PREENS, a Parallel Research Execution Environment for Neural Systems

Louis Vuurpijl, Theo Schouten and Jan Vytopil

University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands
phone: +31 80 652710, fax: +31 80 553450, email: louis@cs.kun.nl

Abstract. PREENS – a Parallel Research Execution Environment for Neural Systems – is a distributed neurosimulator, targeted on networks of workstations and transputer systems. As current applications of neural networks often contain large amounts of data and as the neural networks involved in tasks such as vision are very large, high requirements on memory and computational resources are imposed on the target execution platforms. PREENS can be executed in a distributed environment, i.e. tools and neural network simulation programs can be running on any machine connectable via TCP/IP. Using this approach, larger tasks and more data can be examined using an efficient coarse grained parallelism. Furthermore, the design of PREENS allows for neural networks to be running on any high performance MIMD machine such as a transputer system. In this paper, the different features and design concepts of PREENS are discussed. These can also be used for other applications, like image processing.

1 Introduction

A neurosimulator is a toolbox containing tools for monitoring, manipulating, controlling and executing neural network simulation programs. Current neurosimulators [1] such as the RCS [3], Pygmalion [1] Genesis [13], Planet [8], and others are all based on a general neural network datastructure or some neural network description language (NDL). Using a hierarchical description of its architecture, the neurosimulator is able to access the data associated with the neural network. Also, in many cases the execution of a neural network program is performed based on the data dependency between different neural network components as described in the NDL. This approach requires neural networks to be specified in a prescribed standard manner, which usually means that their implementations are not as efficient as would be the case with implementations specifically designed for a neural network and target execution platform. Furthermore, it forces neural network programmers to program their applications following the NDL syntax or using some set of neural network library routines. PREENS is an execution environment that supports the execution, monitoring and control of neural network simulations. The construction of simulation

[1] For a taxonomy over neurosimulators, see [9]

programs is not supported, to develop a simulation program general programming tools have to be used. Thus unlike existing neurosimulators, in PREENS neural network programs do not have to be specified following the syntax of some general neural network description datastructure. This has the advantage that simulation programs can be efficiently coded and that the programmer has full control over his programs, so new algorithms and applications can be tailor made. Each neural network simulation can be implemented specifically targeted on a certain machine architecture or using the most appropriate datastructures and flow of control.

Based on the observations that neural network simulation programs all have a similar structure and furthermore in general are relatively easy to implement, PREENS uses a conceptual model of *programs* rather than of neural *networks*. In the conceptual model, a program is described via the actions that can be executed. Each program implements a limited number of such actions, e.g. loading of a network or patterns and training, recalling or classifying a set of patterns. Each action can have parameters for initiating it, variables and data that change during execution, and options and settings that guide the program's flow of control. For example, consider table 1 for some sample actions with corresponding parameters, variables etc. In this table, the action `load_patterns` has as parameter the filename from which to load the patterns and as data object the datastructure containing the patterns. Furthermore, it as the setting `normalize` which can be on or off for indicating whether the loaded data has to be normalized or not and it has an option `which_patterns` which can select whether either training data, recall data or data to be classified has to be loaded:

action	parameters	variables	data	settings	options
load_patterns	filename		patterns	normalize	which_patterns
recall		% correct	output		
train	learning rate	error	weights, activations	use_noise	batch_update

Table 1. Actions, parameters, etc.

By examining a program description containing the actions and corresponding objects, PREENS is able to create a user-interface *convis* that enables *con*trol over the program and allows manipulation and *vis*ualization of the objects it contains. Furthermore, it features a data communication interface via which objects can be linked with tools running on any machine connectable to the one convis is running on. Each neural network simulation program is equipped with an interface to convis that "listens" to commands of the user-interface, and examines which program objects have to receive or send information to or from tools. For an overview of PREENS and how it interacts through convis and the data communication interface with running tools and neural network simulation programs, see figure 1.

Note that PREENS fully exploits the availability of workstations in a network environment. As tools, convis and the neural network simulation can be run

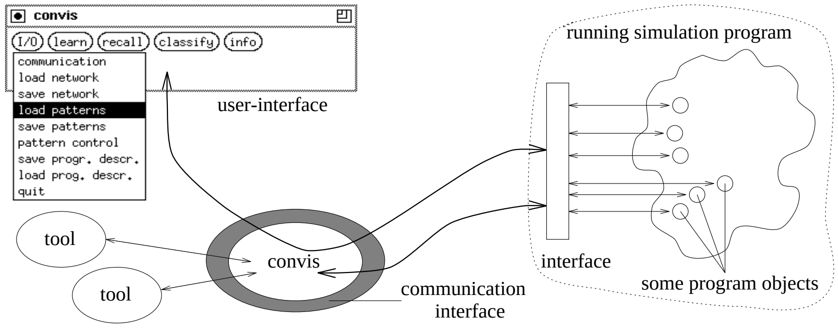

Fig. 1. *The PREENS architecture*

on any machine, the environment can be executed in parallel. Therefore, it can operate faster and handle larger amounts of data than current neurosimulators. Currently, PREENS can handle tools and simulation programs running on workstations and transputer systems. It has been successfully tested on heterogeneous processor networks consisting of Dec and Sun workstations and a Parsytec transputer system.

2 Integrating PREENS with tools

As introduced above, PREENS consist of a manager and user-interface (convis), a data communication interface, a set of tools and an algorithm library. Furthermore, it features a set of interface definitions that can be used to integrate it with (new) tools and neural network algorithms. For any specific neural network application, some standard tools can be used for e.g. monitoring the evolution of weight space during training, monitoring error plots, etc. These tools are also contained in the current PREENS toolset. However in many cases also some application specific tools are used, such as a CCD-camera with framegrabber inputting data online to the running simulation program, some special weight monitoring tools (e.g. a U-matrix displayer [11] or the hyperplane animator [4]), or high quality data processing tools such as Matlab [7].
If new tools have to be added to the environment, only three interface routines have to be used, the first to setup the communication with convis, the second to find out about the data that the tool will be supplied with and the third to communicate data between a neural network simulation program and the tool. The following pseudo-code gives an idea about how this is done:

```
init_communication(host,port); /*   initiates a socket connection  */
ReceiveDataDescription(&data); /*   finds out about  the data and  */
while(...)                     /*           allocates it           */
    ReceiveData(&data);        /* receives the data sent by convis */
    HandleData(data);          /*          tool specific           */
```

The last routine is specific for the tool. For example, we have equipped PREENS with an engine for Matlab following the concept described above by storing the data in matrix format in a file and redirecting the standard input of Matlab to a pipe that sends display or processing commands operating on the data contained in the file.
This approach has two major advantages. The first is that if compute intensive tools like Matlab are used, they can be run on different machines thus reducing the computational load of the machine(s) on which the neural network simulation is running. The second is that tools can be used to examine data online. For example neural network researchers usually store their data at certain time steps during a training process, then start up a tool like Matlab, load the data and process it. Using PREENS, they can define a data object, connect it via convis to a dynamic data evaluation tool, mark the object for examination and watch its evolution online.

3 Integrating PREENS with neural network algorithms

The current PREENS algorithm library contains implementations of backpropagation [10], Kohonen [6], ART[2], Boltzmann Perceptron [5] and other neural network paradigms. If a new neural network algorithm has to be included in the PREENS algorithm library, its program description has to be specified and each action that is contained in the description has to be implemented. Furthermore, a main loop has to be installed that knows about the neural network simulation program by examining its program description. Actions contained in the description can be called via the user-interface. If an action needs to be controllable, i.e. it must be able to interrupt it and view or edit its corresponding objects, a special interface routine called `action_control` has to be incorporated within the code. For example, consider the following code that implements a typical action `train`:

```
int train (Action *train, ...)
    while (error_not_reached && niterations-- && action_control(train))
        do_train;
```

No restrictions are made about the way in which the training algorithm `do_train` is implemented. This involves that neural network programmers can use their own programming style and can tailor their application to their own needs. The conditional statement in the while loop contains two stopping criteria as defined by the training algorithm, and the interface routine `action_control`. The latter examines the program description associated with the action `train` and sends each corresponding object that needs to be monitored to convis. Each object can be annotated with timing information denoting the interval at which it has to be monitored. Furthermore, the routine listens for interrupts from the user-interface or for incoming data. If such events occur, the training procedure is exited and the main loop is re-entered. Any user commands can than be handled or incoming data can be received.

4 PREENS and parallel neural network simulations

As explained above, the main loop listens to update, request and control commands from convis. For each object contained in a neural network simulation program, new values received from input tools or entered by a user via the user-interface are accessed via the program description. If the neural network program is a parallel implementation running on e.g. a transputer system, new values have to be made available through the processor network. Similarly, if values have to be extracted from the simulation program, they have to be somehow gathered from the processor network. PREENS contains a distributed interface for broadcasting and gathering distributed data objects and commands between convis and a parallel simulation program. The current version supports parallel implementations decomposed via the coarse grained dataset decomposition technique [12]. As with this technique each processor hosts the same neural network, there is no problem with gathering and broadcasting distributed information. For applications which use other decomposition techniques, a set of communication routines is being supplied that support gathering and broadcasting of distributed information.
The distributed interface assumes master-slave implementations of the neural networks, where the master distributes commands and transmits data with the slaves. For distributing and gathering data that is distributed over the processor network, a set of data communication routines is provided. This set can be expanded with routines that gather or broadcast distributed data objects that are not structured following the PREENS data formats.

5 Conclusions

We have introduced PREENS, a neurosimulator for distributed environments. PREENS uses the concept of specifying neural network simulation programs via program descriptions containing actions and corresponding objects. Input and output tools for pre- or post processing of data can be bound with individual objects and started on any machine that can be connected with the user-interface convis. New (parallel) neural networks and new tools can be integrated within the PREENS environment using a small set of interface routines and the data communication interface. As neural network programmers do not have to confirm to some general neural network datastructure, the neural network code can be implemented specifically tailored for an application. Furthermore, as the user-interface, tools and neural network program are 'loosely coupled', it is possible to use neural network program code as a standalone program or to fit it in different application codes. This is an important feature for appliers of neural networks who initially want to use a toolbox for tuning and optimizing a specific neural network simulation, and later want to use its code within their own applications. Using action-oriented program descriptions could also facilitate a means of designing toolboxes for other application areas like image processing. Similar to neural network applications, with image processing a relatively small set of actions can be identified such as loading and storing an image, filtering operations, edge detection, segmentation, etc. The possibility of incorporating these within one toolset and exploiting the resources in a distributed environment by executing them on different processors would justify the use of an approach like we have sketched for PREENS.

References

1. M. Azema-Barac, M. Hewetson, M. Recce, J. Taylor, P. Treleavan, and M. Vellasco. Pygmalion Neural Network Programming Environment. In B. Angeniol and B. Widrow, editors, *International Neural Network Conference*, pages 1237–1244, Paris, July 1990. Kluwer Academic Publishers.
2. G.A. Carpenter and S. Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, 37:54–115, 1987.
3. N.H. Goddard, K.J. Lynne, T. Mintz, and L. Bukys. Rochester Connectionist Simulator. Technical Report 233 (revised), University of Rochester, Computer Science Department, Rochester, New York 14627, October 1989.
4. P. Hoeper. The Neural Network Hyperplane Animator Program V. 2.0 General Description and Applications. ftp: cs.rutgers.edu; get pub/hyperplane.animator/Ver2paper.ps, May 1993.
5. B. Kappen. Using Boltzmann Machines for probability estimation. In S. Gielen and B. Kappen, editors, *ICANN '93*, pages 521–526. Springer-Verlag, September 1993.
6. T. Kohonen. *Self-Organization and Associative Memory.* Springer Verlag, Berlin, second edition, 1988.
7. Mathworks, Cochituate Place 24, Prime Pathway, Natic, Mas 01760. *Matlab, High Performance Numeric Computation and Visualization Software*, 1992. Version 4.0 User's Guide.
8. Y. Miyata. *A User's Guide to PlaNet Version 5.6, A Tool for Constructing, Running and Looking into a PDP Network.* University of Boulder, Computer Science Department, Boulder, January 1991.
9. M.L. Recce, P.V. Rocha, and P.C. Treleaven. Neural Network Programming Environments. In I. Aleksander and J. Taylor, editors, *Artificial Neural Networks 2*, pages 1237–1244. Elsevier Science Publishers, September 1992.
10. D.E. Rumelhart and J.L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, 1986.
11. A. Ultsch and H.P. Siemon. Exploratory Data Analysis: Using Kohonen Networks on Transputers. Technical Report Bericht Nr. 329, University of Dortmund, Fachbereich Informatik, Postfach 500500, D-4600 Dortmund 50, December 1989.
12. L.G. Vuurpijl and Th.E. Schouten. Suitability of Transputers for Neural Network Simulations. In W. Joosen and E. Milgrom, editors, *Parallel Computing: From Theory to Sound Practice*, pages 528–537. IOS Press, 1992.
13. M.A. Wilson, U.S. Bhalla, J.D. Uhley, and J.M. Bower. Genesis: A System for Simulating Neural Networks. In D.S. Touretzky, editor, *Advances in Neural and Information Processing Systems*, pages 485–492. Morgan Kaufmann, 1989.