Neural Networks 83 (2016) 1–10

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Simbrain 3.0: A flexible, visually-oriented neural network simulator

Zachary Tosi [a,*], Jeffrey Yoshimi [b]

[a] Cognitive Science, Complex Systems, Indiana University Bloomington, United States
[b] Cognitive and Information Sciences, University of California, Merced, United States

ARTICLE INFO

Article history:
Received 4 December 2015
Received in revised form 4 July 2016
Accepted 13 July 2016
Available online 29 July 2016

Keywords:
Neural networks
Simulator
Education
Network visualization

ABSTRACT

Simbrain 3.0 is a software package for neural network design and analysis, which emphasizes flexibility (arbitrarily complex networks can be built using a suite of basic components) and a visually rich, intuitive interface. These features support both students and professionals. Students can study all of the major classes of neural networks in a familiar graphical setting, and can easily modify simulations, experimenting with networks and immediately seeing the results of their interventions. With the 3.0 release, Simbrain supports models on the order of thousands of neurons and a million synapses. This allows the same features that support education to support research professionals, who can now use the tool to quickly design, run, and analyze the behavior of large, highly customizable simulations.

1. Introduction

Simbrain 3.0 (http://www.simbrain.net/), a major revision and overhaul of Simbrain 2.0 (Yoshimi, 2008), is an open source neural network tool[1] which can simulate many different network architectures using a visually rich and intuitive interface. This makes it ideal for education and rapid prototyping. Simbrain 3.0 has greatly improved performance: it can handle thousands of model neurons and hundreds of thousands of synaptic connections on a standard desktop PC. Thus Simbrain is increasingly viable as a research tool, particularly in computational neuroscience.

Simbrain's flexibility is based on its modular "tool-kit" structure and scripting abilities. In the last few decades, artificial neural networks (ANNs) have become prominent in many fields, including machine learning, neuroscience, and psychology. The thousands of models that have been developed in these fields are largely based on a common core of basic components and functions. Simbrain supports most of these core components as graphical elements and allows them to be combined in arbitrary ways. Thus many existing models can be developed "out of the box" using Simbrain's graphical user interface (GUI). The intention is to create a sandbox for maximally permissive experimentation. In cases where more custom functions are needed, Simbrain provides a powerful scripting interface. In these ways, just about any modern neural network model at the level of abstraction of point neurons can be implemented in Simbrain.

Simbrain's ease of use is based on its graphical interface, which allows individual neurons and synapses, and groups of neurons and synapses, to be created, edited, and connected together using a familiar point and click interface, as well as a set of keyboard shortcuts that, in our experience, are quickly mastered. One of the goals of Simbrain is to make neural networks as accessible (and manipulatable) as possible, in terms of their behavior, dynamics, analysis, and interactions with an environment. Users can modify parameters of running simulations and observe the effects of these interventions in real-time.

These features make it easy to use Simbrain in a broad range of educational settings, with students of different ages and levels of expertise. In fact, an informal design goal of the team is to make it possible for children and young adults to build and study neural networks. Simbrain has been used at the university level at the University of Sydney (Australia), LMU Munich (Germany), University of Indiana Bloomington (USA), and at the University of California, Merced (USA), among others, and the team has been seeking funding to develop educational modules for use in K-12 settings.

However, Simbrain is not exclusively an educational tool. It is also well-suited to research professionals developing more complex models. It has been used by the second author to study the dynamics of small networks embedded in virtual environments (Hotton & Yoshimi, 2011; Yoshimi, 2014), but with the improvements of 3.0 the possibilities have expanded. In as

* Corresponding author.
E-mail addresses: ztosi@umail.iu.edu (Z. Tosi), jyoshimi@ucmerced.edu (J. Yoshimi).

[1] Using the GNU general public license.

http://dx.doi.org/10.1016/j.neunet.2016.07.005
0893-6080/

yet unpublished work, the first author has drawn on Simbrain's enhanced performance capacities and extensibility to study the mechanisms of plasticity that are needed to account for the behavior and structure of cortical microcircuits observed *in vitro* and *in vivo*. In this kind of work, instantaneous visualizations of network data can inspire new hypotheses and lines of research, and make users immediately aware of information about a network which might otherwise go unnoticed, due to the often painstaking process of creating useful data visualizations. In fact, a chance observation of changes in synaptic weight distributions (via Simbrain's histogram plot) was the impetus for the first author's line of research.

Thus in a research/professional context this rich user feedback promotes a development cycle involving some or all of the following steps: (1) Come up with an idea for a novel ANN architecture. (2) Implement the idea using the Simbrain GUI (and possibly scripting). (3) Refine rapidly using the GUI's visual feedback. (4) Fine tune the simulation and potentially develop it without the GUI, using Simbrain as a library.

Simbrain is programmed in Java and thus runs on any machine with a Java Virtual Machine (JRE/JDK 8 or higher required). Neural network graphics are based on the Piccolo library (http://www.cs.umd.edu/hcil/piccolo/), a zoomable scene-graph based library built on top of the Java2D API.[2] The source code is hosted on git-hub (https://github.com/simbrain/simbrain) and efforts have been made to produce a clean, readable codebase.

Simbrain has been in continuous development since the late 1990s. Its original purpose was to provide a visually oriented framework for studying neural networks (few such programs were available at the time). Since then many more neural network packages have emerged, including several that feature rich visualization capacities (Aisa, Mingus, & O'Reilly, 2008; Bekolay et al., 2013; Gewaltig & Diesmann, 2007; Goodman & Brette, 2008).[3] Two of the most similar, in terms of providing a point-and-click GUI for creating neural networks, are Emergent and Nengo. However, these packages have been designed around specific computational frameworks (the Leabra framework in the case of Emergent; the Neural Engineering Framework in the case of Nengo). Though both frameworks are flexible (e.g. backpropagation is available in Emergent), each focuses on a specific class of neuro-computational models. A primary design goal of Simbrain, by contrast, has been to provide a domain general framework for creating and combining arbitrary neuron, synapse, and network models. Programs like Brian and Nest can produce detailed visualizations, but are not fundamentally GUI-based programs. With the emergence of HTML5, beautiful visualizations are more readily accessible directly in the browser, prompting some to create web-based interactive neural networks.[4] However, thus far web-based visualizations have been oriented towards single demonstrations rather than general purpose simulation frameworks.

Simbrain's basic graphical user interface is a desktop (see Fig. 1) with components (neural networks, virtual environments, graphs, etc.) that can be coupled to one another.[5] Network panels allow users to build networks and watch the changing dynamics of neurons and (in some cases) synapses in real-time, alongside corresponding changes in graphs, tables, and virtual environments. The GUI can be disabled (or other GUI objects hidden) to increase a simulation's computational speed and efficiency. Simbrain's logical code is separate from its GUI code and it can be used as a library independently of the GUI.

The shift from Simbrain 2.0 to Simbrain 3.0 focused on scope and scalability/performance. Improved performance made larger-scale simulations possible, which in turn made it possible to add new network types and training algorithms. 3.0 introduces aggregated groups of neurons and synapses ("neuron groups" and "synapse groups") which provide a convenient way to manipulate many neurons or synapses at once. Simbrain's data analysis and visualization software has also been improved. Many of the enhancements have focused on supporting more biologically plausible neural network models, allowing for the simulation of structures like cortical microcircuits in Simbrain.

This article overviews the full scope of Simbrain 3.0's capabilities and features. We first consider its core elements: neurons, synapses, plasticity rules, synaptic transmission rules, neuron groups, synapse groups, and subnetworks. We then consider its performance features, graphical interface design, worlds, and scripting. Examples of applications are given throughout. We end by discussing future directions for Simbrain.

2. Core architecture

2.1. Neurons and synapses

Simbrain supports a wide variety of model neurons (both connectionist and spiking), and rules for synaptic plasticity and transmission. The code was designed with flexibility and extendability in mind. Custom extensions are possible and encouraged. The easiest way to customize the code is via scripts (using beanshell, a type of interpreted Java), though efforts have also been made to make it easy to modify the code directly via the API.[6] Simbrain takes an object-oriented approach to the representation of common ANN constructs like neurons and synapses, as well as the rules governing their behavior. Most customizations can be implemented as a subclass of a Simbrain core class. This results in an intuitive, easy-to-extend API where the conceptual constructs constituting neural networks have a direct relationship to classes in Simbrain.

Neurons correspond to objects with general properties, including an activation value and a boolean value indicating a spike or action potential (in effect, a Dirac or Kronecker delta function depending upon the temporal framework) has occurred. During network updates the neuron class typically handles procedures relating to the calculation of weighted incoming sums from synapses and the appropriate transfer of activation states (including spiking behavior if applicable) to and from synchronizing buffers. The procedure for determining the neuron's state at the next time-step (and thus its overall behavior) is delegated to the neuron's update rule object. For instance, specific point neuron models

[2] Piccolo's main design features have been incorporated into Java FX (the latest iteration of Java's graphics platform), and plans are to gradually port Simbrain to Java FX.

[3] Also see (Krenek et al., 2014) and the list here: https://grey.colorado.edu/emergent/index.php/Comparison_of_Neural_Network_Simulators.

[4] Some examples: http://cs.stanford.edu/people/karpathy/convnetjs/, http://ncase.me/neurons/, and http://scs.ryerson.ca/~aharley/vis/.

[5] All of the screenshots in this paper can be easily reproduced, using simulations that are included with the latest 3.01 release of Simbrain. Figs. 1 and 2 correspond to workspaces *backpropLetters.zip* and *spkExamples.zip*. Figs. 5, 6, 7, 9, and 10 correspond to scripts *CorticalBranching.bsh*, *RealNeuralNetAdEx.bsh*, *braitenberg.bsh*, *elmanPhonemes.bsh*, and *SORN.bsh*.

[6] Custom neurons, synapses, subnetwork types, neuron group types, update methods, gui widgets, and overall simulation configurations can be created using custom scripts. Over 20 sample scripts illustrating these different types of extension (and serving as easy-to-modify templates) are included. Adding new neuron, synapse, neuron group, and many other object types directly to the Simbrain code is also fairly easy. In the most common case (neuron types), it only involves adding two new files following a standard and easy-to-understand template; the process is described here: see https://github.com/simbrain/simbrain/wiki.