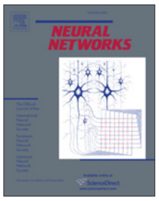
Neural Networks 21 (2008) 1146–1152

Contents lists available at ScienceDirect

Neural Networks

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



2008 Special Issue

The Emergent neural modeling system[☆,☆☆]

Brad Aisa[*], Brian Mingus, Randy O'Reilly

Computational Cognitive Neuroscience Lab, Department of Psychology, University of Colorado at Boulder, United States

ARTICLE INFO

Article history:
Received 1 November 2007
Received in revised form
10 June 2008
Accepted 17 June 2008

Keywords:
Neural networks
Robotics
Simulator

ABSTRACT

Emergent (http://grey.colorado.edu/emergent) is a powerful tool for the simulation of biologically plausible, complex neural systems that was released in August 2007. Inheriting decades of research and experience in network algorithms and modeling principles from its predecessors, PDP++ and PDP, Emergent has been redesigned as an efficient workspace for academic research and an engaging, easy-to-navigate environment for students. The system provides a modern and intuitive interface for programming and visualization centered around hierarchical, tree-based navigation and drag-and-drop reorganization. Emergent contains familiar, high-level simulation constructs such as Layers and Projections, a wide variety of algorithms, general-purpose data handling and analysis facilities and an integrated virtual environment for developing closed-loop cognitive agents. For students, the traditional role of a textbook has been enhanced by wikis embedded in every project that serve to explain, document, and help newcomers engage the interface and step through models using familiar hyperlinks. For advanced users, the software is easily extensible in all respects via runtime plugins, has a powerful shell with an integrated debugger, and a scripting language that is fully symmetric with the interface. Emergent strikes a balance between detailed, computationally expensive spiking neuron models and abstract, Bayesian or symbolic systems. This middle level of detail allows for the rapid development and successful execution of complex cognitive models while maintaining biological plausibility.

1. Introduction

Emergent (http://grey.colorado.edu/emergent) is a powerful tool for the simulation of biologically plausible, complex neural systems that was released in August 2007. The immediate predecessor to Emergent is PDP++ v3.2, a tool used by a variety of researchers for neural modeling and teaching. PDP++ was itself an extension of the PDP software released by McClelland and Rumelhart in 1986 with their groundbreaking book, Parallel Distributed Processing (McClelland & Rumelhart, 1986). Emergent represents a near complete rewrite of PDP++, replacing an aging and largely unsupported graphical user interface (GUI) framework called Interviews with a well supported, more modern one called `Qt` (http://trolltech.com/products/qt). A major benefit of Qt is its seamless integration into all major platforms, allowing Emergent to not only be easily installed on them, but also to adopt their native look and feel. With this in mind, we completely redesigned the user interface, employing a now-familiar tree-based browser approach (with tabbed edit/view panels) for project exploration and interaction (Fig. 1). We also radically redesigned or even replaced several core constructs from the previous product, such as `Environments` and `Processes`, replacing them with the more general-purpose `DataTable` and `Program` constructs that will be discussed later.

More important than technical or interface changes, we also extended the intended scope of the tool. Whereas the previous versions were primarily intended for relatively small research and teaching models, typically aimed at demonstrating some isolated or delimited piece of functionality, the new version is intended to support very large-scale simulations of entire integrated brain-like systems. And whereas the previous versions were primarily designed for closed simulations using simple fixed data patterns as input and output, Emergent has been designed to accommodate external "closed-loop" sensory and motor connections both by plugins and with a built-in simulation environment that includes a rigid-body physics simulation for creating virtual robot-like agents.

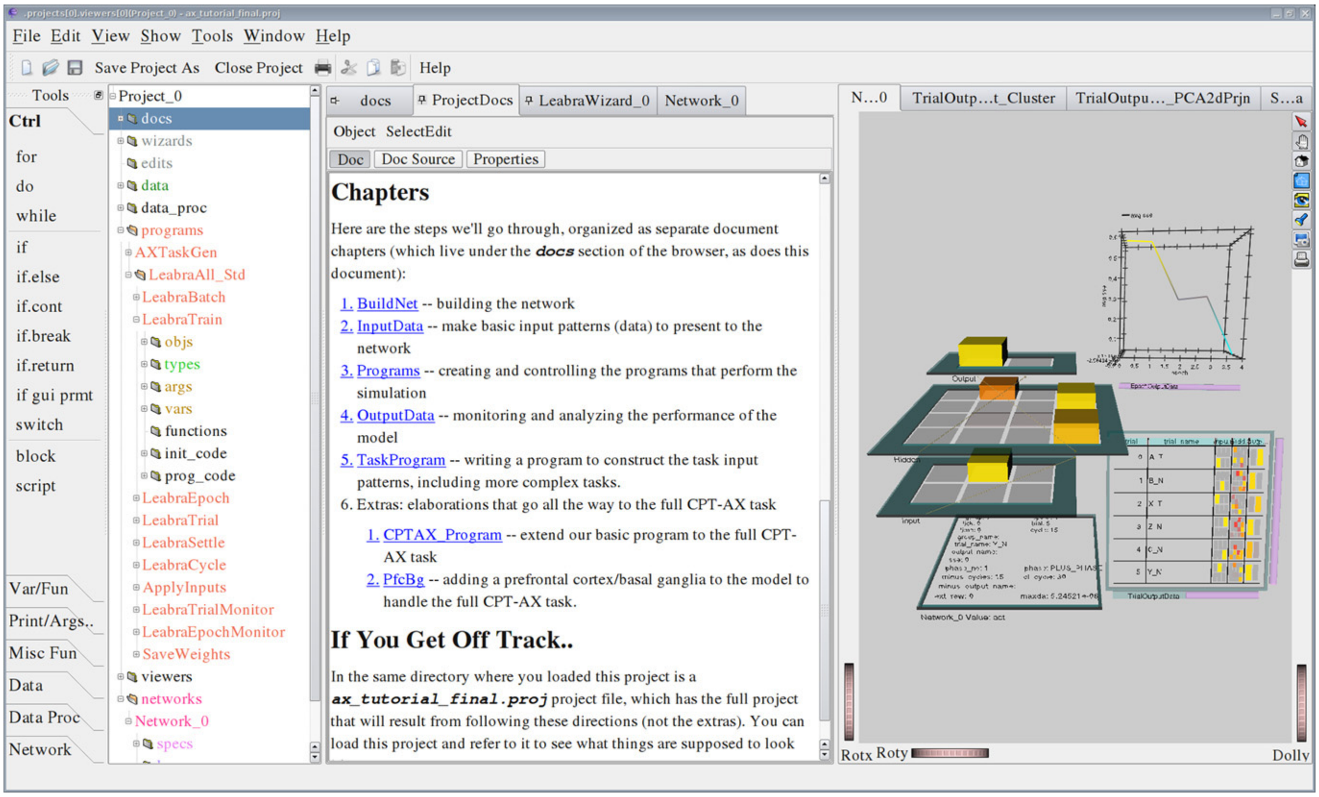
This article will give a general overview of Emergent's features and capabilities, ending with a comparison with other neural network simulators and a discussion of the features we plan to implement in the near future.

☆ Supported by grants: NIH R01 MH069597, ONR N00014-07-1-0651, DARPA/ONR N00014-05-1-0880, ONR N00014-03-1-0428 (O'Reilly); NIH IBSC 1 P50 MH 64445 (McClelland).

☆☆ Thanks go to Dave Jilk for being the intrepid early adopter; Jay McClelland of Carnegie Melon University and Jonathan Cohen of Princeton University for their financial assistance during Emergent's development; and all members of the CCN Lab at CU Boulder for their valuable input and patient testing of the software.

* Corresponding author. Tel.: +1 720 233 0225; fax: +1 303 492 2967.
E-mail address: Brad.Aisa@colorado.edu (B. Aisa).

0893-6080/$ – see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2008.06.016



Chapters

Here are the steps we'll go through, organized as separate document chapters (which live under the ***docs*** section of the browser, as does this document):

1. BuildNet -- building the network
2. InputData -- make basic input patterns (data) to present to the network
3. Programs -- creating and controlling the programs that perform the simulation
4. OutputData -- monitoring and analyzing the performance of the model
5. TaskProgram -- writing a program to construct the task input patterns, including more complex tasks.
6. Extras: elaborations that go all the way to the full CPT-AX task
    1. CPTAX_Program -- extend our basic program to the full CPT-AX task
    2. PfcBg -- adding a prefrontal cortex/basal ganglia to the model to handle the full CPT-AX task.

If You Get Off Track..

In the same directory where you loaded this project is a ***ax_tutorial_final.proj*** project file, which has the full project that will result from following these directions (not the extras). You can load this project and refer to it to see what things are supposed to look

Fig. 1. The Emergent Project Browser. The main workspace in Emergent showing, from left: (a) the Toolbox with widgets for Programming and similar tasks; (b) the main Browser, a hierarchical tree of all objects in the project; (c) the Panel area, for editing and viewing the details of objects, in this case displaying a Doc; and (d) the 3D viewer, for viewing simulation objects in true 3D.

2. Emergent

2.1. *Supported algorithms*

Out of the box, Emergent supports classic back-propagation (BP) (Rumelhart, Hinton, & Williams, 1986), and recurrent back-propagation in several variants (Almeida, 1987; Pineda, 1987; Williams & Zipser, 1989); Constraint-satisfaction (CS) including the Boltzmann Machine (Ackley et al., 1985), Interactive Activation and Competition, and other related algorithms; Self-organized learning (SO) including Hebbian Competitive learning and variants (Rumelhart & Zipser, 1986) and Kohonen's Self-Organizing Maps and variants (Kohonen, 1984); and Leabra (an acronym for "local error-driven and biologically realistic algorithm") which includes key features from each of the above algorithms in one coherent framework (O'Reilly & Munakata, 2000).

The previous version of the software (PDP++ v3.2) also served as the basis for some other neural algorithms or extensions, including the Real-time Neural Simulator RNS++ (http://ccsrv1.psych.indiana.edu/rns++/) (Josh Brown); Long Short Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997); and the oscillating inhibition learning mechanism (Norman et al., 2006). The enhanced user-friendliness of the software and our new plugin technology make these kinds of extensions very easy to implement, hopefully encouraging more researchers to consider Emergent as the architectural base of their research algorithms. Unlike the tools such as MATLAB, Emergent is completely free and open-source; in addition, its network algorithms run at compiled C++ speed, rather than in an interpreter.

2.2. *General features*

Emergent opens to present a familiar tree-based browser (on the left) plus detail panel (on the right.) The user can select any object in the left-hand tree to see its detailed properties on the right, and open container nodes to reveal the sub-contents. Many objects have several detail sub-panels that present the object and its content in different views, depending on the purpose of the user. For example, the table object provides a panel with the properties of the table itself, one that lists the columns, and one that enables the user to browse or edit data. The user can open up any number of new browsers rooted at any point in an existing browser.

Clipboard and drag-and-drop manipulation of objects are supported wherever it makes sense. Many "action-like" operations, such as assigning an object to a program variable element, can be done via drag-and-drop.

When the user opens or creates a project, an additional `viewer` pane appears in the browser; this viewer supports one or more `frames` which display a true 3D rendering of one or more objects in the system, such as networks, graphs and virtual environments.

2.3. *Networks*

The basic unit of modeling in Emergent is the `Unit`, which is a neuron-like object that represents a small population of like-coding spiking neurons, such as might be observed in a cortical column. Its output is typically a time continuous value ranging from 0 to 1, which represent the extremes of "no firing in the population" to "maximal firing" in the population. Maximal firing is a product of the number of individual neurons times the rate of firing per each neuron. For more detailed neural models it is also possible to run the units in discrete spiking mode. These Units perform separate integrations of excitatory, inhibitory, and leak inputs to accommodate shunting inhibition effects, replicating the classic equivalent circuit dynamics of real neurons. The output is typically based on a thresholded, parameterized, bounded and

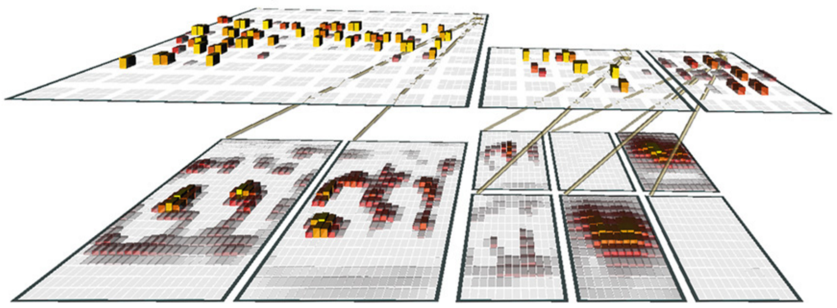


Fig. 2. Emergent Layers, showing Projections and Units.

sigmoidal-like curve. Other transfer function options are provided, and custom functions are possible.

Units are not instantiated or manipulated individually, but are managed in a group called a `Layer`. A Layer in Emergent is a two-dimensional "sheet" of Units, all of which share unit-level parameters (via a `UnitSpec`), inhibitory dynamics and patterns of connectivity to other Layers. A Layer can further be divided into a sub-grid of `UnitGroups`, which enables two levels of inhibitory dynamics, and more sophisticated granularity of connectivity with other Layers. Each Layer has a `LayerSpec` with parameters to control things like inhibitory dynamics, and how input data (if any) is mixed with the existing activation.

Emergent has two distinct constructs for representing connectivity between Units: the `Projection` and the `Connection`. A Projection specifies a logical, unidirectional connection between two layers; a Connection is the actual physical connection between a Unit and its targets, and is analogous to a neural synapse. A Projection specifies the pattern of connectivity between the layers, such as "all to all" or "tessellated", as well as a `ConnectionSpec` that
controls the parameters of the underlying connections that are generated. The ConnectionSpec has parameters that control the physical connections including the weighting of the connection relative to other connections, the weight limits (if any), the local learning rates (Hebbian and error-driven terms) and other miscellaneous parameters. Fig. 2 is an example network showing Layers and Projections.

A set of Layers is aggregated into an overall structure called a Network. There is typically one Network in use during any simulation run, although a Project can contain any number Networks. This can be helpful when testing different approaches as you can share all the other elements, such as control Programs, data input and output and monitoring Programs, to name a few.

2.4. Specs

`Specs` in Emergent are like styles in a word processing program—collections of parameters that can be applied to instances of a specific type, to control or modify their behavior. Specs help the modeler to keep parameters consistent across many instances of a same-type object, such as a Layer of a certain purpose. Sub-specs can be created that automatically inherit their values from a parent Spec, but in which selected parameters can be explicitly overridden. This helps to keep related but distinct Specs coordinated, except for the specific parameter values on which they differ. Specs can be nested to any practical level.

Emergent provides a convenient facility to easily determine which network constructs are associated with each spec. The user can click the Spec in the network control panel, and the items using that Spec are immediately highlighted in the network display. Specs also help to make a model easier to understand. Once an observer has first examined the overall network structure, the next step to understanding the model would be to click on the Specs, which will highlight the objects using them in the 3D viewer.

2.5. Algorithm infrastructure mechanisms

The base classes described above (Connection, Unit, Layer, etc) provide support for a range of common neural network processing mechanisms, such as computing the net input as a function of sending activations times weights. Specific algorithms then add their unique learning and processing mechanisms (e.g., Hebbian learning, inhibitory competition, discrete spiking). Furthermore, all of the implemented algorithms provided with the simulator provide a range of different algorithm variants that can be mixed-and-matched to create novel network architectures. These variations include different learning rules, activation functions, inhibitory mechanisms, etc. These variations are implemented either with a user-selectable switch within a common Spec class, or by a new subclass Spec type that directly implements the new functionality, which the user then selects by applying that spec to the appropriate network elements.

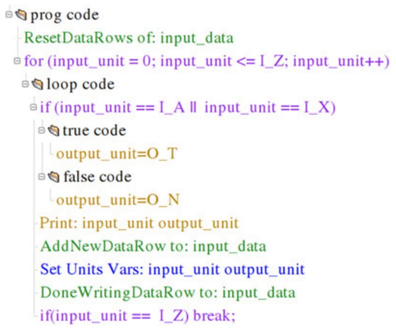
Although all the algorithms are derived from common base classes, each has incompatible optimizations and specializations relative to the others, such that they cannot be mixed in the same network. Thus, it is not possible to directly mix a self-organizing map layer with a backpropagation layer in the same network: supporting such heterogeneous collections would require N^2 kinds of conditional mechanisms and is not efficient and often would lead to nonsensical results. However, it is very straightforward to arrange for the communication of activations or other values between multiple networks of different types, effectively creating hybrid overall architectures. As a perhaps more satisfying alternative, many users leverage the unified framework for many of these different mechanisms provided by the Leabra algorithm, where such architectures can be created by differential parameterization of different layers.

Critically, all of the infrastructure for visualization and data analysis can be automatically applied to any new objects defined in the system, thanks to a powerful type-access system that scans header files and makes the software "self-aware" of very detailed type information for every object defined in the source code (including plugins).

2.6. Network input/output and data monitoring

Most network input and output in an Emergent simulation is facilitated by one or more `DataTable` ("table") objects. A table is similar to a spreadsheet table or database table: it is a set of one or more columns, each of which can contain data of a single type. The table then holds zero or more rows of data. In Emergent, a table cell can hold a matrix in addition to a scalar value. For network input and output data, a matrix column is typically created corresponding to the dimensions of a target or source layer in the network.

Emergent was designed with external connectivity in mind. External sources of input such as video and audio can be



```
prog code
  ResetDataRows of: input_data
  for (input_unit = 0; input_unit <= I_Z; input_unit++)
    loop code
      if (input_unit == I_A || input_unit == I_X)
        true code
          output_unit=O_T
        false code
          output_unit=O_N
      Print: input_unit output_unit
      AddNewDataRow to: input_data
      Set Units Vars: input_unit output_unit
      DoneWritingDataRow to: input_data
      if(input_unit == I_Z) break;
```

Fig. 3. Example program from Emergent's `AX Tutorial`, which walks the user through an implementation of the CPT-AX task used in working memory studies. All elements and groups of elements support copy, paste, drag-and-drop, and lines are color coded according to type.

preprocessed and the raw network input pattern then written to a table. Likewise, network output is first written to a data table, and can then be output to effectors such as simulated muscles, or even a real robot.

The table is the basic construct for network monitoring. Network object parameters, particularly Unit activation values, can be logged to a table during the simulation. The user can log to many tables at multiple levels of time, such as Epoch, Trial, or even Cycle (a single update of all network activations.)

2.7. Simulation control (programs)

Emergent provides a sophisticated but user-friendly general-purpose program environment that is used for sequencing simulations and doing general-purpose data processing tasks. A `Program` is a GUI-based tree of programming widgets called `ProgEls` that enables even a novice user to construct a variety of control sequences such as loops and conditional tests. The application comes with a pre-built library of programs that perform common simulation sequencing functions such as Batch, Epoch and Trial. The Network Wizard will automatically load and connect these to a network.

Programs, an example of which is displayed in Fig. 3, are a simple way of generating C-Super Script (CSS), the native scripting language of Emergent. CSS is best thought of as interpreted C++ and provides full access to the internal objects of Emergent as well as the ability to declare new classes. The resulting script can be examined in a text window. Advanced users can put CSS code in a program script element, but this is usually not necessary because visual programming in Emergent is faster than writing code by hand. CSS scripts are compiled into an efficient object-oriented byte code, which is then run at execution time. The generated network algorithms and data processing primitives are coded in highly optimized C++, scripting being used primarily to control and sequence these optimized workhorse operations.

2.8. Visualizations

Networks, graphs, tabular logs, and physical simulation objects can all be presented in an OpenGL-based 3D visualization environment. The user can create any number of `Frames`, each of which can contain a 3D visualization of one or more of the objects listed. GUI updates can be explicitly disabled to speed up simulations, and are always implicitly suppressed for non-visible panels.

A network display depicts the Unit activation values by default, but the user can select any parameter of Units, Connections, or Projections to monitor.

2.9. Data analysis and graphing

Emergent provides several facilities to help with data analysis. A collection of GUI-accessible objects provide common data processing operations, including: database-style operations (select, sort, join, group, sums, etc.); data analysis (distance, smoothing, dimension reduction such as clustering, PCA, SVD, etc.); data generation (random patterns, line patterns, noise, etc.); and image processing (rotation, translation, scaling, Gaussian and difference-of-Gaussian filtering, etc.). All of these operations are also available to Programs.

As would be expected, data can be readily imported or exported for use with other systems. It is also easy to copy and paste data to and from the clipboard, to exchange data with other programs such as Excel. Additionally, a full range of data graphing operations are available, to present 2D or 3D graphs, to display data from any table. Specialized graph types such as cluster trees are supported.

A 3D `GridView` displays some or all of the columns in a table. It is especially useful for displaying input and output patterns, including photos, and for aggregate Epoch, Train or Trial-level statistics, such as error values.

The `GNU Scientific Library (GSL)` has been incorporated into Emergent, making many of its routines and data structures available. `Matrix` objects, the underlying basis for tables columns, are compatible with GSL routines.

2.10. Virtual environment

Neural network researchers are increasingly testing their simulations of brain processes by embodying them in simulated physical agents that can act in a physically simulated world containing other objects or even other such agents. Emergent includes a powerful built-in simulator along with associated modeling constructs to enable building robot-like agents and connecting them to neural simulations. The simulator is based on two widely used technologies: the `Open Dynamics Engine (ODE)` (http://www.ode.org/) for physical modeling and OpenGL and `Coin3D` (http://www.coin3d.org/) for visualization.

The simulator provides access to all of ODE, including constructs for modeling bodies, including objects such as cylinders, boxes and spheres. Limbs and bodies are connected with Joints that have parameters controlling things like angular stops, degrees of freedom and stiffness. Forces can be applied to joints which results in torques dependent on the bodies they are connected to. Textures can be applied to objects and backgrounds to add visual realism, particularly for vision-based simulations. Objects obey the laws of physics, and phenomena such as momentum, elastic collisions, friction, and gravity are all modeled. Cameras are provided to enable endowing an agent with vision, the result of which can be readily interfaced with network model. 3D placement of sound sources is supported using the Simage library (for real-time playback) or the included `audioproc` plugin for localized sound.

Emergent ships with a simple example model demonstrating how to model a reaching task using an agent with a torso, head, shoulders, and arm, as pictured in Fig. 4.

2.11. Documentation, annotation, and search

Neural models can become quite complex, involving factors ranging from the overall purpose of the experiment, to its architecture, to the parameters being chosen for the elements, to the many experiments that may be conducted upon it, to scientific references and so on. Documenting all these elements can be a challenge. Emergent provides support for this in several areas.

Fig. 4. A simulated agent in the Emergent virtual environment. The top left layers represent the elbow and shoulder forces, as read from ODE at the start of the reach. The middle top layers are the forces that the model produces, and are decoded to apply that force to his elbow and shoulder joints. The inner Gaussian blobs represent the goal location and his hand position at the start of the reach, and the outer blob represents his guess as to where the forces he produces will ultimately land the hand. All of this is orchestrated by the hidden layer, which is in the middle row. Notably, this model does not use error-driven learning, instead using the Primary Value Learned Value (PVLV) reinforcement learning system (O'Reilly et al., 2007), the lower cluster of seven layers, which is based on the detailed anatomy of midbrain dopaminergic neurons. PVLV is available as a Network Wizard for all models to use.

First, most user items include a `desc` field that lets the user include a textual description of the item.

Emergent also provides `Docs`, which are wiki-like pages inside the project. In addition to formatted textual material, Docs can also include hyperlinks (URLs) to external web sources, and "internal URLs" that can link to another doc, or even invoke a procedure on some model object. This latter capability is helpful for developing tutorials or teaching simulations. A Doc can be also be linked to a specific object, in which case the doc appears in that object's property sheet.

Annotation is the ability to add additional fields of information to objects within Emergent. This capability, which comprises a set of name/value pairs, is supported by `User Data`. Most objects in Emergent can have User Data added to them. User Data is also used internally by Emergent for such things as providing format information in table columns and graphs. User Data can be set and retrieved programmatically, adding a powerful metadata facility that can be utilized by advanced models and their control programs.

Emergent includes a text-based search engine, available in the context menu of all tree browsers, that can find objects or docs based on their content, type, method names, etc.

2.12. Select edits

Simulations can often contain many hundreds of disparately located parameters, only a few of which may be intended for modification or exploration. Emergent provides a construct called a `SelectEdit`, which enables parameters and control buttons from anywhere in the simulation (particularly Spec objects) to be displayed and changed in one convenient panel. A combination of Edits and Docs are especially useful for creating tutorial or teaching simulations, with self-contained documentation and a constrained display of key parameters.

2.13. Projects

All of the previously described objects live in a top-level object called a `Project`. A Project contains the Networks, Programs, Tables, Docs, Edits, and Views of a complete simulation. Projects are stored in a textual form that makes them well suited to management under a version control system. We have found the open-source revision control system `Subversion` (http://subversion.tigris.org/) to be highly convenient for this purpose.

2.14. Batch mode

Using the application in GUI mode is convenient when teaching, developing, or debugging models. But modeling often involves long sessions of model training, which can be better handled by a batch scheduling program. So Emergent can be run in -nogui batch mode, with a variety of command-line parameters provided to control which model gets loaded, as well as providing model-specific parameters.

2.15. Distributed memory cluster support and parallel threading

Emergent has support for Linux/Unix distributed memory clusters using the industry standard MPI protocol (e.g., MPICH, OpenMPI). The most efficient parallel speedups are obtained using a parallel training mode, whereby several parallel instances of the model run at the same time (one per node), but get different training patterns applied to them; the weight change differentials are then combined and applied to all the instances. This is very efficient because there is a lot of parallel computation for each communication event — the large size of this communication is insignificant relative to the overall costs of coordinating the timing of the different processors during the communication. For this reason, a more fine-grained parallelism at the level of individual units distributed across different processors has not proven very efficient — the communication events are too frequent relative to