A critical review of computational neurological models

Samantha Zarate – slzarate@stanford.edu – 5787179

June 11, 2014

Abstract

Modeling neural networks computationally has become a field full of possiblities, especially with the influx of data biologists are currently obtaining through improved experimental methods. The objective of this paper is to provide a critical review of computational models simulating the neural networks of the brain from biomedical and computational literature. In this paper, I review three types of approaches for understanding the inner workings of the human brain. The first approach presented is software-based – specifically, the software package NEURON; and the second is hardware-based – specifically, graphics processing units (GPUs) and the novel creation Neurogrid. Though other techniques exist, these two general types of approaches encompass the diversity of models used to simulate neural networks; specific examples within each approach are either revolutionary or extremely widely used, and so are significant in different ways. Each approach is presented with both its benefits and costs as well as possible improvements, and the paper concludes with an overview of the types of methods and future applications of these computational approaches.

1 Introduction

Mathematical theories of perception were among the first attempts at taking a computational approach to understand neuroscience, and those working in biomedical computation have long since attempted to simulate the brain's functions with computers and computational theory (Gerstner *et al*, 2012). Experimental advances in the field now mean that biologists have an enormous amount of data at their disposal. (Hines and Carnevale, 1997) This influx of data combined with the increasing application of computer science in non-computer fields has begun a veritable revolution in molecular biology and

neuroscience: researchers are now using computer simulation and computational theory to attempt to understand, for example, how the brain operates on a systemic level.

Understanding the networks underlying the human brain's functionality has a multitude of implications, including the ability to model computers after human brains, increasing processing power and energy efficiency (Benjamin *et al*, 2014). In recent years, advances in machine learning and computing power have allowed researchers to develop several different ways of approaching modeling the human brain, focusing on different aspects of the brain's functionality.

In this paper, I will briefly review three different computational approaches to modeling the human brain. This review will begin with software simulations, with an emphasis on NEURON; it will then examine hardware approaches, including graphics processing units (GPUs) and Neurogrid. Focus will be given to the *computational* aspects of these approaches – specifically, the algorithms, equations, and concepts behind the models.

2 Software Approach: NEURON

A widely used approach to modeling the human brain uses software packages that aid researchers in crafting and simulating neuronal network models. These include NEURON, which will be covered in-depth here, as well as GENESIS, NEST, and Topographica.

NEURON, along with GENESIS, has a large user base and is currently under active development. (Crook *et al*, 2012) Initially designed to be a flexible framework specifically for dealing with the equations describing nerve cells in which complex membrane properties and currents (Hines and Carnevale, 1997), it supports modeling conductance-based neuronal models with a great amount of biological detail (Crook *et al*, 2012).

Intuitively, NEURON interprets a neuron to be a continuous cable divided into segments. (Hines and Carnevale, 2001) The cable equation, or the basic description between current and voltage in a one-dimensional cable, is as follows:

$$\frac{\partial V}{\partial t} + I(V,t) = \frac{\partial^2 V}{\partial t^2} \tag{1}$$

This equation, combined with boundary conditions expressed in the form of discrete segments – that can later be connected to better reflect the physical shape of the neuron itself, including branching – is the framework behind

the NEURON model, which is, unsurprisingly, most computationally efficient with problems involving small numbers of neurons or a single neuron in which cable properties are crucial (Hines and Carnevale, 1997).

Furthermore, users may define their own membrane and cytoplasm properties (Hines and Carnevale, 2001); specifically, NEURON supports the model description language NeuroML, which is simulator-independent in order to allow communication and documentation of neural network models, as well as PyNN, which is an application programming interface (API) for the Python programming language (Crook *et al*, 2012).

NEURON is known for its flexibility and convenience; specifically, it has a graphical interface (GUI) that can be used to perform tasks such as model creation and variable graphing as well as an object-orientated interpreter providing a programming language that biologists can easily handle. (Hines and Carnevale, 2001) For example, the following code fragment describes a simple neuron with three dendrites connected to one soma connected to an axon:

```
begintemplate Cell1
    public soma, dendrite, axon
    create soma, dendrite[3], axon
    proc init() {
        for i=0,2 connect dendrite[i](0), soma(0)
        connect axon(0), soma(1)
        axon insert hh
    }
endtemplate Cell1
```

Clearly, NEURON was designed specifically for neurons and follows their properties – with a biology-oriented programming language, it becomes easy for neuroscientists to simulate neural networks without learning a massive multipurpose programming language such as Python or C++. (Hines and Carnevale, 1997) Its programming language is extremely intuitive and easy to understand for neuroscientists due to its low *language entropy* – all named keywords are intuitive (e.g. dendrites represent dendrites, etc.), everything is lowercase, similar functions are named similarly, and it completely revolves around neuroscience itself. (Brette, 2012) Due to this specificity, NEURON can also handle large amounts of biological data, which is crucial for modern-day biologists having to deal with unprecedented levels of data.

Unfortunately, like almost all software approaches to problems in neuroscience, NEURON faces the twin challenges of communication and documentation. Formal documentation is critical for reproducible results, yet non-determinism (wherein there is an element of randomness or *choice*, and thus running the

same algorithm twice may not produce the same response) can impede formal descriptions of models. Additionally, neural network models are necessarily scaled down for clarity and human understanding; however, neural networks themselves are extremely complex, so the downscaling must be well-documented, as there are different approaches to downscaling these neural networks. (Crook *et al*, 2012)

While non-determinism cannot be avoided, NEURON has mitigated this randomness issue by employing *random seeds*, which calculators typically use to determine random numbers. Random seeds are numbers used to initialize a pseudorandom number generator; the output of the psuedorandom number generator is deterministic and so, by choosing the seed, can be controlled. Currently, there is no explicit support for downscaling in NEURON, so each user must find their own way to accomplish this. As a result, models can be inconsistent; and in a field where even the smallest inconsistency or discrepancy can lead to dramatically different results, this is a major issue in communication. (Crook *et al*, 2012)

3 Hardware Approach

Another technique of modeling neural networks, which primarily emphasizes rapidity of processing and scale of processes, is a hardware approach, in which physical chips or circuits simulate neural processing (Benjamin *et al*, 2014; Brette and Goodman, 2012; Schemmel *et al*, 2008). This review will focus on the use of graphics processing units (GPUs) and Neurogrid, as they represent both a pre-existing system retooled for neural network modeling and an entirely novel hardware approach bundled with a unique software component, respectively.

3.1 Graphics Processing Units (GPUs)

Graphics processing units (GPUs) are chips of graphics cards; originally designed specifically for computer gaming, they are now being repurposed for parallel computing. GPUs are ideal for parallel computing due to their multiple processor cores, and ultimately researchers wish to be able to simulate neural networks in parallel, thus improving efficiency and reducing the amount of physical hardware needed for large-scale neural modeling. Furthermore, algorithms used by GPUs must be specifically designed for GPUs; individual processor cores are considerably simpler than a typical computer's central processing unit (CPU), and so algorithms failing to account for the GPU architecture will fail to take advantage of the potential efficiency. (Brette and Goodman, 2012) GPUs are multipurpose even within neuronal modeling,

and specialized simulator environments such as NeMo and GeNN have been designed to rise to the challenge. (Crook *et al*, 2012)

However, specifically, I will focus on the use of GPU implementations for *spiking neuron models*, wherein discrete events (spikes) are modeled, rather than gap junctions, as to the best of the author's knowledge, this usage is extremely well-documented. The spiking neuron model can be broken down into three primary steps:

a. Integrating the differential equations used to describe neuronal models, which scales with the number of neurons and easily parallelized.

b. Propagating spikes to target neurons, which scales with the number of synapses.

c. Applying the effects of spikes on target neurons, which also scales with the number of synapses.

Because of how steps 2 and 3 scale, they are the bottlenecks in terms of computational time and cost; in particular, step 2 is more difficult as it does not follow the Single Instruction, Multiple Data paradigm and so is the primary bottleneck. There are two ways to approach this more difficult step of spike propagation: one can either parallelize over neurons, wherein threads update the total input of one neuron by checking whether any presynaptic neurons have spiked (this can be made even more efficient to reduce the number of unnecessary operations); or one can parallelize over synapses, which particularly reduces the number of unnecessary operations with sparse firing, and can even be more efficient than parallelizing over neurons. (Brette and Goodman, 2012)

Overall, large-scale GPU simulations are impeded by memory, which limits overall efficiency (Crook *et al*, 2012) – at each timestep the GPU's kernels need to access a great deal of memory because *every* synaptic variable corresponding to spikes must be accessed; furthermore, the limiting speed is very low due to the fact that synaptic operations are typically addition. Furthermore, due to the high specificity of GPU code, coding for GPUS is a task typically automated using code generation, or the automatic generation of code from a detailed model description, which also allows non-computer scientists to select the most efficient algorithm for their particular hardware device. (Brette and Goodman, 2012)

3.2 Neurogrid

Neurogrid is an extremely novel (developed a little over a month ago) neuromorphic system for simulating large-scale neuronal models in real time –

a physical array of circuit boards designed to emulate both the structure and function of a neuronal system combined with software to perform interactive functions such as visualization, though this review will focus on the hardware aspect of the project.

In particular, Neurogrid uses a series of circuits to simulate neurons – 16 Neurocores, all on a single circuit board and powered by 3 watts, each to simulate a neuron with a soma, a dendrite, and four shared synapse and dendrite circuits. The behavior of these Neurocores are determined by specialized equations, as shown below, wherein v denotes voltage, g denotes conductance, and i denotes current.

Somas are represented by the following equation:

$$\tau_s \dot{v}_s = -v_s + i_{sin} + \frac{v_s^2}{2} - g_K v_s - g_{res} v_s p_{res}(t) + v_d \tag{2}$$

Wherein:

- τ_s is the membrane time constant
- i_{sin} is the input current
- $\frac{v_s^2}{s}$ is the spike-generating sodium current
- g_{res} is the reset conductance for the duration t_{res} of a high-amplitude pulse t_{res}, modeling the refractory period
- g_K denotes high-frequency potassium conductance, modeling spike-frequency adaptation

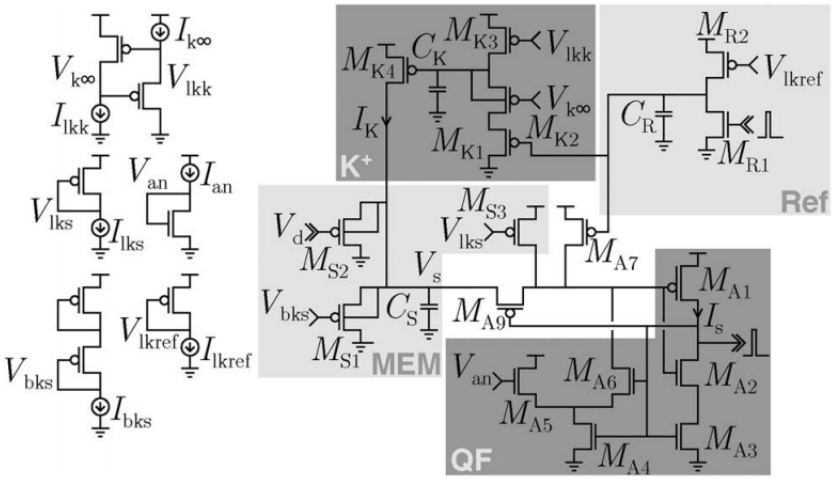
Dendrites are represented by the following equation:

$$\tau_d \dot{v}_d = -v_d + i_{din} + i_{bp} p_{res}(t) + g_{ch}(e_{ch} - v_d) \tag{3}$$

Wherein:

- τ_d is the membrane time constant
- i_{din} is the input current
- i_{bp} is the backpropagating input
- g_{ch} is the channel population's conductance
- e_{ch} is the reverse potential

Figure 1: Example of a circuit modeling a soma used in the Neurogrid. (Benjamin *et al*, 2014)



These equations, including those for synapse population and ion-channel population, are all integrated in order to build separate circuit models for each aspect of the neuron, as shown in figure 1.

Additionally, Neurogrid enables shared electronic circuits – for synapse, dendrites, axons, and more – wherein a common set of wires, resistive network, circuit board, etc. is shared between an entire neuronal population, which allows it to operate much more efficiently than other models. During a million-neuron, eight-billion-synapse real-time simulation, Neurogrid consumed 2.7 watts. Neurogrid also had an overall lower cost than similar methods of simulation by using multilevel axon branching to reduce its fixed area and static energy cost.

Overall, Neurogrid appears to be extremely promising – it can be fully utilized by neural models such as cortical feature maps and cortical columns and is more efficient than its peers. However, there are limitations to Neurogrid: it was very recently published, meaning not many researchers have had a chance to attempt using it; it makes synaptic plasticity all but impossible due to its neurons having fixed neighbors; it has an outdated process; and it is orders of magnitude worse than the human brain in terms of energy efficiency. (Benjamin *et al*, 2014)

4 Conclusion

Each approach has its own unique benefits and challenges:

- Software such as NEURON is operable on different simulators and is easy-to-use for biologists unfamiliar with programming, but it is difficult to reproduce and results are difficult to communicate due to non-standard methods.
  - This can be improved by standardizing methods such as downscaling and generally making code and algorithms easier to communicate between platforms. Standardization without loss of specificity is the key here.
- Hardware repurposed for neural modeling such as graphics processing units (GPUs) are extremely powerful and allow for parallel processing, but it is limited by its own memory.
  - This can be improved by using more specialized chips and utilizing GPUs more efficiently in order to reduce the memory required.
- Hardware built specifically for neuronal modeling such as Neurogrid are also very powerful, specialized, and energy-efficient, but its specialty is limiting in terms of plasticity and it is still considerably less efficient than the human brain itself.
  - This can be improved by both focusing scrutiny on novel methods such as Neurogrid in order to specifically see what should be done to make it more efficient and rethinking the setup to allow for synaptic plasticity – for example, machine learning and short-term memory may be very valuable in permitting Neurogrid to model synaptic plasticity and thus become more efficient.

Modeling of neural networks is an extremely promising, fairly new field, one that has the ability to make computers more powerful than ever before. Additionally, machine learning is a very powerful tool that, when thoroughly integrated into the large-scale modeling of networks, can dramatically improve model accuracy. Though there are different avenues via which researchers approach modeling, each have their own benefits and drawbacks; as a result, both hardware and software modeling approaches will likely continue to be pursued, possibly yielding independent yet equally exciting results for the fields of medicine, neuroscience, computing, and more.