Learning in a Distributed Software Architecture for Large-Scale Neural Modeling

Jasmin Léveillé[1], Heather Ames[2], Benjamin Chandler[2], Anatoli Gorchetchnikov[1], Ennio Mingolla[2], Sean Patrick[1], and Massimiliano Versace[2]

[1] Department of Cognitive and Neural Systems, Boston University, Boston, MA 02215
[2] Center of Excellence for Learning in Education, Science, and Technology, Boston University, Boston, MA 02215
{jasminl,anatoli,versace,ennio}@cns.bu.edu,
{heather.m.ames,sean.patrick.619,bchandle}@gmail.com

Abstract. Progress on large-scale simulation of neural models depends in part on the availability of suitable hardware and software architectures. Heterogeneous hardware computing platforms are becoming increasingly popular as substrates for general-purpose simulation. On the other hand, recent work highlights that certain constraints on neural models must be imposed on neural and synaptic dynamics in order to take advantage of such systems. In this paper we focus on constraints related to learning in a simple visual system and those imposed by a new neural simulator for heterogeneous hardware systems, *CogExMachina* (Cog).

Keywords: Large-scale system, learning laws, neural networks, neural network software, heterogeneous computing.

1 Introduction

Building neural models capable of simulating complex behaviors requires simulating large-scale models linking the macro-scale of behavior to the micro-scale of individual neural events. In recent years a large number of simulation platforms have emerged that satisfy different modeling needs, ranging from simulators that include the low-level spiking behavior of individual neurons [1] to some that abstract away biological details about neurons but try to maintain some of their functions [2]. Although large-scale neural modeling has greatly benefited from the introduction of supercomputers [3], the potential of heterogeneous computing systems still remains largely unexplored. Heterogeneous computing offers many advantages for neural modeling, including the possibility to scale simulations at virtually no cost, and to employ a variety of hardware accelerators to optimize specific model components. On the other hand, heterogeneous computing severely constrains the design of general-purpose neural network modeling software. This paper addresses the issue of how to embed learning in such a system based on our experience with *CogExMachina* (Cog), a new neural simulator designed for heterogeneous computing systems [4].

J. Suzuki and T. Nakano (Eds.): BIONETICS 2010, LNICST 87, pp. 659–666, 2012.

We start in the next section with a brief description of the general modeling framework imposed by Cog. Section 3 examines in more detail the constraints that relate specifically to learning and presents simulation results of a model of the early visual system able to learn orientation preference and ocular dominance maps. We conclude with a description of our current efforts on generalizing the learning capabilities of the system.

2 Neural Modeling Framework

Single neurons in Cog are implemented based on the following neuron model [5]:

$$y = f\left(\mathbf{w}^{\mathbf{T}}\mathbf{x}\right), \tag{1}$$

where $\mathbf{x}$ and $\mathbf{w}$ are the presynaptic input and associated synaptic weight vectors, respectively, f is a scalar-valued activation function, and y is the activation value of the postsynaptic neuron. The product $\mathbf{w}^{\mathbf{T}}\mathbf{x}$ is referred to as the *partial inference*. Cog imposes virtually no restrictions on the choice of the activation function, thereby leaving the modeler free to determine the kind of computation performed by each neuron. However, external input is obtained via partial inference only. This restriction ensures that models implemented remain tractable and efficient by removing the need for sophisticated synchronization mechanisms to handle parallelism.

The segregation of computation into a set of partial inferences followed by an activation function is a critical bridge between biological and silicon computation built into Cog. A neural population can maintain relatively little state, but perform potentially highly nonlinear computations. The web of dendrites feeding that population has states stored in each synapse, but computes in a much more rigid manner.

This dual-natured computation maps extremely well to heterogeneous computers. Conventional general-purpose processors can only work on relatively small sets of data efficiently, but include highly robust strategies for handling irregularity and nonlinearity. Special-purpose accelerators like graphics processors are designed for an opposite set of constraints. They require dramatically higher memory bandwidth, but de-prioritize handling of irregular computation. The signal function component of computation is best mapped to a conventional processor, where the partial inference calculation maps efficiently to a graphics processor.

Beyond graphics processors, single-purpose hardware offers multiple additional orders of magnitude in power efficiency. Graphics processors are vastly more efficient than conventional processors for computations like partial inference, but data is still not as physically local as needed for power efficiency rivaling biology. A graphics card includes its own bank of memory with a very wide connection to the processor, but these two components are still physically separated. This separation means data must be shuttled from a memory chip, across a bus, and finally in to the processor. Efficiency would be dramatically higher if memory could be co-located directly on the processor.

Memristive crossbar memory is a viable contender for this unification of processing and memory. Memristor crossbars can be manufactured directly on top of a conventional chip, but with dramatically higher density than existing memory technologies. This means a massively multi-core chip designed to handle partial inference can localize storage of the weights directly on top of the processing core performing calculations – in particular, partial inference and learning - with those weights. Weight data need not move more than a few millimeters, resulting in a massive increase in energy efficiency. Cog's design is in part motivated by such considerations of locality [4].

In summary, the modeling framework provided by Cog limits communication between neurons to partial inferences and tries to maintain computation related to synaptic weights local to the partial inference processors. The next section addresses the impact of these constraints on learning.

3 Learning

The above considerations lead to the question of how to efficiently introduce synaptic weight learning in a large-scale model instantiated on a distributed, heterogeneous network. One way to answer this question is to restrict learning to a single learning rule which can then be optimized in the same hardware that computes partial inference. From this point of view, one good candidate learning rule might be spike-timing-dependent-plasticity [6] since it appears to give the most complete account of biological synaptic modification mechanisms. Indeed, some large-scale models adopt this rule as their only learning mechanism [7]. However, in a system in which biological realism is only secondary to functionality, such a choice is no longer justified. Moreover, it is fundamentally impossible to determine in advance which learning law is most appropriate without knowing exactly which behavioural task the model needs to perform. Finally, even if the task is known, it is not always clear which learning law will perform best, and if so in what parameter range can it be expected to perform well. Thus, learning in Cog follows the approach of implementing only a generic form of the learning equation which can then be tailored for specific applications. Crucially, the use of a generic form allows for hardware acceleration on the same processors that also compute partial inferences.

3.1 Current General Form of Learning Laws

Cog currently supports learning laws for which weight changes can be implemented in the following general form [8]:

$$\Delta w_{ij} = \lambda s\left(\Delta w_{ij}^{H} + \Delta w_{ij}^{C} + \Delta w_{ij}^{N}\right), \tag{2}$$

where λ is a learning rate, s is a sign factor (-1 or +1), and Δw_{ij}^{H}, Δw_{ij}^{C} and Δw_{ij}^{N} are weight-change terms related to Hebbian, competitive and normalization operations respectively. Presynaptic and postsynaptic units are respectively denoted in Eq. 2 by

indexes *i* and *j*. This general form was shown in [8] to be able to encapsulate a number of learning rules performing independent component analysis and is implemented in Cog as the following sequence of three steps:

$$w_{ij}^1 \leftarrow w_{ij}^0 + h_j x_i \ , \tag{3}$$

$$\begin{aligned} x'_i &\leftarrow w_{ij}^1 h_j \\ w_{ij}^2 &\leftarrow w_{ij}^1 - h_j x'_i \end{aligned} \ , \tag{4}$$

$$w_{ij}^3 \leftarrow w_{ij}^2 - g_j w_{ij}^2 \ . \tag{5}$$

Eqs. 3, 4 and 5 implement the Hebbian, competitive and normalization steps, respectively. Quantities h_j and g_j incorporate the learning rate λ and sign *s* and are computed by the postsynaptic neuron y_j at each time step. Crucially, the forms of h_j and g_j are determined by the user so as to implement a particular learning rule.

Simulation flow in Cog can thus be described as a sequence of two operations performed at each time step. First, all partial inferences are computed and learning is performed based on Eqs. 2-5 with feedback terms h_j and g_j returned from postsynaptic neurons computed at the previous time step. Second, all activation functions are computed as well as feedback terms h_j and g_j, which are sent back to the partial inference processors to be used at the next time step.

3.2 Examples

The simplest example of a learning law that can be implemented in Eqs. 4-6 is Hebbian learning:

$$\dot{w}_{ij} = \lambda x_i y_j \ , \tag{6}$$

where continuous-time notation is used for simplicity. A sequential implementation of this learning rule is easily obtained by setting $h_j = \lambda y_j$ in Eq. 3 and skipping the remaining steps. A slightly more complicated learning rule is the *instar* law [9]:

$$\dot{w}_{ij} = y_j \left(\lambda x_i - \alpha w_{ij}(t) \right) \ , \tag{7}$$

where learning is gated by postsynaptic activity y_j and an additional decay rate, α, needs to be specified by the modeler. Another common law used in the literature on self-organization is the Hebbian rule with postsynaptic normalization [10, 11]:

$$\dot{w}_{ij} = \frac{w_{ij} + \alpha x_i y_j}{\sum_k \left(w_{kj} + \alpha x_k y_j \right)} - w_{ij} \ . \tag{8}$$

Eq.8 can be implemented in Cog as follows. First, the Hebbian step is computed as $h_j = \lambda y_j$. The competitive step is then skipped by setting $w^2_{ij} = w^1_{ij}$, and the normalization step is computed as follows:

$$g_j = 1 - \frac{1}{\sum_k \left(w^0_{kj} + \alpha x_k y_j \right)} = 1 - \frac{1}{1 + \alpha y_j \sum_k x_k} \ , \tag{9}$$

where the fact that all weights sum to 1 (due to postsynaptic normalization) is used to simplify the denominator. Still, Eq. 9 implies that postsynaptic unit y_j needs to have access to the unweighted sum of its presynaptic inputs, requiring additional data transfers between processors computing the signal functions. Nevertheless, this can be accomplished in Cog by allocating an additional synaptic projection consisting of constant unit weights (i.e. $w'_{ij} = 1$) connecting presynaptic inputs x_i to postsynaptic neurons y_j. This costly duplication of weights illustrates well that any learning rule that does not comply naturally with the general form in Eq. 2 cannot be efficiently implemented in a large-scale, distributed, heterogeneous system.

3.3 Simulations

Learning of orientation preference and ocular dominance maps was implemented in Cog as a preliminary test of the simulation framework. The model implemented here is based on the LISSOM architecture [11] which builds on a widely popular tradition of research starting with the model in [10]. Network topology consists of two retinas, each projecting to distinct populations of on-center off-surround and off-center on-surround LGN cells (Fig.1a). Projections to area V1 consist of bottom-up inputs from each LGN population as well as horizontal excitatory and inhibitory connections from within V1. Details about the model and training procedure can be found in [11]. The network was trained using the Hebbian rule with postsynaptic normalization (Eq. 8) which implements competition between synaptic weights and has been shown to be efficient at learning cortical maps.

Figs. 1b-d illustrate the effect of learning using randomly presented inputs on the distribution of ocular dominance across V1 cells, where 0 indicates a cell sensitive to the left eye and 1, a cell sensitive to the right eye. Before learning, all cells are clustered at the center of the *x*-axis, indicating no strong preference for any eye (Fig.1b). After a period of normal rearing, a large proportion of cells become more selectively tuned to a particular eye (Fig.1c). In contrast, if the network is trained with inputs to the right eye only, all cells become sensitive to the right eye (Fig. 1d).

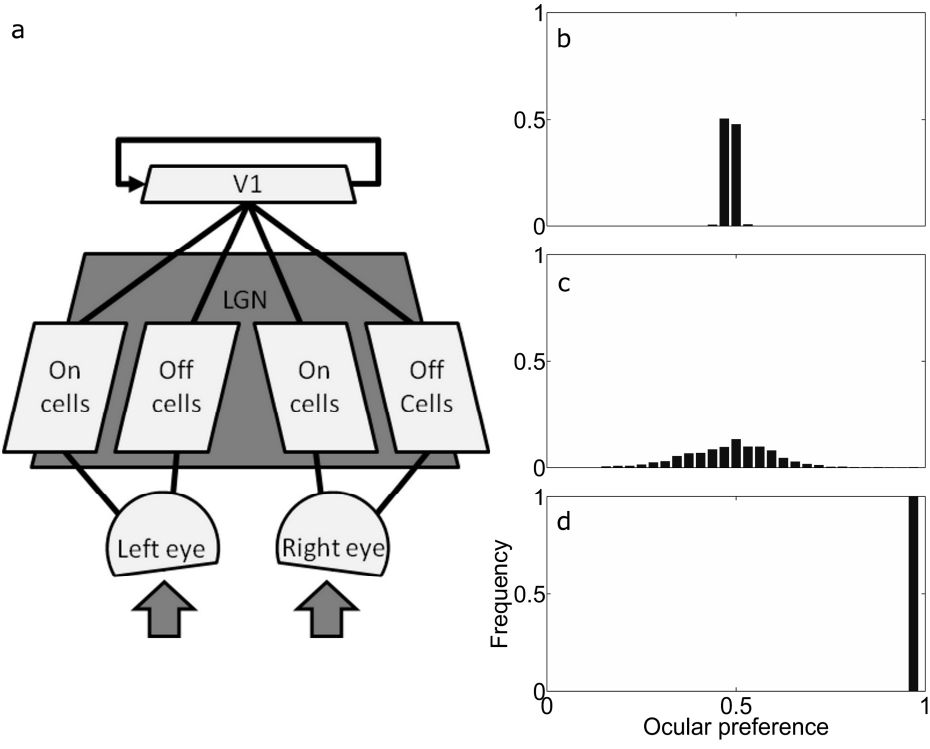

Fig. 1. a) LISSOM model architecture for orientation preference and ocular dominance map development. *b)* Distribution of ocular dominance before learning (0: left eye selectivity; 1: right eye selectivity). *c)* Distribution after a period of binocular training. *d)* Distribution after monocular training.

Figure 2 shows the topography of orientation maps and ocular dominance maps before learning (*a* and *c*, respectively) and after a period of learning (*b* and *d*, respectively).

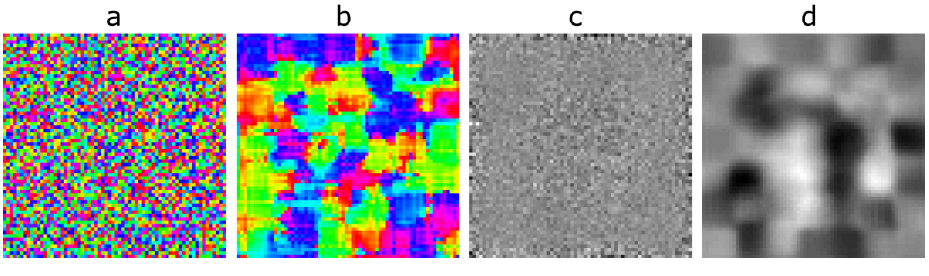

Fig. 2. Topography of learned feature maps. Panels *a)* and *c)* respectively display orientation and ocular dominance maps before learning. Panels *b)* and *d)* display corresponding maps after a period of learning. In *a)* and *b)*, color codes for preferred orientation, and in *c)* and *d)* grayscale values indicate ocular dominance.

As a large-scale learning experiment in Cog, the LISSOM model can be criticized on the grounds that it does not lead to stable representations without concurrently decreasing the learning rate, and that it requires a duplication of weights to implement synaptic weight normalization (Eq. 9). Both issues remain important topics which we are addressing in our current modeling efforts. In particular, on the second issue, the instar rule (Eq. 7) can be shown to lead to normalization based on postsynaptic activity only, thereby escaping the duplication of weights, but requires that inputs be normalized to a fixed constant. Although this assumption does not hold in the LISSOM model, it remains to be shown whether strict normalization is necessary in general. Another candidate learning rule is the BCM law which also implements competition based only on postsynaptic quantities, and also displays stability [12].

3.4 Toward a Generalized Learning Law Equation

In order to allow for a more thorough study of learning in large-scale systems, the general form in Eq. 2 and its associated three-step procedure in Eqs. 3-6 must be generalized to encompass a wider class of learning rules. For example, the *outstar* learning rule [13]:

$$\dot{w}_{ij} = x_i\left(\lambda y_j - \alpha w_{ij}(t)\right), \tag{10}$$

cannot be directly mapped to the existing learning procedure due to the multiplication of weights w_{ij} by the inputs x_i. As in the case of Hebbian learning with postsynaptic normalization, Eq.10 can be implemented by a suitable modification of the network topology, but this would reduce the efficiency of the framework.

Our group recently introduced a new general form of learning law capable of handling several classes of learning rules, including Hebb rule derivatives, threshold-based rules, input reconstruction-based rules and trace-based rules [14]. Crucially, this generalization is achieved by inserting only one additional postsynaptic feedback term to the already existing two terms (h_j and g_j) of the current procedure, making it a suitable candidate for hardware acceleration as mentioned in Section 2.

4 Conclusions

In this paper, we investigated the kind of issues that are likely to be faced when implementing learning in a large-scale neural model instantiated on a distributed, heterogeneous network. In particular, we argue that learning calculations should be local to the partial inference processors, should minimize data transfer to and from signal function processors, and should be implementable in a common general form equation. Our current efforts are aimed at implementing such a general form, which will facilitate the study of learning in large-scale neural network.

References

1. Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J.M., Diesmann, M., Morrison, A., Goodman, P.H., Harris Jr., F.C., Zirpe, M., Natschlager, T., Pecevski, D., Ermentrout, B., Djurfeldt, M., Lansner, A., Rochel, O., Vieville, T., Muller, E., Davison, A.P., El Boustani, S., Destexhe, A.: Simulation of networks of spiking neurons: A review of tools and strategies. J. Comp. Neurol. 23, 349–398 (2007)
2. O'Reilly, R.C., Munakata, Y.: Computational Explorations in Cognitive Neuroscience: Understanding the Mind by Simulating the Brain. MIT Press (2000)
3. Markram, H.: The blue brain project. Nat. Rev. Neurosci. 7, 153–160 (2006)
4. Snider, G.: Intelligent Machines built with Memristive Nanodevices. In: 12th IEEE International Workshop on Cellular Nanoscale Networks and their Applications, CNNA (2010)
5. Haykin, S.: Neural networks: A comprehensive foundation. Prentice-Hall (1999)
6. Levy, W.B., Steward, O.: Temporal contiguity requirements for long-term associative potentiation/depression in the hippocampus. Neuroscience 8, 791–797 (1983)
7. Izhikevich, E.: Large-scale model of the mammalian thalamocortical systems. PNAS 105, 3593–3598 (2008)
8. Hyvärinen, A., Oja, E.: Independent component analysis by general nonlinear Hebbian-like learning rules. Signal Processing 64, 301–313 (1998)
9. Grossberg, S.: Adaptive pattern classification and universal recoding: I Parallel development and coding of neural feature detectors. Biol. Cybern. 23, 121–134 (1976)
10. von der Marlsburg, C.: Self-organization of orientation-selective cells in the striate cortex. Kybernetik 15, 85–100 (1973)
11. Mikkulainen, R., Bednar, J.A., Choe, Y., Sirosh, J.: Computational Maps in the visual cortex. Springer (2005)
12. Bienenstock, E.L., Cooper, L., Munro, P.: Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex. J. Neurosci. 2, 31–48 (1982)
13. Grossberg, S.: Some nonlinear networks capable of learning a spatial pattern of arbitrary complexity. PNAS 59, 368–372 (1968)
14. Gorchetchnikov, A., Versace, M., Ames, H., Léveillé, J., Yazdanbakhsh, A., Chandler, B., Mingolla, E., Snider, G.: General form of learning algorithms for neuromorphic hardware implementation. In: The International Computational Neuroscience Meeting (CNS), San Antonio, TX (July 2010)