithmic issues involved in developing scalable large-scale biologically-inspired spiking neural networks. These neural networks are useful in object recognition and signal processing tasks, but will also be useful in simulations to help understand the human brain. The software is written using object oriented programming and is very general and usable for processing a wide range of sensor data and for data fusion.

1. Introduction

Artificial neural networks (ANN) can broadly be classified into three generations [1, 2] :

- 1st generation had neurons which restricted the output signals to discrete '0' or '1' values.
- 2nd generation models used a continuous activation function allowing for the output to take values between '0' and '1' and used training methods such as back propagation.
- 3rd generation are time-dependent spiking neural networks, as discussed herein.

In a previous paper [3], we showed how the 2nd 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). MPI was used to parallelize the C++ code. The back-propagation algorithm was used to train the network. The training of a problem size with 2 billion neuron weights on an IBM BlueGene/L computer using 1000 dual PowerPC 440 processors required less than 30 minutes. Various comparisons in training time, forward propagation time, and error reduction were also shown. These, however, are not biologically plausible neural networks, even though they have proven useful in numerous engineered systems.

Spiking neural networks belong to the 3rd generation of neural networks and, like their biological counterparts, use spikes or pulses to represent information flow. Information is encoded both in the timing as well as the rate of spiking. The motivation behind exploring the spiking neuron models is that temporal information can also be encoded in the input signals and multiplexing can be achieved using pulse coding. Also, spiking ANNs are more biologically plausible than traditional ANNs. The software required for these neural networks are no more difficult to parallelize than that used for the 2nd generation networks, and therefore massive parallelization is not discussed here. The code developed here is meant to run on multi-processor or multi-core computers using threads. This type of machine is now being used as the nodes of massively parallel computers. A $3000 dual quad-core computer is now roughly the same speed as a 1,000-processor machine from 10 years ago.

In [4], a spiking neural network model was developed to identify characters in a simple character set. Spike time dependent plasticity (STDP) was used for training. These were small-scale simulations programmed in Matlab. The emphasis in the present work is in developing large-scale and scalable systems that can be used for very large neural networks.

A number of other software systems have been developed in the past for simulation of spiking neural networks. They vary in the degree to which they represent biologically realistic neurons, support parallel computing, their complexity, operating system support, performance, etc. Some examples are GENESIS [5], PGENESIS [6], the neo-cortical simulator (NCS) [7], the Neural Simulation Tool (NEST) [8], spikeNET [9], NEURON [10], and Neocognitron [11]. Most of these are medium to high in biological realism [12]. Systems which are more biologically meaningful tend to be more complex, and require more memory and CPU time.

2. Biological Neural Networks

The human brain is the most complicated systems in the known universe. It has roughly 10^{11} neurons and 10^{15} synapses. The number of neurons in the human brain is roughly the same as the number of stars in the Milky Way galaxy. If each synapse represents a byte, then the storage capacity of the brain is 10 times larger than the data in the Library of Congress (500 terabytes). These are staggering numbers, and it would be extremely difficult to simulate this on even the largest current supercomputers. Even more daunting than the size of the problem, however, is the "wiring diagram." Each neuron can be connected to up to 10,000 other neurons, there are feedback connections, and there are different kinds of neurons. In addition, the number of input sensors and output paths are about six orders of magnitude larger than so-called advanced robots.

Figure 1 shows the memory and speed of several biological and human-built systems. The largest computer in the world (at the moment) is the IBM BlueGene [13] with 212,992 processors, which is roughly equivalent to the speed and power of the human brain. This is an overly optimistic comparison however, since the brain is much more complex than a simple feed-forward neural network. In addition, as shown in Figure 2, the power requirements and volume required of the supercomputer are about six

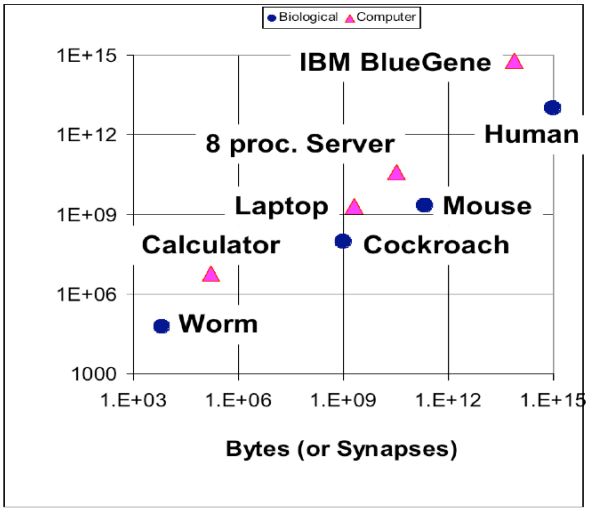

Figure 1. A comparison of biological and computer system speed and memory.

orders of magnitude larger than the human brain. And finally, few people (if any) ever have access to all the processors in this machine at one time. Researchers rarely get more than 500 processors on any supercomputer. As shown in the Figure, however, there are relatively powerful servers (e.g. dual quad-core computers with 32 GB RAM) that have nearly the speed and memory of a

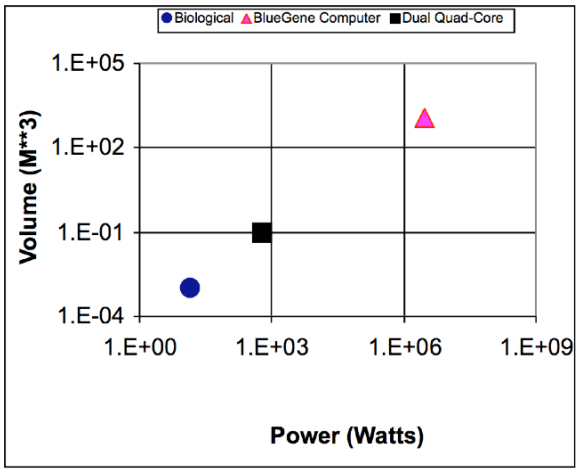

Figure 2 Power and volume required by computers and human brain.

mouse brain (roughly 10^7 neurons and 10^{11} synapses). So a great deal of progress can be made by simply using quad-core chips and thread-based programming.

3. Computational Algorithms

Some of the main issues in developing scalable biologically-inspired neural networks are:

- Neuron modeling
- Synapse modeling
- Connectivity or wiring diagrams
- Learning
- CPU time and memory requirements

Some of the issues related to these topics are discussed below and in the paper by Long and Gupta [1].

3.1 Neuron Modeling

One of the most widely known and detailed models of a biological neuron is due to Hodgkin and Huxley (H-H) [1, 14, 15]. This model is often discussed in neuroscience, but is seldom used in engineering applications since it is very expensive to simulate. This model uses four coupled, nonlinear ordinary differential equations to model each neuron. It must be remembered, however, that the number of neurons in the brain is roughly 10,000 times fewer than the number of synapses.

Another well known, but much simpler, model of a neuron is the leaky integrate and fire (LIF) neuron [1, 14]. For neuron i in a system of neurons, this can be modeled using:

$$\frac{dv_i}{dt} = \frac{1}{RC}\left[(I_{input} + I_i)R - v_i\right] \qquad (1)$$

where v is voltage, R is resistance, C is capacitance, and I_{input} is the input current (usually zero), and I_i is the current from the synapses (discussed in the next section). Table 1 shows some typical values for these parameters.

Table 1. Typical values for LIF neuron model.

Parameter	Typical Values
v	50 millivolts
R	40 megaOhms
C	0.2 nanoFarads
I	0.5 nanoAmps
RC	8.0 milliSeconds
Δt	0.1 milliSeconds

Long and Gupta [1] showed that this method is about 10,000 times faster than solving the H-H equations, but yet the solution is often quite similar. The details of the neuron behavior is probably not crucial if one is primarily interested in developing engineering neural network systems. Also, simple Euler explicit methods are often used to integrate the equations, and it isn't clear that higher order methods will be useful [1] since the solutions are not smooth.

3.2 Synapse Modeling

The neurons connect to one another through synapses. If a neuron fires, a charge is transmitted to the other neurons through the synapses. The current can be modeled as:

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

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 α, represent the synapse weights and the increment of current injected per spike. The weights take on values from -1.0 to 1.0. This current has a 1/N scaling so that consistent behavior is obtained no matter how many synapses a neuron has connected to it. This is somewhat realistic biologically since neurons and synapses have limited surface areas and volumes.

3.3 Connectivity or Wiring Diagrams

The neural connections in the human brain are far more complex than most artificial neural networks. Figure 3 shows an image of the human frontal cortex from Cajal [16]. Most neural network software is restricted to fairly simple feed-forward connections, with possibly some recurrent connections.

For software with truly arbitrary connections, one could allow each synapse to store pointers to the two neurons that connect to it, but this would require two 64-bit numbers for each synapse. For a network with a billion synapses (e.g. a cockroach), this would require 16 gigabytes of data just for these two pointers. The software developed here stores only 1 byte per synapse (for the synaptic weight).

An alternative approach to simulating complex networks is used here. Each neuron stores five integers (20 bytes/neuron). In addition, the network here has distinct layers, possibly six layers like the neocortex. These layers each contain a 2-D array of neurons, so the neurons in each layer can be addressed using i, j indices; and the layer can be addressed with an index k. The five integers that each neuron stores for its connectivity data are: imin, imax, jmin, jmax, and k. So each neuron can connect to a rectangular patch of neurons in any other layer (including recurrent connections). This offers a great deal of flexibility in defining "wiring diagrams" with minimal memory.

3.4 Learning

The learning approach used here is based on the ideas that Hebb proposed in 1949 [17]:

> *"When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."*

Experimental evidence has proven the above idea to be quite accurate [18]. He goes on to say later in the book:

> *"The old, long-established memory would then last, not reversible except with pathological processes in the brain; less strongly established memories would gradually disappear unless reinforced."*

which has also been shown to be true. This second phrase is as important as the first one but is seldom mentioned, partly because the book was fairly difficult to find for many years. Hebb's book has been called the second most important biology book. Darwin's book [19] is clearly first, and it is brilliantly discussed relative to intelligence and consciousness by Dennett [20, 21] and Pinker [22]. While philosophers have debated the subject of consciousness for more than 2000 years, it is now clear that intelligence and consciousness are emergent properties of the neural and synaptic systems in the human brain [20, 23, 24].

Spike-time dependent plasticity (STDP) [4, 25] is one form of Hebbian learning and has been observed by neuroscientists [18]. In one form of STDP the synaptic weights are assumed to change according to:

$$\Delta w = \begin{cases} A^+ e^{(\Delta t / \tau^+)} & , \quad \Delta t \le 0 \\ -A^- e^{(-\Delta t / \tau^-)} & , \quad \Delta t > 0 \end{cases} \qquad (3)$$

where the variables are typically: τ=20 ms, A^+=0.005, A^-=0.006, and $\Delta t=(t_{pre} - t_{post})$ 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

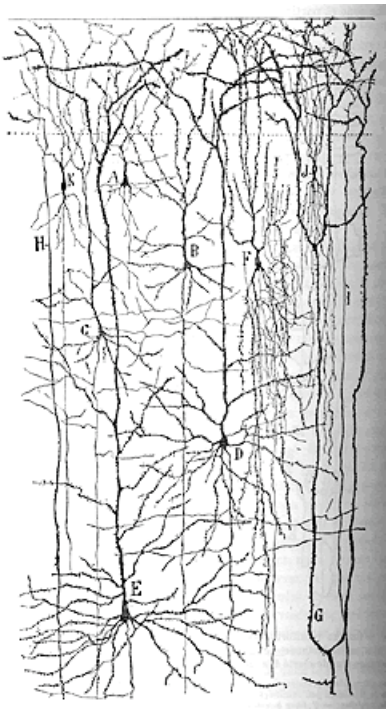

Figure 3 Drawing from Cajal of human frontal cortex.

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. This is one example of Hebbian-style learning, the increase in synapse weights is larger if the two neurons fire closer together.

One approach to implementing the weight change is to use:

$$w = \begin{cases} w_{old} + \Delta w \left(w_{\max} - w_{old} \right) & , \quad \Delta w \geq 0 \\ w_{old} - \Delta w \left(w_{\min} - w_{old} \right) & , \quad \Delta w < 0 \end{cases} \tag{4}$$

But there are many possible ways of implementing the change. The above form allows the use of a minimum and maximum weight, but it is not very robust computationally. The weights often simply end up at the minimum or maximum values or fluctuate endlessly. Also, as the weight approaches either the minimum or the maximum the relative weight change gets reduced.

The above algorithm is not necessarily the optimal learning approach for spiking networks, even though it has worked well on a number of applications and Eqn. (3) has been shown to roughly agree with experimental data.

One issue with STDP involves causality. When a post-synaptic neuron fires in a time-marching code, 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, computational work, and scalabilty are not an issue, but for millions of neurons and billions of synapses we need extremely efficient algorithms and minimal gather/scatter operations (i.e. neurons sending messages to both pre- and post-synaptic connections). The above is relatively straight-forward to implement for small neural networks, but for large systems where memory and computations must be minimized, it is not necessarily effective. This is especially true for complex wiring diagrams, connections that span multiple layers of the network, and connections that have delays.

Hebbian learning based on spike timing is implemented very efficiently in the software developed here. It is implemented in essentially an "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 (but it does not need to know who its post-synaptic connections are). 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 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, and it is scalable and efficient.

The other key to a robust spike-time learning algorithm is homeostatic behavior. This is true for biological and computational systems. Homeostasis is defined as "a relatively stable state of equilibrium." This prevents all the synaptic weights from becoming very large or very small. In the present code, the sum of all the synaptic weights for a particular neuron remains constant.

4. Software approach

The software developed here uses an object-oriented programming (OOP) approach. It is programmed in Java, but it could be easily changed to C++ . We have had a lot of success using the OOP approach for a wide variety of scientific computing applications [26-28]. Long [29] discusses the importance of software engineering, and the need for increased education in this area. The OOP approach (encapsulation, polymorphism, and inheritance) allows one to develop very understandable and maintainable code.

Java has many of the advantages of C++, but without many of the problems of C++. It has many features built

into the language that few other languages can claim, such as threads, exception handling, OOP, graphics, graphical user interfaces, and remote method invocation (RMI). Early Java implementations were fairly inefficient, due to immature compilers and being run in interpreted mode, but now the speed of Java is often equal or close to C++. Reference [1] showed that Java was only 7% slower than C++ for a linear algebra task.

Using an OOP approach allows one to efficiently develop very complex software systems. In this case, we were able to first develop a Neuron class, and we could thoroughly debug that piece before moving on to the higher level functions (Layer class, Network class, and Vision class) of the code. The ability to encapsulate data and methods is especially useful.

Figure 4 shows the time history of neuron voltages in a neural network that used a webcam to provide the input to the first layer of neurons.

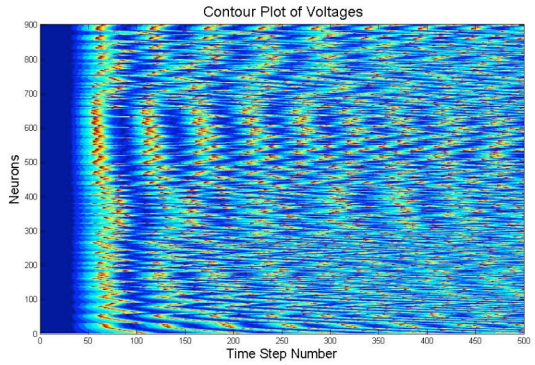

Figure 4. Neuron output.

5. CPU Time and Memory Requirements

Figure 5 shows the performance of the code, running on a 2.4 GHz (Intel Core 2 Duo) MacBook Pro laptop with 2 GB RAM using Java Version 1.5.0_13. The code was run for 250 time steps (67 milliseconds of simulation time). The networks had three layers. The neurons in the first layer each had only one synapse, but in layers 2 and 3 all the neurons were connected to all the neurons in the previous layer. The number of neurons in these simulations ranged from 300 to 67,500. Figure 5 shows the CPU time vs. number of synapses. Other than for very small problems, the performance is linear with the number of synapses. In addition, it shows that due to the low memory requirements the laptop can run up to a billion synapses (which is roughly equivalent to a cockroach). The largest case had each neuron connected to up to 22,000 other neurons. The human brain has neurons with up to 10,000 connections each.

Also shown in Figure 5 are cases that performed the Hebbian learning algorithm. It is extremely encouraging to see that the learning case required essentially the same amount of CPU time/time step as the cases with no learning. The performance of the code is linear with number of neurons also (with no synapses).

The software was designed to use as little memory and CPU time as possible. Since we expect to use hundreds or thousands of synapses/neuron, the synapses dominate both the memory and CPU requirements. Also, in order to minimize memory used, the synapse weights are stored as byte values. When they are needed for calculations, they are converted to floats. Also, the spikes occur relatively rarely, so most of the time steps can be computed quite rapidly with roughly one floating point operation/synapse/step plus some logical operations. If a spike occurs there are additional operations, and if learning is turned on there are some additional operations, but these are relatively rare events. If a neuron spikes at a rate of 100 Hz and the time step size is 0.1 milliseconds, then there is roughly one spike every 100 time steps.

7. Conclusions

In this paper the algorithms and software for spiking neural network simulations have been described. These neural networks offer the promise of better computer vision systems, as well as the hope of increasing our understanding of biological systems. Some preliminary results were included to demonstrate the algorithms and software. These neural networks can be designed to require minimal memory and processing per synapse, but they do require a large number of very small time steps to march the solutions forward in time. We plan to parallelize these codes and to apply them to more complicated applications in the near future.

8. Acknowledgements

I would like to gratefully acknowledge the California Institute of Technology, where most of this work was performed, for supporting me as a Moore Distinguished Scholar. The Office of Naval Research (Grant No. N00014-05-1-0844) also supported this work and is gratefully acknowledged.

9. References

[1] 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: AIAA, 2008.

[2] J. Vreeken, "Spiking neural networks, an introduction," Institute for Information and Computing Sciences, Utrecht University Technical Report UU-CS-2003-008, 2002.

[3] L. N. Long and A. Gupta, "Scalable massively parallel artificial neural networks," *Journal of Aerospace Computing, Information, and Communication,* vol. 5, Jan., 2008.

[4] A. Gupta and L. N. Long, "Character recognition using spiking neural networks," in *International Joint Conference on Neural Networks*, Orlando, FL, 2007, pp. 53-58.

[5] GENESIS, http://www.genesis-sim.org/, Dec. 31, 2007.

[6] R. D. Traub and etal, "A single-column thalamocortical network model exhibiting gamma oscillations, sleep spindles and epileptogenic bursts," *Journal of Neurophysiology,* vol. 93, pp. 2194-2232, 2005.

[7] NeoCortical Simulator, http://brain.unr.edu/ncsDocs/, Dec. 31, 2007.

[8] NEST, www.nest-initiative.uni-freiburg.de, Dec. 31, 2007.

[9] SpikeNet, http://www.sccn.ucsd.edu/~arno/spikenet/, Dec. 31, 2007.

[10] NEURON, http://www.neuron.yale.edu/neuron/, Dec. 31, 2007.

[11] K. Fukushima, "Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition," *Neural Networks,* vol. 1, pp. 119-130, 1988.

[12] T. P. Weldon, W. E. Higgins, and D. F. Dunn, "Gabor filter design for multiple texture segmentation," *Optical Engineering,* vol. 35, pp. 2852-2863, 1996.

[13] TOP500 Computer List, http://www.top500.org/, Dec. 31, 2007.

[14] C. Koch, *Biophysics of Computation: Information Processing in Single Neurons*: Oxford Press, 1999.

[15] 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.

[16] S. Ramón y Cajal, http://nobelprize.org/nobel_prizes/medicine/articles/cajal/,

[17] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*: Erlbaum Pub., 1949.

[18] G. Bi and M. Poo, "Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type," *Journal of Neuroscience,* vol. 18, pp. 10464-10472, 1998.

[19] C. Darwin, *The Origin of Species*: (available at books.google.com), 1859.

[20] D. C. Dennett, *Consciousness Explained*: Back Bay Books, 1992.

[21] D. C. Dennett, *Darwin's Dangerous Idea: Evolution and the Meanings of Life*: Simon and Schuster, 1996.

[22] S. Pinker, *How the Mind Works*: Norton & Company, 1999.

[23] L. N. Long, T. D. Kelley, and M. J. Wenger, "The Prospects for Creating Conscious Machines," in *Toward a Science of Consciousness Conference*, Tucson, AZ, 2008.

[24] C. Koch, *The Quest for Consciousness: A Neurobiological Approach*: Roberts and Company, 2004.

[25] S. Song, K. D. Miller, and L. F. Abbott, "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity," *Nature Neuroscience,* vol. 3, pp. 919-926, 2000.

[26] T. E. Fritz and L. N. Long, "Object-Oriented Unsteady Vortex Lattice Method for Flapping Flight," *Journal of Aircraft,* vol. 41, 2004.

[27] S. D. Hanford, O. Janrathitikarn, and L. N. Long, "Control of a Six-Legged Mobile Robot Using the Soar Cognitive Architecture," in *46th AIAA Aerospace Sciences Meeting, AIAA Paper No. 2008-0878*, Reno, NV, 2008.

[28] P. D. O'Connor, L. N. Long, and J. B. Anderson, "The Direct Simulation of Detonations (Invited Paper)," in *AIAA Joint Propulsion Conference (AIAA Paper No. 2006-4411)*. vol. Paper 2006-4411 Sacramento, CA: AIAA, 2006.

[29] L. N. Long, "The Critical Need for Software Engineering Education," *CrossTalk,* vol. 21, pp. 6-10, 2008.