Topics in Cognitive Science (2012) 1–10
Copyright © 2012 Cognitive Science Society, Inc. All rights reserved.
ISSN: 1756-8757 print / 1756-8765 online
DOI: 10.1111/j.1756-8765.2012.01206.x

Computational Modeling in Cognitive Science: A Manifesto for Change

Caspar Addyman,[a] Robert M. French[b]

[a]*Centre for Brain and Cognitive Development, Birkbeck, University of London*
[b]*LEAD-CNRS, University of Burgundy*

Received 20 January 2012; received in revised form 28 March 2012; accepted 9 April 2012

Abstract

Computational modeling has long been one of the traditional pillars of cognitive science. Unfortunately, the computer models of cognition being developed today have not kept up with the enormous changes that have taken place in computer technology and, especially, in human-computer interfaces. For all intents and purposes, modeling is still done today as it was 25, or even 35, years ago. Everyone still programs in his or her own favorite programming language, source code is rarely made available, accessibility of models to non-programming researchers is essentially non-existent, and even for other modelers, the profusion of source code in a multitude of programming languages, written without programming guidelines, makes it almost impossible to access, check, explore, re-use, or continue to develop. It is high time to change this situation, especially since the tools are now readily available to do so. We propose that the modeling community adopt three simple guidelines that would ensure that computational models would be accessible to the broad range of researchers in cognitive science. We further emphasize the pivotal role that journal editors must play in making computational models accessible to readers of their journals.

Keywords: Computational modeling; Cognitive modeling

1. Introduction

Computational modeling is one of the traditional pillars of cognitive science. Murphy (2011) has succinctly characterized one of the primary reasons for doing computational modeling, compared to verbal theorizing, as follows: ‘‘[It] requires the researcher to be explicit about a theory in a way that a verbal theory does not.’’ In other words, computational models

Correspondence should be sent to Caspar Addyman, CBCD, Birkbeck, University of London, Malet Street, London, WC1E 7HX UK. E-mail: c.addyman@bbk.ac.uk

remove what Murphy calls the ‘‘free wiggle room’’ that comes with all verbal theories. To use his example, a verbal theory that involves the notion of *similarity* between two items is inherently vague because the notion of similarity ‘‘has a number of different senses and ... shifts its meaning somewhat depending on what it is applied to’’ (Murphy, 2011). On the other hand, a computer model must be told, explicitly and precisely, how to compute a similarity measure and, because of this, all wiggle room vanishes.

Computational modeling of cognition has a long and fascinating history (cf. McClelland, 2009). Very soon after the first electronic computing machines were built, computational models of cognition began to appear. These early computational models were implementations of existing mathematical models of neural networks (Farley & Clark, 1954; Rochester, Holland, Haibt, & Duda, 1956). With the development of high-level programming languages and the realization that symbols that ostensibly referred to entities in the world could be input to computer programs, a host of ‘‘cognitive’’ computer models began to appear, beginning with the Logic Theorist (Newell & Simon, 1956), the General Problem Solver (Newell, Shaw, & Simon, 1959), and the perceptron (Rosenblatt, 1958). The computational exploration of cognition was off and running. Myriad models of artificial vision, problem solving, speech recognition, analogy making, action planning, story understanding, etc. were subsequently developed. However, the closest most researchers ever got to any of these early programs was the pages of the research journal in which they were described. This was largely for obvious reasons—namely, the mainframe computers of the day were huge and expensive, the expertise required to program them was formidable, and opportunities to do so were rare. Computational modeling was accessible to only a tiny fraction of the researchers interested in cognition.

Fortunately, times have changed. Unfortunately, computational modeling has not.

Today’s computers bear no more than a passing, theoretical resemblance to their ancestors of 50 years ago. Computing hardware, software, and human-computer interfaces, with the exception of the keyboard, bear essentially no resemblance to interfacing devices of half a century ago. Most important, all cognitive scientists now possess powerful personal computers and have unlimited access to the Web. And yet, in spite of these almost unimaginable changes, for all intents and purposes, modeling is still done as it was in 1985, or even 1975. Everyone still programs in his or her own favorite programming language. Some still use venerable classics like FORTRAN, LISP, or C; others use their modern descendants like SCHEME, or JAVA or C++, C#, and OBJECTIVE C; others still make use of dynamic scripting languages like PYTHON, PERL, or JAVASCRIPT; still more leverage specialist mathematical and statistical environments like MATLAB, MATHEMATICA, or R. The list is long and continually growing.

The problem is that, even if modelers are willing to make their code publicly available, one has to be *very* familiar with the programming language in which it was written to make heads or tails of it. Even then, a major struggle to understand the code is virtually guaranteed, since good-coding practices, including detailed in-line comments and documentation, are rarely, if ever, followed by modelers. A few diehards will attempt to program the model themselves, based on the verbal description in the article in which it appeared, but they will invariably be stymied by details that the authors left out. The end result is always the same: People read the description of the model but never really run it because, for all intents and purposes, it is not accessible. This problem is one faced by connectionists, rational and Bayesian modelers, symbolic and logical modelers, and dynamical systems theorists alike.

Code repositories like SourceForge and Github make it very easy to keep a working copy of the model code available online. Technically, this means that the model can be accessed and run by anyone who cares to do so. But, in reality, code availability does not generally translate into the ability of a non-programming cognitive scientist to test and probe the model. The increasing availability of simulation environments, such as ACT-R (Anderson & Lebiere, 1998; Anderson et al., 2004), Brian (Goodman & Brette, 2008), EMERGENT (Aisa, Mingus, & O'Reilly, 2008), NENGO (Stewart, Tripp, & Eliasmith, 2009), NEURON (Carnevale & Hines, 2006), and SANLAB (Patton & Gray, 2010), should make lives easier for modelers and non-modelers alike, but in many cases, it does not. They are, ultimately, of little use to non-programming cognitive scientists because mastering them, along with their idiosyncratic jargon and conventions, requires a steep learning curve. Even installing and configuring these tools is well beyond the skill level of non-programming users.

In short, even though these general-purpose modeling environments could potentially save other modelers from reinventing the wheel, they tend to be far too complex for researchers who lack a background in programming. Further, no single platform can anticipate all modeling situations and hence, we suggest, as a first step, that modelers make *their own models* easily accessible to the non-modeling community of researchers.

In his article, in the first issue of *topiCS*, McClelland (2009) looked at ''the Place of Modeling in Cognitive Science'' and called for greater transparency *within* modeling. This is fine, of course, but it overlooks the equally important issue of transparency *outside* the core modeling community. Only a two-pronged effort will actually lead to the goal of real accessibility to computational models. First, modelers, henceforth, need to make a real effort to provide user-friendly, working versions of their models to colleagues and students who are not primarily modelers themselves. And second, journal editors, henceforth, need to require user-friendly, working versions of the models described in their journals.

2. Computational modeling: A manifesto for modelers

2.1. Most other researchers interested in your topic will not be modelers. Act accordingly

This means that, at a minimum, all modeling papers should provide readers with a working version of the model and the data used in simulations reported in the paper. Good modeling practice should not only allow access to the source code, but, more important, should also provide a simple, graphic user interface or GUI (preferably, but not necessarily, Web-based) that would allow interested non-programming users to test and probe a particular model without actually having to build it from scratch or having to learn to program. The presentation of any model should have three categories of users in mind—namely:

(1) *Casual users*: people who want to observe the model running essentially as a demo using the built-in, default data supplied by the programmer and reported in the paper.

(2) *Motivated users*: people who want to run their own data on the model and/or explore the parameter space of the model.
(3) *Modelers*: sophisticated programmers who want access to the code, in order to modify not only the parameters of the model but the structure of the model itself.

A well-presented model should be able to simultaneously accommodate all three levels of users. Furthermore, a consideration of these classes of user leads to three core principles that we believe modelers should follow:

1. Let casual users run your simulations—easily.

   Anyone should be able to see your simulations running, regardless of their level of technical competence. A new user should be able to get your simulations running without assistance. There should be a clear user interface with simple, meaningful graphical feedback. Simplicity is important. Beginners should not be overwhelmed by a full bells-and-whistles design.

   Likewise, a user should not be expected to wrestle with complex installation and configuration options to get started. A Web-based version provides the lowest barrier to entry, available to anyone with a browser. If a model is designed as stand-alone software, provide a single package that includes all necessary libraries and data files. If your model lives in a modeling environment, such as Matlab, provide all data and scripts in a single folder with an obvious launch file. If running the model is computationally demanding, consider as an alternative videos of it in operation (see, for example, http://nengo.ca/build-a-brain/spaunvideos).

   Provide good documentation, but most of all, make the user interface self-explanatory.

2. Let motivated users run their own simulations.

   For your model to be of use and relevance to other researchers, they have to be able to probe it by exploring various parameters and must be able to be run it on problems they are interested in.

   Motivated readers of your modeling paper will want to understand the mechanisms that drive it. They may want to see the inner workings of the model and explore its parameter space. In other words, enough of the internal structure of the model should be accessible to users to allow them to be able to understand *why* the model produced the output it did (e.g., hidden unit representations for connectionist models, non-default parameter choices in ACT-R). You should avoid hard-coding ''magic numbers'' and should allow users to adjust model parameters. For example, in order to understand how a neural network model operates, users need to be able to modify learning rates and convergence criteria, as well as having access to network weights and activations and being able to track these over learning. And they must be able to test a trained network on a range of test inputs.

   Researchers may want to run your model on their own data. This should be able to be done in a relatively straightforward manner. The inputs and outputs should be easy to compare to human results on the same experiment and it ought to be easy to get that data in and out of the model. Keep data formats as simple as possible. Complex data structures and functions, however elegant and efficient, are to be avoided for the sake of comprehensibility. A random seed function needs to be included, so that the results from a run of stochastic code can be reproduced exactly. This is also a very useful tool for understanding a program's behavior. Finally, users must be able to export their results for use in their own analyses and publications.

3. Let other modelers use your code.

    Your code is your model. If you are serious about disseminating your work, then sharing your code is the best way to do it. Moreover, source code is the ultimate form of documentation. We believe modelers should always make their original code available.

    Modelers may be reluctant or embarrassed to share their code with others, thereby putting on display their bad coding habits. However, the functionality of your model matters more than the elegance of your code. Having an audience improves your own awareness of what you are doing. Commenting your code and always using meaningful variable names are very good habits to get into, ones that are required if other people are going to be perusing your code. Code that is readable by others is actually simpler for you to maintain, too.

    As the success of the open source movement shows, there can be tremendous benefits to sharing code. For example, our Web application (described below) makes use of many open source libraries to display data and graphs, manipulate arrays and matrices, etc. Likewise sharing your code can save time for others and makes it more likely that they will use your model.

We believe it will also foster new collaborations and make it easier to interest newcomers to cognitive modeling.

3. Making it happen

These ideas seem simple, even obvious. But there are many roadblocks standing in the way of their being put into practice. There have, in fact, been a number of attempts to make modeling more accessible. But these often have had the greater ambition of providing a full modeling environment. In addition, for individual modelers, providing a packaged version of a model is an extra burden above and beyond that of original research and publication. A recently founded cognitive modeling repository (Myung & Pitt, 2010) is still remarkably bare, suggesting that modelers currently see little or no advantage in sharing their models. We believe this is short-sighted. Just as open-access publication leads to higher citation counts (Eysenbach, 2006; Swan, 2010), sharing models will lead to greater exposure.

Finally, it would be hard to overestimate the role of journal editors in ensuring that the changes advocated in this article actually come to pass. Journal editors must also play their part in connecting modelers to the wider audience of cognitive scientists, in particular, non-programming cognitive scientists. *Psychological Review*, for example, is to be applauded for requiring models to be submitted with modeling papers. But for the moment this policy remains an exception. A brief survey of cognitive journals publishing modeling papers found that very few had any explicit policy on the inclusion of source code or models and some even discouraged the inclusion of any supplementary materials. The ready availability and usability of models ensures, at the very least, the reproducibility by third parties of published results, a cornerstone of all scientific investigation.

4. A concrete example: TRACX

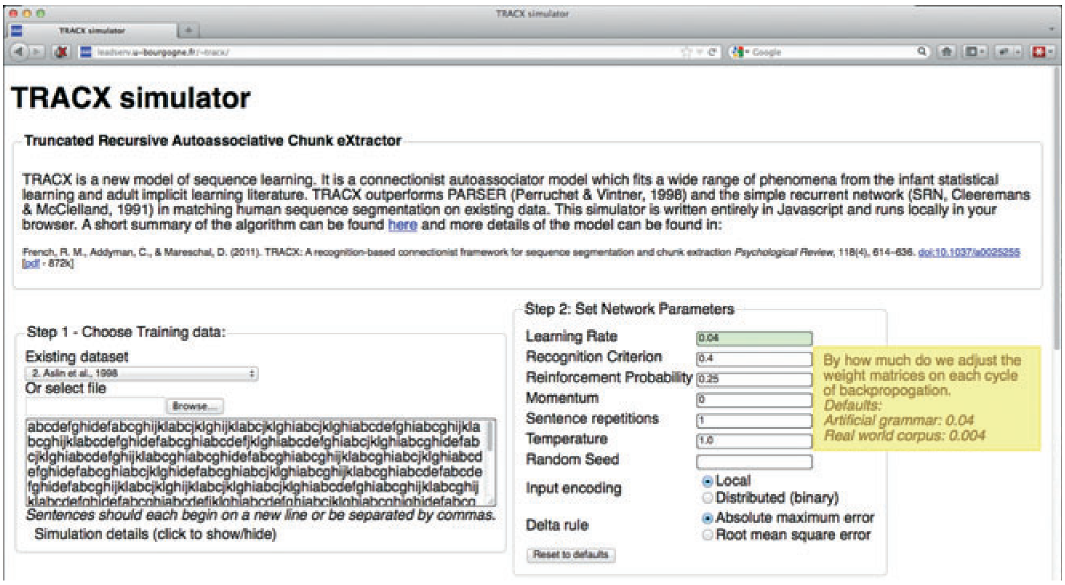
We recently developed TRACX, a connectionist (autoassociator) model of sequence segmentation and chunk extraction (French, Addyman, & Mareschal, 2011), and published an article on it in *Psychological Review.* All of the Matlab code for the simulations described in the article and the results produced are archived on the *Psychological Review* website. For the reasons given above, we also developed a simple Web-based JavaScript version of TRACX that allows users to run it, explore its parameter space, and test it on the original data, as well as on their own data. The Web-based version of the model can be found (and run) here: http://leadserv.u-bourgogne.fr/~tracx/

This Web-based simulator is only meant to illustrate the points above. Basically, the network receives as input a continuous stream of syllables or phonemes and learns to distinguish words from partwords and nonwords. (It is not important for the purposes of the present article to know what these terms mean.) Our Web-based simulator graphically shows how the network learns to distinguish these items. There are certainly many ways in which it could be improved, but it provides an example of one way in which the above modeling principles can be put into application.

The top of the TRACX simulator Web page specifies how to set the model up to be run (Fig. 1). Step 1 specifies the data to be run. The user can either run the data sets that were in the original paper or can enter his or her own data from an Excel file. Step 2 allows the user to set the network parameters. All of the parameters are explained in the paper in which the model was presented. Moreover, tooltips describing each parameter pop up as the user hovers his or her mouse over the relevant field.

After that (Step 3), the program runs (Fig. 2). It can be run either by stepping through the a single training instance with a single network, allowing the user to watch the input, hidden, and output representations evolve, to observe learning as it occurs, etc. Or the program can be run in ''batch mode,'' training a set of networks and outputting average results.

Finally, in the lower part of the page, the results of a run of the program are displayed graphically (Fig. 3). On the left one can follow how the input, hidden, and output representations of the network evolve over time. On the right, one can observe the learning of individual chunks of the input sequence over time. And the Network Error graph shows how the error for words and nonwords is evolving with learning. When the run is completed, there is a record, not only of the final network error for both Words and partwords but also a graph showing how the errors of individual words and partwords evolved over time. Raw numerical data are available too, but this is initially hidden from the user to keep the interface simple. By clicking on the ''Show/Hide data'' below the appropriate section, users can access the initial network weights parameters and random seed, and they can see the raw network activations and the exact values used to plot the mean error scores and errors over time. This allows a user to easily copy and paste their results into Excel to perform his or her own analyses.



TRACX simulator

Truncated Recursive Autoassociative Chunk eXtractor

TRACX is a new model of sequence learning. It is a connectionist autoassociator model which fits a wide range of phenomena from the infant statistical learning and adult implicit learning literature. TRACX outperforms PARSER (Perruchet & Vintner, 1998) and the simple recurrent network (SRN, Cleeremans & McClelland, 1991) in matching human sequence segmentation on existing data. This simulator is written entirely in Javascript and runs locally in your browser. A short summary of the algorithm can be found here and more details of the model can be found in:

French, R. M., Addyman, C., & Mareschal, D. (2011). TRACX: A recognition-based connectionist framework for sequence segmentation and chunk extraction *Psychological Review*, 118(4), 614–636. doi:10.1037/a0025255 [pdf - 872k]

Step 1 - Choose Training data:

Existing dataset: 2. Aslin et al., 1998

Or select file

```
abcdefghidefabcghijklabcjklghijklabcjklghiabcjklghiabcdefghiabcghijkla
bcghijklabcdefghidefabcghiabcdefjklghiabcdefghiabcjklghiabcghidefab
cjklghiabcdefghijklabcghiabcghidefabcghiabcghijklabcghiabcjklghiabcd
efghidefabcghiabcjklghidefabcghiabcjklghiabcghijklabcghiabcdefabcde
fghidefabcghijklabcjklghijklabcjklghiabcjklghiabcdefghiabcghijklabcghij
```

Sentences should each begin on a new line or be separated by commas.

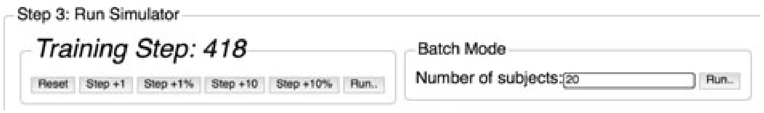
Simulation details (click to show/hide)

Step 2: Set Network Parameters

Learning Rate	0.04
Recognition Criterion	0.4
Reinforcement Probability	0.25
Momentum	0
Sentence repetitions	1
Temperature	1.0
Random Seed	
Input encoding	[✓ Local] Distributed (binary)
Delta rule	[✓ Absolute maximum error] Root mean square error

By how much do we adjust the weight matrices on each cycle of backpropogation.
Defaults:
Artificial grammar: 0.04
Real world corpus: 0.004

Fig. 1. The initial part of the simulator shows how to load data into the model and how to modify the model's parameters. Help is provided in the form of tooltips (yellow box on right) that appear as the user clicks on each field.



Step 3: Run Simulator

Training Step: 418

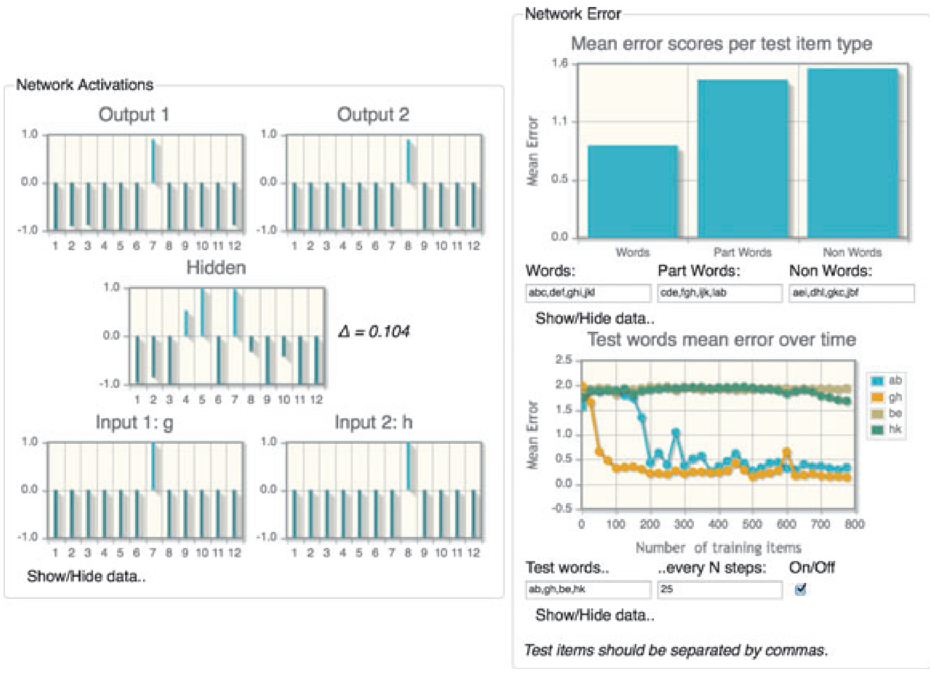
Batch Mode

Number of subjects: 20

Fig. 2. Running the simulator. You can either step through a single run (left) or train a batch of individual simulators and show average outputs (right).

For modelers, the lower left-hand corner of the page (not shown in Fig. 3) also has links to a guide on how to step through the TRACX simulator code, has a bug-report link, and shows where the source code for the page can be downloaded.

We are not suggesting that all GUI versions of computational models follow this Web-based format, but we think that the main features of the present illustration should be included. Whatever language your model is written in, the chances are that a GUI-builder exists to help you add a GUI front-end to the model. For example, Matlab comes fully equipped to develop a graphic user interface to be used with programs and even allows these to be compiled into stand-alone executables. The graphical user interface does not have to be as complex as the Web-based one described above, which was written in JavaScript. But it does need to provide, at a minimum, the possibility of changing model parameters, importing and exporting data files, and displaying results.



Words: abc,def,ghi,jkl
Part Words: cde,fgh,ijk,lab
Non Words: aei,dhl,gkc,jbf

Test words.. ab,gh,be,hk
..every N steps: 25
On/Off ✓

Test items should be separated by commas.

Fig. 3. Displaying the results. Network activations are show on the left and network performance on the right.

5. Conclusion

Computational modeling in cognitive science is badly in need of a makeover to bring it up to date with respect to the tremendous changes in computing techniques and, especially, human-computer interfaces that have taken place in the last quarter of a century. We feel that this would go a long way to bringing back some of the erstwhile enthusiasm for modeling. When was the last time anyone really got excited about a new model, comparable, say, to the excitement generated in 1986 when the PDP books (McClelland & Rumelhart, 1986; Rumelhart & McClelland, 1986) came out? We are convinced that one of the primary reasons for the excitement generated by the PDP models was (and continues to be) due to the simulators available for it, like PLANET (Miyata, 1991), PYGMALION (Azema-Barac et al., 1990), the Rochester Simulator (Goddard, Lynne, Mintz, & Bukys, 1989), and more recently, Oxlearn (Ruh & Westermann, 2009), etc. These simulators, and a host of others, have allowed non-programmers to dip into the world of connectionism and come away impressed. We need to revive the spirit of these early simulators that gave PDP models such a boost. More than ever before, the tools are now available to make computational models truly accessible to non-programming cognitive scientists.

Finally, Gray (2010) points out that the problems related to the inaccessibly of existing computational models are equally damaging to modelers, as well as non-modelers. This inaccessibility reinforces an inflexible approach to modeling, with each modeler tending to stick to his or her own hard-learned and hand-coded approach to modeling, which means that, by and large, modelers work in isolation. While they may be aware of modeling work in other cognitive domains, or on higher and lower cognitive systems, they lack the ability to combine other models with their own. Evolution produced a brain that is not a single one-size-fits-all system, but rather many different systems working in concert. Ultimately, computational modeling of cognition will follow this path, too, which will mean developing reusable computational models that are capable of real integration. The first step in laying the groundwork for the next generation of reusable computational models is to begin by making the current generation's models truly accessible to users of all stripes.

The present manifesto calls on computational modelers to make better use of current computer technology to allow members of the cognitive science community to have real access to their models. We also call on journal editors to require real accessibility to the models that appear in the pages of the journals they oversee. Computational models of cognition should be able to be probed, tested, and applied to new data by cognitive scientists who do not have the programming skills, the time, or the desire to program the models themselves. We believe that the simple suggestions of this short manifesto will, ultimately, not only lead to better models of cognition but will also go a long way to improving the visibility of computational models within cognitive science.

Acknowledgments

This work was funded in part by the French Agence Nationale de la Recherche Grant ANR-10-065-GETPIMA and United Kingdom Economic and Social Research Council Grant RES-062-23-0819 under the auspices of the Open Research Area France–United Kingdom funding initiative. Special thanks to three reviewers for their helpful comments on this article.

References

Aisa, B., Mingus, B., & O'Reilly, R. (2008) The emergent neural modeling system. *Neural Networks*, *21*(8), 1146–1152. doi: 10.1016/j.neunet.2008.06.016

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*(4), 1036–1060.

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, NJ: Erlbaum.

Azema-Barac, M., Hewetson, M., Recce, M., Taylor, J., Treleaven, P., & Vellasco, M. (1990) Pygmalion: Neural network programming environment (1990). In B. Angeniol & B. Widrow (Eds.) *Proceedings of the International Neural Network Conference INNC-90, vol. 2* (pp. 1237–1244). Dordrecht, The Netherlands: Kluwer Academic Publishers.

Carnevale, N. T., & Hines, M. L. (2006) *The NEURON book*. Cambridge, UK: Cambridge University Press.

Eysenbach, G. (2006). Citation advantage of open access articles. *Public Library of Science Biology*, *4*(5), e157. doi:10.1371/journal.pbio.0040157

Farley, B., & Clark, W. A. (1954). Simulation of self-organizing systems by digital computer. *IRE Transactions on Information Theory*, *4*, 76–84.

French, R. M., Addyman, C., & Mareschal, D. (2011). TRACX: A recognition-based connectionist framework for sequence segmentation and chunk extraction. *Psychological Review*, *118*(4), 614–636. doi:10.1037/a0025255

Goddard, N. H., Lynne, K., Mintz, T., & Bukys, L. (1989). The Rochester Connectionist Simulator, Technical Report 233 (revised), Computer Science Department, University of Rochester, New York.

Goodman, D. F., & Brette, R. (2008) Brian: A simulator for spiking neural networks in Python. *Frontiers in Neuroinformatics*, *8*(5). doi:10.3389/neuro.11.005.2008

Gray, W. D. (2010). The shape of things to come: An emerging constellation of interconnected tools for developing the right cognitive model at the right scale. Keynote address delivered to the Behavior Representation in Modeling and Simulation Conference (BRiMS). Charleston, SC.

McClelland, J. L. (2009). The place of modeling in cognitive science. *Topics in Cognitive Science*, *1*, 11–38.

McClelland, J. L., & Rumelhart, D., & the PDP research group. (1986). *Parallel distributed processing, Volume II*. Cambridge, MA: MIT Press.

Miyata, Y. (1991). A user's guide to PlaNet version 5.6, *The Reference Manual for PlaNet Version 5.6*, University of Colorado, Boulder, Computer Science Department.

Murphy, G. L. (2011). Models and concepts. In E. M. Pothos & A. J. Wills (Eds.), *Formal approaches in categorization* (pp. 299–312). Cambridge, UK: Cambridge University Press.

Myung, J. I., & Pitt, M. A. (2010). Cognitive modeling repository. Columbus, OH: Department of Psychology, Ohio State University (Distributor). Retrieved from http://cmr.osu.edu/

Newell, A., Shaw, J. C., & Simon, H. A. (1959). Report on a general problem-solving program. *Proceedings of the International Conference on Information Processing*, pp. 256–264.

Newell, A., & Simon, H. A. (1956). The logic theory machine: A complex information processing system. *P-868. The Rand Corporation*.

Patton, E. W., & Gray, W. D. (2010). SANLab-CM: A tool for incorporating stochastic operations into activity network modeling. *Behavior Research Methods*, *42*(3), 877–883.

Rochester, N., Holland, J. H., Haibt, L. H., & Duda, W. L. (1956). Test on a cell assembly theory of the action of the brain using a large digital computer. *IRE Transactions on Information Theory*, IT-2, 80-93.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and optimization in the brain. *Psychological Review*, *65*(6), 386–408.

Ruh, N., & Westermann, G. (2009). OXlearn: A new MATLAB based simulation tool for connectionist models. *Behavior Research Methods*, *41*(4), 1138–1143.

Rumelhart, D., & McClelland, J., & the PDP research group. (1986). *Parallel distributed processing, vol. I*. Cambridge, MA: MIT Press.

Stewart, T. C., Tripp, B., & Eliasmith, C. (2009). Python scripting in the Nengo simulator. *Frontiers in Neuroinformatics*, *3*(7). doi:10.3389/neuro.11.007.2009.

Swan, A. (2010) The Open Access citation advantage: Studies and results to date. Technical Report, School of Electronics & Computer Science, University of Southampton.