UNIVERSITY OF TÜBINGEN

WILHELM-SCHICKARD-INSTITUTE
FOR COMPUTER SCIENCE

Department of Computer Architecture

JavaNNS

Java Neural Network Simulator

User Manual, Version 1.0 beta

Igor Fischer, Fabian Hennecke, Andreas Zell

Contents

1. Introduction

Java Neural Network Simulator (JavaNNS) is a simulator for neural networks developed at the Wilhelm-Schickard-Institute for Computer Science (WSI) in Tübingen, Germany. It is based on the Stuttgart Neural Network Simulator (SNNS) 4.2 kernel, with a new graphical user interface written in Java set on top of it. As a consequence, the capabilities of JavaNNS are mostly equal to the capabilities of the SNNS, whereas the user interface has been newly designed and -- so we hope -- become easier and more intuitive to use. Some complex, but not very often used features of the SNNS (e.g. three-dimensional display of neural networks) have been left out or postponed for a later version, whereas some new, like the log panel, have been introduced.

Besides the new user interface, a big advantage of JavaNNS is its increased platform independence. Whereas SNNS was developed with primarily Unix workstations in mind, JavaNNS also runs on PCs, provided that the Java Runtime Environment is installed. As of writing of this manual JavaNNS has been tested on:

- Windows NT
- Windows 2000
- RedHat Linux 6.1
- Solaris 7

with more to follow soon.

1.1. How to read this manual

Because of large similarities between SNNS and JavaNNS, this manual covers only the differences between the two. It should be read as a companion to the SNNS User Manual, available from the WSI web site:

`http://www-ra.informatik.uni-tuebingen.de/SNNS/`

We suggest that you first read the SNNS Manual, in order to become acquainted with the theory of neural networks, the way they are implemented in the SNNS kernel and to get a basic idea of the SNNS graphical user interface. If you are already familiar with SNNS, you can skip this step and start directly with this manual.

In the next chapter, you will find the license agreement. Please read it carefully and make sure that it is acceptable for you before installing and using JavaNNS. The installation process differs slightly for Windows and Unix machines and is described separately for each case. After installing, we suggest that you follow our quick tour through the simulator to get the first impression of how it is organized and used. The rest of the manual covers in more detail creating, manipulation and analyzing neural networks. You can skim it in the first reading and use it later as a reference.

2. Licensing and Acknowledgements

JavaNNS is Copyright (c) 1996-2001 JavaNNS Group, Wilhelm-Schickard-Institute for Computer Science (WSI), University of Tübingen, Sand 1, 72076 Tübingen, Germany. It uses the kernel of SNNS (Stuttgart Neural Network Simulator), which is Copyright (c) 1990-95 SNNS Group, Institute for Parallel and Distributed High-Performance Systems (IPVR), University of Stuttgart, Breitwiesenstrasse 20-22, 70565 Stuttgart, Germany.

Currently, JavaNNS is distributed by the University of Tübingen and only as a binary. Although it is distributed free of charge, please note that it is NOT PUBLIC DOMAIN.

The JavaNNS License is gives you the freedom to give away verbatim copies of the JavaNNS distribution (which include the license). We do not allow modified copies of our software or software derived from it to be distributed. You may, however, distribute your modifications as separate files along with our unmodified JavaNNS software. We encourage users to send changes and improvements which would benefit many other users to us so that all users may receive these improvements in a later version. The restriction not to distribute modified copies is also useful to prevent bug reports from someone else's modifications.

For our protection, we want to make certain that everyone understands that there is NO WARRANTY OF ANY KIND for the JavaNNS software.

2.1. License Agreement

1. This License Agreement applies to the JavaNNS program and all accompanying programs and files that are distributed with a notice placed by the copyright holder saying it may be distributed under the terms of the JavaNNS License. 'JavaNNS', below, refers to any such program or work, and a 'work based on JavaNNS' means either JavaNNS or any work containing JavaNNS or a portion of it, either verbatim or with modifications. Each licensee is addressed as 'you'.
2. You may copy and distribute verbatim copies of the JavaNNS distribution as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty, keep intact all the notices that refer to this License and to the absence of any warranty, and give any other recipients of JavaNNS a copy of this license along with JavaNNS.
3. You may modify your copy or copies of JavaNNS or any portion of it only for your own use. You may not distribute modified copies of JavaNNS. You may, however, distribute your modifications as separate files (e.,g. new network or pattern files) along with our unmodified JavaNNS software. We also encourage users to send changes and improvements which would benefit many other users to us so that all users may receive these improvements in a later version. The restriction not to distribute modified copies is also useful to prevent bug reports from someone else's modifications.
4. If you distribute copies of JavaNNS you may not charge anything except the cost for the media and a fair estimate of the costs of computer time or network time directly attributable to the copying.
5. You may not copy, modify, sublicense, distribute or transfer JavaNNS except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, distribute or transfer JavaNNS is void, and will automatically terminate your rights to use JavaNNS under this License. However, parties who have received copies, or rights to use copies, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

6. By copying, distributing or modifying JavaNNS (or any work based on javaNNS) you indicate your acceptance of this license to do so, and all its terms and conditions.
7. Each time you redistribute JavaNNS (or any work based on JavaNNS), the recipient automatically receives a license from the original licensor to copy, distribute or modify JavaNNS subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein.
8. Because JavaNNS is licensed free of charge, there is no warranty for it, to the extent permitted by applicable law. The copyright holders and/or other parties provide JavaNNS 'as is' without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of JavaNNS is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.
9. In no event will any copyright holder, or any other party who may redistribute JavaNNS as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use JavaNNS (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of JavaNNS to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

2.2. Acknowledgments

JavaNNS is a joint effort of a large number of people, computer science students, research assistants as well as faculty members at the Institute for Parallel and Distributed High Performance Systems (IPVR) at University of Stuttgart, the Wilhelm Schickard Institute of Computer Science at the University of Tübingen, and the European Particle Research Lab CERN in Geneva.

The project to develop an efficient and portable neural network simulator which later became SNNS was lead since 1989 by Dr. Andreas Zell, who designed SNNS and acted as advisor for more than two dozen independent research and Master's thesis projects that made up SNNS, JavaNNS and some of its applications. Over time the source grew to a total size of now 5MB in 160.000+ lines of code. Research began under the supervision of Prof. Dr. Andreas Reuter and Prof. Dr. Paul Levi. We are grateful for their support and for providing us with the necessary computer and network equipment.

The following persons were directly involved in the SNNS project. They are listed in the order in which they joined the SNNS team.

Table 1: SNNS and JavaNNS project members

Andreas Zell	Design of the SNNS simulator, SNNS project team leader.
Niels Mache	SNNS simulator kernel (really the heart of SNNS), parallel SNNS kernel on MasPar MP-1216.
Tilman Sommer	original version of the graphical user interface XGUI with integrated network editor, PostScript printing.
Ralf Hübner	SNNS simulator 3D graphical user interface, user interface development (version 2.0 to 3.0).

Table 1: SNNS and JavaNNS project members

Thomas Korb	SNNS network compiler and network description language Nessus.
Michael Vogt	Radial Basis Functions. Together with Günter Mamier implementation of Time Delay Networks. Definition of the new pattern format.
Günter Mamier	SNNS visualization and analyzing tools. Implementation of the batch execution capability. Together with Michael Vogt implementation of the new pattern handling. Compilation and continuous update of the user manual. Maintenance of the ftp server. Bugfixes and installation of external contributions.
Michael Schmalzl	SNNS network creation tool Bignet, implementation of Cascade Correlation, and printed character recognition with SNNS.
Kai-Uwe Herrmann	ART models ART1, ART2, ARTMAP and modification of the BigNet tool.
Artemis Hatzigeorgiou	documentation about the SNNS project, learning procedure Backpercolation 1.
Dietmar Posselt	ANSI-C translation of SNNS.
Sven Döring	ANSI-C translation of SNNS and source code maintenance. Implementation of distributed kernel for workstation clusters.
Tobias Soyez	Jordan and Elman networks, implementation of the network analyzer.
Tobias Schreiner	Network pruning algorithms.
Bernward Kett	Redesign of C-code generator snns2c.
Jens Wieland	Design and implementation of batchman.
Jürgen Gatter	Implementation of TACOMA and some modifications of Cascade Correlation.
Igor Fischer	Java user interface design and development.
Fabian Hennecke	Java user interface development.

There are a number of important external contributions by: Martin Reczko, Martin Riedmiller Mark Seemann, Marcus Ritt, Jamie DeCoster Jochen Biedermann, Joachim Danz, Christian Wehrfritz Randolf Werner, Michael Berthold and Bruno Orsier.

3. Installation

To be able to use JavaNNS, you have to have Java Runtime Environment (or JDK, which contains it) installed. JavaNNS has been tested to work with Java 1.2.2, but we recommend Java 1.3 for better performance and appearance.

The installation process differs slightly for Windows and Unix machines, therefore we describe it separately.

3.1. Windows Installation

JavaNNS is distributed as the zip file JavaNNS-Win.zip. Unzip the file into a folder of your choice. You should get:

1. JavaNNS.jar - the Java archive file containing the JavaNNS user interface classes
2. SNNS_jkr.dll - the SNNS kernel shared library
3. JavaNNSini.html - JavaNNS initialization file
4. JavaNNS.bat - batch file for starting the JavaNNS
5. examples - folder with example networks, patterns etc.
6. manual - folder containing this manual

In order to make JavaNNS functional, make sure that the kernel library SNNS_jkr.dll is placed in your path. There are two ways of doing it:

1. Moving the file into a folder that is already in the path (e.g. Windows\system32; or
2. Setting the Path system variable through the Control Panel to point to the JavaNNS folder.

3.2. Unix Installation

JavaNNS is distributed in gzipped tar archives for different operating systems, like JavaNNS-LinuxIntel.tar.gz and JavaNNS-Solaris.tar.gz. Unpack the archive into a directory of your choice. You should get:

1. JavaNNS.jar - the Java archive file containing the JavaNNS user interface classes
2. libSNNS_jkr.so - the SNNS kernel shared library
3. JavaNNSini.html - JavaNNS initialization file
4. examples - directory with example networks, patterns etc.
5. manual - directory containing this manual

In order to make JavaNNS functional, make sure that the kernel library (libSNNS_jkr.so) is placed in your library path. There are two ways of doing it:

1. Moving the file into a directory that is already in the library path. You can check the LD_LIBRARY_PATH environment variable to see the directories in the path; or
2. Setting the LD_LIBRARY_PATH environment variable, e.g.

```
setenv LD_LIBRARY_PATH mylibrarypath
```

3.3. Setting up JavaNNS

Settings for JavaNNS are stored in the initialization file:

```
JavaNNSini.html
```

located in the same directory (folder) as other JavaNNS files. It is a HTML file, containing only HTML links. It's structure is simple: each link represents a JavaNNS variable, with the HREF part representing the variable value, and the text its name. For example, the link

```
<a href=
  "file:///JNNS/manual/JavaNNS-manual.html">User Manual URL</a>
```

states that the HTML version of the user manual is to be found in the file JavaNNS-manual.html, placed in the directory (folder) JNNS/manual. In this link, “User Manual URL” is the variable name, and “file:///JNNS/manual/JavaNNS-manual.html” its value.

You can use an ordinary text editor for editing JavaNNSini.html.

In order to be able to access the manual from JavaNNS, you should set the value of this variable according to the path where you installed JavaNNS. Also, because JavaNNS uses a Web browser for displaying help, you should set the variable “Browser name” to point to the browser you wish to use. Use naming conventions common for your operating systems (i.e. normal slash (/) for Unix and backslash (\) for Windows) to specify the path.

You can also set the initial width and height of the JavaNNS screen by adjusting the corresponding variables.

3.4. Running JavaNNS

That's all! Now you can start JavaNNS by typing:

```
java -jar JavaNNS.jar
```

from the command prompt. If you are running Windows, you can also click the JavaNNS.bat batch file from Windows Explorer.

4. A Quick Tour of JavaNNS

JavaNNS is a simulator for artificial neural networks, i.e. computational models inspired by biological neural networks. It enables you to use predefined networks or create your own, to train and to analyze them. If any of these terms is unknown to you, please refer to a book about neural networks or to the SNNS User Manual - this manual describes only the usage of JavaNNS.

4.1. Starting JavaNNS

To begin the tour, let's start JavaNNS, as described in "Installing": type java -jar JavaNNS.jar or, if using Windows, click the JavaNNS.bat file. After starting the program, its main window opens. As we have started the program no parameters in the command line, the window is empty, containing only the usual menu bar. Also, no network files have been loaded.

4.2. Loading Files

Use File/Open menu to open an example file: navigate to the examples directory and open the files xor_untrained.net, and xor.pat - a simple network and a corresponding pattern file.

4.3. View Network

The main window still remains empty, so choose View/Network to display the network. You should see a new window appearing, schematically showing a network, consisting of 4 units (neurons) and links between them, in its main part. Neurons and links have different colors, representing different values of unit activations and link weights. The colored bar on the left edge of the window shows which color corresponds to which value and can be used as reminder. The colors - and appearance in general - can be adjusted through View/Display Settings, which corresponds to the Display/Setup window in SNNS.

4.4. Training Network

Let us now train the network - reprogram its weights, so that it gives the desired output when presented an input pattern. For that purpose, open the Control Panel in the Tools menu. The Control Panel is, as in the SNNS, the most important window in the simulator, because almost all modifications and manipulations of the network are done through it. We shall also open the Error Graph window, in order to watch the training progress. Finally, to receive some textual and numerical information, we can open the Log window. Both are accessible through the View menu.

A sample screen shot with the windows open is shown in Figure 1.

The Control Panel is, contrary to the one in SNNS, divided into six tabs, each containing controls for specific purpose. For this introduction, let us switch directly to the learning tab. Here, the user can choose the learning function, set its parameters, number of learning cycles and update steps and finally perform network initialization and learning. The classic Backpropagation (equals Std_Backpropagation in SNNS) is the default learning function. As you can see, for each learning function default parameters are provided.

Learning is performed by pressing one of the buttons: "Learn current" which performs training with the currently selected pattern, and "Learn all", which trains the network with all patterns from the pattern set. During learning, the error graph displays the error curve - the type of error to be drawn is set on the left edge of the window. The error is also written into the log window.

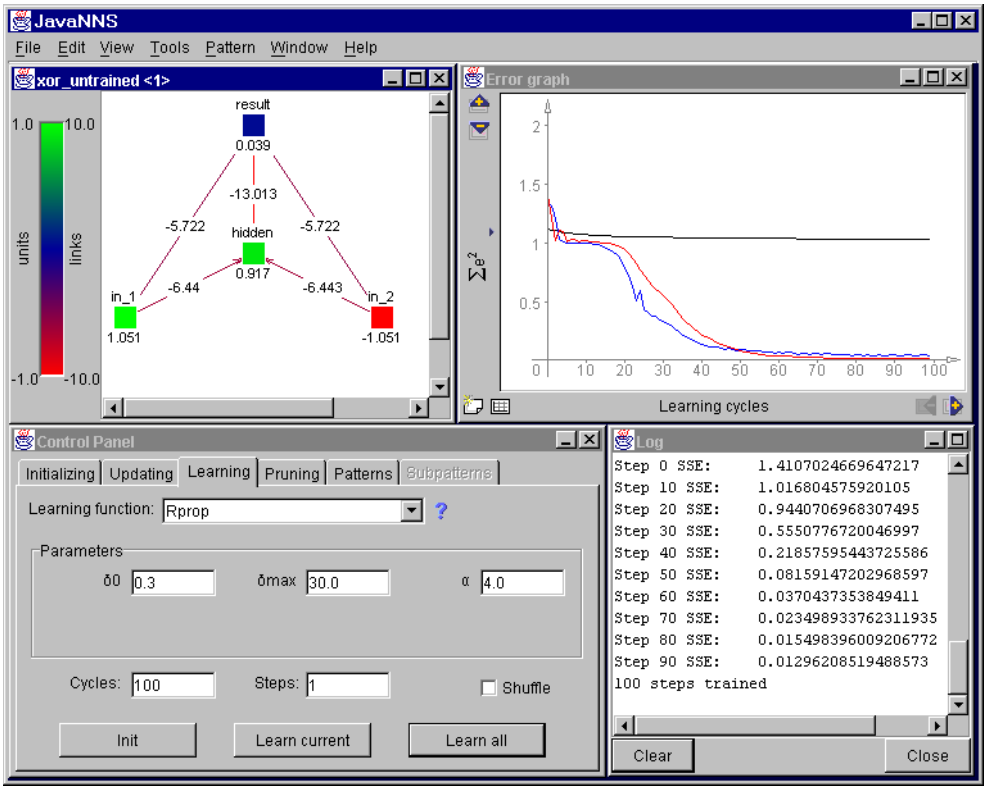

Figure 1: JavaNNS with XOR network, error graph, control and log panel

4.5. Analyzing Network

For analyzing the network and its performance, tools like Analyzer (in the Tools menu) and Projection (in the View menu), already familiar to SNNS users, can be used. For Projection, two input units and a hidden or output unit have to be selected in order for the menu item to become enabled. The Projection Panel than displays the activation of the hidden or output unit as a function of the two inputs. The activation is represented by color, so a colored rectangle is obtained. Analyzer is used to show output or activation of a unit as a function of other unit's activation or of the input pattern. Its usage is similar to the Analyze panel in the SNNS.

4.6. Creating a Network

Now let's create a network of our own. Choose File/New to remove the current network from the simulator. Then, choose Create/Layers from the Tools menu. A window resembling the Bignet tool of SNNS appears. Choose width and height "1", unit type "Input" and click "Create" to create a new layer. For the next layer, set height to five and the unit type to "Hidden" and click "Create" again. Finally, create the output layer with the height of one and unit type "Output" and close the window. To connect the created units, use Create/Connections from the Tools menu. Simply choose "Connect feed-forward" and click "Connect". Doing that, you have created a simple feed-forward neural network, with one input, five hidden and one output unit. You can now close the Connections window, too.

4.7. Graphical Network Display

You can arrange units on the display manually, by clicking and dragging them with the mouse. In fact, clicking a unit selects it, and dragging moves all selected units. To deselect a unit, press the CTRL-Key on the keyboard and click it while still holding the key pressed. Using

View/View Settings, tab Units and Links, you can choose what to display above and under each unit. Make sure that "Name" is selected as top label. Since the units have just been created, they are all called "noName". To change the names, choose "Names" from the Edit menu. The top labels turn to text fields. Use the mouse to place the caret into each one and enter some names. After you have finished, press "Enter" or click in an empty are of the display to turn the text fields to labels again.

4.8. Training and Validation Pattern Sets

To see how two pattern sets can be used for training and validation, load two pattern sets from "examples" directory: trainMAP.pat and validMAP.pat. In the Control Panel, tab "Patterns", select trainMap as the training set and validMAP as the validation set. Switch back to the "Learning" tab and train the network. During training two curves are displayed in the Error Graph: one, who's color depends on the number of already displayed curves and which represents the error of the training set, and the other, pink one, which represents the error of the validation set. The validation set is normally used to avoid overtraining of a network. For more information refer to the SNNS User Manual and other neural networks literature.

5. Network Creation and Editing

5.1. Network View and Display Settings

Although not necessary, it is recommended that a network view is open when creating and editing networks. Network view is opened through View/ Network menu. The network view displays a visual representation of the network, which comprises of units and connections (links) between them. Units are drawn as colored squares with 16 pixels side length, and connections as colored lines. For both units and links the color represents a value: activation for units and weight for links. The colored bar on the left edge of the network view serves as a quick reminder for color-to-value correspondence. (Figure 2)

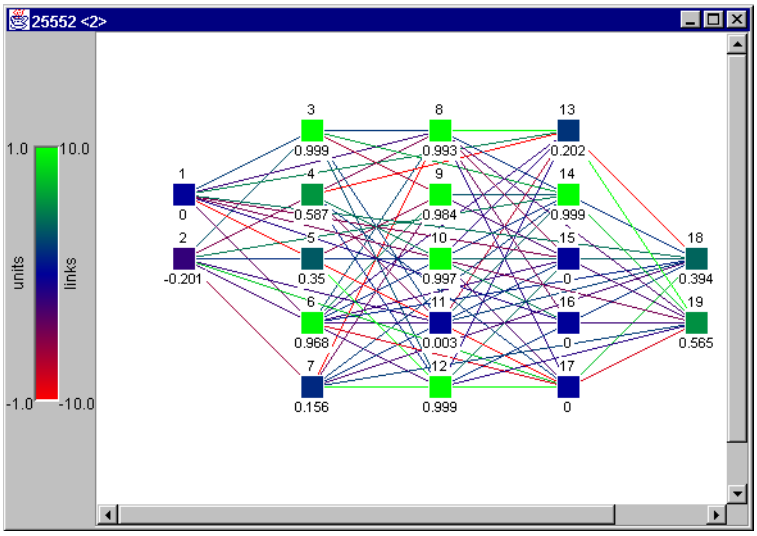

Figure 2: Network view

Units are placed along an invisible grid in the network view. Optionally, above and below each unit diverse unit properties can be displayed. Which ones, as well as grid size, chroma coding for units and links and some more data are set in the Display Settings panel, accessible from the View menu. This panel corresponds to the Display/Setup panel in SNNS.

Display Settings comprise of two tabs: General and Units&Links (or SOM for Kohonen tool). In tab General, grid size (in pixels), subnet number and chroma codes for different values can be set. In Units&Links, the user can set which properties, like name, unit activation etc. are to be shown above (Top label) and below (Base label) of each unit. Also, the user can decide if the connections are to be shown, if their weights are to be displayed numerically and if the direction arrows should be drawn. Sliders "Maximum expected value" and "Maximum expected weight" control the chroma coding for units and links, since they determine which value corresponds to the full color, as set in the "General" tab. Minimum values are simply taken to be negatives of the maximums, and for values between the color is interpolated. The Slider "Weakest visible link" is self-explanatory and helps keeping the network view more comprehensible.

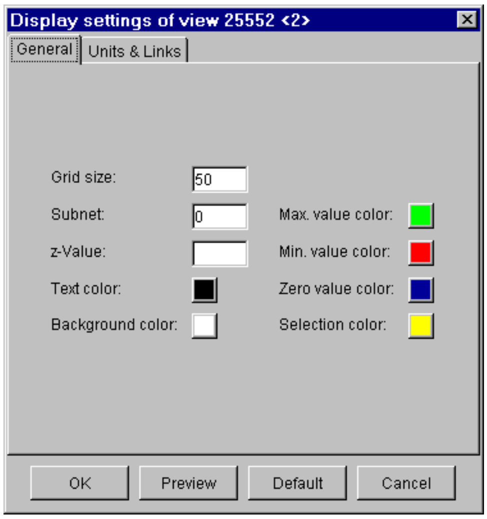

Figure 3: Display Settings - General

Since more than one network view can be open at the same time, Display Settings refer to the currently selected one.

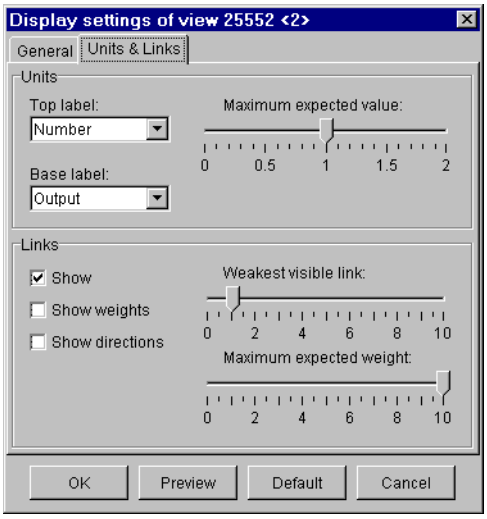

Figure 4: Display Settings - Units and Links

5.2. Tools for Creating Networks

Networks are created using two tools from the "Tools" menu, both from the "Create" submenu: "Layers" and "Connections". They together correspond to the "Bignet" tool in SNNS.

In JavaNNS layer has a different meaning as in the SNNS. In JavaNNS, layer corresponds to a physical layer of units that is being created. When creating layers, width and height determine the number of units in horizontal and vertical direction for the layer. Top left position is updated automatically, but can also be entered manually, and controls the layer's position in the display area. For all the data - width, height and coordinates of the top left position - the measuring unit is "grid size unit", which is set in the View/Display Settings panel.

In "Unit detail" segment of the window, the unit type (e.g. Input or Hidden), activation function of the units (Logistic by default), output function of the units (Identity by default), the layer number and the subnet number are set.

The "Connections" window provides for creating links (connections) between units. Three different ways are possible for creating links: by manually selecting units to connect ("Connect selected units"), by automatically connecting the whole network in a feed-forward style ("Connect feed-forward") and by interconnecting those in the same layer (Auto-associative). In case of feed-forward networks, shortcut connections (links connecting units form non-adjacent layers) can optionally be created. For auto-associative networks, self-connections (feedback connections from the output to the input of a same unit) can be allowed.

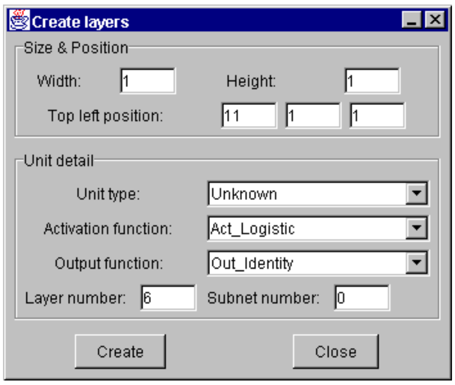

Figure 5: Create Layers tool

Except for automatic generation of feed-forward connections, the user has to select units to be connected. Units are selected using the mouse, either by clicking each unit, or by clicking the mouse and dragging a rectangle around units to be selected. Units are deselected by clicking them while holding the CTRL key pressed. A simple click in an empty area in a network view deselects all units.

Connecting selected units is a two-step process. In the first step, the user selects units where the connections are to originate (source units) and presses the button "Mark selected units as source". In the second step, the user selects the receiving units (targets) and presses the button, which is now labeled "Connect source with selected units". For auto-associative connections it suffices to select the desired units and press the "Connect selected units" button.

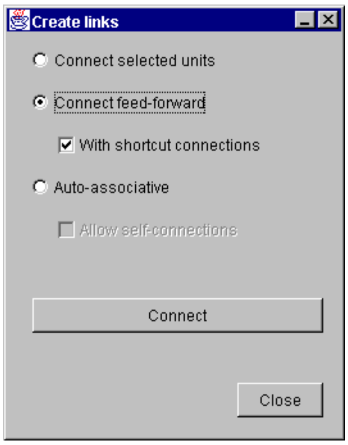

Figure 6: Create Links tool

Selected units can be dragged with the mouse in order to change their positions.

5.3. Editing Units

Existing units can be edited by selecting them and then choosing Unit Properties from the Edit menu or Edit Units in the context-sensitive menu, which is accessed by pressing the right mouse button while over a unit. An extra window appears, displaying all editable unit properties, like name, type, activation etc. This method allows only for setting the same values for all selected units. Alternatively, the user can edit values displayed as top and base labels of each unit individually. For that purpose, the user has to choose from the Edit menu which property he or she wants to edit. The labels displaying the property turn to entry fields, which can now be edited. The fields are selected by the mouse and can be traversed by pressing the Tab key. Pressing Enter accepts changes and turns the fields back to labels.

Changing activation values of units is useful if patterns are created manually.

6. Pattern Management

Like in SNNS, patterns are organized in pattern sets, which are stored as text files. They can be loaded using the Open option and saved using "Save data" (not Save!) from the File menu. Further manipulation is primarily performed from the Control Panel (accessible from the Tools menu), in the "Patterns" tab. Some simple manipulations (adding, modifying, deleting) can be also performed from the Patterns menu in the main menu bar.

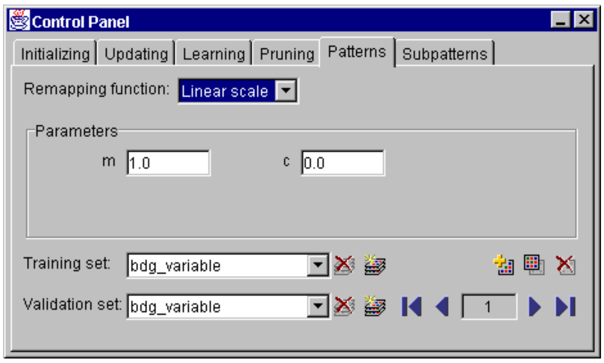

Figure 7: Control Panel - Patterns

In the Control Panel, a pattern remapping function and its parameters can be selected. The two combo boxes - Training set and Validation set - are used for selecting the active training and validation set, respectively. Also, when new pattern sets are created (by pressing the second button next to each of the combo boxes), the corresponding combo box becomes editable, so that the new pattern set can be given a name. The other button, adjacent to the combo box, deletes the current pattern set from the memory.

Near the right edge of the panel, in the pre-last row, three more buttons serve for modifying the current pattern set. Their function, from left to right, is: add, copy and delete pattern. Add creates a new pattern from current input and output unit activations and adds it to the current pattern set. Copy creates a new pattern, which is a verbatim copy of the currently selected one, and adds it to the pattern set. Finally, the delete button deletes the currently selected pattern.

The current pattern is identified by its ordinal number in the pattern set. This number is displayed in a text field between arrow buttons in the bottom right corner of the panel. The arrow buttons provide for navigating through the patterns in the currently selected set.

Some patterns can contain subpatterns of variable length. In that case, the tab "Subpatterns" is enabled and provides for defining size and shape of subpatterns, as well as for navigating through them. This corresponds to the Subpattern window in SNNS.

Propagating patterns through the network is done in the Update tab of the Control Panel. Same navigational controls are provided as in the Patterns tab. Besides, the button between the arrows propagates the current pattern through the network.

The same panel is also used for selecting the updating function and its parameters to be used in training.

7. Training and Pruning Networks

Training is also performed through the Control Panel. In the Initializing tab, an initialization function and its parameters can be set. The Init button (also available in the Learning tab) performs the initialization.

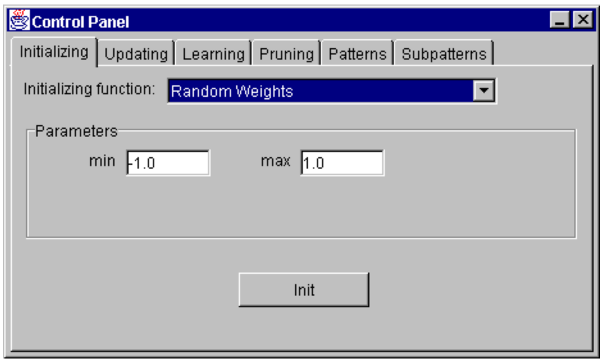

Figure 8: Control Panel - Init

Under the Learning tab, the user can choose the learning function, set its parameters, number of learning cycles and update steps and finally perform network initialization and learning. The classic Backpropagation (equals Std_Backpropagation in SNNS) is the default learning function. For each learning function default parameters are provided.

The "Learn current" button performs training with the currently selected pattern and "Learn all" with all patterns from the pattern set. In order to monitor learning progress, it is useful to open the Error Graph and/or Log window, both available from the View menu. During learning, the error graph displays the error curve. The type of the error to be drawn is set through the middle button located on the left edge of the window. The arrow buttons near the axes are used for scaling. The two buttons in the left bottom corner clear the error graph and toggle grid, respectively.

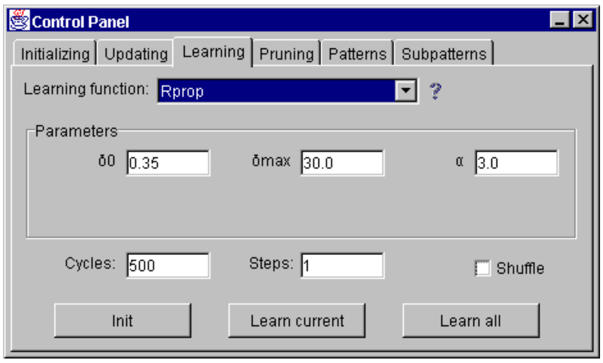

Figure 9: Control Panel - Learning

During training, the error is also written into the log window. Also, many other useful information about the network are written there on diverse occasions. The log window corresponds roughly to the command shell window from which SNNS is started in a Unix system.

Options and controls for pruning networks are found under the Pruning tab in the Control Panel. Its contents corresponds mostly to the Pruning window in SNNS. However, contrary to the SNNS, the user does not have to set the learning function to PruningFeedForward. In JavaNNS it is done automatically and transparently for him/her. The learning function, as set under the Learning tab, as well as number of cycles, correspond to the data entered in "General parameters for Training" section of the SNNS' Pruning window. In JavaNNS, pruning is performed by pressing the Prune button.

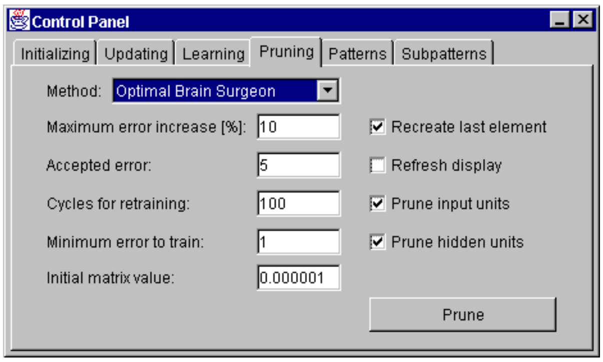

Figure 10: Control Panel - Pruning

Cascade correlation and TACOMA learning are the only exceptions to the concept of the Control Panel being the central place for manipulating networks. Because of the large number of parameters needed by the two learning methods, a separate window, accessible from the Tools menu is used. Contrary to SNNS, where parameters for cascade correlation and TACOMA are dispersed between the Control Panel and the Cascade window, in JavaNNS the Cascade window alone covers all data and parameters needed for applying the two learning algorithms.

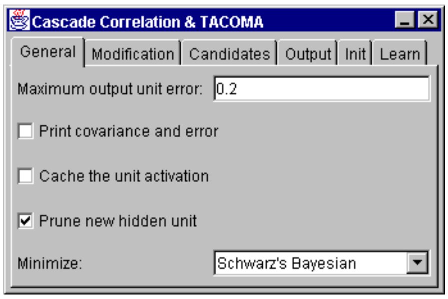

Figure 11: Cascade Correlation and TACOMA - General

The window is divided into six tabs. Tabs "General", "Modification" and "Learn" cover the parameters set in SNNS in the section "General Parameters for Cascade" of the Cascade window. Under the "Lear" tab of the JavaNNS Cascade window, the learning function, together with its parameters and the maximal number of cascades (hidden units generated during learning) are set. The "Init" tab is introduced for convenience and provides for initializing network. Tabs "Candidates" and "Output" correspond to "Parameters for Candidate Units" and "Parameters for Output Units" sections in the SNNS window. The default learning method invoked from the window is cascade correlation, TACOMA can be set as a modification under the corresponding tab.

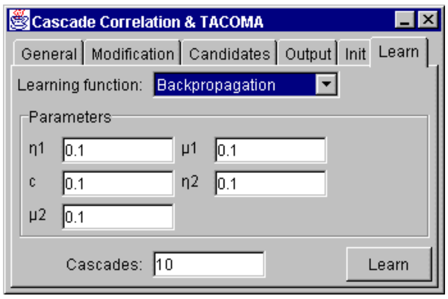

Figure 12: Cascade Correlation and TACOMA - Learning

8. Analyzing Networks

Weights and Projection panels, accessible through the View menu, and Analyzer, accessible from the Tools menu, can be used for getting insight into a network. All the panels correspond to their SNNS counterparts and differ only in appearance.

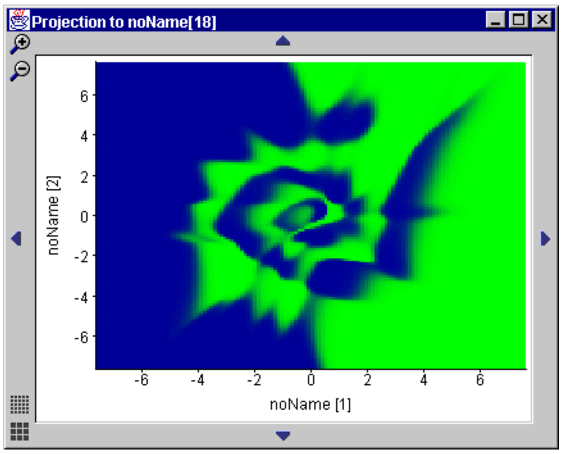

Figure 13: Projection panel

8.1. Projection Panel

The Projection panel shows activation of a hidden or output unit as a function of activations of two input units. The panel can only be opened when the three units are selected in a network view. The activations of the input units are drawn on the x- and y-axis, while corresponding activations of the output unit are represented by different pixel colors. For the chroma coding, the same values as for the network view apply.

Arrows at the panel edges are used for moving the projection window in the input space. The two buttons on in the top left corner are used for zooming, and the buttons in the bottom left corner for adjusting the view resolution. Zooming can also be performed manually, by dragging a rectangle in the projection area.

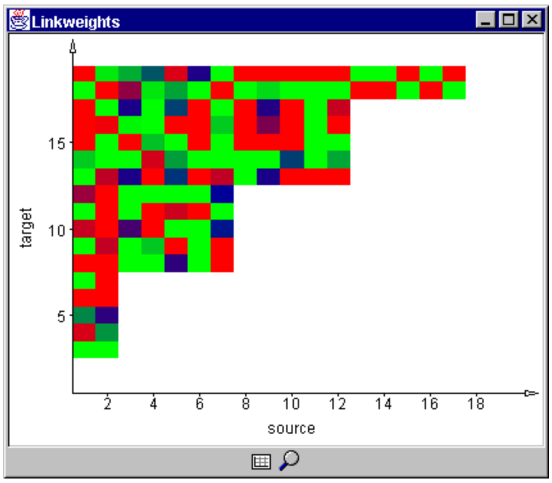

Figure 14: Weights panel

8.2. Weights Panel

The Weights panel presents link weights as colored rectangles. The x-axis is used for source units and the y-axis for target units of the links. The two buttons at the panel bottom are used for toggling grid and for auto zoom for optimal display. As in the projection panel, zooming can be performed manually.

8.3. Analyzer

The Analyzer is used to show output or activation of a unit as a function of other unit’s activation or of the input pattern. Its usage is similar to the Analyze panel in the SNNS. The control buttons are also familiar and have the same function as in the Error Graph, Projection and Weights panel.

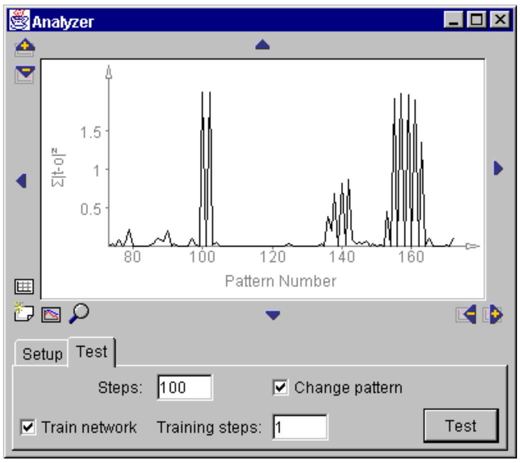

Figure 15: Analyzer panel

9. Loading, Saving and Printing

File loading, saving and printing of results is performed through the File menu. Whereas "Open" can be used for loading any type of file (network, pattern, text...), "Save", as well as "Save as" are used only for saving the current network. Other file types are saved through "Save data", by choosing the appropriate file type in the combo box at the bottom of the dialog window. For result files, additional options (start and end pattern, inclusion of input and output files and file creation mode) like in SNNS can be set.

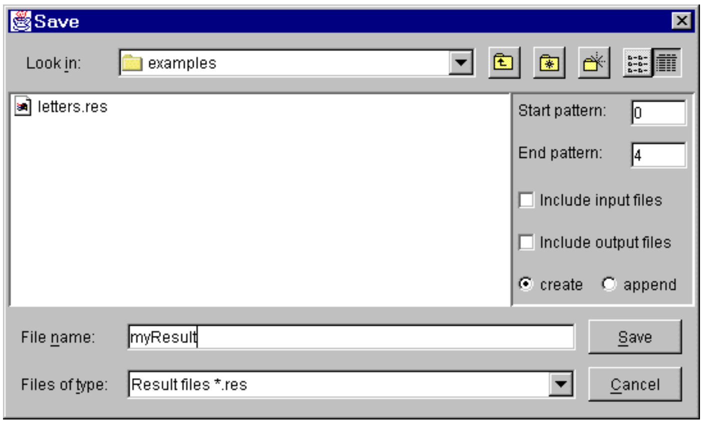

Figure 16: File Save dialog

When choosing files for loading in the file dialog window it is possible to select multiple files, even of different types. That way, the user can load a network, configuration and multiple pattern files in only one step. (This currently doesn't work in the Linux implementation)

Print always refers to the currently active window. Therefore, anything that can be displayed in JavaNNS can also be printed by making the desired window active (i.e. clicking it with the mouse) and choosing "Print" from the File menu.