Collaborative modelling: the future of computational neuroscience?

Andrew P. Davison

Unité de Neurosciences, Information et Complexité (UNIC),
CNRS UPR 3293, Gif sur Yvette, France

This is the pre-peer reviewed version of the following article:

Davison A.P. (2012) Collaborative modelling: the future of computational neuroscience? *Network: Computation in Neural Systems* doi: 10.3109/0954898X.2012.718482

which has been published in final form at

`http://informahealthcare.com/doi/abs/10.3109/0954898X.2012.718482`

Abbreviated title: Collaborative modelling in computational neuroscience

Declaration of interest: The author reports no conflicts of interest. The author alone is responsible for the content and writing of the paper.

Abstract

Given the complexity of biological neural circuits and of their component cells and synapses, building and simulating robust, well-validated, detailed models increasingly surpasses the resources of an individual researcher or small research group. In this article, I will briefly review possible solutions to this problem, argue for open, collaborative modelling as the optimal solution for advancing neuroscience knowledge, and identify potential bottlenecks and possible solutions.

Introduction

Biological nervous systems, from the entire 358-cell system of *C. elegans* to the human brain, are very complex, with very many moving parts interacting in many ways. Understanding how brains or brain regions work will require modelling these biological neuronal networks at many levels of abstraction and at many spatial and temporal scales. In this article I will focus on data-driven modelling with a high level of detail, aimed at understanding low-level mechanisms and how higher-order properties arise from the interactions of individual elements.

Given that we wish to model a complicated structure with a very large number of elements, that multiplexes a large number of different computations within the same physical structure, whose properties are difficult to measure, and yet about which there is a vast and rapidly increasing experimental literature, is it possible for an individual scientist or a small research group to develop and simulate an accurate and complete model of a brain region or other neuronal network?

Until recently, almost all detailed modelling studies of individual neurons and of neuronal networks have been the work of individual researchers and small teams, from the foundational squid axon model of Hodgkin and Huxley (1952) to the whole-brain simulations of Izhikevich and Edelman (2008). There have been exceptions to this rule, such as the "Eternal Purkinje Cell", a model originally developed by De Schutter and Bower (1994a,b) which has since been refined, extended and incorporated into network models in many laboratories, but even

in these cases different research groups take the original model in different directions, and such changes are not generally merged together.

In the last few years, several projects have emerged that take a more industrial approach to neuronal modelling, assembling experimental biologists, computational neuroscientists, computer scientists and professional programmers into large teams that attempt to model neural circuits with tens of thousands of detailed multicompartmental neurons and millions of synapses, and simulate these models on supercomputers. The first of these was the Blue Brain Project (Markram, 2006); the Allen Institute for Brain Science and Brain Corporation have announced similar plans (Allen Institute for Brain Science, 2012; Brain Corporation, 2012), while the Human Brain Project is seeking funding to scale up the Blue Brain Project's cortical column model to whole-brain models (Walker, 2012).

The need to assemble large teams to work collaboratively on a single model arises when the size and complexity of a model passes a certain threshold. Managing the data used to build and test the model, simulating the model on large clusters and supercomputers, managing storage and analysis of output data, developing the software framework that links everything together, all become large tasks in their own right, and a division of labour is needed.

There are several ways in which to assemble a team for collaborative modelling. One is through formal, hierarchical structures such as the Blue Brain Project or Allen Institute. Most of neuroscience, however, is structured into individual labs: is it possible for such labs, through more informal collaborations, to assemble the large teams necessary to develop large-scale, detailed, data-driven models of neural circuits?

The example of the open source software movement suggests that it certainly is possible. A huge amount of high quality software is now developed by informal teams working on different continents, working for different companies or in their spare time, rarely or never meeting in person but using assorted electronic methods of communication to structure their work. In neuroscience, many scientists have started to use the same approaches to develop scientific software, initially mainly for tool development but now also specifically for model development. The OpenWorm project (`http://www.openworm.org/`) is an informal, unfunded collaboration of mostly junior scientists which aims to develop a complete model of *C. elegans* and has already made considerable progress. The Open Source Brain Initiative (`http://www.opensourcebrain.org`) is building a platform dedicated to open, public, collaborative development of neuroscience models, while the well-established ModelDB database (Hines et al., 2004) is adding support for models with multiple versions, which develop over time.

Such initiatives are only in their infancy at present. Whether or not they will succeed depends both on solving technical problems and on resolving a number of social issues related to professional recognition and scientific career structures. In the rest of this article, I will review some of the technical requirements for open, collaborative modelling before discussing the social issues that must be addressed.

Technical considerations

From a technical perspective, collaborative modelling has a number of requirements.

Interoperability between simulators and/or common modelling languages Given that collaborative network models are likely to consist of multiple components developed in different labs, it is unlikely that they will all have been developed using a single simulation environment or a single programming language. One option is to convert all components to run in a single simulation environment, but this is likely to be extremely time consuming, and must be repeated every time a new simulator is used. Two better solutions, which may be used separately or together, are tools for simulator run-time communication and tools for simulator-independent model and experiment description.

If two or more simulators can be made to communicate the times of spike events and/or values of state variables during a simulation, it is

possible to run each component in the environment for which it was originally developed and avoid translation completely. This can be done in an *ad hoc* fashion – advancing each simulator for a small time interval, swapping data via some mechanism such as the filesystem, and then advancing again – or using a dedicated interface library. The *ad hoc* approach is particularly suitable when there is a large difference in time scales between the two simulators. Examples of dedicated interfaces are MUSIC (Djurfeldt et al., 2010) and PLATONIC (Kannon et al., 2011). MUSIC, for example, takes care of efficiently transmitting spike times and the values of state variables between two or more parallel simulators (or other tools such as visualizers) while respecting the possibly different time steps used in the different simulators.

Simulator-independent model descriptions allow a model to be developed without considering which simulator will be used to simulate it. This has several advantages, in particular making it simple to run models on different simulators and compare the results, and allowing different labs, using different simulation platforms, to collaborate on the same model. There are two general approaches to simulator-independent model descriptions. The first is to use a declarative, XML-based language, which can then be interpreted by a simulator or used to generate simulator-specific code. The main examples of this approach in computational neuroscience are NeuroML (Gleeson et al., 2010) and NineML (Raikov and the INCF Multiscale Modeling Taskforce, 2010). The second approach is to define a common application programming interface (API) for neuronal simulations and then to implement this interface using the native programming language of each simulator. The latter approach is taken by PyNN (Davison et al., 2009).

Version control systems Collaborative modelling implies that more than one person will be contributing code to the model and that the model will change over time. This implies a need for merging (possibly conflicting) contributions, having a unique identifier for a particular version/snapshot of the code and, probably, being able to identify who contributed what. All of these functions are provided by modern version control systems (VCS), specifically those that support atomic operations: this includes Subversion, Mercurial, Git, Bazaar, Perforce, Darcs, Monotone, etc. but excludes the once-popular CVS.

An important factor in choosing a VCS is whether it is client-server (like Subversion) or distributed (like Git, Mercurial). A client-server system has a central server containing the repository, with all the history of contributions and changes. Each developer has a copy of the files, but does not have a copy of the history. A commit or check-in (merging local changes into the repository) always requires a network connection. With a distributed system, every developer has a copy of the entire repository with all the history. A commit can be done to the local repository without needing a network connection. A network connection is needed only when merging changes made to different copies of the repository. It is possible to have a central server with the "definitive" copy of the repository, or developers can work in an entirely peer-to-peer manner.

For collaborative modelling, there is a clear advantage to having a central, definitive copy of the repository, in order to avoid the model splintering into multiple, incompatible versions, each specific to one lab. Using a distributed VCS requires greater discipline from developers to merge changes into the central repository frequently, and hence does increase the ease with which splintering into incompatible versions can occur, while using a client-server VCS *requires* the merges to be performed at each check-in. However, the advantages of using a distributed VCS (local commits, better automated merging, easier set-up) outweigh this single advantage of the client-server approach in most circumstances.

The final aspect to consider in the choice of a VCS is tool support, in particular web-based repository hosting. Of the most popular repository hosting platforms, Sourceforge (`http://sourceforge.net`) supports Subversion, Git, Mercurial, Bazaar and CVS, Google Code (`http://code.google.com`) supports Subversion, Mercurial and Git, Bitbucket (`https://bitbucket.org`) supports Mercurial and Git, GitHub (`https://github.com`) supports only

Git.

Legal clarity Source code is protected by copyright laws. Collaborative modelling therefore requires that all contributors explicitly declare that other people are allowed to copy and modify their code. The best way to do this is to use a standard open-source licence for source code, such as the GNU Public License (GPL) or the Modified Berkeley Software Distribution (BSD) License. The Open Source Initiative maintains a list of approved open source licenses at `http://www.opensource.org/licenses/`. The most important consideration in choosing a licence is the distinction between "copyleft" and "permissive" licences. Copyleft means that modified versions of the software must use the same licence, while permissive licences do not have such a requirement.

In the past, it has been rare for shared model code to have an explicit licence - a large number of models in ModelDB, for example, have no licence information, nor information about allowed use of the code. In most of these cases, the authors probably believe they are placing the code in the public domain, but from a legal point of view it is unlikely they have achieved this aim.

A much more comprehensive study of the legal framework for sharing scientific code is presented by Stodden (2009).

Communication tools Like other collaborative activities, collaborative modelling requires good communication tools. In the age of social media, it is hardly worthwhile to list the currently most popular tools. It is useful, however, to list the capabilities that such tools should support when used for collaborative modelling. First, there is a need for both asynchronous and synchronous communication. Since collaborators on a model may be in different time zones, there is a clear advantage in being able to leave messages that others will read later, but sometimes there is no substitute for live interaction. Second, communications should be archived. When new people join the project, it is very helpful to be able to understand the reasons for past decisions by reference to the discussions that took place at the time. Third, there should be support for structuring communication according to discrete issues, to keep discussion of a particular point focused and accessible in one place. Fourth, it should be possible to participate in discussion and have access to an overview of the project activity without being a member of the core team and without having to follow all the small details. This allows occasional contributions from people who are interested in the project but do not have time to be involved on a regular basis.

It is clear that most of the technical requirements for collaborative modelling are easily satisfied, but a small number of bottlenecks remain. Where a requirement for code sharing is not specific to neuroscience – for version control, clear licensing and communication tools – there are many available solutions, mostly due to the rise of the open-source software movement. Inasmuch as problems remain they are due to a lack of knowledge by neuroscientists of these tools and as such can be remedied by education and training.

The requirements for simulator operability and common modelling languages have come a long way towards being fully satisfied in the last few years, but there are still several points of friction. Version 1 of the NeuroML specification (Gleeson et al., 2010) makes it much easier to share morphologically detailed, biophysical models of neurons and of small networks between users of different simulators, but it lacks general support for phenomenological models such as those of the integrate-and-fire family, lacks the ability to explicitly specify the mathematics of model components, and has weak support for describing the connectivity of large-scale networks. All of these shortcomings are being addressed in the efforts to develop NeuroML version 2, as well as in the INCF-initiated NineML project. The PyNN API makes it possible to develop simulator-independent models for any simulator that has a Python interface, which is currently most of the widely-used simulators, and greatly simplifies porting models from one simulator to another by allowing incremental, rather than all-at-once, conversion , but it does not cur-

rently support morphologically detailed models, and it is of course restricted to a single programming language. Projects underway to support import/export of NeuroML-definitions by/from PyNN and generation of Python code from a NeuroML description could potentially provide the best of both worlds, combining the stability and language-independence of NeuroML descriptions with the flexibility of software development in Python. Finally, run-time interoperability using libraries such as MUSIC has now been demonstrated, but has only recently begun to be incorporated into existing simulation tools and so has not yet been widely used for real-world problems.

Social and scientific considerations

Science has long had a tension between the need for open communication of results and methods between researchers so that science can progress, and the wish of individual scientists to gain and maintain an advantage in the competition for resources and advancement in the scientific career structure. An increase in open, collaborative modelling would definitely increase the ease and pace of communication between researchers, and as such should be expected to have a beneficial effect on the rate of advancement of science, but it is unclear what the effect would be on the careers of individual scientists.

A number of considerations arise for scientists engaged in open, collaborative modelling.

Publication An open model would be expected to continuously evolve as it is incrementally improved, with occasional major code changes to reduce complexity, improve performance or take advantage of some new technology. The original model developers may drop out of active development as their career progresses or research interests change, and new collaborators may come along. A publication about, or using, the model captures a snapshot of the model at one point in time. The question of authorship then becomes particularly challenging. Who should be an author on the manuscript? Everyone who has ever contributed to the model? Only those who have contributed since the last publication about/using the model? Those who have contributed to the particular features that are investigated in the paper? If someone has made a large contribution to the code but has had minimal input into writing the paper, should they be listed? In what order should contributors be listed? None of these questions are unique to collaborative modelling projects, and most journals have guidelines which can help to resolve these issues, but the more informal the collaboration, the more acute they will become.

Recognition and impact There is a wider conversation going on about how to evaluate scientific careers, and on using metrics for accomplishment other than the traditional journal impact factor (e.g., Neylon and Wu, 2009). Open, collaborative modelling is clearly an area where scientists can make a large contribution and have considerable impact without publishing, or in addition to publishing, traditional scientific papers. The number of downloads or checkouts of the code base, the number of derivative projects, and the number of lines of code written are all possible metrics that could be used to evaluate scientists' productivity. Each of these metrics has flaws; if several metrics are combined, how should they be weighted with respect to each other and with respect to traditional measures of impact? Adding a new feature to a model is more likely to result in a publication than adding new tests of the code, but it might well be the tests that add the most value by increasing the confidence we can have in the model results. How to motivate scientists to work on coding activities that are less personally rewarding but of higher value for the project and for science?

Bug fixes and retractions Almost all code, with the exception of safety-critical software for aircraft, nuclear reactors, etc., has bugs and other errors. Scientific code, developed by the most part by non-professional programmers and with a small number of users, is likely to have a higher rate of errors than the average. It is therefore probable that a large proportion of the code used to generate published arti-

cles has bugs, most of which will have small effects that do not affect the studies' conclusions, but some of which will completely invalidate them (see for example Chang et al., 2006; Brunini, 2006). At the present time, since most scientific code is not shared, most of these bugs are not found, but as code sharing increases, and especially with code in open repositories, more such bugs will be found. For the major errors, formal retractions are a workable mechanism, although not without problems. For the more numerous small bugs, what should happen when one is found? It may be very straightforward to regenerate the figures or re-run the statistical tests with the bug corrected, but where should the new figures, or the new version of the manuscript be placed, and how can someone who discovers the original version be notified that a corrected version exists? Most journals do not have any mechanism for this. Repositories such as arXiv (`http://arxiv.org/`), which allow posting of multiple versions of a manuscript, demonstrate one way forward. Tools, such as FigShare (`http://figshare.com/`), that allow more fine-grained sharing of scientific results, are another.

While finding bugs in scientific code creates problems, it also of course allows them to be fixed, increasing the quality of the science. This is one of the major value propositions of open, collaborative modelling (and of open-source software generally), that by having more people examining and using the code, more of the bugs will be found. In the words of Raymond (1999, p. 30), "given enough eyeballs, all bugs are shallow".

Competitive advantage If a scientist has invested x years of effort in developing a model, and then publishes a description of the model but does not share the code, then it will take another scientist y years of effort, where y is probably less than x (since the description is public) but is certainly not zero, to re-implement the model and then test or extend it. If the first scientist publishes their code at the same time as the manuscript, the second scientist will take a much smaller time z (the time taken to download, install and understand the code) to arrive at the point where the model can be tested or extended. The difference between y and z, which will be at least a few months and could easily be several years, is a net loss to the progress of science , but could be of considerable benefit to the first scientist, since he or she can spend this time extending the model him/herself, or in developing a new model, which will lead to more publications and will advance his or her career. The disincentive to sharing models, especially for young scientists who are not yet established, is very evident.

For open, collaborative modelling to become more widely adopted, which is to the benefit of science as a whole, then the disincentive described above must be counterbalanced by an equally large or larger incentive to share models. Here there is anecdotal evidence that sharing scientific code increases the number of citations of an article, and generally boosts the authors' reputations. I am not aware of any systematic studies of this issue, however, and so the decision to share code remains a leap of faith. This situation could be changed by three things: first, if studies are conducted that demonstrate a net benefit of sharing to scientific reputations; second, if the weighting of formal publications in evaluating impact is reduced relative to other scientific outputs; third, if funding agencies or journals begin to require sharing of the code related to a publication.

So far in this section I have not addressed the distinction between sharing after publication and placing code in an open, publicly accessible repository from the beginning of a research project. Having code be completely open from the beginning adds a risk of being "scooped", of someone else using your code for a publication before you've had time to publish yourself. Set against that is the possibility of attracting strong collaborators, and of the final publication being of higher quality (due to fewer bugs, more discussion prior to publication) and being more highly cited (as it relates to an already open model that is more likely to be built-on than grudgingly-shared one-off code).

Model diversity One possible scientific concern with open, collaborative modelling is that the reduction of diversity in the ecology of mod-

els in a given area (as more scientists work on the same model, rather than each group developing its own) will lead to fewer ideas being explored, as everyone concentrates on only a few, well-accepted lines of research. I think this concern is reasonable, and as a community we should always be aware of the dangers of such "group-think", I think it is unlikely to be a problem in practice for the type of complex, data-driven model that will most benefit from collaborative modelling. Here there are so many details that there is plenty of space for many scientists to push the model forward in different directions at once, while at the same time many of the details will not have a strong bearing on any given scientific question under study , and so there is a considerable advantage in being able to use well-established, uncontroversial, common components for such parts of a model: this improves model robustness and makes model comparison easier.

Conclusion

If one accepts that large-scale, data-driven modelling is an essential tool in achieving a thorough understanding of brain function, then it follows that models must increasingly be developed by large groups, to cover the various competencies in data management, simulation technologies, high performance computing, and data analysis that are needed, and to follow up the many pathways of investigation that such detailed models provide. Such large groups may be assembled through centralised, formal projects or corporate structures, but may also be assembled in more informal, distributed collaborations, or through some combination of these approaches. This way of working has some parallels with high-energy particle physics, in which there are a small number of central facilities (accelerators in physics, computer clusters and supercomputer centres in large-scale modelling), each of which hosts a small number of experiments (centred around detectors in particle physics, common models in neuroscience), each of which is developed by distributed teams from many universities. Each individual research group then pursues their own research interests based on shared data obtained from the common infrastructure they helped develop. This works very well in high-energy physics. For it to work in computational neuroscience requires the solution of a number of technical challenges, but this is well underway. More important are the social challenges, for which solutions are further away. To commit to open, collaborative modelling, in which most of the scientists from a sub-field of computational neuroscience work on a small number of shared models, requires a certain bravery, especially for young scientists, but the potential rewards are enormous.

Acknowledgments

References

Allen Institute for Brain Science 2012. Paul G. Allen commits $300M to expand the Allen Institute for Brain Science to drive toward a complete understanding of how the brain works [Internet]. Seattle, WA [cited 2012 Jun 19]. Available from: `http://www.alleninstitute.org/Media/documents/press_releases/2012_0321_PressRelease_ExpansionAnnouncement.html`.

Brain Corporation 2012. Technology [Internet]. San Diego, CA [cited 2012 Jun 19]. Available from: `http://braincorporation.com/index.php/category/technology`.

Brunini, A. 2006. Retraction: Origin of the obliquities of the giant planets in mutual interactions in the early solar system. *Nature* 443:1013.

Chang, G., Roth, C. B., Reyes, C. L., Pornillos, O., Chen, Y.-J., and Chen, A. P. 2006. Retraction. *Science* 314:1875.

Davison, A. P., Brüderle, D., Eppler, J. M., Kremkow, J., Muller, E., Pecevski, D. A., Perrinet, L., and Yger, P. 2009. PyNN: a common interface for neuronal network simulators. *Front Neuroinform* 2:10.3389/neuro.11.011.2008.

De Schutter, E. and Bower, J. M. 1994a. An active membrane model of the cerebellar purk-

inje cell. i. simulation of current clamps in slice. *J Neurophysiol* 71:375–400.

De Schutter, E. and Bower, J. M. 1994b. An active membrane model of the cerebellar purkinje cell. ii. simulation of synaptic responses. *J Neurophysiol* 71:401–419.

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

Gleeson, P., Crook, S., Cannon, R. C., Hines, M. L., Billings, G. O., Farinella, M., Morse, T. M., Davison, A. P., Ray, S., Bhalla, U. S., Barnes, S. R., Dimitrova, Y. D., and Silver, R. A. 2010. NeuroML: A language for describing data driven models of neurons and networks with a high degree of biological detail. *PLoS Comput Biol* 6.

Hines, M. L., Morse, T., Migliore, M., Carnevale, N. T., and Shepherd, G. M. 2004. ModelDB: A Database to Support Computational Neuroscience. *J Comput Neurosci* 17:7–11.

Hodgkin, A. L. and Huxley, A. F. 1952. A quantitative description of membrane current and its application to conduction and excitation in nerve. *J Physiol* 117:500–544.

Izhikevich, E. M. and Edelman, G. M. 2008. Large-scale model of mammalian thalamocortical systems. *Proc Natl Acad Sci USA* 105:3593–3598.

Kannon, T., Inagaki, K., Kamiji, N. L., Makimura, K., , and Usui, S. 2011. PLATO: Data-oriented approach to collaborative large-scale brain system modeling. *Neural Networks* 24:918–26.

Markram, H. 2006. The Blue Brain Project. *Nat Rev Neurosci* 7:153–160.

Neylon, C. and Wu, S. 2009. Article-level metrics and the evolution of scientific impact. *PLoS Biol* 7:e1000242.

Raikov, I. and the INCF Multiscale Modeling Taskforce 2010. NineML—a description language for spiking neuron network modeling: the abstraction layer. *BMC Neuroscience* 11 (Suppl 1):P66.

Raymond, E. S. 1999. The Cathedral and the Bazaar. O'Reilly Media.

Stodden, V. 2009. The legal framework for reproducible research in the sciences: Licensing and copyright. *IEEE Computing in Science and Engineering* 11:35–40.

R. Walker (ed.) 2012. The Human Brain Project: A report to the European Commission [Internet]. The HBP-PS Consortium, Lausanne, Switzerland. Available from: `http://www.humanbrainproject.eu/files/HBP_flagship.pdf`.