A Flexible Graphical User Interface for Embedding Heterogeneous Neural Network Simulators

Radu Drossu
rdrossu@eecs.wsu.edu

Zoran Obradović[1]
zoran@eecs.wsu.edu

Justin Fletcher
jfletche@eecs.wsu.edu

School of Electrical Engineering and Computer Science
Washington State University, Pullman, Washington, 99164-2752

Abstract— **The graphical user interface (GUI) to heterogeneous neural network simulators proposed in this article is intended to be of use both for the novice and for the experienced neural network user. For the novice, it provides an easy to use neural network simulation package that insulates the user from the need of knowing the simulator implementation details or the configuration file syntax. For the experienced neural network professional it provides an interface that is easily extensible to include any additional neural network simulator in its binary form. To satisfy both academic and personal computer environments, the GUI has been developed by using the free TCL/TK software package, available both on workstations running UNIX and on PC's running the free Linux operating system. Although the GUI and the embedded simulators have been successfully tested both in neural network research and training programs, a more extensive testing in undergraduate and graduate level classes is in progress.**

I. Introduction

Neural network technology has the potential to solve an impressive variety of today's technological problems which conventional computers are quite powerless to deal with [5]. Popularity of this technology is rapidly growing since it proved its applicability in everyday applications as diverse as signature verification [2], power consumption predictions [14], or financial markets forecasting [4].

Neural network data modelling techniques are applicable to two general types of problems:

- Classification problems, in which the goal is to classify a number of patterns as belonging to different categories (an example would be to automatically recognize hand written zip codes and classify each symbol in one of ten categories corresponding to the digits from 0 to 9 [15]).
- Prediction problems, in which the goal is to predict a future value of a specific process parameter (an example would be the prediction of the U.S. power consumption for the year 1997). The prediction problems can be regarded as limit cases of classification problems in which the number of classes is infinite.

In addition, these techniques are applicable not only to time independent, but also to time dependent problems (time series), where the patterns to be classified are temporal sequences of previous process values.

Neural network computational models consist of a number of relatively simple, interconnected processing units called *neurons* working in parallel. Different neural network *architectures* can be used to solve a given classification or prediction problem. In an architecture the neurons are interconnected through *synaptic links* and are grouped in *layers*, with synaptic links usually connecting neurons in adjacent layers. Typically three different layer types can be distinguished: *input layer* (the layer that external stimuli are applied to), *output layer* (the layer that outputs results to the exterior world) and one or more *hidden layers* (intermediate computational layers between input and output layer). The neural network architectures can be, roughly speaking, divided in two categories: *feedforward*, in which the signal flow between different layers is unidirectional (from input layer towards output layer), or *recurrent*, in which the flow between different layers is bidirectional (both from input layer towards output layer as well as opposite).

The most distinctive element in neural networks, as opposed to traditional computational structures, is denoted as *learning*. Learning represents the adaptation of some characteristic neural network parameters by using a set of examples (known outcomes of the problem for given conditions) so that for given input patterns the outputs reflect the value of a specific function. The final data modelling goal is not to memorize some known patterns, but to be able to *generalize* from prior examples (to be able to estimate the outcomes of the problem under unknown conditions).

Neural network data modeling represents the combination of an architecture and an appropriate learning technique. Traditional approaches assume learning on a prespecified architecture, whereas *constructive learning* creates the appropriate architecture during the learning process. Most neural networks have a number of architectural and learning parameters that have to be appropri-

[1] Sponsored in part by the NSF research grant NSF-IRI-9308523 and by the NSF education grant NSF-ESI-9254358.

ately specified. Selection of appropriate parameters for large scale applications is an important experimental problem. If parameters are not appropriate, the algorithms may take a long time to converge or may not converge at all [21]. Consequently, in training novice neural networks users, an emphasis on techniques and intuition development is necessary for properly setting these parameters.

Currently, most neural networks modeling is performed on standard sequential computers (rather than on specialized neural networks hardware) using simulation software. Until recently the choice of a simulation software was not a primary issue among neural network professionals, as applications of neural networks technology have been limited to relatively small problems. However, the choice of an efficient simulation software plays an important role when dealing with large scale, real life applications as desired by industry. The approach proposed in this paper provides developers with a number of simulator modules implementing various learning algorithms, all using a single user interface. In such an environment it is easier to select the model that is appropriate for a particular application than to become accustomed to various independent simulators.

Until recently, most neural network modeling work was performed by highly trained neural networks experts. In practice, such experts learn how to use several commercial or freely available packages and in addition develop their own software to deal with special issues not supported by the packages. Experience seems to indicate that for dealing with features not supported by the standard packages, even inside a single group, most neural networks experts prefer building their own software rather than using code developed by other experts, as it usually takes a significant time to learn how to use somebody else's neural networks code properly. We strongly believe that the software reuse among experts can be significantly improved by using a flexible user interface, as proposed in this paper, that can easily support developers' needs.

As the applicability of artificial neural networks seems to extend far beyond the computer science domain to areas like electrical engineering, biology, medicine, business, the need for rapidly trained personnel able to quickly assimilate neural network techniques and to apply them to the corresponding domain becomes more and more acute. Thus the necessity to acquire some basic neural network knowledge rapidly, without covering underlying mathematical and programing details, but rather concentrating on understanding the conceptual model as well as on designing adequate experiments tuned to a particular problem. Our experience indicates that the choice of a graphical user interface, as proposed in this paper, plays a crucial role when trying to transfer neural network technology rapidly to non-expert users.

The graphical user interface (GUI) proposed in this article is intended to be of use both for the novice and for the experienced user. The software package included with this paper provides an interface to both the most used traditional backpropagation learning through gradient descent optimization on a prespecified analog architecture [17] and to a novel learning technique developed in our lab, known as hyperplane determination from examples (HDE), that automatically grows a problem tailored discrete architecture [10, 11]. The fine-tuning of each neural network learning process is done by adjusting specific learning parameters. For the novice, it provides an easy to use neural network simulation package supporting both backpropagation, as well as HDE, that insulates the user from the need of knowing the implementation details, as well as the configuration file syntax for running the simulators. For the experienced neural network professional it provides an interface that is easily extensible to include any additional neural network simulator in its binary form.

The paper is organized as follows: Section 2 contains a description of the graphical user interface and its underlying menu structure; Section 3 illustrates the flexibility, portability and extendibility of the GUI by means of embedding two radically different neural network simulators; finally, Section 4 discusses the applications of the GUI in research and teaching, as well as suggestions for further improvements of the GUI.

II. GUI Design and Implementation

The objective of our effort was to develop a portable GUI that would be able to serve neural network professionals, as well as a large number of novice neural network users, who might have access only to personal computers (PCs). With the introduction of Linux, a free UNIX-like operating system that runs on Intel 386/486/Pentium based personal computers (PCs), the use of UNIX and X Window environments becomes affordable not only to institutions, but also to the large PC users community [1]. Linux provides the PC user with the power and flexibility of the UNIX operating system, as well as with an overwhelming amount of free, good quality software. One of the software packages, included in most of the Linux distributions, available also on UNIX workstations in academic environments, is Dr. John Ousterhout's TCL/TK package [16]. TCL, standing for "tool command language" is a general purpose scripting language for controlling and extending applications, whereas TK is a toolkit designed to develop X Window applications. For those who aren't familiar with X Windows, this represents a windowing environment, similar in appearance to PC window environments, that is used in the UNIX environment. Both TCL and TK are implemented as libraries of C procedures that allow the extension of their core features. In its actual form, TCL is interpreted, thus suggesting that there might be a slowdown as compared to a compiled program. This potential slowdown isn't apparent for scripts of a few thousand lines, as is sufficient for developing a GUI for neural network simulation. The CD-ROM software accompanying this paper includes the GUI and simulators' executable files for both HP 9000 series workstations, as well as for PC's running Linux. To run the included software on one of these platforms, the existence of TCL ver. 7.3 and TK ver. 3.6 is also required.

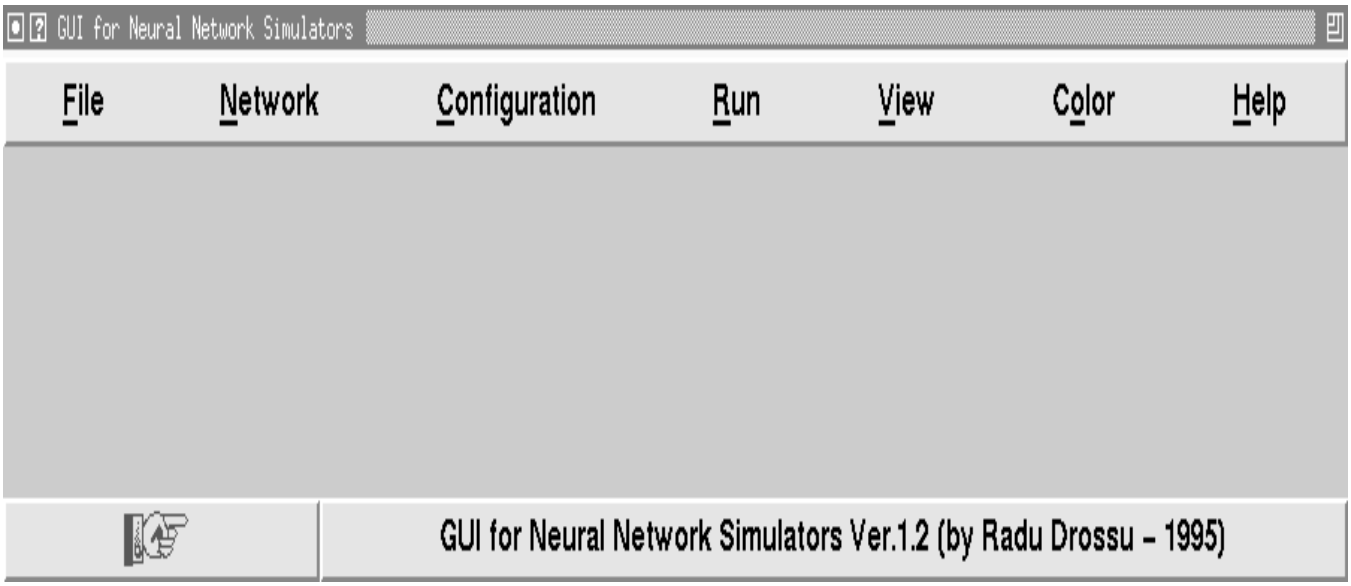

Figure 1: GUI Main Window

The GUI proposed in this paper is developed for UNIX and Linux environments, to satisfy both expert and novel users' needs. For ensuring portability, the GUI has been written using just the TCL/TK core features without any additional C procedures. The GUI allows an interactive setting of neural network simulator parameters as well as the execution of the simulations. In its current form, the GUI embeds two radically different simulators, a backpropagation simulator and the HDE constructive learning simulator. A basic understanding of TCL/TK should allow the user to easily embed additional simulators by "linking" their executable files to the GUI interface.

The GUI provides the following basic features:

1. A basic text editor for editing plain text files.
2. The color configuration of all the GUI items.
3. Setting the characteristic features and parameters for the underlying neural network simulators.
4. Saving the neural network configuration in the simulator specific file formats, as well as retrieving a neural network configuration from a file in the case of the backpropagation simulator.
5. Viewing the neural network configuration both in a tutorial style form (in which the significance of the different configuration options is briefly explained), as well as in the format in which the configuration would appear in the simulators' configuration files.
6. Running the simulators both as foreground and as background processes (for an explanation of foreground and background processes, the reader can consult [19, 22]).
7. A help file that provides information on running the simulators both by using the GUI, as well as by editing the corresponding configuration files for the simulators.

In order to run the GUI, the contents of either the directory called NNGUI/UNIX (for HP 9000 series workstations), or of NNGUI/LINUX (for PC's running Linux) from the CD-ROM needs to be copied to a user's directory, say MYNNGUI. The current directory needs to be changed to MYNNGUI and the GUI can then be started by issuing the command *nngui*. The GUI main window is then displayed, as shown in Fig. 1. The different menus can be accessed either by clicking them with the left mouse button or keeping the ALT key pressed while pressing the underlined letter of the menu name. During application execution, certain GUI options can be accessed directly from the main window through the use of a keyboard accelerator (*hot key*). The options that have corresponding hot keys have the key combination displayed in the pop-up menus (read also the help file for a list of options), where CTRL+A stands for "press the CTRL key and keep it pressed while pressing the A key "(the GUI is not case sensitive, so lowercase a is the same as uppercase A). In each file selection box, a file is selected by double-clicking it with the left mouse button.

The menu structure of the GUI can be represented hierarchically as in Fig. 2.

A brief explanation of each of the different menu items supported under the current GUI implementation follows. For a more detailed explanation, install the program from the CD-ROM and select the *Help* option under the *Help* menu item, or press CTRL+H in the main menu window.

A. The File Menu

The following file access options are included in this menu.

- The *Edit File* menu item provides access to a simple general purpose text editor, that allows the GUI user to easily change parameters in a previously used neural network configuration file. A file selection box allows the user to select either an existing or a new file to edit (see Fig. 3). An editor window is then displayed, having the basic editor commands printed at the bottom

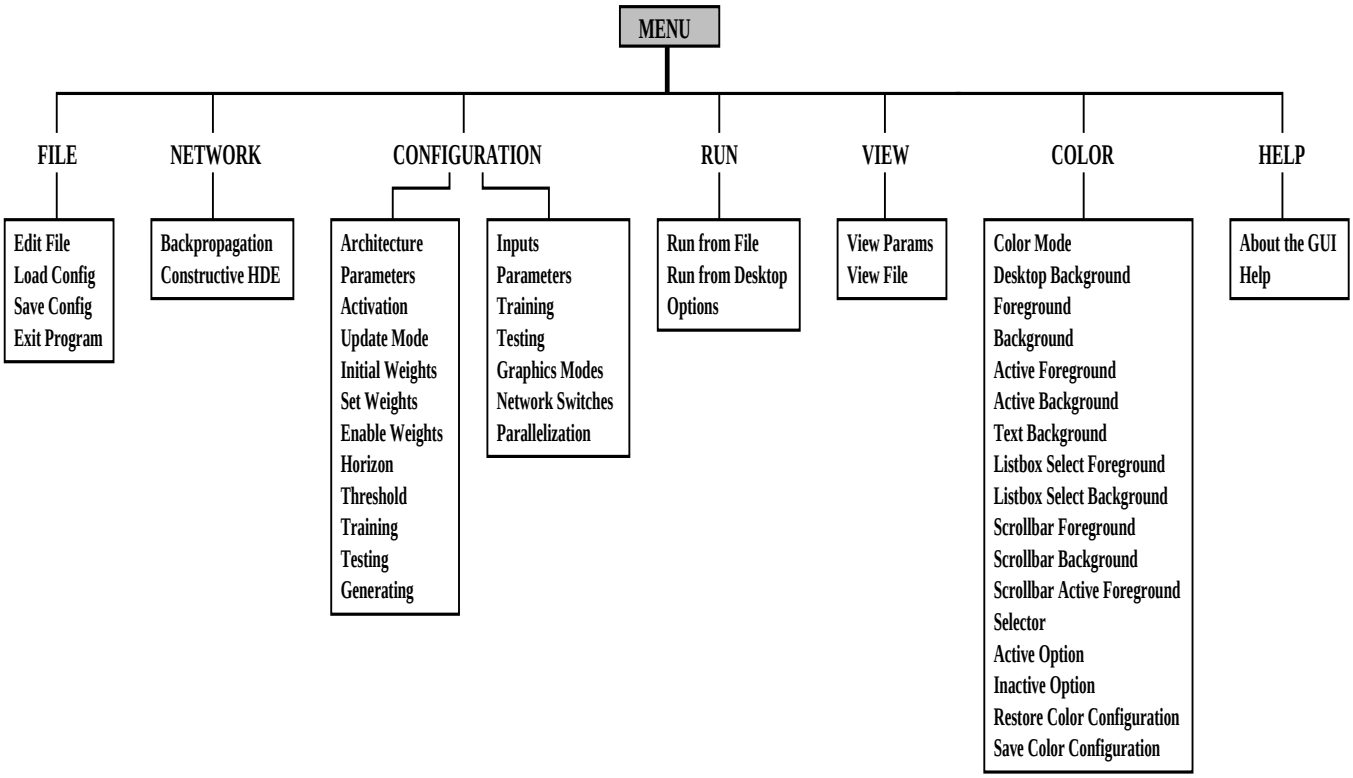

Figure 2: GUI Menu Structure

of the screen (see Fig. 4).

- The *Load Config* menu item allows the loading of a previously saved configuration file. It also allows the more advanced user to load a configuration file that was created directly (without the use of the GUI). In its current implementation, the GUI supports this option just for the backpropagation simulator.
- The *Save Config* menu item allows the saving of the neural network configuration defined in the GUI to a file. This allows the user to save a certain neural network configuration for future use.
- The *Exit Program* menu item exits the application.

B. The Network Menu

This menu allows the selection of the desired simulator. The current version of the GUI supports two different neural network models:

- The *Backpropagation* menu item selects the backpropagation simulator. By selecting this option, the settings specific to the backpropagation simulator will be displayed under the Configuration menu.
- The *Constructive HDE* menu item selects the HDE constructive learning simulator. This option will also enable the display of constructive learning specific settings under the Configuration menu.

C. The Configuration Menu

Parameters specific to the supported neural network models are set by using this menu. This menu is neural network model sensitive, meaning that the characteristics displayed depend on the simulator set in the Network menu. A more detailed description of this menu in the current GUI implementation is given in Section 3.

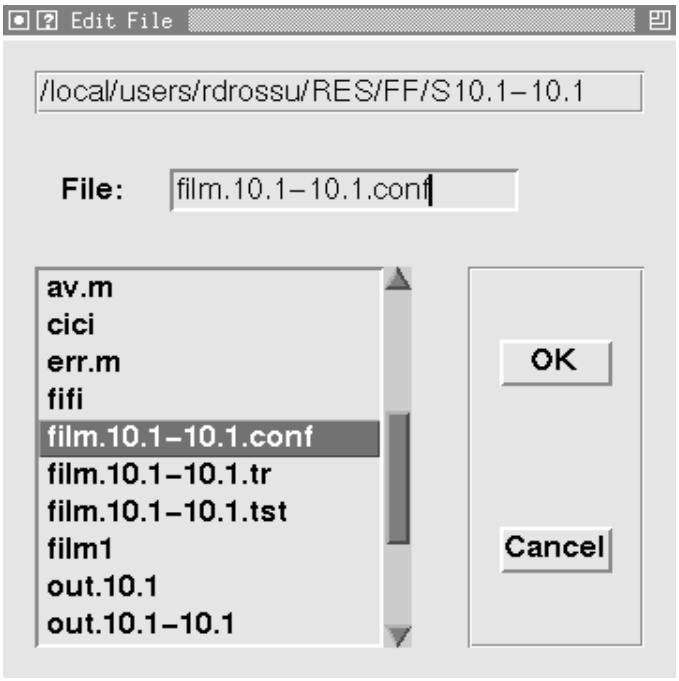

Figure 3: File Selection Box

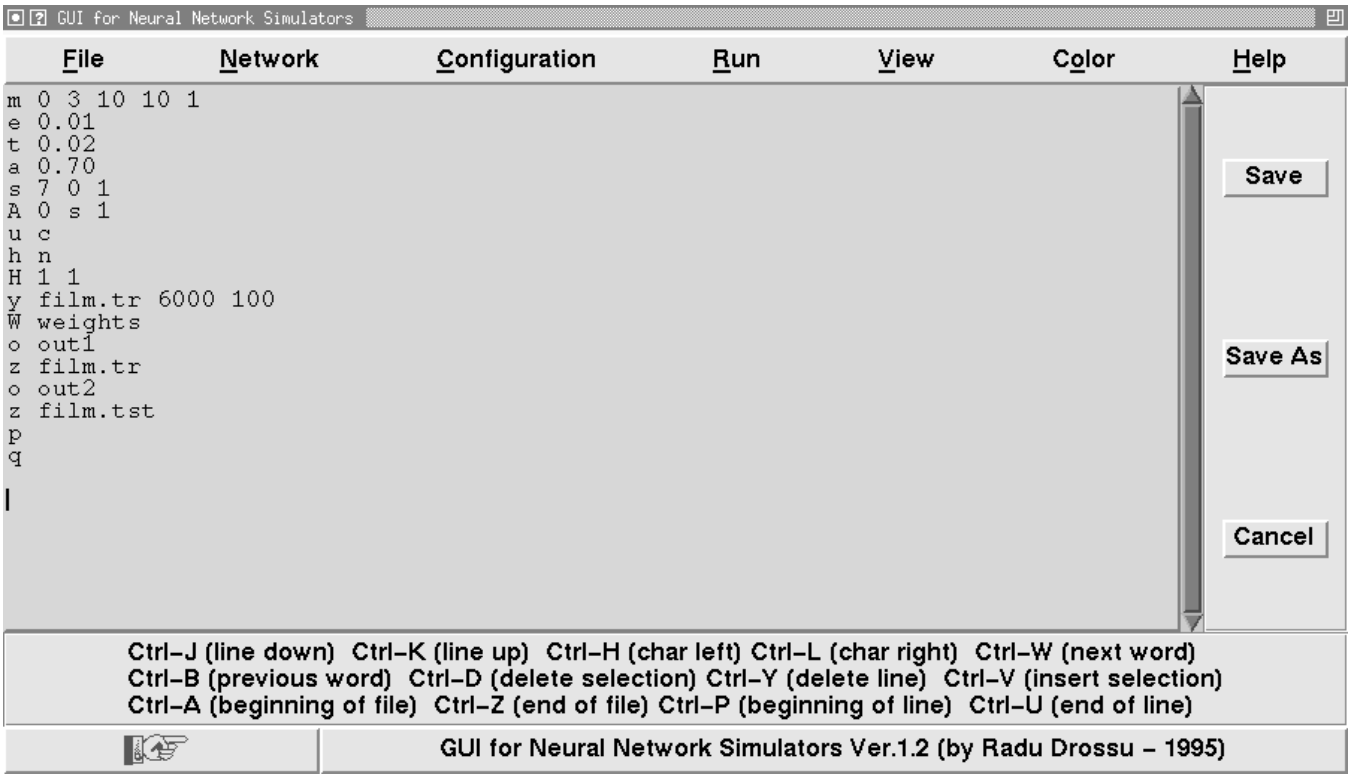

Figure 4: Edit File

D. The Run Menu

The Run menu starts the selected neural network simulator. It consists of three menu items:

- *Run from File* starts the simulator using a simulator specific configuration file. This is the option that an expert in a particular neural network model would most likely use.
- *Run from Desktop* starts the simulator using the parameters that have been configured by using the GUI. This is the preferred option for a novice user, who is unfamiliar with a particular neural network simulator.
- *Options* allows the choice of whether to run the simulator as a foreground or as a background job (in which case the output of the simulation will be redirected to a file). The foreground option is usually preferred in the case of a short simulation run (of the order of minutes), whereas the background option is used in the case of long simulation runs (that last hours or even days).

E. The View Menu

This menu allows the display of the current simulator settings.

- *View Params* displays the settings in a fairly self explanatory way, thus helping the user to decide whether all the settings are appropriate or not.
- *View File* displays the settings in a more cryptic way, characteristic to the specific configuration file syntax of the underlying simulators.

F. The Color Menu

For a more pleasant appearance of the GUI, a multitude of color options can be set in this menu. The *Color Mode* menu item allows the selection of colors in three different ways, either by using X Windows' default names, or by using two more sophisticated, but fine tunable, color selection techniques, HSV and RGB [9]. A setup screen for an HSV color selection is shown in Fig. 5.

The default color configuration can be restored by selecting the *Restore Color Configuration* menu item, whereas the preferred color configuration can be saved by choosing the *Save Color Configuration* menu item. Whenever a new color configuration is saved, it will overwrite the previously saved color configuration stored in the file config.crt.

G. The Help Menu

This menu contains a release number of the current GUI version under the *About the GUI* menu item, as well as a help file under *Help*.

III. The Configuration Menu as a Key to Embedding Heterogeneous Neural Network Simulators

The two neural network simulators embedded in the current GUI are drastically different in implementation. The backpropagation simulator represents a traditional learning technique applied to a fixed architecture, whereas the HDE constructive learning simulator updates the architecture during the learning process. Implementation-wise also, the two simulators are also radically different. Although both

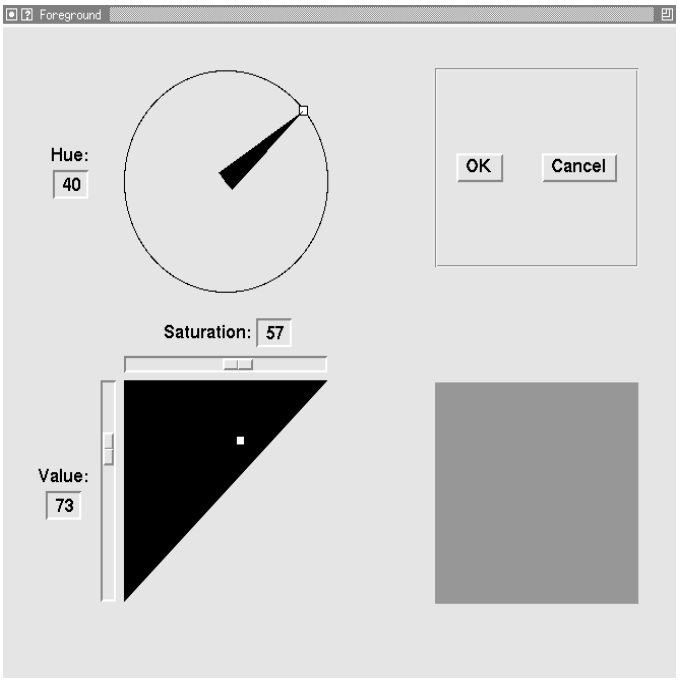

Figure 5: HSV Color Mode Selection

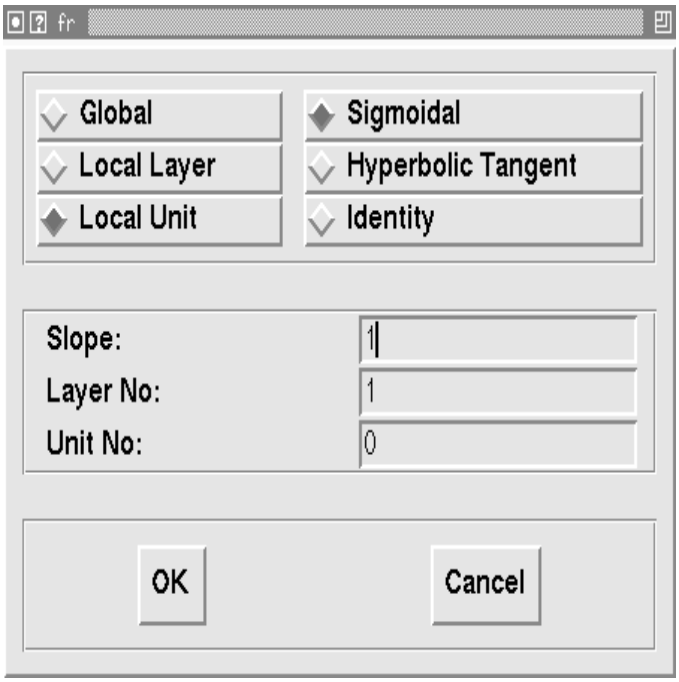

Figure 6: Setting the Activation Function for the Backpropagation Simulator

of them were programmed in C, the HDE simulator also uses XLib calls to implement a graphical visualization of the learning process. In addition, whereas the backpropagation simulator is intended to be run only sequentially (on a single sequential computer), the HDE simulator can be run both sequentially or in parallel, either on a highly parallel machine, such as the Paragon at the San Diego Supercomputer Center [18], or in a local distributed system (a computer network). Effective support of two so different neural network models by a single GUI is a strong indication that the GUI is *flexible*. In addition, as long as a compiled (binary) version of the simulators is available on a certain computer that has the TCL/TK (ver. 7.3/3.6) package installed, the GUI can be successfully used, thus showing that the GUI is *portable*. Finally, when additional neural network simulators need to be embedded in the GUI, this can be accomplished without significant effort, as long as the binary files of the simulators are available, proving that the GUI is also *extensible*.

From the perspective of the end user, who should be insulated from implementation details, the only differences that appear between various simulators are reflected just in the Configuration menu. As mentioned before, this menu depends on the neural network simulator selected in the Network menu. If additional simulators are added to the GUI, these will be reflected in the corresponding Configuration menu options when selected.

A brief description of the configuration options for the two neural network simulators included in this version of the GUI follows. For a detailed description of the parameters that are set in this menu, the reader is advised to install the software from the CD-ROM and to consult the help file (by pressing CTRL+H in the main window). For configuring model parameters, basic concepts for the backpropagation learning technique can be found in [12, 13], whereas for the HDE constructive learning in [10, 11].

The corresponding Configuration menu settings for the backpropagation simulator are:

- *Architecture* allows to select either a feedforward or a recurrent neural network, as well as to establish the desired layer structure (number of layers and number of neurons per layer). In the case of recurrent networks it also allows the selection of the feedback type and of the feedback layer, as well as of the scaling parameters for the feedback signals.
- *Parameters* sets the backpropagation specific learning parameters, known as learning rate, momentum and tolerance.
- *Activation* allows the setting of the neurons' activation function to either a sigmoidal, a hyperbolic tangent or an identity function. The setting can be done either globally (for all the neurons in the network) or locally, either at layer level or at neuron (unit) level. An actual screen of the activation function window is presented in Fig. 6.
- *Update Mode* sets one of the two specific modes in which the weights of the synaptic links are updated, either continuously (after each training example) or in batch mode (after each presentation of all examples from the training set).
- *Initial Weights* allows to set the initial value of the synaptic links' weights either randomly, or with values read from a file.
- *Set Weights* allows the setting of particular weight values either globally (for all the synaptic links), or locally (for all the links connecting two adjacent layers, for all

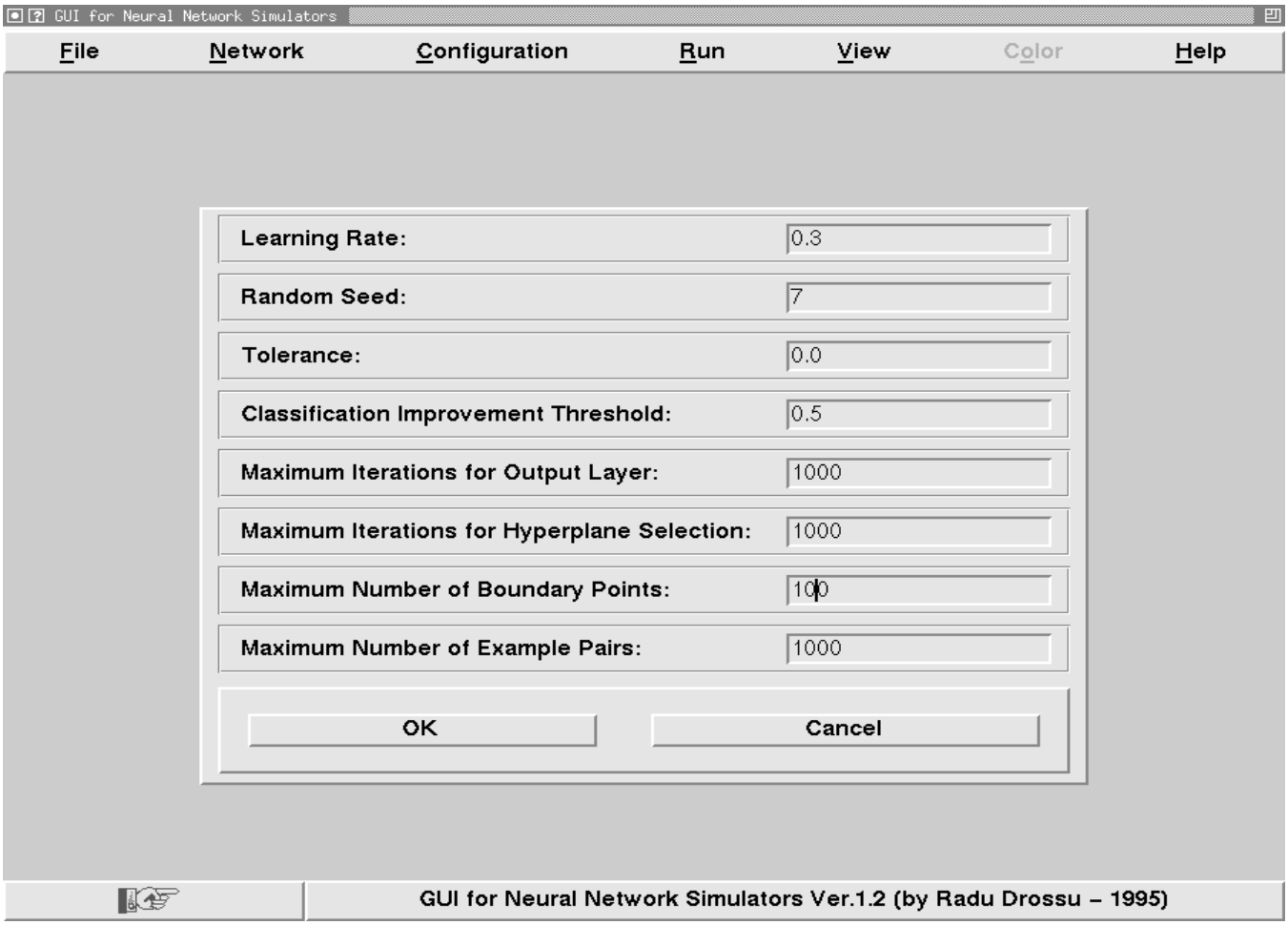

Figure 7: Setting the Parameters for the HDE Simulator

the links emanating from a given unit, for all the links ending at a given unit, or for a single link).

- *Enable Weights* allows the enabling or disabling of certain synaptic links in ways similar to setting weight values. A disabled link behaves like a disconnected link.
- *Horizon* allows to set the desired testing horizon for time series prediction problems (a testing horizon of one indicates the prediction of just the next process value, whereas a prediction horizon larger than one indicates the prediction of a process value further in future). This menu item also allows the setting of the training horizon, an internal learning parameter that can't exceed the testing horizon.
- *Threshold* allows a brute force elimination of training examples with outputs above or below a certain level (*None* stands for no elimination of training examples).
- *Training* selects the training file (file containing all the examples used during the learning process), as well as the number of training epochs (number of repetitive passes through the example file) and the number of epochs after which to display the learning status.
- *Testing* selects the testing file (file containing examples not seen during the learning process, on which to test the generalization ability of the neural network).
- *Generating* allows the generation of a time series.

The Configuration menu settings for the HDE constructive learning simulator are:

- *Inputs* allows to set the number of neural network inputs (the number of outputs is one, since the simulator is used just for binary classification problems).
- *Parameters* allows the setting of HDE characteristic parameters like learning rate, random seed, tolerance, classification improvement threshold, maximum iterations for output layer, maximum iteration for hyperplane selection, maximum number of boundary points, maximum number of example pairs. The parameters setting screen is shown in Fig. 7.
- *Training* and *Testing* allow the setting of training and testing files, similar to the case of the backpropagation simulator.
- *Graphics Modes* allows the setting of the graphic display modes for tracing the constructive learning process. These include the display of training and test patterns, interpolation steps, points on separating hyperplanes, candidate and actual hyperplanes, as well as the final separating surface. An actual graphics

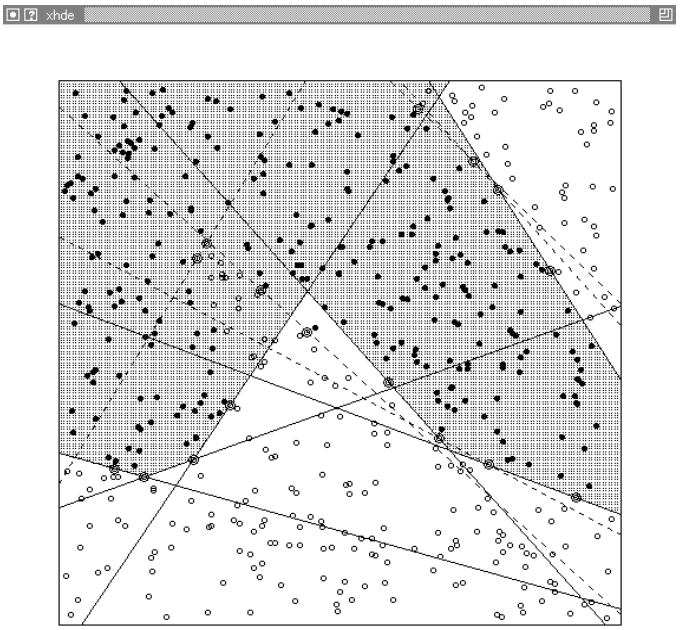

Figure 8: Graphics in HDE Simulation

screen resulting from a simulation in which all the graphics options, except the display of the test patterns are set, is presented in Fig. 8.

- *Network Switches* allows the setting of different additional HDE options such as creating hidden units and printing the network, the training and the test sets.
- *Advanced Options* allows the selection of two sophisticated features. The Hybrid System option incorporates prior knowledge to the HDE learning, provided in the form of an expert system [10]. The Parallel Computation option allows the parallel execution of the simulation on either a distributed system or on a highly parallel machine [11]. This also implies the existence of the p4 software package on the user's computer, which might not be installed on all systems, although it is available cost free from Argonne National Labs [3].

IV. Application in Research and Teaching

The primary goal of the GUI development was to assist both neural network professionals and novel users in performing neural network experimentations on heterogeneous simulators.

From the professionals' perspective, the two embedded neural network simulators were successfully used either directly or by means of the GUI in our lab for several research projects, covering various problems [6, 7, 8, 10, 11].

In summer 1995, the GUI was also used by neural network first time users during the WSU/NSF Teacher Institute for Science/Mathematics Education through Engineering Experiences training program. Two mathematics teachers who had no prior knowledge of both the UNIX operating system and neural networks spent 6 weeks in our lab with the objective of learning neural network techniques for designing backpropagation experiments. After getting accustomed with the GUI in a fast learning process, the results obtained by them on a few neural network benchmark classification problems like the Monk problems [20] and the breast cancer diagnosis problem [23] were comparable to the best known results for the corresponding problems. As a final result, they also developed a neural network teaching module to be used in high school.

Although the GUI and the embedded simulators have been successfully tested both in neural network research and training programs, a more extensive in class testing is needed. Hence, the GUI is intended for use in two courses offered in Fall '95 at the School of Electrical Engineering and Computer Science at Washington State University. One is an undergraduate course on Artificial Intelligence, whereas the other one is a graduate course on Neural Networks. From these two diverse groups of students we expect to gain valuable suggestions on future extensions to the current version of the GUI. Finally, any feedback from the readers of this article and users of the software provided on the accompanying CD-ROM would be extremely beneficial for improving the current GUI implementation and embedding additional neural network simulators.

ACKNOWLEDGEMENTS

We would like to thank LeeAnn Wagner and Bud Wright, participants in the 1995 WSU/NSF Teacher Institute for Science/Mathematics Education through Engineering Experiences program, who successfully completed the neural network training program, supporting our hypothesis that the current GUI implementation is an appropriate teaching tool for students with no prior experience in this domain. We also thank Profs. R. Zollars, D. Orlich, J. Petersen and W. Thomson, principal investigators for the NSF grant ESI-9254358, who approved Wagner and Wright's summer project in our lab and Prof. Jack Meador for his willingness to use our software in his neural networks course.

Finally, we thank Ioana Danciu for her constructive comments on a preliminary version of the manuscript.

References

[1] S. Bokhari, "The Linux Operating System," in *IEEE Computer*, pp. 74-79, August 1995.

[2] J. Bromley et al., "Signature Verification Using a Siamese Time Delay Neural Network," in *Advances in Neural Information Processing Systems*, vol.6, pp. 737-744, 1994.

[3] R. Butler, E. Lusk, "User's Guide to the p4 Parallel Programming System," *Technical Report ANL-92117*, Argonne National Laboratory, Argonne, IL, 1992.

[4] T. Chenoweth and Z. Obradovic, "A Multi-Component Nonlinear Prediction System for the SP 500 Index," in *Neurocomputing Journal* (in press).

[5] DARPA, "DARPA Neural Network Study, October 1987-February 1988," *AFCEA International Press*, Fairfax, VA, 1988.

[6] R. Drossu et al., "Single and Multiple Frame Video Traffic Prediction Using Neural Network Models," in *Computer Networks, Architecture and Applications*, S. V. Raghavan and B. N. Jain eds., Chapman & Hall, pp. 146-158, 1995.

[7] R. Drossu, Z. Obradovic, "Stochastic Modelling Hints for Neural Network Prediction," in *World Congress on Neural Networks*, Washington D.C., vol. 2, pp. 16-19, 1995.

[8] R. Drossu, Z. Obradovic, "Novel Results on Stochastic Modelling Hints for Neural Network Prediction," in *World Congress on Neural Networks*, Washington D.C., vol. 3, pp. 230-233, 1995.

[9] J. D. Foley et al., "Computer Graphics: Principles and Practice. Second Edition," *Addison-Wesley*, 1993.

[10] J. Fletcher and Z. Obradovic, "Combining Prior Symbolic Knowledge and Constructive Neural Networks," in *Connection Science: Journal of Neural Computing, Artificial Intelligence and Cognitive Research*, vol. 5, nos. 3-4, pp. 365-375, 1993.

[11] J. Fletcher and Z. Obradovic, "A Discrete Approach to Constructive Neural Network Learning," in *Neural, Parallel and Scientific Computations* (in press).

[12] S. Haykin, "Neural Networks: A Comprehensive Foundation," *MacMillan Publishing Company*, 1994.

[13] J. Hertz et al., "Introduction to the Theory of Neural Computation," *Addison-Wesley*, 1991.

[14] M. Mangeas, A. S. Weigend, "Forecasting Electricity Demand Using Nonlinear Mixture of Experts," in *World Congress on Neural Networks*, Washington D.C., 1995, vol. 2, pp. 48-53.

[15] O. Matan et al., "Multi-Digit Recognition Using a Space Displacement Neural Network," in *Advances in Neural Information Processing Systems*, vol. 4, pp. 488-495, 1992.

[16] J. K. Ousterhout, "Tcl and the Tk Toolkit," *Addison-Wesley*, 1994.

[17] D. E. Rummelhart et al., "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing*, vol. 1, MIT Press, 1986, pp. 318-362.

[18] San Diego Supercomputer Center, "Parallel User Guide," San Diego, CA, 1993.

[19] M. G. Sobell, "UNIX System V: A Practical Guide. Third Edition," *Benjamin Cummings*, 1995.

[20] S.B. Thrun et al. "The MONK's Problems: A Performance Comparison of Different Learning Algorithms," *Technical Report*, Department of Computer Science, Carnegie Mellon University, 1991.

[21] Venkateswaran, R. and Obradovic, Z. "Efficient Learning through Cooperation," in *World Congress on Neural Networks*, San Diego, CA, vol. 3, pp. 390-395, 1994.

[22] M. Welsh, L. Kaufman, "Running LINUX," *O'Reilly and Associates*, Sebastopol, California, 1995.

[23] W. H. Wolberg and O. L. Mangasarian, "Multisurface Method of Pattern Separation for Medical Diagnosis Applied to Breast Cytology," in *Proc. of the National Academy of Sciences*, U.S.A., vol. 87, pp 9193-9196, 1990.