After starting the package a banner will appear which will vanish after you click the left mouse button in the panel. You are then left with the SNNS manager panel.

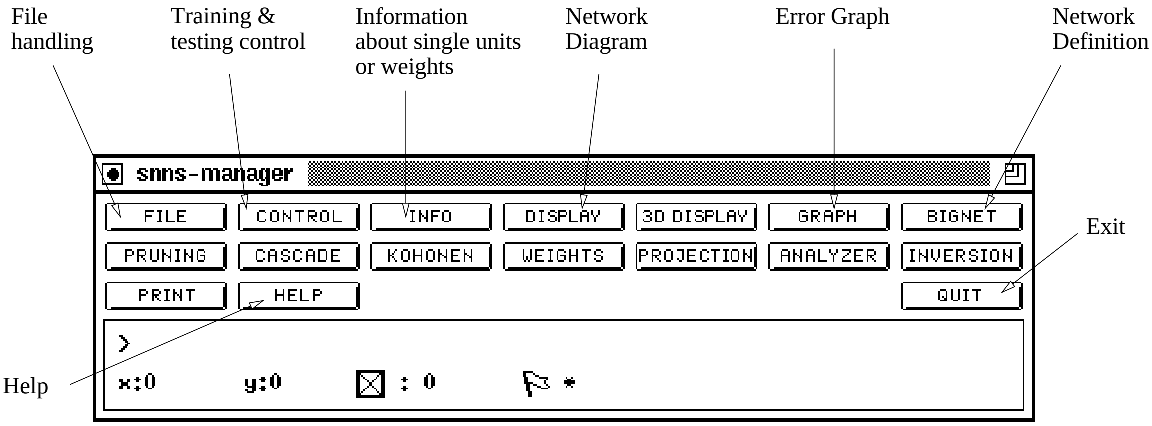

Figure 4.1: The SNNS manager panel

The SNNS manager allows you to access all functions offered by the package. It is a professional tool and you may find it a little intimidating. You will not need to use the majority of the options. You should read this introduction while running the simulator - the whole thing is quite intuitive and you will find your way around it very quickly.

4.1.2 Reading and Writing Files

SNNS supports five types of files, the most important ones are:

NET Network definition files containing information on network topology and learning rules. The files end in the extension '.net'.

PAT Pattern files, containing the training and test data. All pattern files end in '.pat'.

RES Results files. Network output is interpreted in many possible ways, depending on the problem. SNNS allows the user to dump the network outputs into a separate file for later analysis.

The final two file types are not important for a first exploration of SNNS.

The first thing you are likely to use is the FILE option in the manager panel to read network and pattern definition files. The window that will appear is given in figure 4.2.

The top text field shows the current directory. The main field shows all files for each of the file types that SNNS can read/write. Directories are marked by square brackets. To load an example network change the directory by entering the example directory path in the top field (do not press return):

`SNNSv4.1/examples`

Changes will only be apparent after one of the file selectors has been touched:click on PAT and then NET again. You should now see a list of all network definition files

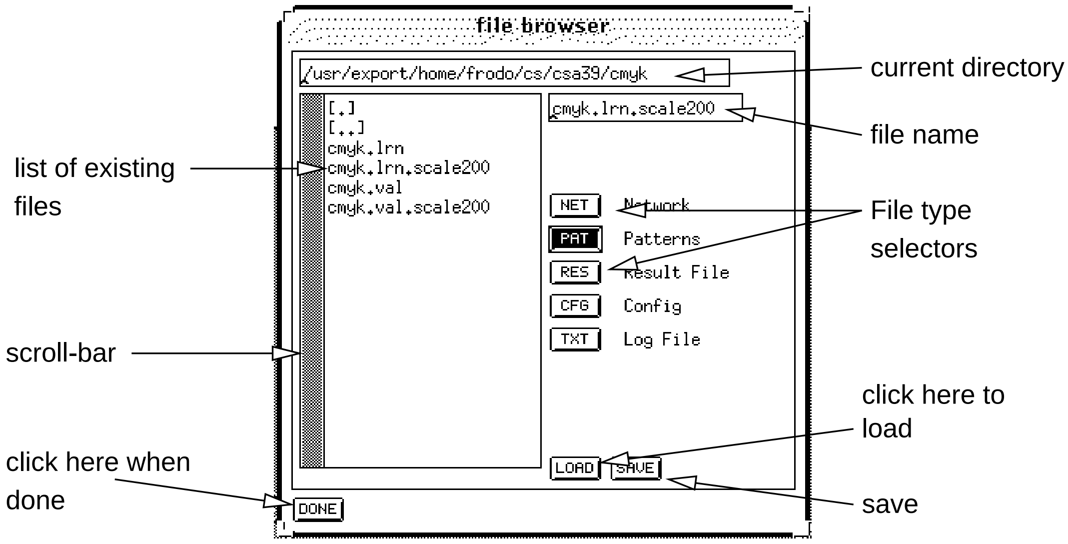

Figure 4.2: The SNNS file browser

currently available. Double-clicking on one of the filenames, say 'letters' will copy the network name into the file name window. To load the network simply click on $\boxed{\text{LOAD}}$. You can also enter the filename directly into the file name window (top left).

4.1.3 Creating New Networks

You will need to create your own networks. SNNS allows the creation of many different network types. Here is an example of how to create a conventional (fully connected) feed-forward network.

First select the FEEDFORWARD option, hidden under the $\boxed{\text{BIGNET}}$ button in the manager panel. You are then faced with the panel in figure 4.3. Only two parts of the panel are required.

The top allows the definition of the network topology, that is how many units are required in each layer and how they should appear if the network is displayed on the screen. The lower part allows you to fully connect the layers and to create the network.

Note that much of what you are defining here is purely cosmetic. The pattern files contain a given number of inputs and outputs, they have to match the network topology; how they are arranged in the display is immaterial.

First you have to define the input layer by filling the blanks in the top right hand corner (edit plane) of the panel. The panel shows the current settings for each group of units (a plane in SNNS terminology). Each group is of a given type (i.e. input, hidden or output), and each plane contains a number of units arranged in an x-y-z coordinate system. This is used for drawing networks only!

You can change the entries by entering values into the boxes or by clicking on $\boxed{\text{TYPE}}$ and

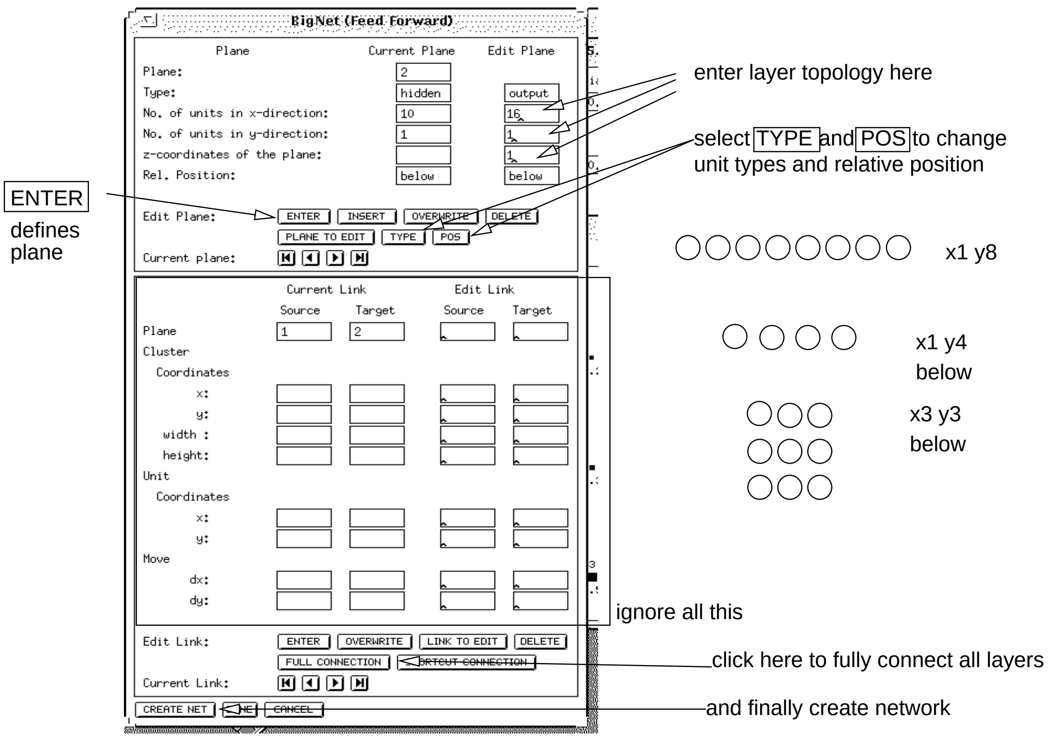

Figure 4.3: The SNNS BigNet Feedforward network designer panel

POS to change the unit type and relative position. The relative position is not used for the first plane of units (there is nothing to position it relatively to). The layers will, for instance, be positioned below the previous layers if the 'Rel. Position' has been changed to 'below' by clicking on the POS button.

Here is an example of how to create a simple pattern associator network with a 5x7 matrix of inputs inputs, 10 hidden units and 26 outputs:

```
Leave 'Type' as input
set no 'x' direction to 5
set no 'y' direction to 7
and click on \fbox{ENTER}
```

If the input is acceptable it will be copied to the column to the left. The next step is to define the hidden layer, containing 10 units, positioned to the right the inputs.

```
Change 'Type' from input to hidden by clicking on \fbox{TYPE} once.
set no 'x' direction to 1
set no 'y' direction to 10
change 'Rel.Position' to 'below' by clicking on \fbox{POS}
and click on \fbox{ENTER}
```

You are now ready to define the output plane, here you want 26 output units to the right of the input. You may want to save space and arrange the 26 outputs as two columns of

13 units each.

```
Change 'Type' from hidden to output by clicking on \fbox{TYPE} again.
set no 'x' direction to 2
set no 'y' direction to 13
and click on \fbox{ENTER}
```

After defining the layer topology the connections have to be made. Simply click on FULL CONNECTION (bottom left of lower panel). Then select CREATE NET and DONE. You may have to confirm the destruction of any network already present.

Selection of DISPLAY from the SNNS-manager panel should result in figure 4.4.

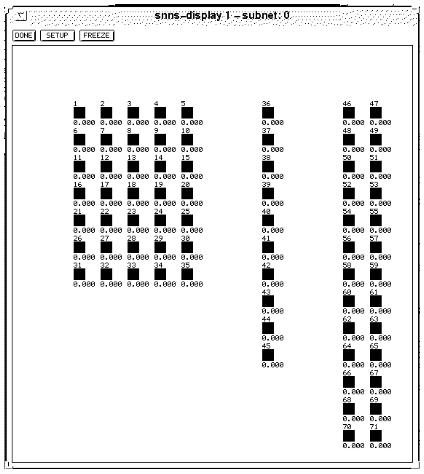

Figure 4.4: SNNS network display panel

The lines showing the weights are not normally visible, you have to switch them on by select SETUP, and then click on the ON button next to 'links' option. You will find that SNNS refuses any further input until you have selected DONE.

After creating the network and loading in the pattern file(s) - which have to fit the network topology - you can start training the net. The network you have just created should fit the 'letters' patterns in the SNNS examples directory.

An alternate way to construct a network is via the graphical network editor build into SNNS. It is best suited to alter an existing large network or to create a new small one. For the creation of large networks use bignet. The network editor is described in chapter 6.

4.1.4 Training Networks

Load the 'letters' pattern file (in the SNNS-`examples` directory) at this stage. The network is a pattern associator that can be trained to map an input image (5x7 pixel representation of letters) into output units where each letter is represented by an output unit.

All training and testing is done via the CONTROL option. I will discuss the most important features one by one. The panel consists of two parts. The top part controls the parameters defining the training process, the bottom three rows are blanks that have to be filled in to define the learning rates etc. and the range over which weights will be randomly distributed when the network is initialised. The defaults for the learning parameters are (0.2 0 0 0 0) while the default weight setting is between 1 and -1 (1.0 -1.0 0 0 0).

4.1.4.1 Initialisation

Many networks have to be initialised before they can be used. To do this click on INIT (top line of buttons in control). You can change the range of random numbers used in the initialisation by entering appropriate values into the fields to the right of 'INIT' at the lower end of the control panel.

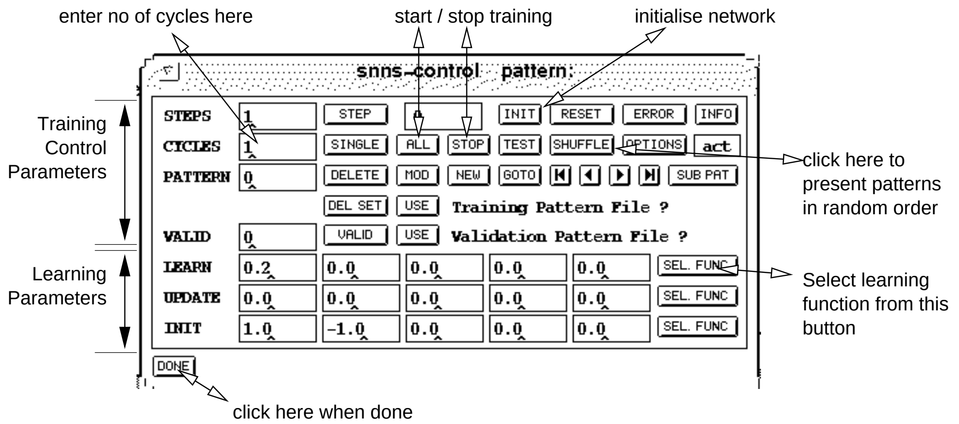

Figure 4.5: SNNS network training and testing control panel

4.1.4.2 Selecting a learning function

The default learning function for feed-forward nets is Std_Backpropagation, you may want something a little more extravagant. Simply click on SEL FUNC (Select function next to the learning parameters, see figure 4.5) and pick what you want to use. The routines you may want to consider are Std_Backpropagation, BackpropMomentum or Rprop). Use BackpropMomentum for the letters example.

Each learning function requires a different parameter set: here are the important ones, details are given in the manual:

Std_Backpropagation 1: α learning rate (0-1), 2: d_{max}, the maximum error that is tolerated. use 0 or a small value.

BackpropMomentum 1: α learning rate (0-1), 2: μ momentum term (0-0.99), 3: c flat spot elimination (ignore) and 4: d_{max} max ignored error.

Rprop 1: starting values of Δ_{ij} (0-0.2) 2: Δ_{max} maximum update value (30 works well..) 3: α the weight decay term as an exponent (5 works for most problems) $x = 10^{-\alpha} =$

0.00001.

Once all parameters are set you are ready to do some training. Training is done for a number of 'CYCLES' or epochs (enter a number, say 200 - see fig. 4.5). All training patterns are presented once during each cycle. It is sometimes preferable to select the patterns randomly for presentation rather than in order: Click on SHUFFLE to do this.

For the pattern associator example leave the learning rate at 0.2 and set the momentum term (second field) to 0.5; leave everything else at 0.

Before starting the learning process you may like to open a GRAPH panel (from manager) to monitor the progress during training.[1]

Click on ALL to start training and STOP to interrupt training at any time. The graph will start on the left whenever the network is initialised so that it is easy to compare different learning parameters. The current errors are also displayed on the screen so that they could be used in any graph plotting package (like xmgr).

It is impossible to judge the network performance from the training data alone. It is therefore sensible to load in a 'test' set once in a while to ensure that the net is not over-training and generalising correctly. There is no test set for the letters example. You can have up to 5 different data sets active at any one time. The two USE buttons on the control panel allow you to select which data sets to use for training and validation. The top button selects the training set, the bottom one the 'validation set'. If you enter a non-zero value into the box next to VALID a validation data set will be tested and the root-mean-square error will be plotted on the graph in red every N cycles (N is the number you entered in the box).

You can also step through all the patterns in a data set and, without updating any weight, calculate the output activations. To step through the patterns click on TEST.

You can go to any pattern in the training data set by either specifying the pattern number in the field next to 'PATTERN' and clicking on GOTO or by using the 'tape player controls' positioned to the right of GOTO. The outputs given by the network when stepping though the data are the targets, not the calculated outputs (!).

4.1.5 Saving Results for Testing

Network performance measures depend on the problem. If the network has to perform a classification task it is common to calculate the error as a percentage of correct classifications. It is possible to tolerate quite high errors in the output activations. If the network has to match a smooth function it may be most sensible to calculate the RMS error over all output units etc.

The most sensible way to progress is to save the output activations together with target values for the test data and to write a little program that does whatever testing is required.

[1] If you do this scale the y-range to lie between 0 and 26 by clicking on the 'right-arrow' next the 'Scale Y:' a few times. You can also resize the window containing the graph.

The RES files under FILE are just the ticket: Note that the output patterns are always saved. The 'include output patterns' actually means 'include target (!) patterns'.

4.1.6 Further Explorations

It is possible to visualise the weights by plotting them, just like the output values as boxes of different sizes/colour. Sometimes examining the weights gives helpful insights into how the networks work. Select WEIGHTS from the manager panel to see the weight diagram.

4.1.7 SNNS File Formats

4.1.7.1 Pattern files

To train a network on your own data you first have to massage the data into a format that SNNS can understand. Fortunately this is quite easy. SNNS data files have a header component and a data component. The header defines how many patterns the file contains as well as the dimensionality of the input and target vectors. The files are saved as ASCII test. An example is given in figure 4.6.

```
SNNS pattern definition file V3.2
generated at Wed Aug  9 11:01:29 1995


No. of patterns : 4772
No. of input units : 16
No. of output units : 1
```

File header

```
# 2
 0 2 2 0 -1 -1 0.91 0.91 0 1 0 0 0.01 0.01 0.34 -0.09 0.0
# 4
 1 2 2 1 -1 4 2.93 2.93 0 0.95 0 0 0 0 0 0.75 1.0
# 10
 0 2 2 0 1 2 3.43 2.15 0 0.94 0 0 0 0 0 0.65 1.0
# 12
 0 0 2 0 1 -1 2.89 2.89 0 0.95 0 0 0 0 0 0.78 0.0
# 14
 0 2 0 0 -1 -1 2.59 ...
```

data

every new pattern starts with `#label`

could also use

```
# input
0 1 0 0 1
# target
1
```

etc

Figure 4.6: Pattern file diagram

The header has to conform exactly to the SNNS format, so watch out for extra spaces etc. I found it easiest to copy headers from one of the example pattern files and to edit the numbers. The data component of the pattern file is simply a listing of numbers that represent the activations of the input and output units. For each pattern the number of values has to match the number of input plus thenumber of output units of the network as defined in the header. For clarity you may wish to put comments (lines starting with a hash (#)) between your patterns like shown in figure 4.6. They are ignored by SNNS but may be used by some pattern processing tools. The pattern definitions may have 'CR' characters in them.

Note that while the results saved by SNNS use (almost) the same file format as used for the pattern files, the label values defined in the pattern files are not used.

4.1.7.2 Network files

The networks files, just as the pattern and result files are stored as ASCII files, they are relatively easy to read and you may find it easier to hand-edit the network definition file than to use the graphical user interface to perform tasks such as changing the unit transfer function or to change the network topology.

4.2 XGUI Files

The graphical user interface consists of the following files:

`snns` or `xgui`	SNNS simulator program (XGUI and simulator kernel linked together into one executable program)
`default.cfg`	default configuration (see chapter 4.3.2)
`help.hdoc`	help text used by XGUI

The file `snns` in the home directory of SNNS is only a symbolic link to the file

```
xgui/bin/<architecture>/xgui
```

where `<architecture>` is one of the currently supported machine architectures, like `dec_alpha`, `dec_mips`, `hp_ux`, `ibm_aix`, `sgi_irix`, `pc_linux`, `sun_os4x`, or `sun_solaris`.

The file `Readme_xgui` contains changes performed after printing of this document. The user is urged to read it, prior to using XGUI. The file `help.hdoc` is explained in chapter 4.3.10.

XGUI looks for the files `default.cfg` and `help.hdoc` first in the current directory. If not found there, it looks in the directory specified by the environment variable `XGUILOADPATH`. By the command

`setenv XGUILOADPATH` *Path*

this variable can be set to the path where `default.cfg` and `help.hdoc` are located. This is best done by an entry to the files `.login` or `.cshrc`. Advanced users may change the help file or the default configuration for their own purposes. This should be done, however, only on a copy of the files in a private directory.

SNNS uses the following extensions for its files:

`.net`	network files (units and link weights)
`.pat`	pattern files
`.cfg`	configuration settings files
`.txt`	text files (log files)

`.res` result files (unit activations)

A simulator run is started by the command

`snns [<netfile>.net] [<pattern>.pat] [<config>.cfg] [options]` Return

where valid options are

-font `<name>` : font for the simulator
-dfont `<name>` : font for the displays
-mono : black & white on color screens
-help : help screen to explain the options

in the home directory of SNNS or by directly calling

`<SNNS-directory>/xgui/bin/<architecture>/xgui`

from any directory. Note that the shell variable XGUILOADPATH must be set properly before, or SNNS will complain about missing files `default.cfg` and `help.hdoc`.

The executable `xgui` may also be called with X-Window parameters as arguments.

Setting the display font can be advisable, if the font selected by the SNNS automatic font detection looks ugly. The following example starts the display with the 7x13bold font

`snns -font 7x13bold` Return

The fonts which are available can be detected with the program xfontsel (not part of this distribution). The current version of SNNS can not handle fonts wider than 8 pixels.

4.3 Windows of XGUI

The graphical user interface has the following windows, which can be positioned and handled independently (*toplevel shells*):

- **Manager** panel with buttons to open other windows, a message line, and a line with status information at the bottom .
- **File** browser for loading and saving networks and pattern files.
- **Control** panel for simulator operations.
- **Info** panel for setting and getting information about unit and link attributes.
- several **Displays**, to display the network graphically in two dimensions.
- **3D View** panel to control the three dimensional network visualization component.
- **Graph** display, to explain the network error during teaching graphically.
- **Bignet** panel to facilitate the creation of big regular networks.
- **Pruning** panel for control of the pruning algorithm.