Creating, documenting and sharing network models

Sharon M. Crook[1,2], James A. Bednar[3], Sandra Berger[2], Robert Cannon[4], Andrew P. Davison[5], Mikael Djurfeldt[6], Jochen Eppler[7], Birgit Kriener[8], Steve Furber[9], Bruce Graham[10], Hans E. Plesser[8], Lars Schwabe[11], Leslie Smith[10], Volker Steuber[12], Sacha van Albada[7]

[1] School of Mathematical and Statistical Sciences, Arizona State University, Tempe, AZ, USA

[2] School of Life Sciences, Arizona State University, Tempe, AZ, USA

[3] School of Informatics, University of Edinburgh, Edinburgh, UK

[4] Textensor Limited, Edinburgh, UK

[5] Unité de Neuroscience, Information and Complexité, CNRS, Gif sur Yvette, France

[6] PDC Center for High Performance Computing, Royal Institute of Technology, Stockholm, Sweden

[7] Institute of Neuroscience and Medicine, Forschungszentrum, Juelich, Germany

[8] Department of Mathematical Sciences and Technology, Norwegian University of Life Sciences, Ås, Norway

[9] School of Computer Science, University of Manchester, Manchester, UK

[10] Department of Computing Science and Mathematics, University of Stirling, Stirling, UK

[11] Institute of Computer Science, University of Rostock, Rostock, Germany

[12] School of Computer Science and Science and Technology Research Institute, University of Hertfordshire, Hertfordshire, UK

Abstract

As computational neuroscience matures, many simulation environments are available that are useful for neuronal network modeling. However, methods for successfully documenting models for publication and for exchanging models and model components among these projects are still under development. Here we briefly review existing software and applications for network model creation, documentation and exchange. Then we discuss a few of the larger issues facing the field of computational neuroscience regarding network modeling and suggest solutions to some of these problems, concentrating in particular on standardized network model terminology, notation, and descriptions and explicit documentation of model scaling. We hope this will enable and encourage computational neuroscientists to share their models more systematically in the future.

1 Introduction

While the availability of a diverse array of general purpose and more specialized neuronal network simulators facilitates the development of models in neuroscience, the specialized languages and model descriptions that they utilize generally are not interoperable, limiting model reproducibility, exchange and re-use. As models become increasingly more complex and cross multiple scales, attempting to convert code to a different simulator format becomes even more time consuming. Recent work comparing numerical results across simulators (Gleeson et al. 2010, Henker et al. 2012) and examining implementation issues that are inherent to network modeling (Crook et al. 2012) point to the need for benchmarks for simulator testing and also reinforce the need for simulator-independent descriptions for model publication and exchange. The lack of standardized terminology, notation, and graphical representations for documenting networks also negatively impacts progress in our field (Nordlie et al. 2009).

From the early days of computer design (von Neumann 1986), the computer hardware community has addressed many of the same issues facing the network modeling community. For example, how should a designed structure be described, and how can one check that the architecture is what is intended? Because different hardware simulator tools are used for the various modules of a design, hardware designers must be able to interface tools easily. Thus, simulator interoperability is critical. Originally hardware design was performed at the circuit level, where early design tools aided the creation of full schematics and provided automatic simulation. However, gradually, design became more abstract as hardware became more complex. Toward the end of the 1980's, the level of abstraction made design at the circuit level impossible, and design tools began to use libraries of pre-built functions. Diverse notations for these components were developed and the rapidly moving target of the state-of-the-art circuit made standardization difficult. However, these notations slowly became standardized, developing into a formal notation for digital circuits composed of gates and boxes that are described in a hierarchical manner. Attempts to develop functional languages for describing hardware have not been widely adopted, although currently Bluespec's (http://bluespec.com) high-level language facilities are gaining interest.

Like the hardware community, the network modeling community needs an extremely concise, high-level description of model architecture that can be mapped onto a simulator. It is important that it be possible to implement different modules of the architecture at different levels of abstraction. As model complexity grows and development relies on high-level tools, validation is critical so that modelers are able to trust that the model is the intended one and that it is implemented correctly. In this article we briefly discuss some of the ways in which our community is slowly building an infrastructure for efficiently creating, documenting and sharing network models. Then we address a few of the larger issues facing the field of computational neuroscience as we move forward, concentrating in particular on standardized model descriptions and explicit documentation of model scaling.

2 Current resources

Simulation environments

The creation and simulation of network models is facilitated by a large number of freely available software packages (Brette et al. 2007). For more than 20 years, the NEURON (Hines 1989, Hines and Carnevale 1997) and GENESIS (Bower and Beeman 1997) simulation environments have supported the modeling of networks of conductance based neuronal models that include a large amount of biological detail. These simulators have a large user base, and are both under active development, with the original GENESIS simulation software now being superseded by the GENESIS-3 / Neurospaces (Cornelis and De Schutter 2003, Cornelis et al. 2012a, Cornelis et al. 2012b) and MOOSE (Ray and Bhalla 2008) initiatives. Other software packages including NEST (Gewaltig and Diesmann 2007), Brian (Goodman and Brette 2008), PCSIM (Pecevski et al. 2009), and Topographica (Bednar 2009) are more appropriate for the simulation of large-scale networks of abstract neuronal models such as integrate-and-fire (see review by Burkitt 2006), Izhikevich (Izhikevich 2004) and firing-rate models (Wilson and Cowan 1972). The increasing scale of such models is also driving the development of alternative computer architectures and simulation paradigms such as the use of graphics processing units (GPUs) (Nageswaran et al. 2009) and specialized chips (Furber and Temple 2007), described in more detail below. Moreover, a number of simulators such as Cx3D (Zubler and Douglas 2009), NETMORPH (Koene et al. 2009) and NeuGen (Eberhard et al. 2006) have been designed to specifically model the biological development of neurons and neuronal networks.

While the existence of such a wide range of neural simulators is beneficial for the field of computational neuroscience as it provides researchers with ample flexibility and opportunities to choose a simulator that has been optimized for a specific research question, it also complicates the exchange of computational models and therefore collaboration between different laboratories. A complex neuronal network model can take months or even years to develop, analyze, and document, and a full understanding and further development of the simulator-specific scripts can be challenging for users of the same simulator, let alone for someone who is not familiar with the specific simulator that has been used. The desire for model exchange among laboratories that use different simulation platforms and for portability of models between different simulators has stimulated the development of several interoperability frameworks. For example, many simulators such as NEURON, GENESIS-3, MOOSE, Cx3D, and others, now provide support for the simulator-independent model description language NeuroML (Goddard et al. 2001, Crook et al. 2007, Gleeson et al. 2010). Other software packages allow the simulator-independent development of computational models that can then be run on a number of different simulation environments. For example, the PyNN software package (Davison et al. 2009) provides a Python API for the creation of neural network simulations for use with several different simulation platforms. Similarly, the Multi-Simulation Coordinator (MUSIC; Djurfeldt et al. 2010) supports the runtime interaction of multiple simulator tools in multi-level simulations. The neuroConstruct software (Gleeson et al. 2007) facilitates the development, visualization and analysis of biologically detailed neuronal networks in three-dimensional space. These network models are stored in NeuroML format, and neuroConstruct can automatically generate scripts for several simulators. More details regarding some of these approaches are provided below.

Code sharing

There are several possible formal methods for sharing code such as on a publisher's website or in a public source-code repository. Although many journals offer the possibility of making model code available as supplementary material attached to a

journal article, there are disadvantages to this option, including lack of standardization in the code format or the associated metadata, difficulty in updating the code archive if bugs are found, improvements are made or contact details are changed, and quality control. Curated model repositories, where a curator verifies that the code reproduces one or more figures from the published article, and which often have standardized metadata making it easier to find models of a certain type, address the issues of quality control and standardization (Lloyd et al. 2008). Some examples are ModelDB (Peterson et al. 1996, Davison et al. 2002, Migliore et al. 2003, Hines et al. 2004), the Visiome platform (Usui 2003), the BioModels database (Le Novère et al. 2006), and the CellML Model Repository (http://models.cellml.org). The model database most relevant to neuronal network models is ModelDB, a well-established database of computational models of neurons, cellular mechanisms and networks in a variety of different simulators and programming languages, which is curated by the SenseLab initiative at Yale University. In addition, the use of ModelDB is strongly supported by journals such as the Journal of Computational Neuroscience that recommend that all models described by articles in the journal should be uploaded into this database.

A further step towards enabling collaborative model development has been taken by the Open Source Brain (OSB; http://opensourcebrain.org) project initiated by the Silver Laboratory at University College London, which currently involves 11 laboratories as well as partners outside academia in Europe and the United States. The OSB initiative provides a public repository for detailed models of neurons and networks that can be developed collaboratively in any simulator format. The aim is to facilitate collaboration by storing the models in simulator-independent NeuroML format, and to provide access to curated models that reflect the latest experimental findings and that will evolve in parallel with the development of new simulation technology and modeling paradigms.

Code sharing does not ensure reproducibility or model exchange

As detailed by Crook et al. (2012), sharing code may provide a means for replicating results using the same code, but it does not ensure independent reproducibility of model results, which requires a simulator-independent approach. In fact, it is sometimes the case that a given result from a published paper cannot be re-created with code that has been made available, although the use of curated model code repositories is helping in this regard. Reasons for lack of reproducibility may involve differences in the version of the simulator, the compiler, or of shared libraries that are used by either the simulator or the code, differences in the computing platform, or simply poor record keeping on the part of the researcher. Ideally, models and their sub-components should be exchanged easily for re-use across many different simulators. This is the motivation behind approaches discussed in the next section.

3 Formal approaches for describing networks

To facilitate independent reproduction of neuronal network modeling studies, a systematic approach is needed for reporting models. Here we discuss approaches for both human-readable and machine-readable standardized formats that can provide descriptions of network models that are independent of any particular simulator.

Tables and graphical descriptions

Nordlie et al. (2009) provide a checklist for model descriptions, requiring information on the following aspects of a model: (i) model composition, (ii) coordinate systems and topology, (iii) connectivity, (iv) neurons, synapses, and channels, (v) model input, output, and free parameters, (vi) model validation, and (vii) model implementation. They further propose a concise tabular format for summarizing this information in publications. NeuroML tools can generate tables in this format from a formal model description.

Another popular approach is to represent networks graphically in publications, which is an important tool for providing a quick overview to the reader. Although graphs are not sufficient for describing all aspects of a model, they are useful and would be much more useful if a standard approach were adopted by the network modeling community, similar to the use of the Systems Biology Graphical Notation, or SBGN, by the systems biology community (Le Novère et al. 2009). Often, graphical representations must be hierarchical to depict the details at different levels of spatial scale, and the use of *ad hoc* notations with conflicting symbols from one publication to the next makes it difficult to share these complex ideas. The issues with the current use of graphical representations for network models in neuroscience are articulated well by Nordlie and Plesser (2010), who also advocate a connectivity matrix approach. This method of visualizing network connectivity can be used at different levels for either full details or summary information, and is currently available by using the ConnPlotter package (Nordlie and Plesser 2010) with the NEST simulator. However, there are some aspects of networks that cannot be provided in a connectivity matrix. In particular, some network descriptions require procedural information about the order and manner of creating connections, as discussed in more detail below.

Description languages

Software and database developers in many fields, including neuroscience, have enthusiastically adopted Extensible Markup Language (XML) technology (Bray et al. 2008) as an ideal representation for complex structures such as models and data. A major advantage of XML is that it provides a machine-readable format that is independent of any particular programming language or software encoding, which is ideal for a structured, declarative description that can provide a standard for an entire community. Like HTML, XML is composed of text and tags that explicitly describe the structure and the semantics of the content of the document; however, the tags are defined by developers as a specific XML-based markup language that is appropriate for a particular application.

A number of ongoing projects focus on the development of these self-documenting markup languages that are extensible and can form the basis for specific implementations covering a wide range of modeling scales in neuroscience. The Systems Biology Markup Language, SBML, (Hucka et al. 2003) and CellML (Hedley et al. 2000, Lloyd et al. 2004) are two popular languages for describing systems of interacting biomolecules that comprise models often used in systems biology, and both languages are relevant to network models since they can be used to describe complex models of synaptic signaling processes. NeuroML (Goddard et al. 2001, Crook et al. 2007, Gleeson et al. 2010) differs from these languages in that it is a domain specific model description language, and neuroscience concepts such as cells, ion channels and synaptic connections are an integral part of the language. Recently, the International Neuroinformatics Coordinating Facility facilitated the initiation of a markup language for models of spiking neural networks composed of abstract cell types (Network Interchange

format for Neuroscience; http://nineml.org), which is complementary to NeuroML. Additionally, the Simulation Experiment Description Markup Language (SED-ML) (Köhn and Le Novère 2008) is a language for encoding the details of simulation experiments, which follows the requirements defined in the MIASE (Minimal Information About a Simulation Experiment) guidelines (http: //biomodels.net/miase). Taken together, these markup languages cover the majority of network models. The use of namespaces allows for unambiguous mixing of several XML languages; thus, it is possible to use multiple languages for describing different modules of a multiscale model. This is the approach employed by NeuroML to include very detailed models of synaptic processes using SBML for example.

Tools that support simulator interoperability

neuroConstruct is an example of a successful software application that uses declarative descriptions to its advantage (Gleeson et al. 2007). This software facilitates the creation, visualization, and analysis of networks of multicompartment neurons in 3D space, where a graphical user interface allows model generation and modification without programming. Models within neuroConstruct are based on the simulator-independent NeuroML standards, allowing automatic generation of code for multiple simulators. This has facilitated the testing of neuroConstruct and the verification of its simulator independence, through a process where published models were re-implemented using neuroConstruct and run on multiple simulators as described in Gleeson et al. (2010).

In a different approach, PyNN (Davison et al. 2009), provides a programmatic simulator-independent format. In particular, it provides an API in the Python programming language that supports computational studies using the software simulators NEURON, NEST, PCSIM and Brian, as well as a number of neuromorphic hardware systems (Brüderle et al. 2009, Galluppi et al. 2010). This allows the code for a simulation to be written once and then run on different simulator engines. Unlike declarative specifications, this description is immediately executable without an intermediate translation step, which gives a more direct link between description and results. The use of a programming language also provides the full power of such a language, with loops, conditionals, subroutines and other programming constructs. The great flexibility and extensibility this gives can be a strong advantage, especially in an exploratory phase of model building. It may also be a disadvantage if misused, leading to unnecessary complexity, bugs, and difficulty understanding the essential components of the model, which are less common with declarative specifications.

A third approach involves the standardization of interfaces through which different software components communicate at runtime. This has been used to good effect in MUSIC (Djurfeldt et al. 2010) to allow different parts of a network model to be run on separate simulators. It is also a key aspect of the design of MOOSE, which takes an object oriented approach for the simulation engine and supports a wide range of pluggable components which perform different parts of the calculation. The benefits here are much the same as with plugin architectures for more mainstream applications such as web browsers and word processors. These mainstream applications have benefited from software design philosophies developed over many years (see Gamma et al. (1994) for an overview). In computational neuroscience a first such design philosophy for appropriate modularization of a simulator is described by the GENESIS-3 / Neurospaces CBI architecture (Cornelis et al. 2012b). The benefits of such a modularization are that the plugin developer does not need to master the entire system,

but only the interfaces it must implement, and the same plugin may be used in different contexts. As such, this approach has the potential to reduce the overhead for developing new modules, which is particularly important in a loosely coupled software community like ours. However, some of the pitfalls are also the same as for browser plugins. Development schedules are rarely synchronized, leading to considerable potential for version compatibility problems, and it is much harder to achieve good performance and scalable behavior with a combination of plugins working through a restricted interface than it is with a monolithic system designed as a single entity. This last issue can be addressed by an alternative approach to runtime interaction in which the simulation problem is separated by processing tasks rather than by model components. Each of the different computational tasks involved in running a simulation such as processing model descriptions, operating on cell morphologies, discretizing meshes or solving differential equations can form the basis of a specialized library. This practice is already widespread for some parts of the problem, with independently developed packages often used for XML processing or solving differential equations, but has potential to be extended to finer scales with highly specialized libraries dedicated to particular aspects of model processing and simulation.

Formal specification and verification of connection primitives

Desirably, connectivity on various levels should be describable in one common framework and terminology, irrespective of what is to be connected, e.g. synaptic contact points on dendritic tree structures, point neurons, or also whole populations or areas. These descriptions should be clear and concise, but must not lack crucial details.

For example, if connectivity in a network of N nodes is described as random, it might appear clear that every possible connection between two nodes $i, j \in \{1,2, \dots, N\}, i \neq j$, is established in a Bernoulli-trial fashion with a certain probability p. This network ensemble strictly corresponds to the class of Erdos-Rényi (ER) networks, implying that no connection is established twice (no *multapses*) and no neuron is synaptically connected to itself (no *autapses*), properties that modelers however often allow for. Moreover, for ER random networks there might be nodes that are not connected to any node at all, another feature that is often explicitly excluded. Another assumption often made is that nodes have a pre-described distribution of the number of connections per node, e.g. that all nodes receive exactly (Brunel 2000) or at least (Watts and Strogatz 1998) k connections, while for an ER network both the number of incoming and outgoing connections are distributed binomially (Albert and Barabási 2002).

Even though such connectivity details may appear minor, they can have measurable impact on the dynamics of spiking neuron networks. Assume for a moment that all spike trains are Poisson processes with intensity v and that a neuron receives k input spike trains, all with the same weight w. If all k input currents are independent, the variance of the input current is proportional to w^2kv, while if all currents are sampled from the same neuron it rather corresponds to one spike train of weight wk and the variance is thus instead proportional to w^2k^2v. Similar differences due to multapses are induced in the input current covariances due to common input. Thus minor details potentially alter the entire covariance structure of the network activity and can lead to problems in reproducing results that relate to second order properties if not properly documented.

If the aim is conciseness of description, even the simple balanced random network of Brunel (2000) can become cumbersome. A possible, already lengthy yet incomplete

description could be: “every neuron receives k synapses from randomly drawn subsets of k_e excitatory neurons and $k_i = k - k_e$ inhibitory neurons, such that no connection is established more than once and no neuron connects to itself”. A more concise and formal description in tabular form was suggested in Nordlie et al. (2009). When it comes to more complex network models with additional biologically motivated detail (e.g. Hill and Tononi 2005, Izhikevich and Edelman 2008, Phoka et al. 2012) the situation soon becomes worse, and the benefit of a formalized tabular representation is evident (Nordlie et al. 2009).

A first step towards standardizing the description of connectivity is to agree on a common terminology or ontology, such as the Computational Neuroscience Ontology that is currently under development (http://purl.bioontology.org/ontology/CNO) to unambiguously describe and annotate network models. Along these lines, we suggest unambiguous definitions for the terminology and structure of *connectivity primitives*, defined by community agreement. These can be high-level, relating to network classes such as “ER random networks”, “all-to-all” or “ring networks”, but also can be very low-level primitives, specifying connection patterns on the basis of individual nodes and connections, such as “random convergent connect” (Nordlie et al. 2009). This approach allows for both declarative, shorthand descriptions of networks as a whole, if the network class is well-defined and the procedure of network generation does not matter. It also allows for more refined procedural descriptions in terms of connectivity building blocks, which is particularly important when operations during network generation need to be performed in a certain order. The approach is also useful when connectivity patterns are highly stereotypic and only the parameters vary (as in the models of Hill and Tononi (2005) and Phoka et al. (2012)). Such connectivity patterns can have different properties in that they can be:

node-centric versus set-centric: For a node-centric approach, one might ask, “given a node, what nodes is it connected to?”. Random convergent connections are an example (Nordlie et al. 2009). For a set-centric approach, in contrast, one asks “what characterizes a set of connections?”. As an example, form a set by drawing N connections with random sources and targets.

local versus global: In a local approach, individual connections are established irrespective of the state of the rest of the network. Global requirements establish connections in a way that depends on the state of other nodes or connections.

deterministic versus probabilistic: A deterministic pattern invariably will result in the same connectivity with every instantiation, whereas a probabilistic connectivity pattern specifies the statistics of connectivity across instantiations.

value-dependent versus attribute-dependent: An example of value-dependent connectivity is distance-dependent connections. Attribute-dependent connections are of the form “has property A” or “is of type B” for example.

Finally, boundary conditions should be specified, and if networks are embedded in some type of metric space, also this metric and the node conditions should be given.

Finding naming conventions however often collides with the inertia of established terminology, or simply the complexity of the network objects to be described. Thus a formally minimal, i.e. mathematical, specification of connectivity would resolve the problem of terminological ambiguities. This was recently put forward in the form of the *Connection Set Algebra* (CSA; Djurfeldt 2012). This operator-based approach

automatically resolves the problem of expressing the procedural order of certain network generation operations, since this is inherently expressed in the order of operator composition. In CSA, a set of network connections is represented by an object called a *connection-set*. This object can be subdivided into a *mask*, expressing the existence of connections, and zero or more *value sets*, expressing parameters associated with connections, such as a weight or delay. The CSA uses *index sets* to refer to the nodes (synapses, neurons, etc.) to be connected. For example, when connecting a source and target neuron population, source neurons are enumerated using non-negative integers, which together form an index set I. Similarly, an index set J enumerates the targets. The mask can then be regarded as a set of pairs $i,j,\ i \in I,\ j \in J,$ with one pair per existing connection. This is equivalent to a connection matrix. A value set is a function of source and target indices $I \times J \to \mathbb{R}$.

Connection-sets typically express a type of connectivity, such as “ER random” or “all-to-all” rather than a specific finite set of connections. This is possible because connection-sets are allowed to be of infinite size rather than adapted to the sizes of specific source and target populations. CSA is an algebra over connection-sets, where operators are applied to elementary connection-sets to form the desired connectivity. Given source and target populations of definite sizes, finite portions of a connection type can be “cut out” using the CSA intersection operator. CSA objects and operators can be efficiently implemented as iterators. A demonstration of the CSA implementation in Python is available at the INCF software center (http://software.incf.org/software/csa).

Since the aim of such precise network structure descriptions is reproducibility, there is also a need to specify how to test whether a generated network actually corresponds to the intended structure. This might be a straightforward task, especially if the network is small and connectivity is simple and stereotypic. For example, for a grid network, one might check if each node is connected to its k nearest-neighbors or generate the adjacency matrix and determine whether it has the typical, expected band structure. However, what if the connectivity is probabilistic and each instantiation will be slightly different, or connectivity is dependent on pairwise distances, but the number of potential target nodes in a given distance is not homogeneous? The latter is the case when nodes are embedded on a grid with open boundary conditions: a node in the center of the grid will have the same number of potential targets in all directions, while a node sitting at the boundary of the grid will have none beyond that boundary. So if the connection rule is to connect to all nodes within a certain distance, the number of established connections per node (the degree) depends on the location of the node. An additional complication is that the number of nodes and connections is also often very large so that reading out or storing the complete connectivity for testing purposes may not be practical.

For deterministic networks it is often sufficient to check connectivity for subsets of neurons. For example, for the grid with open boundary conditions mentioned above, one could check if nodes in the center, on the edges and in the corners have the expected number of connections, given the spatial connection profile. For probabilistic networks, measuring the distribution of the number of connections k per node, the degree distribution $P(k)$ can be a useful way to validate the network structure. For the ER network, the expected degree distribution is a Binomial distribution and a Kolmogorov-Smirnov (KS) test can be employed to quantify significance. If connectivity is probabilistic and also dependent on pairwise distance, the number of expected connections of a node at position r, $P(k|r),$ is in general given by the convolution of the spatial connection probability profile and the node density distribution. If this is soluble, as in the case of uniform node density and a Gaussian connectivity profile, the

cumulative density function can be derived and KS testing is again possible. For the topology library in NEST, a comprehensive test suite for all offered standard connection routines is currently under development.

4 Implementation issues and solutions

Because of their high computational processing and memory requirements, and to ease analysis, nearly all existing computational models of neuronal networks are significantly downscaled versions of the corresponding biological system. Downscaling involves using a smaller number of model elements to represent a larger population in the underlying system. Unfortunately, neuronal simulators rarely provide explicit support for such downscaling, leading to *ad hoc* approaches for model scaling that currently make it difficult to interpret, share, and connect models. Moreover, publications very often fail to state explicitly what type of downscaling was used, precisely how the model relates to the underlying system, and what limitations result from the downscaling.

Model scaling should be explicit

Here we primarily focus on neuron, synapse, and dendrite downscaling for clarity, but similar arguments apply to other model elements. Two fundamentally different approaches to downscaling may be distinguished: lumping and subsampling. For lumping, multiple neurons or synapses are combined into larger units for simulation, with properties averaged or summed as appropriate. For subsampling, each model element retains a one-to-one relationship with an element in the biological system, but is treated as a representative of a larger population not explicitly modeled, with adjustments to parameters to compensate for the missing elements. Two types of subsampling can be further distinguished, depending on the spatial layout of the elements: either modeling a small patch at full density (a clustered approach), or a larger patch at low density (a distributed approach). Of course, combinations of approaches are also possible. Clearly, each of these types of downscaling requires different adjustments to parameter values and has different implications for the analysis and interpretation of results. For instance, lumped models will tend to have longer effective time constants than subsampled ones (see for example Borisyuk et al. (2002)), and time delays between neurons will be larger in a distributed sample than in a clustered sample.

For simple firing-rate point neurons, downscaling by lumping or distributed subsampling is reasonably well defined, with linear scaling that works well over a large parameter range (Bednar et al. 2004). For example, each synaptic input to a neuron in a distributed subsampled firing-rate network that simulates 10% of the actual neurons in a region will have to be scaled up by a factor of 10 to represent the contribution from the 90% of the neurons not being modeled. The Topographica simulator provides explicit support for downscaling networks of firing-rate neurons, requiring all parameter values to be expressed independently of the type and amount of downscaling (Bednar 2008). With scale-independent parameters, the amount of downscaling can then be varied easily for each run (e.g. to test that results are robust to downscaling), and the specific scaling assumptions can be reported explicitly in publications.

However, other simulators rarely provide any direct support for downscaling neurons or synapses in networks, and the various *ad hoc* approaches in use can dramatically affect model behavior. Some theoretical results are available that help to systematically downscale numbers of synapses while preserving basic characteristics of the network

dynamics under certain conditions. In the asynchronous irregular state, characteristic of large cortical networks, the summed current-based synaptic input to each neuron is well approximated by a Gaussian noise. The mean and variance of this noise determine the firing rate in networks of binary (van Vreeswijk and Sompolinsky 1998) and integrate-and-fire model neurons (Brunel 2000). Maintaining the same firing rate is one particular choice for defining equivalence between the full-size system and the downscaled version. This can be achieved by an appropriate choice of synaptic weights, the ratio between excitatory and inhibitory weights, and external input (van Vreeswijk and Sompolinsky 1998, Brunel 2000, Burkitt 2006). These results remain useful even under sufficiently mild deviations from the conditions under which they were derived. Modelers should use such theoretical findings in order to increase the chance that downscaled models are in fact comparable to their full-scale counterparts.

Scaling networks of compartmental model neurons connected through conductance-based synapses presents its own unique challenges. The central issue is preserving the effects on the postsynaptic cell of hundreds or thousands of synapses spread across highly branched dendrites. To produce a computationally tractable network model, two compromises must be considered at the cellular level: (1) a reduction in the number of afferent cells, and (2) a reduction in the complexity of the modeled dendrites (by lumping).

A reduction in the number of afferent cells can be handled as a reduced number of synapses on the target cell, with each synapse having a suitably increased peak conductance (a lumping approach). With such scaling, the target cell will receive stronger, more spatially localized inputs. This could distort the postsynaptic response by enhancing nonlinear interactions between inputs and active membrane currents and result in significant distortion of network dynamics (Djurfeldt et al. 2008). For example, dendritic calcium spikes could occur more frequently than expected, due to a few, strong inputs driving the dendritic membrane to the spiking threshold. An alternative is to preserve, as far as possible, the actual number of synapses, but with groups of synapses being driven by the same presynaptic cell (rather than separate cells, as in the real system). The disadvantage of this subsampling approach is that the inputs, though of a realistic number, will have unnatural temporal correlations. For intrinsic network inputs, one of these approaches must be adopted, despite the limitations, if a full-scale network model is infeasible. In order to properly interpret simulation results, different sized networks should be simulated with whichever scaling scheme is adopted in order to assess the likely effects, for example, changing the size of particular neuronal populations while preserving the number of synapses on each target cell (Orban et al. 2006). Extrinsic inputs can be handled with further alternatives. If a cell receives a large number of inputs from outside the network being modeled, then those inputs can be modeled by a reduced population of synapses that preserve the synaptic strength expected in the full population, but receive inputs at a higher mean frequency. This preserves the mean driving conductance across the population of extrinsic inputs, and may still maintain appropriate target cell firing statistics due to this input. This has been demonstrated for a 100-fold reduction in the number of synapses on modeled Purkinje cell dendrites (De Schutter and Bower 1994, Steuber et al. 2007). The fluctuating current arriving at the cell body due to spatially and temporally distributed inputs to the dendrites may possibly be captured by a suitable statistical model, allowing extrinsic inputs to be modeled as a simple current injection into the cell body. Background input to neocortical pyramidal cells has been fitted by a single-variable stochastic model similar to an Ornstein-Uhlenbeck process (Destexhe et al. 2001).

Scaling compartmental models of single neurons is also a necessity when building a large-scale biologically realistic network model, but remains something of an art form when dealing with highly branched dendrites that have active, nonlinear membrane properties. Such nonlinear properties may induce specific local processing of synaptic inputs (see for example Poirazi et al. (2003)). This provides constraints on the level of morphological detail that needs to be preserved in any reduced compartmental model. Detailed single cell modeling may need to be undertaken to try to understand how to preserve such local processing in a simplified dendritic morphology. A two or three compartment model that captures the distinction between input to the cell body and to the dendrites may, in fact, be too simple in many cases.

Overall, due to the complex relationship between the downscaled system in these simulations and the underlying biological system, it is critical for simulators to start to provide explicit support for downscaling. One possible ideal (implemented in Topographica, but not yet in spiking simulators) would be to specify all parameters and architecture in terms of the biological system being modeled, and then separately specify the scale to be used in a particular simulation, as well as whatever scaling (linear or nonlinear) is necessary to map between the original and downscaled systems. Publications can then report both the unscaled network, as an explicit statement of what assumptions are being made about the underlying system; and the scaling equations, as an explicit statement of what assumptions are being made about how the simulation relates to the real system.

Hardware approaches to large-scale modeling

An alternative approach to dealing with scaling issues is to attempt to simulate networks at the scale of the brain; however, the memory demands of such large networks are prohibitive. The use of generators rather than specific positions for connectivity, where data are stored efficiently and accessed as needed, allows for the implementation of larger networks using traditional software approaches (Smith, 1992). But recently, the desired increase to brain-scale neuronal network models has driven the development of novel neuromorphic computer architectures. The use of GPU implementations (see for example Nageswaran et al. (2009)) has resulted in the development of specialized simulator environments as well, such as NeMo (http://nemosim.sourceforge.net) and GeNN (http://sourceforge.net/projects/genn). However, these large-scale GPU based simulations still require a great deal of memory, limiting overall speed-up. The use of specialized chips with on-chip communication networks for direct spike transfer can dramatically improve run-time requirements. This innovative approach is the basis of the SpiNNaker (Spiking Neural Network Architecture) machine, which is a multi-processor machine designed specifically to run large-scale networks of point neuron models in biological real time using millisecond integration steps (Furber and Temple 2007). It has a bespoke communications infrastructure tuned to carry very large numbers of small packets, where each packet conveys information about one “spike”. As neuron, synapse and plasticity models are implemented in software, there is considerable flexibility in how these are used, and hybrid networks can be accommodated. In principle neural network models can be mapped onto SpiNNaker from any suitable high-level description language. An automated design flow from PyNN has been established, which maps the network topology into the SpiNNaker packet routing hardware and maps neuron populations onto one or more processors, loading leaky integrate and fire or Izhikevich neuron models (Izhikevich 2004) from a library as required, where some synaptic plasticity models, such as spike-time-depending plasticity, are available.

The SpiNNaker execution model employs asynchronous concurrency, so model results are to a degree non-deterministic, although there is a millisecond synchronous mode that supports deterministic operation provided that the neuron input processes are linear (as it is impractical to control the order in which input spike packets arrive within each millisecond). The non-determinism of the normal operating mode makes direct comparison at the level of individual spike times with the results from other platforms problematic, although higher-level network properties normally will be comparable. The SpiNNaker machine also performs all computations using fixed-point arithmetic, again making direct comparisons difficult with most other platforms that use floating-point arithmetic. These compromises allow for the execution of spiking neural networks of up to a billion point neurons distributed across a million processors in biological real time, but lead to questions about how network results should be interpreted and how they can be compared across architectures. However, similar questions arise when comparing results across supposedly deterministic simulation environments, as described by Crook et al. (2012), and these issues should be the subject of further discussion in the community.

5 What can our community do to help?

Although recent efforts to create an infrastructure throughout the computational neuroscience community for describing and sharing models are promising, much more is needed to improve efficiency and ensure reproducibility in our field. There are several tasks that should be explicitly embedded in simulator environments to aid with some of the issues outlined here. In particular, simulator developers should ensure that models and model components can be shared easily using multiple description standards such as tables, graphs, connection matrices and simulator-independent description languages, which would aid modelers in effectively documenting models for publication and exchange. In addition, to promote reproducibility, simulators should provide more self-documentation such as unit tracking, records of parameter values, version control approaches, and explicit descriptions of model assumptions. As outlined above, a more formalized approach to model scaling is needed to aid in interpreting results and linking models across scales. The community should also continue to invest in efforts to standardize terminology through the development of ontologies and formal notation, and to standardize libraries and interfaces across tools, and a sincere effort to create benchmarks for simulator testing is needed.

Acknowledgments

The workshop that resulted in this work was supported in part by the National Institute of Mental Health under grant R01MH061905 to SMC and in part by the Research Council of Norway under grant 178892/V30 eNeuro. Additional funding was provided by the Institute of Adaptive and Neural Computation in the School of Informatics at the University of Edinburgh and the Scottish Informatics and Computer Science Alliance.

References

Albert R, Barabási A-L. 2002. Statistical mechanics of complex networks. Reviews of Modern Physics, 74:47–97.

Bednar J. 2008. Understanding neural maps with Topographica. Brains, Minds, and Media 3:bmm1402.

Bednar JA. 2009. Topographica: Building and analyzing map-level simulations from Python, C/C++, MATLAB, NEST, or NEURON components. Frontiers in Neuroinformatics 3:8. doi: 10.3389/neuro.11.008.2009

Bednar JA, Kelkar A, Miikkulainen R. 2004. Scaling self-organizing maps to model large cortical networks. Neuroinformatics 2:275–302.

Borisyuk A, Semple MN, Rinzel J. 2002. Adaptation and inhibition underlie responses to time-varying inter aural phase cues in a model of inferior colliculus neurons. Journal of Neurophysiology 88:2134– 2146.

Bower J, Beeman D. 1997. The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System. New York: Springer.

Bray T, Paoli J, Sperberg-McQueen CM, Maler E, Yergeau F. 2008. Extensible markup language (XML) 1.0. [cited Jun 2012]. Available: http://www.w3.org/TR/REC-xml

Brette R, Rudolph M, Carnevale T, Hines M, Beeman D, Bower J M, Diesmann M, Morrison A, Goodman P H, Jr F C H, Zirpe M, Natschläger T, Pecevski D, Ermentrout B, Djurfeldt M, Lansner A, Rochel O, Vieville T, Muller E, Davison A P, Boustani S E, Destexhe A. 2007. Simulation of networks of spiking neurons: A review of tools and strategies. Journal of Computational Neuroscience 23:349–398.

Brüderle D, Müller E, Davison A, Muller E, Schemmel J, Meier K. 2009. Establishing a novel modeling tool: A Python-based interface for a neuromorphic hardware system. Frontiers in Neuroinformatics 3:17. doi:10.3389/neuro.11.017.2009

Brunel N. 2000. Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. Journal of Computational Neuroscience 8(3):183–208.

Burkitt AN. 2006. A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. Biological Cybernetics 95:1–19.

Carnevale NT, Hines ML. 2006. The NEURON Book. Cambridge: Cambridge University Press.

Cornelis H, De Schutter E. 2003. Neurospaces: Separating modeling and simulation. Neurocomputing 52-54:1079–1084.

Cornelis H, Coop AD, Bower JM. 2012a. A federated design for a neurobiological simulation engine: the CBI federated software architecture. PLoS ONE 7(1):e28956.

Cornelis H, Rodriguez AL, Coop AD, Bower JM. 2012b. Python as a federation tool for GENESIS 3.0. PLoS ONE 7(1):e29018.

Crook S, Davison AP, Plesser HE. 2012. Learning from the past: Approaches for reproducibility in computational neuroscience. In: Bower JM, editor. 20 Years of Computational Neuroscience, Springer Series in Computational Neuroscience. New York: Springer.

Crook S, Gleeson P, Howell F, Svitak J, Silver RA. 2007. MorphML: Level 1 of the NeuroML standards for neuronal morphology data and model specification. Neuroinformatics 5:96–104.

Davison AP, Brüderle D, Eppler JM, Kremkow J, Muller E, Pecevski DA, Perrinet L, Yger P. 2009. PyNN: a common interface for neuronal network simulators. Frontiers in Neuroinformatics 2:11. doi:10.3389/neuro.11.011.2008

Davison AP, Morse TM, Migliore M, Marenco L, Shepherd GM, Hines ML. 2002. ModelDB: a resource for neuronal and network modeling. In: Kötter R, editor. Neuroscience Databases: A Practical Guide. Norwell, MA: Kluwer Academic Publishers. p99–122.

De Schutter E, Bower JM. 1994. An active membrane model of the cerebellar Purkinje cell: II. Simulation of synaptic responses. Journal of Neurophysiology 71:401–419.

Destexhe A, Rudolph M, Fellous J-M, Sejnowski TJ. 2001. Fluctuating synaptic conductances re-create *in vivo*-like activity in neocortical neurons. Neuroscience 107:13–24.

Djurfeldt M. 2012. The Connection-set Algebra—a novel formalism for the representation of connectivity structure in neuronal network models. Neuroinformatics 10:287–304.

Djurfeldt M, Hjorth J, Eppler J, Dudani N, Helias M, Potjans T, Bhalla U, Diesmann M, Hellgren-Kotaleski J, Ekeberg O. 2010. Run-time interoperability between neuronal network simulators based on the music framework. Neuroinformatics 8:43–60.

Djurfeldt M, Ekeberg O, Lansner A. 2008. Large-scale modeling - a tool for conquering the complexity of the brain. Frontiers in Neuroinformatics 2:1. doi: 10.3389/neuro.11.001.2008

Eberhard JP, Wanner A, Wittum G. 2006. Neugen: A tool for the generation of realistic morphology of cortical neurons and neural networks in 3D. Neurocomputing 70:327–342.

Eppler JM, Helias M, Muller E, Diesmann M, Gewaltig M-O. 2008. PyNEST: A convenient interface to the NEST simulator. Frontiers in Neuroinformatics 2:12. doi: 10.3389/neuro.11.012.2008

Furber S, Temple S. 2007. Neural systems engineering. Journal of the Royal Society Interface 4:193-206.

Galluppi F, Rast A, Davies S, Furber S. 2010. A general-purpose model translation system for a universal neural chip. In: Wong K, Mendis B, Bouzerdoum A, editors. Neural Information Processing. Theory and Algorithms, volume 6443 of Lecture Notes in Computer Science. Berlin/Heidelberg: Springer. p58–65.

Gamma E, Helm R, Johnson R, Vlissides J. 1994. Elements of Reusable Object-Oriented Software. AW Publishing.

Gewaltig M-O, Diesmann M. 2007. NEST (NEural Simulation Tool). Scholarpedia, 2(4):1430.

Gleeson P, Crook S, Cannon RC, Hines ML, Billings GO, Farinella M, Morse TR, Davison AP, Ray S, Bhalla US, Barnes SR, Dimitrova YD, Silver RA. 2010. NeuroML: A language for describing data driven models of neurons and networks with a high degree of biological detail. PLoS Comput Biol 6(6):e1000815.

Gleeson P, Steuber V, Silver RA. 2007. neuroConstruct: a tool for modeling networks in 3D space. Neuron 54:219–235.

Goddard N, Hucka M, Howell F, Cornelis H, Shankar K, Beeman D. 2001. NeuroML: model description methods for collaborative modelling in neuroscience. Philosophical Transactions of the Royal Society B 356:1209–1228.

Goodman D, Brette R. 2008. Brian: a simulator for spiking neural networks in Python. Frontiers in Neuroinformatics 2:5. doi: 10.3389/neuro.11.005.2008

Hedley WJ, Nelson MR, Nielsen PF, DP Bullivant, Hunter PJ. 2000. XML languages for describing biological models. In Proceedings of the Physiological Society of New Zealand, volume 19.

Henker S, Partzsch J, Schüffny R. 2012. Accuracy evaluation of numerical methods used in state-of-the-art simulators for spiking neural networks. J Comput Neurosci 32:309–326.

Hill S, Tononi G. 2005. Modeling sleep and wakefulness in the thalamocortical system. Journal of Neurophysiology, 93(3):1671–1698.

Hines M. 1989. A program for simulation of nerve equations with branching geometries. Int. J. Biomed. Comput. 24:55–68.

Hines ML, Carnevale NT. 1997. The NEURON simulation environment. Neural Comput. 9(6):1179–1209.

Hines ML, Morse T, Migliore M, Carnevale NT, Shepherd GM. 2004. ModelDB: A database to support computational neuroscience. J Comput Neurosci 17(1):7–11.

Hucka M, Finney A, Sauro HM, Bolouri H, Doyle JC, Kitano H, Arkin AP. 2003. The systems biology markup language (SBML): A medium for representation and exchange of biochemical network models. Bioinformatics 19:524–531.

Izhikevich EM. 2004. Which model to use for cortical spiking neurons? IEEE Transactions on Neural Networks 15:1063-1070.

Izhikevich EM, Edelman GM. 2008. Large-scale model of mammalian thalamocortical systems. Proceedings of the National Academy of Sciences 105(9):3593–3598.

Koene RA, Tijms B, van Hees P, Postma F, de Rikker S, Ramakers G, van Pelt J, van Ooyen A. 2009. Netmorph: A framework for the stochastic generation of large scale neuronal networks with realistic neuron morphologies. Neuroinformatics 7:195–210.

Köhn D, Le Novère N. 2008. SED-ML—an XML format for the implementation of the MIASE guidelines. In: Heiner M, Uhrmacher A, editors. Computational Methods in Systems Biology, volume 5307 of Lecture Notes in Computer Science. Berlin / Heidelberg: Springer. p176–190.

Le Novère N, Bornstein B, Broicher A, Courtot M, Donizelli M, Dharuri H, Li L, Sauro H, Schilstra M, Shapiro B, Snoep JL, Hucka M. 2006. BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. Nucleic Acids Research 34(Database issue):D689–691.

Le Novére N, Hucka M, Moodie S Mi H, Schreiber F, Sorokin A, Demir E, Wegner K, Aladjem MI, Wimalaratne SM, Bergman FT, Gauges R, Ghazal P, Kawaji H, Li L, Matsuoka Y, Villger A., Boyd SE, Calzone L, Courtot M, Dogrusoz U, Freeman TC, Funahashi A, Ghosh S, Jouraku A, Kim S, Kolpakov F, Luna A, Sahle S, Schmidt E, Watterson S, Wu G, Goryanin I, Kell DB, Sander C, Sauro H, Snoep JL Kohn K, Kitano H. 2009. The systems biology graphical notation. Nat Biotechnol 8: 735–741.

Lloyd CM, Lawson JR, Hunter PJ, Nielsen PF. 2008. The CellML model repository. Bioinformatics 24(18):2122–2123.

Lloyd CM, Halstead MDB, Nielsen PF. 2004. CellML: Its future, present and past. Progress in Biophysics and Molecular Biology 85:433–450.

Migliore M, Morse TM, Davison AP, Marenco L, Shepherd GM, Hines ML. 2003. ModelDB: Making models publicly accessible to support computational neuroscience. Neuroinformatics 1(1):135–139.

Nageswaran JM, Dutt N, Krichmar JL, Nicolau A, Veidenbaum AV. 2009. A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors. Neural Networks 22:791-800.

Nordlie E, Gewaltig M-O, Plesser HE. 2009. Towards reproducible descriptions of neuronal network models. PLoS Computational Biology 5(8).

Nordlie E, Plesser HE. 2010. Visualizing neuronal network connectivity with connectivity pattern tables. Frontiers in Neuroinformatics 3(39):1-15. doi: 10.3389/neuro.11.039.2009

Orban G, Kiss T, Erdi P. 2006. Intrinsic and synaptic mechanisms determining the timing of neuron population activity during hippocampal theta oscillations. Journal of Neurophysiology 96:2889–2904.

Pecevski D, Natschläger T, Schuch K. 2009. PCSIM: A parallel simulation environment for neural circuits fully integrated with Python. Frontiers in Neuroinformatics 3:11. doi: 10.3389/neuro.11.011.2009

Petersonn BE, Healy MD, Nadkarni PM, Miller PL, Shepherd GM. 1996. ModelDB: an environment for running and storing computational models and their results applied to neuroscience. J Am Med Inform Assoc 3:389–398.

Phoka E, Wildie M, Schultz SR, Barahona M. 2012. Sensory experience modifies spontaneous state dynamics in a large-scale barrel cortical model. Journal of Computational Neuroscience DOI: 10.1007/s10827-012-0388-6.

Poirazi P, Brannon T, Mel BW. 2003. Pyramidal neuron as two-layer neural network. Neuron 37: 989–999.

Ray S, Bhalla US. 2008. PyMOOSE: interoperable scripting in Python for MOOSE. Frontiers in Neuroinformatics. 2:6. doi: 10.3389/neuro.11.006.2008

Smith LS. 1992. A framework for neural net specification. IEEE Transactions on Software Engineering 18: 601–612.

Steuber S, Mittmann W, Hoebeek FE, Silver RA, De Zeeuw CI, Hausser M, De Schutter E. 2007. Cerebellar LTD and pattern recognition by Purkinje cells. Neuron 54:121–136.

Usui S. 2003. Visiome: neuroinformatics research in vision project. Neural Networks 16:1293–1300.

van Vreeswijk C, Sompolinsky H. 1998. Chaotic balanced state in a model of cortical circuits. Neural Computation 10:1321–1371.

von Neumann J. 1986. Papers of John von Neumann on Computers and Computing Theory. Cambridge, MA: MIT Press.

Watts DJ, Strogatz SH. 1998. Collective dynamics of small-world networks. Nature 393:440–444.

Wilson HR, Cowan JD. 1972. Excitatory and inhibitory interactions in localized populations of model neurons. Biophysical Journal 12:1-24.

Zubler F, Douglas R. 2009. A framework for modeling the growth and development of neurons and networks. Frontiers in Computational Neuroscience, 3:25. doi: 10.3389/neuro.10.025.2009