Behavior Research Methods
2009, 41 (4), 1138-1143
doi:10.3758/BRM.41.4.1138

OXlearn: A new MATLAB-based simulation tool for connectionist models

Nicolas Ruh and Gert Westermann
Oxford Brookes University, Oxford, England

OXlearn is a free, platform-independent MATLAB toolbox in which standard connectionist neural network models can be set up, run, and analyzed by means of a user-friendly graphical interface. Due to its seamless integration with the MATLAB programming environment, the inner workings of the simulation tool can be easily inspected and/or extended using native MATLAB commands or components. This combination of usability, transparency, and extendability makes OXlearn an efficient tool for the implementation of basic research projects or the prototyping of more complex research endeavors, as well as for teaching. Both the MATLAB toolbox and a compiled version that does not require access to MATLAB can be downloaded from http://psych.brookes.ac.uk/oxlearn/.

Over 20 years after the publication of seminal work by David E. Rumelhart and his colleagues from the PDP Research Group (McClelland, Rumelhart, & the PDP Research Group, 1986; Rumelhart, Hinton, & Williams, 1986; Rumelhart & McClelland, 1986a, 1986b), the connectionist approach to modeling cognitive processes has diversified into a bewildering number of different network architectures, learning algorithms, and training paradigms, but also various simulation environments in which the actual models can be implemented. Naturally, most of these simulation tools have become more complex over time as they have integrated an ever-increasing amount of functionality (see, e.g., Emergent [Aisa, Mingus, & O'Reilly, 2008] or SNNS [Stuttgart Neural Network Simulator; Zell, Mache, Sommer, & Korb, 1991]), but by the same token, they have become increasingly difficult to access for the uninitiated researcher or student. Although such powerful simulation environments are invaluable for complex research projects conducted by experts in the field, they are not necessarily expedient for an introduction to neural network modeling, or even for the implementation of basic projects that do not make use of all the extended features.

Simpler simulation tools, conversely, provide only limited functionality, which may or may not suit the needs of a specific research project but can be the more suitable choice from an educational point of view. The *t-learn* simulator that accompanied the book on "rethinking innateness" (Elman et al., 1996) is the best known of these basic tools. With its combination of graphical interface and text-based controls, however, t-learn has never been the most user-friendly of programs. What is more, the t-learn software has long since stopped being actively developed and is not compatible with modern operating systems (e.g., Windows XP), thus leaving a gap not only in terms of an accessible educational tool, but also as a basic simulation software in which student projects can be run, new ideas can be quickly tested, or existing simulations can be reimplemented and scrutinized.

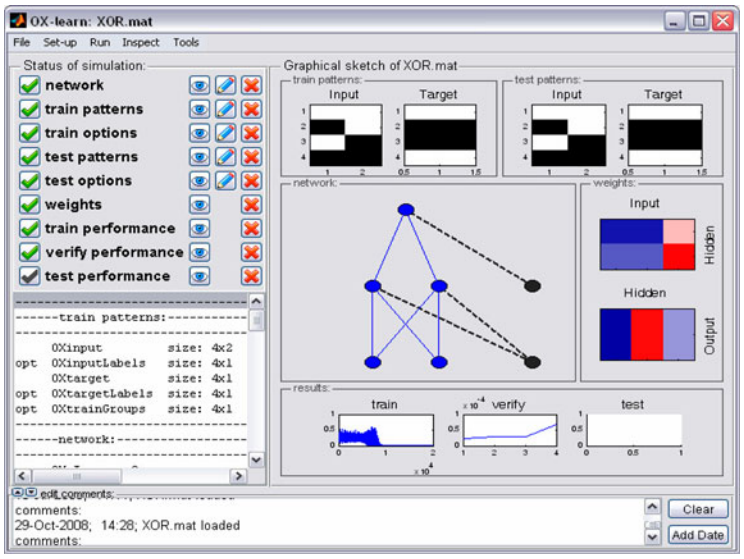
The OXlearn simulation software is designed to fill this gap. OXlearn supports the full functionality of the now outdated t-learn simulator, along with additional tools for network creation, data manipulation, visualization, and analysis. Although OXlearn's inbuilt capabilities are limited to standard architectures and algorithms (i.e., single- and multilayer feedforward networks and SRNs [simple recurrent networks] to be trained with the delta-rule/backpropagation algorithm), these basic setups continue to be used in a wide array of current neural-network-related research—for example, for modeling categorization in infancy (French, Mareschal, Mermillod, & Quinn, 2004; Mareschal, Quinn, & French, 2002), lexical segmentation (Davis, 2003), morphological inflection (Nakisa, Plunkett, & Hahn, 2000; Plunkett & Juola, 1999), abnormal language development (Thomas & Karmiloff-Smith, 2003), the mental lexicon (Elman, 2004), visual deficits in children with Williams syndrome (Abreu, French, Annaz, Thomas, & de Schonen, 2005), and so forth.

All inbuilt functionality of OXlearn is accessible through the extended, user-friendly graphical user interface (GUI). Being implemented as a set of MATLAB functions, however, OXlearn has the additional advantages (1) of being inspectable and customizable at every level within the MATLAB programming environment and (2) of running on every MATLAB-enabled computer, independently of the underlying operating system.

OXlearn: The Graphical User Interface (GUI)

Figure 1 depicts the simulation overview display of the OXlearn GUI with a classical XOR simulation already set

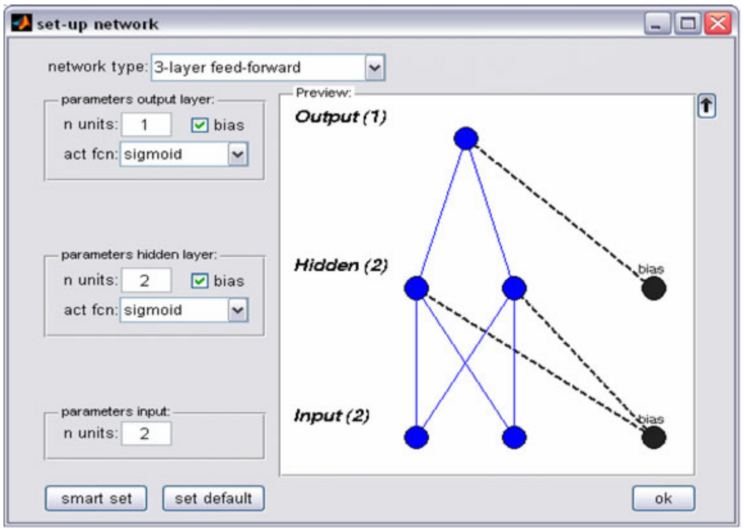
N. Ruh, nruh@brookes.ac.uk



```
------------------------------------
------train patterns:---------------
------------------------------------
      OXinput          size:  4x2
opt   OXinputLabels    size:  4x1
      OXtarget         size:  4x1
opt   OXtargetLabels   size:  4x1
opt   OXtrainGroups    size:  4x1
------------------------------------
------network:----------------------
------------------------------------
```

comments:
29-Oct-2008; 14:28; XOR.mat loaded
comments:

Figure 1. The simulation overview display of the OXlearn GUI.



network type: 3-layer feed-forward

parameters output layer:
n units: 1 [✓ bias]
act fcn: sigmoid

parameters hidden layer:
n units: 2 [✓ bias]
act fcn: sigmoid

parameters input:
n units: 2

Figure 2. The interface for defining a network's architecture.

up. OXlearn simulations can be controlled directly from this display or through the menu bar at the top of the window. The menu bar reflects the general structure of a typical workflow: The leftmost item, "File," is concerned with general data management—that is, importing simulation data in various formats (e.g., .txt, .rtf, .csv, .xls, .mat, etc.), opening existing simulations, and saving/exporting simulation data. Note that a set of example simulations (including the XOR network and other examples adapted from the literature on t-learn) are bundled into the OXlearn simulation tool. The "Set-up" menu holds tools to define a model (see Figure 2) and its training paradigm and to create/manipulate the patterns the model is to process.

The "Run" menu holds submenus to train or test one or several models, and the "Inspect" menu gives access to various ways in which a simulation project and its outcomes may be visualized (see, e.g., Figure 3). Note that all graphics produced by OXlearn are fully customizable (permitting changes in colors, line styles, labels, etc.) and can be exported to all graphical formats supported by MATLAB (e.g., .jpg, .tiff, .gif, .eps, .bmp, .fig, etc.).

"Tools," finally, comprises a collection of advanced analysis and pre-/postprocessing instruments, such as principal component analyses (Figure 4), cluster plots,[1] translation routines, and a tool for comparing performance over several runs, networks, or snapshots (dumps) of one network's state during training.

OXlearn: Why MATLAB?

Most existing simulation software comes in the form of precompiled programs. As a result, it is impossible to take a peek under the hood—for example, to comprehend the underlying computations. The noncompiled MATLAB programming language, by contrast, allows exactly that.

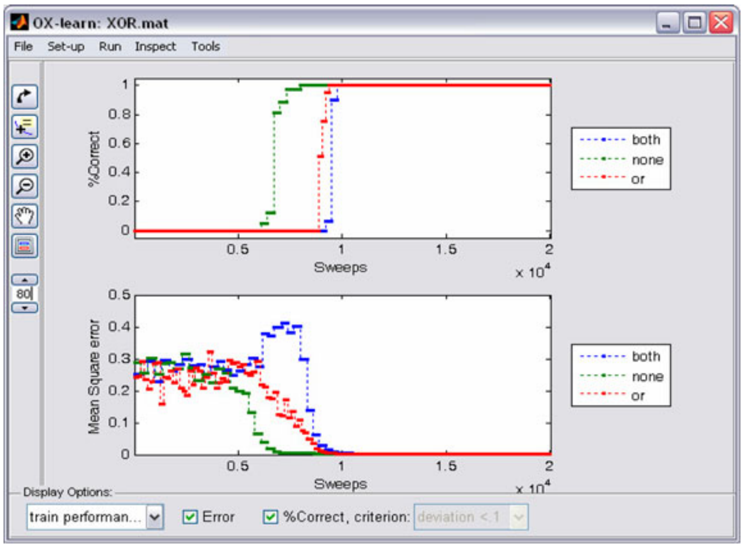


Figure 3. Performance display depicting the learning profiles of three groups of stimuli.

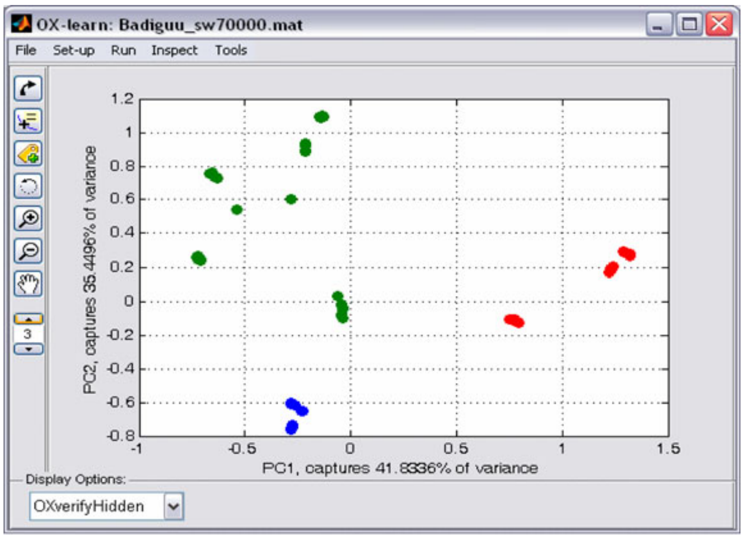

OX-learn: Badiguu_sw70000.mat
File Set-up Run Inspect Tools
PC2, captures 35.4496% of variance
PC1, captures 41.8336% of variance
3
Display Options:
OXverifyHidden

Figure 4. Scatterplot depicting the distribution of stimuli along the first two principal components.

All MATLAB functions can be inspected at run time, their execution can be paused at any point, and the values of all variables can be monitored or visualized. Together with native interactive tools, such as the workspace browser for the inspection of current variable values, the array editor for Excel-style manipulation of tabular data, and the various data visualization and plotting tools, MATLAB itself provides a programming environment that is extremely transparent and accessible even for the computationally less inclined. As a result, even for nonprogrammers, it is relatively simple to inspect, understand, and possibly adjust existing MATLAB code.

A further advantage of OXlearn's integration within the MATLAB programming environment can be seen in the fact that MATLAB commands or interactive tools can be used to perform necessary and often nontrivial pre- and postsimulation steps pertaining to a specific project, such as the creation or manipulation of input patterns, statistical analysis of simulation outcomes, or the preparation of final graphs for publication.

With respect to the inevitable trade-off between speed in compiled programs and flexibility in interpreted ones, the choice of MATLAB emphasizes the latter. As a result, OXlearn is at a disadvantage when compared with other (compiled) simulation tools in terms of pure computing speed. Due to the availability of powerful personal computers and technical advances in the implementation of math-centered interpreted languages such as MATLAB, however, these speed differences are of little practical consequence in moderately complex projects. As an example, training a

```
%testing network with sigmoid activation function

%-------------forward pass------------------------------------

OXtestHidden = 1./(1+exp(-(OXweightsInputToHidden * OXtestInput ...
                          + OXweightsToHiddenBias * OXbH)));

OXtestOutput = 1./(1+exp(-(OXweightsHiddenToOutput * OXtestHidden ...
                          + OXweightsToOutputBias * OXbO)));
```

$20 \times 30 \times 20$ multilayer backpropagation network (= 1,200 connections) for 100,000 sweeps[2] takes about 18 sec on a 2.40-GHz machine running 32-bit Windows XP. Note also that there are several simple ways of tweaking performance, such as disabling optional features in OXlearn (e.g., refrain from saving detailed log data or updating the display throughout training) or ensuring that MATLAB makes full use of its multithreading capabilities[3] when run on multicore computers. The choice of implementing OXlearn in MATLAB thus should rarely amount to a practical problem in terms of computing speed, while buying considerable advantages in terms of flexibility and transparency.[4]

OXlearn: Software or Toolbox?

The OXlearn software is written in the MATLAB programming language in order to make full use of the aforementioned advantages. Like any other MATLAB toolbox, OXlearn functions and parameters can be manipulated directly within the programming environment. In addition, however, some of the MATLAB functions bundled into the OXlearn software implement a comprehensive and intuitive GUI that grants access to all inbuilt functionality by means of standard graphical controls (buttons, check boxes, menus, etc.).

This general setup gives OXlearn a somewhat dual nature. On the one hand, it is possible to utilize OXlearn like any other predefined software by means of interacting with its GUI. As long as the intended use stays within the bounds of the inbuilt functionality, this is a comfortable and efficient way of building, training, and testing standard neural network models. When used in this way, even the MATLAB version of OXlearn does not require any interaction with the underlying MATLAB platform, apart from starting the program by typing "OXlearn" in the command window. For educational purposes, for example, it will rarely be necessary to go beyond this. On the other hand, however, more proficient or inquisitive users have the option of inspecting and/or manipulating OXlearn's functioning within the MATLAB environment—be it through MATLAB commands or through the use of various interactive tools provided by the MATLAB interface. This, of course, is most advantageous when nonstandard functionality is required—for example, to implement custom analyses or visualization techniques. The additional flexibility provided by the seamless integration into the MATLAB programming environment is what makes OXlearn a viable research tool.

To give an example, comparing breakdown behavior in lesioned networks with human breakdown patterns has been one major line of enquiry in neural network research. However, the various ways in which a network can be artificially lesioned (e.g., disabling of, or adding [different kinds of] noise to, all/selected/a proportion of patterns/layers/units/connections) are hard to formalize in any kind of user interface, and consequently, the OXlearn GUI does not include such a *lesioning* feature. This notwithstanding, any kind of lesioning can be implemented by adjusting the code of the function that performs the testing. For a three-layer feedforward network, for example, this function is called "`OX_testFF3.m`," and its content can be inspected by opening this file in the MATLAB editor (or any other text-editing program). The textbox above shows the implementation of the forward pass of the test patterns through the (trained) network.

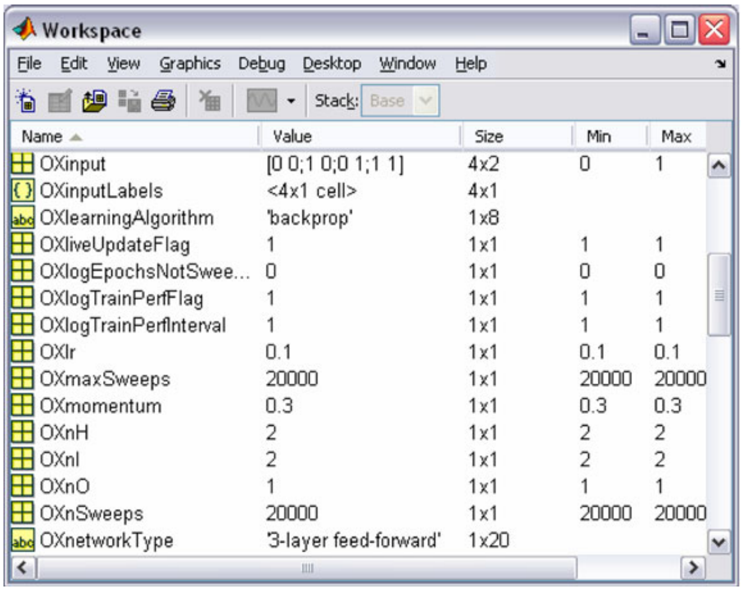
Because of MATLAB's specialization in matrix-based computation, the forward pass requires only two lines of code. In the first step, the entire test corpus (`OXtestInput`, a matrix of values where columns correspond to input units and rows correspond to individual patterns) is transformed into hidden layer activations. The second step performs the transformation of hidden layer activations into output layer activations and, thus, the response of the network to each of the test patterns. Most of the possible ways of lesioning the network can be implemented by adding one line to this code. For example, disrupting a random sample of 20% of the input-hidden connections could be achieved by including the following code before Step 1:

```
OXweightsInputToHidden =
  OXweightsInputToHidden .* ...
  (rand(size(OXweightsInputToHidden)) > 0.2);
```

Adding normally distributed noise with $M = 0$ and $SD = 0.5$ would be implemented by

```
OXweightsInputToHidden =
 OXweightsInputToHidden + ...
 (randn(size(OXweightsInputToHidden)) * 0.5);
```

Although the example of testing lesioned networks demonstrates the relative ease of source code adaptation, this should rarely be necessary. The fact that the underlying code is readily accessible, however, can also help in deepening the interested user's grasp of the inner workings of neural network models.



Name	Value	Size	Min	Max
OXinput	[0 0;1 0;0 1;1 1]	4x2	0	1
OXinputLabels	<4x1 cell>	4x1		
OXlearningAlgorithm	'backprop'	1x8		
OXliveUpdateFlag	1	1x1	1	1
OXlogEpochsNotSwee...	0	1x1	0	0
OXlogTrainPerfFlag	1	1x1	1	1
OXlogTrainPerfInterval	1	1x1	1	1
OXlr	0.1	1x1	0.1	0.1
OXmaxSweeps	20000	1x1	20000	20000
OXmomentum	0.3	1x1	0.3	0.3
OXnH	2	1x1	2	2
OXnI	2	1x1	2	2
OXnO	1	1x1	1	1
OXnSweeps	20000	1x1	20000	20000
OXnetworkType	'3-layer feed-forward'	1x20		

Figure 5. Content of MATLAB's workspace browser with the XOR simulation loaded into OXlearn.

OXlearn: Designed for Transparency

The idea of making the OXlearn toolbox open to inspection and customizations is supported by two specific design features. The first is the deliberate choice of a non-object-oriented approach, which means that the user who chooses to have a look at OXlearn's internal functioning will have to deal only with simple data formats such as (matrices of) numbers, Boolean values, or strings. The second feature is that all data used by the simulations are mediated by MATLAB's basic workspace. For example, when the command to train a model is executed (by calling the respective function from the command line or by pressing the "train network" button in the GUI), all relevant parameters will be collected from the basic workspace, the simulation will be run, and the results (i.e., new connection weights, output activations of the network, etc.) will be given back to the basic workspace. Similarly, subsequent analysis of the model's performance operates on the outcome data stored in the basic workspace. This ensures easy access to all relevant simulation data within the MATLAB environment—for example, by looking at the workspace browser (Figure 5). As an optional alternative to using the OXlearn GUI, all the intuitively labeled parameters and data containers can be inspected, saved, plotted, and changed from here. More proficient users can also use MATLAB's editor window to inspect the function that performs the training of the network (e.g., `OX_trainFF3.m`) and adapt it.

OXlearn: Two Versions

MATLAB is rapidly becoming something like the lingua franca of scientific computing. It is thus reasonable to assume that many practitioners will have access to a license and, thus, will be able to utilize OXlearn as a software or extendable MATLAB toolbox as suits their needs. Please note that OXlearn has been tested for MATLAB 7.3 (Release 2006b) and above; earlier versions are not supported. When the toolbox version is opted for, interaction with MATLAB is *required* only at two points. After the OXlearn folder has been downloaded and extracted on the hard drive, the first step is to let MATLAB know where to find OXlearn. This can be done by using the standard browsing interface at the top of MATLAB's main window to browse to the location where the OXlearn folder was saved. Alternatively, the OXlearn folder can be added to the MATLAB path (i.e., a list of locations known to MATLAB)—for example, through the interface under "File → Set Path." The latter solution has the advantage of adding the new location permanently, provided the user has administrator rights for MATLAB. The second step entails starting the program—for example, by typing "OXlearn" in the command window. As explained in the manual, it is also possible to create a shortcut button to replace this step.

For users without access to MATLAB, OXlearn is also available in a compiled version (platform dependent; currently, Windows XP only). This stand-alone version is fully functional in the software sense (similar to most existing simulation tools) but lacks the potential flexibility and extendability provided by the MATLAB environment. In order to use the stand-alone version, a free program called "MATLAB Runtime Component" (MRC) must be installed prior to opening the OXlearn executable. Both versions (and the MRC) can be downloaded free of charge from http://psych.brookes.ac.uk/oxlearn/. This Web page also provides an extensive user manual where OXlearn's usage and functionality are explained in more detail.

Conclusion

In this article, we have presented OXlearn, a new MATLAB-based simulation software for the implementation and analysis of neural network models. The main rationale in designing OXlearn was to combine ease of use with maximal transparency to facilitate extension of the existing functionality. Instead of trying to predict all the possible architectures, algorithms, and training paradigms that a potential user might want to implement and building them into the software, we have chosen to include only a core set of model types, thus keeping the complexity of the software within bounds. However, using a set of MATLAB functions to implement this core functionality and the accompanying GUI enables flexible use of the software by making all relevant data within the simulation tool easily accessible through the MATLAB environment. This inbuilt transparency not only allows the interested user to directly inspect the inner working of the simulation tool, thus facilitating insights into the computational underpinnings of neural network models, but also permits OXlearn to be used in conjunction with native MATLAB tools and commands—for example, to provide alternative ways of visualizing data, creating stimuli, or implementing custom analyses. Proficient MATLAB users can also adjust or extend OXlearn's core functionality by editing the existing routines or scripting new code. However, users who do not want/need to adapt OXlearn will find it an easy-to-use, intuitive tool for quickly setting up and testing standard connectionist models. When used in this way (or in the stand-alone version), OXlearn does not require any knowledge of MATLAB or programming.

Its dual nature as software and MATLAB toolbox makes OXlearn well suited for both educational and research purposes. Using the software side only, standard models can be swiftly (re-)implemented and tested, which makes OXlearn a convenient tool for researchers using standard connectionist algorithms and for hands-on exercises—for example, in an introductory class on connectionist modeling. At the same time, the transition into a more profound understanding of the computations involved in neural network modeling is facilitated by the fact that the inner workings of the software are designed for maximum transparency and can be directly inspected and manipulated. The insights gained can then be applied to customize parts of OXlearn's functionality so as to encompass the nonstandard aspects that a specific research project might require. Due to this toolbox side, OXlearn can also be used as a flexible tool in more specialized research.

AUTHOR NOTE

The development of the OXlearn software was supported by ESRC Grant Res-061-23-0129 to G.W. Correspondence concerning this article should be addressed to N. Ruh, Postdoctoral Researcher, Oxford Brookes University, Gipsy Lane, Buckley Building, Room B1.02, Oxford OX3 0BP, England (e-mail: nruh@brookes.ac.uk).

REFERENCES

Abreu, A. M., French, R. M., Annaz, D., Thomas, M., & de Schonen, S. (2005). A "visual conflict" hypothesis for global–local visual deficits in Williams syndrome: Simulations and data. In *Proceedings of the 27th Annual Meeting of the Cognitive Science Society* (pp. 45-50). Mahwah, NJ: Erlbaum.

Aisa, B., Mingus, B., & O'Reilly, R. (2008). The emergent neural modeling system. *Neural Networks*, **21**, 1146-1152.

Davis, M. H. (2003). Connectionist modelling of lexical segmentation and vocabulary acquisition. In P. T. Quinlan (Ed.), *Connectionist models of development* (pp. 125-159). Hove, U.K.: Psychology Press.

Elman, J. L. (2004). An alternative view of the mental lexicon. *Trends in Cognitive Sciences*, **8**, 301-306.

Elman, J. L., Bates, E. A., Johnson, M. H., Karmiloff-Smith, A., Parisi, D., & Plunkett, K. (1996). *Rethinking innateness: A connectionist perspective on development*. Cambridge, MA: MIT Press.

French, R. M., Mareschal, D., Mermillod, M., & Quinn, P. C. (2004). The role of bottom-up processing in perceptual categorization by 3- to 4-month-old infants: Simulations and data. *Journal of Experimental Psychology: General*, **133**, 382-397.

Mareschal, D., Quinn, P. C., & French, R. M. (2002). Asymmetric interference in 3- to 4-month-olds' sequential category learning. *Cognitive Science*, **26**, 377-389.

McClelland, J. L., Rumelhart, D. E., & the PDP Research Group (1986). *Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 2: Psychological and biological models*. Cambridge, MA: MIT Press.

Nakisa, R. C., Plunkett, K., & Hahn, U. (2000). Single- and dual-route models of inflectional morphology. In P. Broeder & J. Murre (Eds.), *Models of language acquisition: Inductive and deductive approaches* (pp. 201-224). Oxford: Oxford University Press.

Plunkett, K., & Juola, P. (1999). A connectionist model of English past tense and plural morphology. *Cognitive Science*, **23**, 463-490.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, & the PDP Research Group (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 1: Foundations* (pp. 318-362). Cambridge, MA: MIT Press.

Rumelhart, D. E., & McClelland, J. L. (1986a). On learning the past tense of English verbs: Implicit rules or parallel distributed processing? In J. L. McClelland, D. E. Rumelhart, & the PDP Research Group (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 2: Psychological and biological models* (pp. 216-271). Cambridge, MA: MIT Press.

Rumelhart, D. E., & McClelland, J. L. (1986b). PDP models and general issues in cognitive science. In D. E. Rumelhart, J. L. McClelland, & the PDP Research Group (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 1: Foundations* (pp. 110-146). Cambridge, MA: MIT Press.

Thomas, M. S. C., & Karmiloff-Smith, A. (2003). Modeling language acquisition in atypical phenotypes. *Psychological Review*, **110**, 647-682.

Zell, A., Mache, N., Sommer, T., & Korb, T. (1991). Recent developments of the SNNS neural network simulator. *Applications of Artificial Neural Networks II*, **1469**, 708-718.

NOTES

1. Note that some of these tools depend on MATLAB's Statistics Toolbox.

2. One sweep consists of a forward pass of a single pattern through the network.

3. This is influenced by a setting in the MATLAB Preferences and differs between MATLAB versions.

4. At this point, a note on MATLAB's commercial Neural Networks toolbox might be warranted. Although the Neural Networks toolbox shares some of the advantages mentioned above, it trades others (especially transparency) for the sake of computational speed, the implementation of more sophisticated algorithms/architectures, and the possibility of integrating it into engineering applications (e.g., Simulink or Stateflow). Even GUI-centered usage presupposes a basic understanding of neural networks and MATLAB programming, and the object-oriented programming approach makes it difficult for anyone without a thorough understanding of the underlying concepts to see through (or customize) the code. Then again, as compared with OXlearn, the Neural Networks toolbox offers a much wider set of functionalities and might be the appropriate, if more expensive, choice for users with a background in engineering or computer science.

(Manuscript received December 16, 2008;
revision accepted for publication July 11, 2009.)