Modular Object-Oriented Neural Network Simulators and Topology Generalizations

G. Thimm, R. Grau, and E. Fiesler

IDIAP, Case postale 609, CH-1920 Martigny, Switzerland

Published in
Proceedings of the International Conference on Artificial Neural Networks (ICANN 94)
M. Marinaro and P. G. Morasso, ed., volume 1, Part 2: Mathematical Model,
pages 747–750, 26–29 May, 1994, Springer-Verlag.

1 Introduction

In current neural network research, simulation plays a crucial role. Although there is a wide range of neural network simulators available, it is impossible to keep up with the continuous surge of new neural networks and their variations. Consequently, the extensibility and modularity of neural network simulation software is an important issue. Implementation and modification of neural networks and their embedding into an simulation environment should be possible with minimal effort.

Object-oriented programming languages facilitate the fulfillment of these demands and allows the software developer to reuse software models and therefore reduce the overall implementation effort.

OpenSimulator[1] *3.1* and *Sesame*[2] *4.5*, two simulation packages written in *C++*, have the potential of fulfilling the need. Besides ready to use modules, these packages also provide a graphical user interface. Their flexibility is tested using higher order ontogenic neural networks as an example of non standard neural network topologies (see for example [Fiesler-94.1] and [Fiesler-94.2]).

2 Some Neural Network Topology Generalizations

Topologies of higher order ontogenic neural networks differ from standard neural networks topologies in two points. Firstly, the interconnections, which usually connect a source neuron with a sink neuron, are generalized by allowing a connection to combine several inputs by means of a so called *splicing function*, which maps one or more input values to a single output value.

The properties of the used connections characterize the network: the number of inputs of a connection define its order and the maximal order of a connection in a neural network determines the order of the network.

A second generalization allows a variable number of units (neurons or connections), where the number and kind of units to be added or removed is determined by a learning algorithm. This kind of network is called an *ontogenic* neural network.

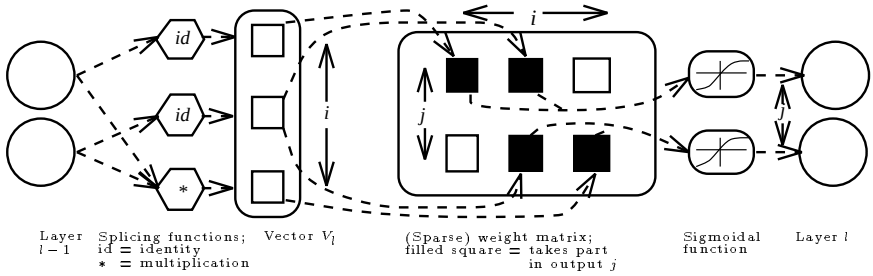
3 Data Structures for High Order Ontogenic Networks

An intuitive approach for implementing higher order neural networks is to store the weights of connections of order ω in $\omega+1$-dimensional matrices, where each dimension of the matrix either refers to one of the inputs, or to one of the outputs of a connection. Unfortunately, this data structure has some drawbacks, since several matrices have to be reserved for the connections between two layers of neurons, one for each group of connections having the same order and splicing function. These matrices are often largely unused, since ontogenic neural networks are likely to be sparsely connected. Besides this, the numerous matrices complicate both the propagation of values through the network and the updating of weights.

An improved approach introduces only necessary connections, using a projection of the activation values of layer $l-1$ to a variable sized vector V_l. Each vector element stores the result of a splicing function applied to the activation values of layer $l-1$ (see figure). In the implementation, this projection is realized by instances of a class defining connections. Each connection instance has pointers to some elements of the output vector of layer $l-1$, as well as to an element of vector V_l. Whenever such a connection instance is called, it applies a splicing function to the values indicated as its inputs and the outcome is written to result vector V_l. This vector provides the inputs for a variable sized perceptron. Only if the ith connection (writing its output to the ith element of vector V_l) is expected to take part in the input for the

[1] *OpenSimulator* is copyrighted by the ETH Zürich.
[2] *Sesame* is copyrighted by the GMD Schloß Birlinghoven.

jth neuron, the element in the ith column and the jth row of the weight matrix is allowed to be non-zero. If a connection is introduced, and another connection with the same splicing function, order, and combination of inputs exists and writes to the ith element of the vector V_l, the corresponding entry in the weight matrix is allowed to become non-zero. Otherwise, the size of the weight matrix is increased by a column, the size of the vector V_l by an element, and a new instance of the connection class is initialized.



The second approach is more efficient, both in memory and execution time. Furthermore, the vector V_l can be regarded as an input vector for a standard perceptron layer, which implies only minor modifications to a Perceptron layer implementation.

4 The Neural Network Simulators

4.1 *OpenSimulator*

The *OpenSimulator* software package was developed by J.-F. Leber at the Institute for Signal and Information Processing of the ETH Zürich [Leber-92] [Leber-93]. The proliferation of the software is restricted. *OpenSimulator 3.1* requires *UNIX*[3], *XView* and *X-Windows*[4]. The user can enter commands using a menu based interactive graphical interface or a command interpreter, or restore a previously saved session, since all data, such as the network topology, the patterns, and the weight matrices, are saved in files. *OpenSimulator* allows a group of users to share a common version of the simulator and to generate an extended version in the user's private workspace.

The functions of *OpenSimulator* are divided into two categories: functions of the *simulation kernel* and functions of their *interactive counterparts*. Both have to be created in order to introduce a new building block for a neural network. This creation is supported by a menu, which allows the creation of templates for all necessary files, to which the user is supposed to add code. In the source code, macros are used to add variables, replace or add functions, add menus, and make plots. New macros can easily be defined by the user and neural network topologies can either be defined using the graphical interface or by a file containing calls to *C++* functions.

For the construction of neural networks the following tools are available:

- matrix libraries for bytes, shorts, integers, floats, doubles, and complex data types,
- graphics libraries, and
- a basic layer class for backpropagation training, a perceptron layer class, and a topology-preserving feature map class.

4.2 *Sesame*

Sesame was developed by A. Linden and Ch. Tietz at GMD Schloß Birlinghoven in Germany [Linden-92] and is available as a freeware software package[5]. *Sesame*[6] 4.1 requires *UNIX*, *InterViews*[7], *X-Windows*, and several GNU tools and libraries. The user interface is based on a command interpreter and the current state of a simulation can be stored. *Sesame* is based on modules which can contain anything ranging from a simple counter or a complex neural network to a graphical display. New modules may be assembled from existing modules or designed in *C++*. In the latter case the export of variables or command names to the command interpreter or the declaration of *typed communication sites* is done by function calls and their handling is completely

[3] *UNIX* is a registered trademark of AT&T.

[4] *XView* and *X-Windows* are registered trademarks of the MIT.

[5] *Sesame* is at least available via ftp at the anonymous ftp site ftp.gmd.de.

[6] Meanwhile version 4.5 has been released. This version has some more features but also some minor incompatibilities with version 4.1.

[7] *InterViews* is copyrighted by the Stanford University.

covered by their classes. This includes visualization of parameters in the user interface and saving in files for future experiments.

A simulation is defined by a sequence of interpreted commands (typed or read from files) which organize the static and dynamic data flow. The static data flow is described using connection commands, whereas the dynamic data flow is described by one or more procedures, which may contain calls to other procedures, calls to modules, or simple control structures. Assembly of these commands and procedures can be done in the command interpreter, and saved together with the current state of modules, in intelligible files.

The existing modules supply the following features:

- available neural network models: one and two hidden layer backpropagation, Kohonen's self organizing feature map, and Kanerva's sparse distributed memory,
- basic modules for neural networks, like neuron layers for backpropagation or radial basis function neural networks,
- modules for data handling like vector comparison, statistical analysis, and graphical displays,
- vector/matrix handling (only completely implemented for double floating point), and
- list, stack, pattern, and file handling modules.

4.3 Comparison of the Simulators

In the table below, some of the main differences between the two simulators are summarized, and marked as advantages (⊕) or disadvantages (⊖).

Sesame 4.5	*OpenSimulator 3.1*
⊕ Provides on-line help functions, has good documentation, and a tutorial.	⊖ Provides limited on-line help and user documentation (tutorial only in German).
⊖ Has no interactive graphical user interface, therefore the interactive construction of experiments, must be done with the command interpreter.	⊕ Has an interactive graphical surface and a command interpreter.
⊕ Is easy to understand.	⊕ Its simulation kernel and interactive counterpart are clearly distinct.
⊕ Its source is easy to maintain and extend.	⊖ Its graphical interface is mainly based on a vast hierarchy of menus. Often, one has to open a confusing amount of menus to get a particular piece of information or to set parameter.
⊕ Is very flexible in combining basic modules to a complete simulation.	⊕ Is based on shared libraries.
⊖ Its functionalities are highly divided up into small modules; this causes sometimes confusion in the conception of an experiment.	

4.3.1 Remark: *OpenSimulator 4.0* has been announced

For December 1993, J.-F. Leber and A. Jarosch have announced version 4.0 of OpenSimulator, wherein the objects of a simulation can be distributed over a network containing different computer architectures, as the communication is quick and easy. Computationally expensive algorithms can therefore be executed efficiently on a super computer while the graphical interaction takes place on a workstation. Moreover, the programmer of a new module does not need to concern himself with any graphics, as they are generated on-line from an automatically generated ASCII description. The algorithms are thus ideally separated from the graphics. OpenSimulator 4.0 is more modular and flexible, simpler to use, and easier to maintain.

5 The Implementation

The first approach to implement higher order ontogenic neural networks was done using *OpenSimulator*. For the reasons given in section 3, the authors later switched to the second approach, which was implemented using *Sesame*.

5.1 Extending *OpenSimulator*

For a neural network implementation, *OpenSimulator* offers classes for neuron layers with input and output handling and optional forward, backward, and lateral feedback weights. It also provides different sigmoidal functions and functions controlling the training and testing phase.

For the implementation of higher dimensional matrices it is not appropriate to inherit from the available classes for two dimensional matrices, since these classes are highly specialized for other purposes. Also, the matrix class definition may cause some confusion, as they are distributed over a "kernel" file and other header files. This is done with the aim of realizing a kind of *C++*-template concept (which was not available when the library was written). The most specialized classes for the implementation of a neural network in *OpenSimulator* represent layers with optional weight matrices, which can be used as models for a first implementation approach

to higher order ontogenic neural networks. Although these classes do have to be rewritten, the implementation does profit from existing classes through inheritance.

Changes to *OpenSimulator* are usually spread out over the software package: both the functions in the simulation kernel and the macros used in the simulation description files have to be changed (compare paragraph 4.3.1). In the interactive counterpart, facilities for handling the new features need to be created.

For an implementation of the second approach, the above mentioned modifications as well as the problems with using the existing matrix implementation are similar.

5.2 Extending *Sesame*

Due to the design of *Sesame*, a class representing a high order ontogenic neural network needed to be written, since no appropriate module exists. The new module reuses code of a backpropagation network class, which provides a good model. The variable size weight matrix and the vector V are realized by an oversized matrix and vector respectively. A more elaborate implementation requires the extension to variable sized subclasses, which is easy. A minor problem in the implementation is the access of array elements via addresses, which was not foreseen in the design.

All changes to the *Sesame* code are confined to the new class module (except for three lines to let *Sesame* know about the existence of the class). The concepts for communication, matrix handling, and other basic concepts are clear and simple to use, and the high order neural network class is therefore easily embedded.

Regarding the first approach, similar statements can be asserted: *Sesame* has no matrix class for higher dimensional matrices but offers a suitable basic matrix class.

6 Conclusions

Both the *OpenSimulator* and the *Sesame* software packages are very useful tools for the simulation of neural networks. Besides that both simulators have the (hopefully temporary) disadvantage of still being under development, they have other characteristics in common: the functionalities presented for the implementation of higher order and ontogenic neural networks are similar. As both software packages are written in *C++*, new code can often be easily generated using the inheritance mechanism.

The overall concept of *Sesame* is clearer and the method of modular and object oriented programming is better fulfilled, and therefore the implementation of a neural network and embedding in the simulator is simpler (a revised version of *OpenSimulator* has been announced, see paragraph 4.3.1). This can be seen from the fact that changes to *Sesame* are localized in the new module, whereas *OpenSimulator* requires extensions innon-neural networks parts. However, only *OpenSimulator* allows the manipulation of the neural networks through a graphical interface. It should also be noted that, *Sesame* and *OpenSimulator* can be considered generic simulator construction tools with special features for neural networks.

Acknowledgements

The authors would like to thank Jean-François Leber at the Institute for Signal and Information Processing of the ETH Zürich and Alexander Linden and Thomas Sudbrak at the GMD in Schloß Birlinghoven and their colleagues for their swift responses to questions, helpful discussions, and their overall support of the software.

References

[Fiesler-94.1] E. Fiesler (1994), **Neural Network Classification and Formalization**, accepted by *Computer Standards & Interfaces*, 16, special issue on Neural Network Standards, J. Fulcher (ed.), North-Holland/Elsevier, Amsterdam, The Netherlands.

[Fiesler-94.2] E. Fiesler (1994), **Comparative Bibliography of Ontogenic Neural Networks**, in these proceedings (ICANN'94).

[Leber-92] J.-F. Leber and G. S. Moschytz (1992), **An Acoustical Signal Recognizer Implemented on a Novel Interactive Object-Oriented Neural Network Simulator**, in I. Alexander and J. Taylor (ed.), *Artificial Neural Networks, 2*, pp. 1291–1294, North-Holland/Elsevier, Amsterdam, The Netherlands, Sep. 4–7.

[Leber-93] J.-F. Leber (1993), **The Recognition of Acoustical Signals Using Neural Networks and an Open Simulator**. Series in Microelectronics, 20, W. Fichtner, W. Guggenbühl, H. Melchior, and G. S. Moschytz (ed.), Hartung-Gorre Verlag, Konstanz, Germany.

[Linden-92] A. Linden and C. Tietz (1992). **SESAME - A Software Environment for Combining Multiple Neural Network Paradigms and Applications**, in I. Alexander and J. Taylor (ed.), *Artificial Neural Networks, 2*, pp. 1265–1268, North-Holland/Elsevier, Amsterdam, The Netherlands, Sep. 4–7.