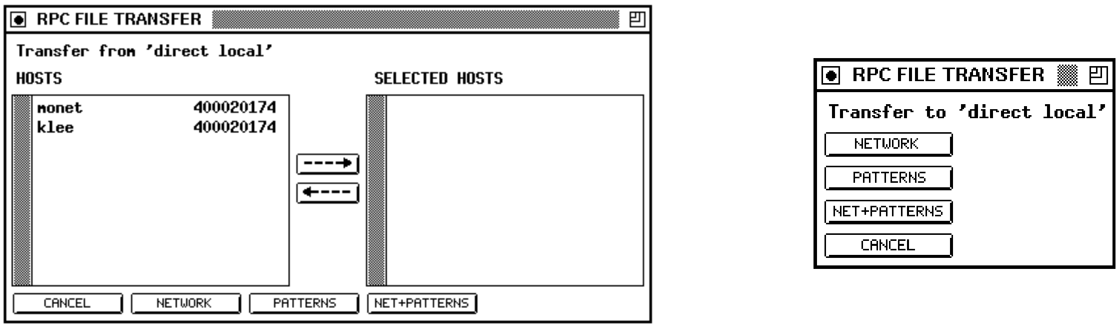

Figure 12.4: The RPC Transfer Panels. Put on the left, Get on the right.

12.2.5 Show Info Panel

With this panel it is possible to select the information, displayed in the **SNNS RPC** panel. Items can be added or deleted with the two arrows. The order of the selected items can be chosen with `UP IN LIST` or `DOWN IN LIST`.

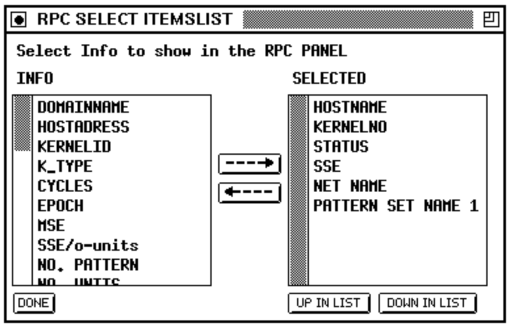

Figure 12.5: The RPC Info Panel

The possible fields are:

- `HOST-NAME`: the computer name on which the simulator kernel is running,
- `KERNEL-NAME`: the version number under which the simulator kernel is registered at the portmapper (this information can also be displayed with 'rpcinfo -p'),
- `STATUS`: the last known state of the simulator kernel,
- `DOMAINNAME`: the domain name of the network,
- `HOSTADDRESS`:the host address of the simulator kernel,
- `K-TYPE`: the type of the simulator kernel ("coop-master" and "normal" are possible, but coop-master is not implemented in the current version),
- `KERNEL_ID`: id of the simulator kernel in xgui,
- `CYCLES`: number of cycles the kernel used to train,

- `EPOCH`: number of epochs the kernel used to train,
- `SSE`: the sse error value,
- `MSE`: the mse error value,
- `SSE/o-units`: the sse error value divided bv the number of output units,
- `NO. PATTERN`: the number of patterns,
- `NO. UNITS`: the number of units,
- `NO. INPUT`: the number of input units,
- `NO. HIDDEN`: the number of hidden units,
- `NO. OUTPUT`: the number of output units,
- `NET NAME`: the network name,
- `PATTERN SET NAME`: the name of the pattern sets,
- `LEARN`: the selected learning parameters,
- `UPDATE`: the selected update parameters,
- `INIT`: the selected initialization parameters.

Note that all information is usually not displayed immediately, but with a small delay.

12.3 RPC Errors

If an rpc function is called, there may be additional errors besides regular SNNS errors, such as the call of the `DeletePattern` function without a defined pattern in the network. Because `rpcgen` is not generating error code within the rpc routines the result of these functions has to be checked for errors. Since these routines return a NULL value in case of an error, the return value can be checked for NULL and the system can then call an error routine, which determines the exact cause of the error. The rpc library provides such an error routine. Because the rpc mechanism can not determine if there is a net-adapter malfunction or if one of the cables is defect, the rpc mechanism only determines that there is no response from anyone within a given time interval. The error cause, delivered by the error function is not that an adapter card is defect, but that the time interval was exceeded (time-out error). This is the most common type of error and the communication module tries to detect the error with the help of the graphical user-interface.

To accomplish such a task, the communication module remembers if the last inquiry was performed synchronously or asynchronously. If the inquiry was asynchronous the probable cause is that the server is not ready for a new task, because it has to finish the current one. Therefore the simulator kernel reports to the graphical user-interface as soon as it has finished its task. This enables the communication module to update all status informations about the simulator kernel. The user will be informed about an error, and if available what kind of error, over a 'pop up' window.

Due to the separation of the graphical user-interface and the simulator kernel, the error tolerance has increased. If there is for example an error in the graphical user-interface, only the interface will be terminated and after a new start of the graphical user-interface and its linking to the simulator kernel, the user is able to proceed exactly where the error occured.

If an error occurs in the kernel the situation is less complicated. In that case simply a new net has to be started, since the communication module intercepts all maskable signals and saves the current net before it ends its operation correctly.

This kind of independence can be used to train multiple nets in parallel. All that has to be done is to load multiple nets with multiple simulator kernels and to start the training process asynchronously with the desired parameters. Now the graphical user-interface can be terminated, while the simulator kernel trains the nets. It is possible to interrupt the training and to evaluate the current state of the training process with the graphical user-interface. After that, the training can be resumed. In case the simulator kernel has already finished training the net it will wait a given period of time before it saves the net and ends its operations. In contrast to the batch version of SNNS, `snnsbat`, it is possible to select all actions of the kernel from a menu and to evaluate the current training, e.g. the current epochs and error rate.

Chapter 13

Batchman

Since training a neural network may require several hours of CPU time, it is advisable to perform this task as a batch job during low usage times. SNNS offers the program `batchman` for this purpose. It is basically an additional interface to the kernel that allows easy background execution.

13.1 Introduction

This newly implemented batch language is to replace the old `snnsbat`. Programs which are written in the old `snnsbat` language will not be able to run on the newly designed interpreter. `Snnsbat` is not supported any longer, but we keep the program for those users who are comfortable with it and do not want to switch to `batchman`. The new language supports all functions which are necessary to train and test neural nets. All non-graphical features which are offered by the graphical user interface (`XGUI`) may be accessed with the help of this language as well.

The new batch language was modeled after languages like `AWK`, `Pascal`, `Modula2` and `C`. It is an advantage to have some knowledge in one of the described languages. The language will enable the user to get the desired result without investing a lot of time in learning its syntactical structure. For most operators multiple spellings are possible and variables don't have to be declared before they are used. If an error occurs in the written batch program the user will be informed by a displayed meaningful error message (warning) and the corresponding line number.

13.1.1 Styling Conventions

Here is a description of the style conventions used:

Input which occurs on a Unix command line or which is part of the batch program will be displayed in `typewriter` writing. Such an input should be adopted without any modification.

For example:

```
/Unix> batchman -h
```

This is an instruction which should be entered in the Unix command line, where `/Unix>` is the shell prompt which expects input from the user. Its appearance may change depending on the Unix-system installed. The instruction `batchman -h` starts the interpreter with the `-h` help option which tells the interpreter to display a help message. Every form of input has to be confirmed with Enter (Return). Batch programs or part of batch programs will also be displayed in typewriter writing. Batch programs can be written with a conventional text editor and saved in a file. Commands can also be entered in the interactive mode of the interpreter. If a file is used as a source to enter instructions, the name of the file has to be provided when starting the interpreter. Typewriter writing is also used for wild cards. Those wild cards have to be replaced by real names.

13.1.2 Calling the Batch Interpreter

The Interpreter can be used in an interactive mode or with the help of a file, containing the batch program. When using a file no input from the keyboard is necessary. The interactive mode can be activated by just calling the interpreter:

```
/Unix> batchman
```

which produces:

```
SNNS Batch Interpreter V1.0.  Type batchman -h for help.
No input file specified, reading input from stdin.
batchman>
```

Now the interpreter is ready to accept the user's instructions, which can be entered with the help of the keyboard. Once the input is completed the interpreter can be put to work with `Ctrl-D`. The interpreter can be aborted with `Ctrl-C`. The instructions entered are only invoked after `Ctrl-D` is pressed.

If the user decides to use a file for input the command line option `-f` has to be given together with the name of the interpreter:

```
/Unix> batchman -f myprog.bat
```

Once this is completed, the interpreter starts the program contained in the file myprog.bat and executes its commands.

The standard output is usually the screen but with the command line option `-l` the output can be redirected in a protocol file. The name of the file has to follow the command line option:

```
/Unix> batchman -l logfile
```

Usually the output is redirected in combination with the reading of the program out of a file:

```
/Unix> batchman -f myprog.bat -l logfile
```

The order of the command line options is arbitrary.

More command line options are:

`-p`: Programs should only be parsed but not executed. This option tells the interpreter to check the correctness of the program without executing the instructions contained in the program. Run time errors can not be detected. Such a run time error could be an invalid SNNS function call.

`-q`: No messages should be displayed except those caused by the `print()`-function.

`-s`: No warnings should be displayed.

`-h`: A help message should be displayed which describes the available command line options.

All following input will be printed without the shell-text.

13.2 Description of the Batch Language

This section explains the general structure of a batch program, the usage of variables of the different data types and usage of the print function. After this an introduction to control structures follows.

13.2.1 Structure of a Batch Program

The structure of a batch program is not predetermined. There is no declaration section for variables in the program. All instructions are specified in the program according to their execution order. Multiple blanks are allowed between instructions. Even no blanks between instructions are possible if the semantics are clear. Single instructions in a line don't have to be completed by a semicolon. In such a case the end of line character (Ctrl-D) is separating two different instructions in two lines. Also key words which have the responsibility of determining the end of a block (`endwhile, endif, endfor, until` and `else`) don't have to be completed by a semicolon. Multiple semicolons are possible between two instructions. However if there are more than two instructions in a line the semicolon is necessary. Comments in the source code of the programs start with a '#' character. Then the rest of the line will be regarded as a comment.
A comment could have the following appearance:

```
#This is a comment
a:=4 #This is another comment
```

The second line begins with an instruction and ends with a comment.

13.2.2 Data Types and Variables

The batch language is able to recognize the following data types:

- Integer numbers
- Floating point numbers
- Boolean type 'TRUE' and 'FALSE'
- Strings

The creation of float numbers is similar to a creation of such numbers in the language C because they both use the exponential representation. Float numbers would be: 0.42, 3e3, or 0.7E-12. The value of 0.7E-12 would be $0.7 * 10^{-12}$ and the value of 3e3 would be $3 * 10^{3}$

Boolean values are entered as shown above and without any kind of modification.

Strings have to be enclosed by " and can not contain the tabulator character. Strings also have to contain at least one character and can not be longer than one line. Such strings could be:

```
"This is a string"
"This is also a string (0.7E-12)"
```

The following example would yield an error

```
"But this
is not a string"
```

13.2.3 Variables

In order to save values it is possible to use variables in the batch language. A variable is introduced to the interpreter automatically once it is used for the first time. No previous declaration is required. Names of variables must start with a letter or an underscore. Digits, letters or more underscores could follow. Names could be:

```
a, num1, _test, first_net, k17_u, Test_buffer_1
```

The interpreter distinguishes between lower and upper case letters. The type of a variable is not known until a value is assigned to it. The variable has the same type as the assigned value:

```
a = 5
filename := "first.net"
init_flag := TRUE
NET_ERR = 4.7e+11
a := init_flag
```

The assignment of variables is done by using '=' or ':='. The comparison operator is '=='. The variable 'a' belongs to the type integer and changes its type in line 5 to boolean. `Filename` belongs to the type string and `NET_ERR` to the type float.

13.2.4 System Variables

System variables are predefined variables that are set by the program and that are read-only for the user. The following system variables have the same semantics as the displayed variables in the graphical user interface:

SSE	Sum of the squared differences of each output neuron
MSE	SSE divided by the number of training patterns
SSEPU	SSE divided by the number of output neurons of the net
CYCLES	Number of the cycles trained so far.

Additionally there are two more system variables:

PAT	The number of patterns in the current pattern set
EXIT_CODE	The exit status of an execute call

13.2.5 Operators and Expressions

An expression is usually a formula which calculates a value. An expression could be a complex mathematical formula or just a value. Expressions include:

```
3
TRUE
3 + 3
17 - 4 * a + (2 * ln 5) / 0.3
```

The value or the result of an expression can be assigned to a variable. The following operators exist, ordered by priority from top to bottom:

Operator	Function
+, −	Sign for numbers
not, !	Logic negation for boolean numbers
sqrt	Square root
ln	Natural logarithm to the basis e
log	Logarithms to the basis 10
**, ^	Exponential function
*	Multiplication
/	Division
div	Even number division with an even result
mod, %	Result after an even number division
+	Addition
−	Subtraction

Operator	Function
<	smaller than
<=, =<	smaller equal
>	greater than
>=, =>	greater equal
==	equal
<>, !=	not equal
and, &&	logic AND for boolean values
or, $\Vert$	logic OR for boolean values

If more than one expression occurs in a line the execution of expressions starts at the left and proceeds towards the right. The order can be changed with parentheses '(' ')'.

The type of an expression is determined at run time and is set with the operator except in the case of integer number division, the modulo operation, the boolean operation and the compare operations.

If two integer values are multilpied, the result will be an integer value. But if an integer and a float value are multilpied, the result will be a float value. If one operator is of type string, then all other operators are transformed into strings. Partial expressions are calculated before the transformation takes place:

```
a := 5 + " plus " + 4 + " is " + ( 8 + 1 )
```

is transformed to the string:

```
5 plus 4 is 9
```

Please note that if the user decides to use operators such as sqrt, ln, log or the exponential operator, no parentheses are required because the operators are not function calls:

Square root:	`sqrt 9`
natural logarithm:	`ln 2`
logarithm to the base of 10:	`log alpha`
Exponential function:	`10 ** 4` oder `a^b`

However parentheses are possible and some times even necessary:

```
sqrt (9 + 16)
ln (2^16)
log (alpha * sqrt tau)
```

13.2.6 The Print Function

So far the user is able to generate expressions and to assign a value to a variable. In order to display values, the print function is used. The print function is a real function call of the batch interpreter and displays all values on the standard output if no input file is declared. Otherwise all output is redirected into a file. The print function can be called with multiple arguments. If the function is called without any arguments a new line will be produced. All print commands are automatically terminated with a newline.

Instruction:	generates the output:
`print(5)`	`5`
`print(3*4)`	`12`
`print("This is a text")`	`This is a text`
`print("This is a text and values:",1,2,3)`	`This is a text and values:123`
`print("Or: ",1," ",2," ",3)`	`Or: 1 2 3`
`print(ln (2^16))`	`11.0904`
`print(FALSE)`	`FALSE`
`print(25e-2)`	`0.25`

If a variable, which has not been assigned a value yet, is tried to be printed, the print function will display $<\ >$ `undef` instead of a value.

13.2.7 Control Structures

Control structures are a characteristic of a programming language. Such structures make it possible to repeat one or multiple instructions depending on a condition or a value. `BLOCK` has to be replaced by a sequence of instructions. `ASSIGNMENT` has to be replaced by an assignment operation and `EXPRESSION` by an expression. It is also possible to branch within a program with the help of such control structures:

```
if EXPRESSION then BLOCK endif
if EXPRESSION then BLOCK else BLOCK endif
for ASSIGNMENT to EXPRESSION do BLOCK endfor
while EXPRESSION do BLOCK endwhile
repeat BLOCK until EXPRESSION
```

The If Instruction

There are two variants to the `if` instruction. The `first` variant is:

```
If EXPRESSION then BLOCK endif
```

The block is executed only if the expression has the boolean value TRUE.

`EXPRESSIONS` can be replaced by any complex expression if it delivers a boolean value:

```
if sqrt (9)-5<0 and TRUE<>FALSE then print("hello world") endif
```

produces:

```
hello world
```

Please note that the logic operator ‘and’ is the operator last executed due to its lowest priority. If there is confusion about the execution order, it is recommended to use brackets to make sure the desired result will be achieved.

The `second` variant of the `if` operator uses a second block which will be executed as an alternative to the first one. The structure of the second `if` variant looks like this:

```
if EXPRESSION then BLOCK1 else BLOCK2 endif
```

The first BLOCK, here described as BLOCK1, will be executed only if the resulting value of EXPRESSION is 'TRUE'. If EXPRESSION delivers 'FALSE', BLOCK2 will be executed.

The For Instruction

The `for` instruction is a control structure to repeat a block, a fixed number of times. The most general appearance is:

for ASSIGNMENT to EXPRESSION do BLOCK endfor

A counter for the `for` repetitions of the block is needed. This is a variable which counts the loop iterations. The value is increased by one if an loop iteration is completed. If the value of the counter is larger then the value of the `EXPRESSIONS`, the `BLOCK` won't be executed anymore. If the value is already larger at the beginning, the instructions contained in the block are not executed at all. The counter is a simple variable. A `for` instruction could look like this:

for i := 2 to 5 do print (" here we are: ",i) endfor

produces:

```
here we are:  2
here we are:  3
here we are:  4
here we are:  5
```

It is possible to control the repetitions of a block by assigning a value to the counter or by using the `continue`, `break` instructions. The instruction `break` leaves the cycle immediately while `continue` increases the counter by one and performs another repetition of the block. One example could be:

```
for counter := 1 to 200 do
a := a * counter
c := c + 1
if test == TRUE then break endif
endfor
```

In this example the boolean variable test is used to abort the repetitions of the block early.

While and Repeat Instructions

The `while` and the `repeat` instructions differ from a `for` instruction because they don't have a count variable and execute their commands only while a condition is met (while) or until a condition is met (repeat). The condition is an expression which delivers a boolean value. The formats of the `while` and the `repeat` instructions are:

```
while EXPRESSION do BLOCK endwhile
repeat BLOCK until EXPRESSION
```

The user has to make sure that the cycle terminates at one point. This can be achieved by making sure that the EXPRESSION delivers once the value 'TRUE' in case of the `repeat` instruction or 'FALSE' in case of the `while` instruction. The `for` example from the previous section is equivalent to:

```
i := 2
while i <= 5 do
print ( "here we are:  ",i)
i := i + 1 endwhile
```

or to:

```
i := 2
repeat
print ( "here we are:  ",i)
i := i + 1
until i > 5
```

The main difference between `repeat` and `while` is that repeat guarantees that the BLOCK is executed at least once. The `break` and the `continue` instructions may also be used within the BLOCK.

13.3 SNNS Function Calls

The SNNS function calls control the SNNS kernel. They are available as function calls in `batchman`. The function calls can be divided into four groups:

- Functions which are setting SNNS parameters :
  - setInitFunc()
  - setLearnFunc()
  - setUpdateFunc()
  - setPruningFunc()
  - setSubPattern()
  - setShuffle()
  - setSubShuffle()
- Functions which refer to neural nets :
  - loadNet()
  - saveNet()
  - saveResult()
  - initNet()
  - trainNet()
  - testNet()

- Functions which refer to patterns :
  - loadPattern()
  - setPattern()
  - delPattern()
- Special functions :
  - pruneNet()
  - execute()
  - print()
  - exit()
  - setseed()

The format of such calls is:

function_name (parameter1, parameter2...)

No parameters, one parameter, or multiple parameters can be placed after the function name. Unspecified values take on a default value. Note, however, that if the third value is to be modified, the first two values have to be provided with the function call as well. The parameters have the same order as in the graphical user interface.

Now a description of the function calls of the first group follows.

13.3.1 Function Calls To Set SNNS Parameters

The following functions calls to set SNNS parameters are available:

`setInitFunc()`	Selects the initialization function and its parameters
`setLearnFunc()`	Selects the learning function and its parameters
`setUpdateFunc()`	Selects the update function and its parameters
`setPruningFunc()`	Selects the pruning function and its parameters
`setSubPattern()`	Defines the subpattern shifting scheme
`setShuffle()`	Change the shuffle modus
`setSubShuffle()`	Change the subpattern shuffle modus

The format and the usage of the function calls will be discussed now. It is an enormous help to be familiar with the graphical user interface of the SNNS especially with the chapters “Parameters of the learning functions”, “Update functions”, “Initialization functions”, “Handling patterns with SNNS”, and “Pruning algorithms”.

setInitFunc

This function call selects the function with which the net is initialized. The format is:

```
setInitFunc (function name, parameter...)
```

where `function name` is the initialization function and has to be selected out of:

ART1_Weights	DLVQ_Weights	Random_Weights_Perc
ART2_Weights	Hebb	Randomize_Weights
ARTMAP_Weights	Hebb_Fixed_Act	RBF_Weights
CC_Weights	JE_Weights	RBF_Weights_Kohonen
ClippHebb	Kohonen_Rand_Pat	RBF_Weights_Redo
CPN_Weights_v3.2	Kohonen_Weights_v3.2	RCC_Weights
CPN_Weights_v3.3	Kohonen_Const	RM_Random_Weights
CPN_Rand_Pat	PseudoInv	

It has to be provided by the user and the name has to be exactly as printed above. The function name has to be embraced by "".

After the name of the initialization function is provided the user can enter the parameters which influence the initialization process. If no parameters have been entered default values will be selected. The selected parameters have to be of type float. Function calls could look like this:

```
setInitFunc ("Randomize_Weights")
setInitFunc("Randomize_Weights", 1.0, -1.0)
```

where the first call selects the `Randomize_Weights` function with default parameters. The second call uses the `Randomize_Weights` function and sets two parameters. The batch interpreter displays:

```
Init function is now Randomize_Weights
Parameters are:  1.0 -1.0
```

setLearnFunc

The function call `setLearnFunc` is very similar to the `setinitFunc` call. `setLearnFunc` selects the learning function which will be used in the training process of the neural net. The format is:

```
setLearnFunc (function name, parameters....)
```

where function name is the name of the desired learning algorithm. This name is mandatory and has to match one of the following strings:

ART1	Dynamic_LVQ	RadialBasisLearning
ART2	Hebbian	RBF-DDA
ARTMAP	JE_BP	RCC
BackPercolation	JE_BP_Momentum	RM_delta
BackpropBatch	JE_Quickprop	Rprop
BackpropMomentum	JE_Rprop	Sim_Ann_SS
BackpropWeightDecay	Kohonen	Sim_Ann_WTA
BPTT	Monte-Carlo	Sim_Ann_WWTA
BBPTT	PruningFeedForward	Std_Backpropagation
CC	QPTT	TimeDelayBackprop
Counterpropagation	Quickprop	

After the name of the learning algorithm is provided, the user can specify some parameters. The interpreter is using default values if no parameters are selected. The values have to be of the type float. A detailed description can be found in the chapter "Parameter of the learning function". Function calls could look like this:

```
setLearn("Std_Backpropagation")
setLearn( "Std_Backpropagation", 0.1)
```

The first function call selects the learning algorithm and the second one additionally provides the first learning parameter. The batch interpreter displays:

```
Learning function is now:  Std_backpropagation
Parameters are:  0.1
```

setUpdateFunc

This function is selecting the order in which the neurons are visited. The format is:

```
setUpdateFunc (function name, parameters...)
```

where function name is the name of the update function. The name of the update algorithm has to be selected as shown below.

```
Topological_Order     BAM_Order              JE_Special
ART1_Stable           BPTT_Order             Kohonen_Order
ART1_Synchronous      CC_Order               Random_Order
ART2_Stable           CounterPropagation     Random_Permutation
ART2_Synchronous      Dynamic_LVQ            RCC_Order
ARTMAP_Stable         Hopfield_Fixed_Act     Serial_Order
ARTMAP_Synchronous    Hopfield_Synchronous   Synchonous_Order
Auto_Synchronous      JE_Order               TimeDelay_Order
```

After the name is provided several parameters can follow. If no parameters are selected, default values are chosen by the interpreter. The parameters have to be of the type float. The update functions are described in the chapter `Update functions`. A function call could look like this:

```
setUpdateFunc ("Topological_Order")
```

The batch interpreter displays:

```
Update function is now Topological_Order
```

setPruningFunc

This function call is used to select the different pruning algorithms for neural networks. (See chapter `Pruning algorithms`). A function call may look like this:

```
setPruningFunc (function name1, function name2, parameters)
```

where function name1 is the name of the pruning function and has to be selected from:

```
MagPruning               OptimalBrainSurgeon    OptimalBrainDamage
Noncontributing_Units    Skeletonization
```

Function name2 is the name of the subordinated learning function and has to be selected out of:

```
BackpropBatch        Quickprop        BackpropWeightDecay
BackpropMomentum     Rprop            Std_Backpropagation
```

Additionally the parameters described below can be entered. If no parameters are entered default values are used by the interpreter. Those values appear in the graphical user interface in the corresponding widget of the pruning window.

1. Maximum error increase in % (float)
2. Accepted error (float)
3. Recreate last pruned element (boolean)
4. Learn cycles for first training (integer)
5. Learn cycles for retraining (integer)
6. Minimum error to stop (float)
7. Initial value for matrix (float)
8. Input pruning (boolean)
9. Hidden pruning (boolean)

Function calls could look like this:

```
setPruningFunc("OptimalBrainDamage","Std_Backpropagation")
setPruningFunc("MagPruning", "Rprop", 15.0, 3.5, FALSE, 500, 90,
1e6, 1.0)
```

In the first function call the pruning function and the subordinate learning function is selected. In the second function call almost all parameters are specified. Please note that a function call has to be specified without a carriage return. Long function calls have to be specified within one line. The following text is displayed by the batch interpreter:

```
Pruning function is now MagPruning
Subordinate learning function is now Rprop
Parameters are:  15.0 3.5 FALSE 500 90 1.0 1e-6 TRUE TRUE
```

The regular learning function `PruningFeedForward` has to be set with the function call `setLearnFunc()`. This is not necessary if `PruningFeedForward` is already set in the network file.

setSubPattern

The function call `setSubPattern` defines the *Subpattern-Shifting-Scheme* which is described in chapter "Variable size pattern". The definition of the *Subpattern-Shifting-Scheme* has to fit the used pattern file and the architecture of the net. The format of the function call is:

```
setSubPattern(InputSize, InputStep1, OutputSize1, OutputStep1)
```

The first dimension of the subpatterns is described by the first four parameters. The order of the parameters is identical to the order in the graphical user interface (see chapter "Sub Pattern Handling"). All four parameters are needed for one dimension. If a second dimension exists the four parameters of that dimension are given after the four parameters of the first dimension. This applies to all following dimensions. Function calls could look like this:

```
setSubPattern (5, 3, 5, 1)
setSubPattern(5, 3, 5, 1, 5, 3, 5, 1)
```

A one-dimensional subpattern with the InputSize 5, InputStep 3, OutputSize 5, OutputStep 1 is defined by the first call. A two-dimensional subpattern as used in the example network `watch net` is defined by the second function call. The following text is displayed by the batch interpreter:

```
Sub-pattern shifting scheme (re)defined
Parameters are:  5 3 5 1 5 3 5 1
```

The parameters have to be integers.

setShuffle, setSubShuffle

The function calls `setShuffle` and `setSubShuffle` enable the user to work with the shuffle function of the SNNS which selects the next training pattern randomly. The shuffle function can be switched on or off. The format of the function calls is:

```
setShuffle (mode)
setSubShuffle (mode)
```

where the parameter mode is a boolean value. The boolean value TRUE switches the shuffle function on and the boolean value FALSE switches it off. `setShuffe` relates to regular patterns and `setSubShuffle` relates to subpatterns. The function call:

```
setSubShuffle (TRUE)
```

will display:

```
Subpattern shuffling enabled
```

13.3.2 Function Calls Related To Networks

This section describes the second group of function calls which are related to network or network files. The second group of SNNS functions contains the following function calls:

`loadNet()`	Load a net
`saveNet()`	Save a net
`saveResult()`	Save a result file
`initNet()`	initialize a net
`trainNet()`	train a net
`testNet()`	test a net

The function calls `loadNet` and `saveNet` both have the same format:

```
loadNet (file_name)
saveNet (file_name)
```

where file_name is a valid Unix file name enclosed by `" "`. The function `loadNet` loads a net in the simulator kernel and `saveNet` saves a net which is currently located in the simulator kernel. The function call `loadNet` sets the system variable `CYCLES` to zero. This variable contains the number of training cycles used by the simulator to train a net. Examples for such calls could be:

```
loadNet ("encoder.net")
...
saveNet ("encoder.net")
```

The following result can be seen:

```
Net encoder.net loaded
Network file encoder.net written
```

The function call `saveResult` saves a SNNS result file and has the following format:

```
saveResult (file_name, start, end, inclIn, inclOut, file_mode)
```

The first parameter (file_name) is required. The file name has to be a valid Unix file name enclosed by `" "`. All other parameters are optional. Please note that if one specific parameter is to be entered all other parameters before the entered parameter have to be provided also. The parameter `start` selects the first pattern which will be handled and `end` selects the last one. If the user wants to handle all patterns the system variable PAT can be entered here. This system variable contains the number of all patterns. The parameters `inclIn` and `inclOut` decide if the input patterns and the output patterns should be saved in the result file or not. Those parameters contain boolean values. If `inclIn` is TRUE all input patterns will be saved in the result file. If `inclIn` is FALSE the patterns will not be saved. The parameter `inclOut` is identical except for the fact that it relates to output patterns. The last parameter `file_mode` of the type string, decides if a file should be created or if data is just appended to an existing file. The strings "create" and "append" are accepted for file mode. A `saveResult` call could look like this:

```
saveResult ("encoder.res")
saveResult ("encoder.res", 1, PAT, FALSE, TRUE, "create")
```

both will produce this:

```
Result file encoder.res written
```

In the second case the result file encoder.res was written and contains all output patterns.

The function calls `initNet, trainNet, testNet` are related to each other. All functions are called without any parameters:

```
initNet()
trainNet()
testNet()
```

`initNet()` initializes the neural network. After the net has been reset with the function call `setInitFunc`, the system variable `CYCLE` is set to zero. The function call `initNet` is necessary if an untrained net is to be trained for the first time or if the user wants to set a trained net to its untrained state.

```
initNet()
```

produces:

```
Net initialized
```

The function call `trainNet` is training the net exactly one cycle long. After this, the content of the system variables `SSE, MSE, SSEPU` and `Cycles` is updated.

The function call `testNet` is used to display the user the error of the trained net, without actually training it. This call changes the system variables `SSE , MSE, SSEPU` but leaves the net and all its weights unchanged.

Please note that the function call `trainNet` is usually used in combination with a repetition control structure like `for`, `repeat`, or `while`.

13.3.3 Pattern Function Calls

The following function calls relate to patterns:

`loadPattern()`	Loads the pattern file
`setPattern()`	Replaces the current pattern file
`delPattern()`	Deletes the pattern file

The simulator kernel is able to store several pattern files (currently 5). The user can switch between those pattern files with the help of the `setPattern()` call. The function call `delPattern` deletes a pattern file from the simulator kernel. All three mentioned calls have `file_name` as an argument:

```
loadPattern (file_name)
setPattern (file_name)
delPattern (file_name)
```

All three function calls set the value of the system variable `Pat` to the number of patterns of the pattern file used last. The handling of the pattern files is similar to the handling of such files in the graphical user interface. The last loaded pattern file is the current one. The function call `setPattern` (similar to the USE button of the graphical user interface of the SNNS.) selects one of the loaded pattern files as the one currently in use. The call `delPattern` deletes the pattern file currently in use from the kernel. The function calls:

```
loadPattern ("encoder.pat")
loadPattern ("encoder1.pat")
setPattern("encoder.pat")
delPattern("encoder.pat")
```

produce:

```
Patternset encoder.pat loaded; 1 patternset(s) in memory
Patternset encoder1.pat loaded; 2 patternset(s) in memory
Patternset is now encoder.pat
Patternset encoder.pat deleted; 1 patternset(s) in memory
Patternset is now encoder1.pat
```

13.3.4 Special Functions

There are four miscelaneous functions for the use in `batchman`

`pruneNet()`	Starts network pruning
`execute()`	Executes any unix shell comand or program
`exit()`	Quits `batchman`
`setseed()`	Sets a seed for the random number generator

The function call pruneNet() is pruning a net equivalent to the pruning in the graphical user interface. After all functions and parameters are set with the call `setPruningFunc` the `pruneNet()` function call can be executed. No parameters are necessary:

pruneNet()

An interface to the Unix operation system can be created by using the function `execute`. This function call enables the user to start a program at the Unix command line and redirect its output to the batch program. All Unix help programs can be used to make this special function a very powerful tool. The format is:

```
execute (instruction, variable1, variable2.....)
```

where 'instruction' is a Unix instruction or a Unix program. All output, generated by the Unix command has to be separated by blanks and has to be placed in one line. If this is not done automatically please use the Unix commands `AWK` or `grep` to format the output as needed. Those commands are able to produce such a format. The output generated by the program will be assigned, according to the order of the output sequences, to the variables `variable1`, `variable2`.. The data type of the generated output is automatically set to one of the four data types of the batch interpreter. Additionally the exit state of the Unix program is saved in the system variable `EXIT_CODE`. An example for `execute` is:

```
execute ("date", one, two, three, four)
print ("It is ", four, " o'clock")
```

This function call calls the command `date` and reads the output `"Fri May 19 16:28:29 GMT 1995"` in the four above named variables. The variable 'four' contains the time. The batch interpreter produces:

```
It is 16:28:29 o'clock
```

The `execute` call could also be used to determine the available free disk space:

```
execute ("df .| grep dev", dmy, dmy, dmy, freeblocks)
print ("There are ", freeblocks, "Blocks free")
```

In this examples the Unix pipe and the grep command are responsible for reducing the output and placing it into one line. All lines, that contain `dev`, are filtered out. The second line is read by the batch interpreter and all information is assigned to the named variables. The first three fields are assigned to the variable `dmy`. The information about the available blocks will be stored in the variable `freeblocks`. The following output is produced:

```
There are 46102 Blocks free
```

The examples given above should give the user an idea how to handle the `execute` command. It should be pointed out here that `execute` could as well call another batch interpreter which could work on partial solutions of the problem. If the user wants to accomplish such a task the command line option -q of the batch interpreter could be used to suppress output not caused by the print command. This would ease the reading of the output.

```
exit()
```

The last special function call is `exit`. This function call leaves the batch program immediately and terminates the batch interpreter. The parameter used in this function is the exit state, which will be returned to the calling program (usually the Unix shell). If no parameter is used the batch interpreter returns zero. The format is:

```
exit (state)
```

The integer `state` ranges from -128 to +127. If the value is not within this range the value will be mapped into the valid range and an error message displayed. The following example will show the user how this function call could be used:

```
if freeblocks < 1000 then
print ("Not enough disk space")
exit (1)
endif
```

The function `setseed` sets a seed value for the random number generator used by the initialization functions. If `setseed` is not called before initializing a network, subsequent initializiations yield the exact same initial network conditions. Thereby it is possible to make an exact comparison of two training runs with different learning parameters.

```
setseed(seed)
```

`Setseed` may be called with an integer parameter as a seed value. Without a parameter it uses the value returned by the shell command 'date' as seed value.

13.4 Batchman Example Programs

13.4.1 Example 1

A typical program to train a net may look like this:

```
loadNet("encoder.net")
loadPattern("encoder.pat")
setInitFunc("Randomize_Weights", 1.0, -1.0)
initNet()

while SSE > 6.9 and CYCLES < 1000 do
  if CYCLES mod 10 == 0 then
    print ("cycles = ", CYCLES, "  SSE = ", SSE) endif
  trainNet()
endwhile

saveResult("encoder.res", 1, PAT, TRUE, TRUE, "create")
saveNet("encoder.trained.net")

print ("Cycles trained: ", CYCLES)
print ("Training stopped at error: ", SSE)
```

This batch program loads the neural net 'encoder.net' and the corresponding pattern file. Now the net is initialized. A training process continues until the SSE error is smaller or equal to 6.9. The trained net and the result file are saved once the training is completed. The following output is generated by this program:

```
Net encoder.net loaded
Patternset encoder.pat loaded; 1 patternset(s) in memory
Init function is now Randomize_Weights
Net initialised
cycles = 0  SSE = 3.40282e+38
cycles = 10  SSE = 7.68288
cycles = 20  SSE = 7.08139
cycles = 30  SSE = 6.95443
Result file encoder.res written
```

```
Network file encoder.trained.net written
Cycles trained: 40
Training stopped at error: 6.89944
```

13.4.2 Example 2

The following example program reads the output of the network analyzation program `analyze`. The output is transformed into a single line with the help of the program `analyze.gawk`. The net is trained until all patterns are classified correctly:

```
loadNet ("encoder.net")
loadPattern ("encoder.pat")
initNet ()

while(TRUE)
  for i := 1 to 500 do
    trainNet ()
  endfor

  resfile := "test.res"
  saveResult (resfile, 1, PAT, FALSE, TRUE, "create")
  saveNet("enc1.net")

  command := "analyze -s -e WTA -i " + resfile + " | analyze.gawk"
  execute(command, w, r, u, e)
  print("wrong: ",w, "  right: ",r, "  unknown: ",u, "  error: ",e)
  if(right == 100) break
endwhile
```

The following output is generated:

```
Net encoder.net loaded
Patternset encoder.pat loaded; 1 patternset(s) in memory
-> Batchman warning at line 3:
   Init function and params not specified; using defaults
Net initialised
Result file test.res written
Network file enc1.net written
wrong: 87.5  right: 12.5  unknown: 0  error: 7
Result file test.res written
Network file enc1.net written
wrong: 50  right: 50  unknown: 0  error: 3
Result file test.res written
Network file enc1.net written
wrong: 0  right: 100  unknown: 0  error: 0
```

13.4.3 Example 3

The last example program shows how the user can validate the training with a second pattern file. The net is trained with one training pattern file and the error, which is used to determine when training should be stopped, is measured on a second pattern file. Thereby it is possible to estimate if the net is able to classify unknown patterns correctly:

```
loadNet ("test.net")
loadPattern ("validate.pat")
loadPattern ("training.pat")
initNet ()

repeat
  for i := 1 to 20 do
    trainNet ()
  endfor
  saveNet ("test." + CYCLES + "cycles.net")
  setPattern ("validate.pat")
  testNet ()
  valid_error := SSE
  setPattern ("training.pat")
until valid_error < 2.5

saveResult ("test.res")
```

The program trains a net for 20 cycles and saves it under a new name for every iteration of the repeat instruction. Each time the program tests the net with the validation pattern set. This process is repeated until the error of the validation set is smaller than 2.5

13.5 Snnsbat – The predessor

This section describes `snnsbat`, the old way of controling SNNS in batch mode. Please note that we do encourage everybody to use the new `batchman` facility and do not support `snnsbat` any longer!

13.5.1 The Snnsbat Environment

`snnsbat` runs very dependably even on unstable system configurations and is secured against data loss due to system crashes, network failures etc.. On UNIX based systems the program may be terminated with the command 'kill -15' without loosing the currently computed network.

The calling syntax of snnsbat is:

snnsbat [< configuration_file > [< log_file >]]

This call starts `snnsbat` in the foreground. On UNIX systems the command for background execution is 'at', so that the command line

echo 'snnsbat default.cfg log.file' | at 22:00

would start the program tonight at 10pm[1].

If the optional file names are omitted, `snnsbat` tries to open the configuration file './snnsbat.cfg' and the protocol file './snnsbat.log'.

13.5.2 Using Snnsbat

The batch mode execution of SNNS is controlled by the configuration file. It contains entries that define the network and parameters required for program execution. These entries are tuples (mostly pairs) of a keyword followed by one or more values. There is only one tuple allowed per line, but lines may be separated by an arbitrary number of comment lines. Comments start with the number sign '#'. The set of given tuples specify the actions performed by SNNS in one execution run. An arbitrary number of execution runs can be defined in one configuration file, by separating the tuple sets with the keyword 'PerformActions:'. Within a tuple set, the tuples may be listed in any order. If a tuple is listed several times, values that are already read are overwritten. The only exception is the key 'Type:', which has to be listed only once and as the first key. If a key is omitted, the corresponding value(s) are assigned a default.

Here is a listing of the tuples and their meaning:

Key	Value	Meaning
InitFunction:	<string>	Name of the initialization function.
InitParam:	<float> ···	'NoOfInitParam' parameters for initialization function, separated by blanks.
LearnParam:	<float> ···	'NoOfLearnParam' parameters for learning function, separated by blanks.
UpdateParam:	<float> ···	'NoOfUpdateParam' parameters for the update function, separated by blanks.
LearnPatternFile:	<string>	Filename of the learning patterns.
MaxErrorToStop:	<float>	Network error when learning is to be halted.
MaxLearnCycles:	<int>	Maximum number of learning cycles to be executed.
NetworkFile:	<string>	Filename of the net to be trained.
NoOfInitParam:	<int>	Number of parameters for the initialization function.
NoOfLearnParam:	<int>	No of parameters for learning function.
NoOfUpdateParam:	<int>	No of parameters for update function.
NoOfVarDim:	<int> <int>	Number of variable dimensions of the input and output patterns.
PerformActions:	none	Execution run separator.

[1] This construction is necessary since 'at' can read only from stdin.

Key	Value	Meaning
PruningMaxRetrainCycles:	<int>	maximum no. of cycles per retraining
PruningMaxErrorIncrease:	<float>	Percentage to be added to the first net error. The resulting value cannot be exceeded by the net error, unless it is lower than the accepted error
PruningAcceptedError:	<float>	Maximum accepted error.
PruningRecreate:	[YES | NO]	Flag for reestablishing the last state of the net at the end of pruning
PruningOBSInitParam:	<float>	initial value for OBS
PruningInputPruning:	[YES | NO]	Flag for input unit pruning
PruningHiddenPruning:	[YES | NO]	Flag for hidden unit pruning
ResultFile:	<string>	Filename of the result file.
ResultIncludeInput:	[YES | NO]	Flag for inclusion of input patterns in the result file.
ResultIncludeOutput:	[YES | NO]	Flag for inclusion of output learning patterns in the result file.
SubPatternOSize:	<int> ···	NoOfVarDim[2] int values that specify the shape of the sub patterns of each output pattern.
SubPatternOStep:	<int> ···	NoOfVarDim[2] int values that specify the shifting steps for the sub patterns of each output pattern.
TestPatternFile:	<string>	Filename of the test patterns.
TrainedNetworkFile:	<string>	Filename where the net should be stored after training / initialization.
Type:	<string>	The type of grammar that corresponds to this file. Valid types are: 'SNNSBATCH_1': performs only one execution run. 'SNNSBATCH_2': performs multiple execution runs.
ResultMinMaxPattern:	<int> <int>	Number of the first and last pattern to be used for result file generation.
Shuffle:	[YES | NO]	Flag for pattern shuffling.
ShuffleSubPat:	[YES | NO]	Flag for subpattern shuffling.
SubPatternISize:	<int> ···	NoOfVarDim[1] int values that specify the shape of the sub patterns of each input pattern.
SubPatternIStep:	<int> ···	NoOfVarDim[1] int values that specify the shifting steps for the sub patterns of each input pattern.

Please note the mandatory colon after each key and the upper case of several letters.

`snnsbat` may also be used to perform only parts of a regular network training run. If the network is not to be initialized, training is not to be performed, or no result file is to be computed, the corresponding entries in the configuration file can be omitted.

For all keywords the string '<OLD>' is also a valid value. If <OLD> is specified, the value of the previous execution run is kept. For the keys 'NetworkFile:' and 'LearnPatternFile:' this means, that the corresponding files are not read in again. The network (patterns) already in memory are used instead, thereby saving considerable execution time. This allows for a continuous logging of network performance. The user may, for example, load a network and pattern file, train the network for 100 cycles, create a result file, train another 100 cycles, create a second result file, and so forth. Since the error made by the current network in classifying the patterns is reported in the result file, the series of result files document the improvement of the network performance.

The following table shows the behavior of the program caused by omitted entries:

missing key	resulting behavior
InitFunction:	The net is not initialized.
InitParam:	Init function gets only zero values as parameters.
LearnParam:	Learning function gets only zero values as parameters.
UpdateParam:	Update function gets only zero values as parameters.
LearnPatternFile:	Abort with error message if more than 0 learning cycles are specified. Initialization can be performed if init function does not require patterns.
MaxErrorToStop:	Training runs for 'MaxLearnCycles:' cycles.
MaxLearnCycles:	No training takes place. If training is supposed to run until *MaxErrorToStop*, a rather huge number should be supplied here. (skipping this entry would inhibit training completely).
NetworkFile:	Abort with error message.
NoOfInitParam:	No parameters are assigned to the initialization function. Error message from the SNNS kernel possible.
NoOfLearnParam:	No parameters are assigned to the learning function. Error message from the SNNS kernel possible.
NoOfUpdateParam:	No parameters are assigned to the update function.
NoOfVarDim:	Network can not handle variable pattern sizes.
PerformActions:	Only one execution run is performed. Repeated keywords lead to deletion of older values.
ResultFile:	No result file is generated.
ResultIncludeInput:	The result file does NOT contain input Patterns.
ResultIncludeOutput:	The result file DOES contain learn output Patterns.
ResultMinMaxPattern:	All patterns are propagated.
Shuffle:	Patterns are not shuffled.
ShuffleSubPat:	Subpatterns are not shuffled.
SubPatternISize:	
SubPatternIStep:	
SubPatternOSize:	
SubPatternOStep:	Abort with error message if 'NoOfVarDim:' was specified.

missing key	resulting behavior
TestPatternFile:	Result file generation uses the learning patterns. If they are not specified either, the program is aborted with an error message when trying to generate a result file.
TrainedNetworkFile:	Network is not saved after training / initialization. It is used for result file generation.
Type:	Abort with error message.

Here is a typical example of a configuration file:

```
#
Type: SNNSBATCH_2
#
# If a key is given twice, the second appearance is taken.
# Keys that are not required for a special run may be omitted.
# If a key is omitted but required, a default value is assumed.
# The lines may be separated with comments.
#
# Please note the mandatory file type specification at the beginning and
# the colon following the key.
#
#######################################################################
NetworkFile: /home/SNNSv4.1/examples/letters.net
#
InitFunction: Randomize_Weights
NoOfInitParam: 2
InitParam: -1.0 1.0
#
LearnPatternFile: /home/SNNSv4.1/examples/letters.pat
NoOfVarDim: 2 1
SubPatternISize: 5 5
SubPatternOSize: 26
SubPatternIStep: 5 1
SubPatternOStep: 1
NoOfLearnParam: 2
LearnParam: 0.8 0.3
MaxLearnCycles: 100
MaxErrorToStop: 1
Shuffle: YES
#
TrainedNetworkFile: trained_letters.net
ResultFile: letters1.res
ResultMinMaxPattern: 1 26
ResultIncludeInput: NO
ResultIncludeOutput: YES
#
#This execution run loads a network and pattern file with variable
#pattern format, initializes the network, trains it for 100 cycles
#(or stops, if then error is less than 0.01), and finally computes
#the result file letters1.
PerformActions:
#
NetworkFile: <OLD>
#
LearnPatternFile: <OLD>
NoOfLearnParam: <OLD>
LearnParam: <OLD>
MaxLearnCycles: 100
MaxErrorToStop: 1
Shuffle: YES
#
ResultFile: letters2.res
ResultMinMaxPattern: <OLD>
ResultIncludeInput:  <OLD>
ResultIncludeOutput: <OLD>
```

```
#
#This execution run continues the training of the already loaded file
#for another 100 cycles before creating a second result file.
#
PerformActions:
#
NetworkFile: <OLD>
#
LearnPatternFile: <OLD>
NoOfLearnParam: <OLD>
LearnParam: 0.2 0.3
MaxLearnCycles: 100
MaxErrorToStop: 0.01
Shuffle: YES
#
ResultFile: letters3.res
ResultMinMaxPattern: <OLD>
ResultIncludeInput:  <OLD>
ResultIncludeOutput: <OLD>
TrainedNetworkFile: trained_letters.net
#
#This execution run concludes the training of the already loaded file.
#After another 100 cycles of training with changed learning
#parameters the final network is saved to a file and a third result
#file is created.
#
```

The file <log_file> collects the SNNS kernel messages and contains statistics about running time and speed of the program. An example protocol file is listed in appendix **??**.

If the <log_file> command line parameter is omitted, `snnsbat` opens the file 'snnsbat.log' in the current directory. To limit the size of this file, a maximum of 100 learning cycles are logged. This means, that for 1000 learning cycles a message will be written to the file every 10 cycles.

If the time required for network training exceeds 30 minutes of CPU time, the network is saved. The log file then shows the message:

```
#####  Temporary network file 'SNNS_Aaaa00457' created.  #####
```

Temporay networks always start with the string 'SNNS_'. After 30 more minutes of CPU time, `snnsbat` creates a second security copy. Upon normal termination of the program, these copies are deleted from the current directory. The log file then shows the message:

```
#####  Temporary network file 'SNNS_Aaaa00457' removed.  #####
```

In an emergency (powerdown, kill, alarm, etc.), the current network is saved by the program. The log file, resp. the mailbox, will later show an entry like:

```
Signal 15 caught, SNNS V4.1 Batchlearning terminated.
```

```
SNNS V4.1 Batchlearning terminated at Tue Mar 23 08:49:04 1995
System:  SunOS Node:  matisse Machine:  sun4m
Networkfile './SNNS_BAAa02686' saved.
Logfile 'snnsbat.log' written.
```

13.5.3 Calling Snnsbat

`snnsbat` may be called interactively or in batch mode. It was designed, however, to be called in batch mode. On Unix machines, the command 'at' should be used, to allow logging the program with the mailbox. However, 'at' can only read from standard input, so a combination of 'echo' and 'pipe' has to be used.

Three short examples for Unix are given here, to clarify the calls:

unix>echo 'snnsbat mybatch.cfg mybatch.log' | at 21:00 Friday

starts `snnsbat` next Friday at 9pm with the parameters given in mybatch.cfg and writes the output to the file mybatch.log in the current directory.

unix>echo 'snnsbat SNNSconfig1.cfg SNNSlog1.log' | at 22

starts `snnsbat` today at 10pm

unix>echo 'snnsbat' | at now + 2 hours

starts `snnsbat` in 2 hours and uses the default files snnsbat.cfg and snnsbat.log

The executable is located in the directory '.../SNNSv4.1/kernel/bin/<machine_type>/'.
The sources of snnsbat can be found in the directory '.../SNNSv4.1/kernel/sources/'.
An example configuration file was placed in '.../SNNSv4.0/examples'.

Chapter 14

Tools for SNNS

14.1 Overview

There are the following tools available to ease the use of SNNS:

`analyze:`	analyzes result files generated by SNNS to test the classification capabilities of the corresponding net
`bignet:`	feedforward network generator
`Convert2snns:`	pattern conversion tool for Kohonen Networks
`feedback-gennet:`	generator for network definition files
`mkhead:`	writes SNNS pattern file header to stdout
`mkout:`	writes SNNS output pattern to stdout
`mkpat:`	reads 8 bit rawfile and writes SNNS pattern file to stdout
`netlearn:`	backpropagation test program
`netperf:`	benchmark program
`pat_sel:`	produces pattern file with selected patterns
`snns2c:`	compiles an SNNS network file into an executable C source.

14.2 Analyze

The purpose of this tool is to analyze the result files that have been created by SNNS. The result file which you want to analyze has to contain the `teaching output` and the `output of the network`.

Synopsis: `analyze` [-options]

It is possible to choose between the following options in any order:

`-w`	numbers of patterns which were classified wrong are printed
`-r`	numbers of patterns which were classified right are printed
`-u`	numbers of patterns which were not classified are printed
`-a`	same as -w -r -u

`-v`	verbose output. Each printed number is preceded by one of the words 'wrong', 'right' or 'unknown' depending on the result of the classification.
`-s`	statistic information containing wrong, right and not classified patterns. The network error is printed also.
`-c`	same as -s, but statistics for each output unit (class) is displayed.
`-i <file name>`	name of the 'result file' which is going to be analyzed.
`-o <file name>`	name of the file which is going to be produced by `analyze`.
`-e <function>`	defines the name of the 'analyzing function'. Possible names are: 402040, WTA, band (description see below)
`-l <real value>`	first parameter of the analyzing function.
`-h <real value>`	second parameter of the analyzing function.

Starting `analyze` without any options is equivalent to:

```
analyze -w -e 402040 -l 0.4 -h 0.6
```

14.2.1 Analyzing Functions

The classification of the patterns depends on the analyzing function. `402040` stands for the '402040' rule. That means on a range from 0 to 1 h will be 0.6 (upper 40%) and l will be 0.4 (lower 40%). The middle 20% is represented by h-l. The classification of the patterns will depend on h, l and other constrains (see `402040` below).
`WTA` stands for winner takes all. That means the classification depends on the unit with the highest output and other constrains (see `WTA` below). `Band` is an analyzing function that checks a band of values around the teaching output. This is the only analyzing function that treats teaching outputs of value zero correctly.

402040:
A pattern is classified correctly if:

- the output of exactly one output unit is $\geq$ h.
- the 'teaching output' of this unit is the maximum teaching output of the pattern.
- the output of all other output units is $\leq$ l.

A pattern is classified incorrectly if:

- the output of exactly one output unit is $\geq$ h.
- the 'teaching output' of this unit is `NOT` the maximum 'teaching output' of the pattern.
- the output of all other units is $\leq$ l.

A pattern is unclassified in all other cases.

WTA:
A pattern is classified correctly if:

- there is an output unit with the value greater than the output value of all other output units (this output value is supposed to be 0).
- 0 > h.
- the 'teaching output of this unit is the maximum 'teaching output' of the pattern.
- the output of all other units is < a-1.

A pattern is classified incorrectly if:

- there is an output unit with the value greater than the output value of all other output units (this output value is supposed to be 0).
- 0 > h.
- the 'teaching output ' of this unit is `NOT` the maximum 'teaching output' of the pattern.
- the output of all other output units is < a-1.

A pattern is unclassified in all other cases.

Band:
A pattern is classified correctly if for all output units:

- the output is >= the teaching output - low.
- the output is <= the teaching output + high.

A pattern is classified incorrectly if for all output units:

- the output is <= the teaching output - low.
  or
- the output is >= the teaching output + high.

Default values used for l and h are: l=0.0 h=0.0

14.3 Bignet

This program is a generator for special 3-layered feedforward networks. It is a demo for using the SNNS kernel interface to create networks. It has nothing to do with the XGUI bignet tool.

Synopsis: `bignet`

example:

```
unix> bignet
```

produces

SNNS 3D-Kernel 4.1 — Network Generator for 3 Layer Feedforward Networks —

No. of input units: 35
No. of output units: 26
No. of hidden units: 10

Create Units now...
make connections

Name of the network : test
Filename of the network to save: test.net
Saving the network...

14.4 Convert2snns

In order to work with the KOHONEN tools in SNNS, a pattern file and a network file with a special format are necessary.

Convert2snns will accomplish three important things:

- Creation of a 2-dimensional Kohonen Feature Map with n components
- Weight files are converted in a SNNS compatible .net file
- A file with raw patterns is converted in a .pat file

When working with convert2snns, 3 files are necessary:

1. A control file, containing the configuration of the network
2. A file with weight vectors
3. A file with raw patterns

14.4.1 Setup and Structure of a Control, Weight, Pattern File

Each line of the control file begins with a `KEYWORD` followed by the respective declaration. The order of the keywords is arbitrary.

Example of a control file:

`PATTERNFILE` eddy.in **
`WEIGHTFILE` eddy.dat
`XSIZE` 18 ←
`YSIZE` 18 ←
`COMPONENTS` 8 ←
`PATTERNS` 47 **

For creation of a network file you need at least the statements marked ← and for the .pat file additionally the statements marked **.
Omitting the `WEIGHTFILE` will initialize the weights of the network with 0.

The `WEIGHTFILE` is a simple ASCII file, containing the weight vectors row by row.

The `PATTERNFILE` contains in each line the components of a pattern.

If convert2snns has finished the conversion it will ask for the name of the network and pattern files to be saved.

14.5 Feedback-gennet

The program `feedback-gennet` generates network definition files for fully recurrent networks of any size. This is not possible by using bignet.

The networks have the following structure:

- input layer with no intra layer connections
- fully recurrent hidden layer
- output layer: connections from each hidden unit to each output unit
`AND`
optionally fully recurrent intra layer connections in the output layer
`AND`
optionally feedback connections from each output unit to each hidden unit.

The activation function of the output units can be set to sigmoidal or linear. All weights are initialized with 0.0. Other initializations should be performed by the init functions in SNNS.

Synopsis: `feedback-gennet`

example:

`unix>` feedback-gennet

produces

```
Enter # input units:   3
Enter # hidden units:  3
Enter # output units:  1
INTRA layer connections in the output layer        (y/n) :n
feedback connections from output to hidden units   (y/n) :n
Linear output activation function                  (y/n) :n
Enter name of the network file: xor-rec.net
working...
generated xor-rec.net
```

14.6 Mkhead

This program writes a SNNS pattern file header to stdout. This program can be used together with mkpat and mkout to produce pattern files from raw files in a shell script.

Synopsis: `mkhead` <pats> <in_units> <out_units>

where:

`pats`	are the number of patterns in the file
`in_units`	are the number of input units in the file
`out_units`	are the number of output units in the file

14.7 Mkout

This program writes a SNNS output pattern to stdout. This program can be used together with mkpat and mkhead to produce pattern files from raw files in a shell script.

Synopsis: `mkout` <units> <active_unit>

where:

`units`	is the number of output units
`active_unit`	is the unit which has to be activated

14.8 Mkpat

The purpose of this program is to read a binary 8-Bit file from the stdin and writes a SNNS pattern file entry to stdout. This program can be used together with mkpat and mkout to produce pattern files from raw files in a shell script.

Synopsis: `mkpat` <xsize> <ysize>

where:

`xsize`	is the xsize of the raw file
`ysize`	is the ysize of the raw file

14.9 Netlearn

This is a SNNS kernel backpropagation test program. It is a demo for using the SNNS kernel interface to train networks.

Synopsis: `netlearn`

example:

```
unix> netlearn
```

produces

```
SNNS 3D-Kernel V 4.1
—Network learning—

Filename of the network file: letters_untrained.net
Loading the network...

Network name: letters
No. of units:            71
No. of input units:      35
No. of output units:     26
No. of sites:            0
No. of links:            610

Learning function:       Std_Backpropagation
Update function:         Topological_Order

Filename of the pattern file: letters.pat
loading the patterns...

Number of pattern:       26

The learning function Std_Backpropagation needs 2 input parameters:
Parameter [1]:           0.6
Parameter [2]:           0.6
Choose number of cycles: 250

Shuffle patterns (y/n)   n
Shuffling of patterns disabled

learning...
```

14.10 Netperf

This is a benchmark program for SNNS. Propagation and backpropagation tests are performed.

Synopsis: `netperf`

example:

```
unix> netperf
```

produces

```
SNNS 3D-Kernel V4.1          — Benchmark Test —
Filename of the network file: nettalk.net
loading the network...       Network name: nettalk1
No. of units:                349
No. of input units:          203
No. of ouput units:          26
No. of sites:                0
No. of links:                27480

Learningfunction:            Std_Backpropagation
Updatefunction:              Topological_Order

Do you want to benchmark
Propagation                  [1] or
Backpropagation              [2]
Input:                       1

Choose no. of cycles
Begin propagation...

No. of units updated:        34900
No. of sites updated:        0
No. of links updated:        2748000
CPU Time used:               3.05 seconds

No. of connections per second (CPS) : 9.0099e+05
```

14.11 Pat_sel

Given a pattern file and a file which contains numbers, `pat_sel` produces a new pattern file which contains the subset of the first one. This pattern file consists of the patterns whose numbers are given in the number file.

Synopsis: `pat_sel` <number file> <input pattern file> <output pattern file>

Parameters:

`<number file>`	ASCII file which contains positive integer numbers (one per line) in ascending order.
`<input pattern file>`	SNNS pattern file.
`<output pattern file>`	SNNS pattern file which contains the selected subset (created by `pat_sel`)

`Pat_sel` can be used to create a pattern file which contains only the patterns that were classified 'wrong' by the neural network. That is why a 'result file' has to be created using

SNNS. The result file can be analyzed with the tool analyze. This 'number file' and the corresponding 'pattern file' are used by pat_sel. The new 'pattern file' will be created.

14.12 Snns2c

Synopsis: `snns2c` <network> [<C-filename> [<function-name>]]

where:

`<network>`	is the name of the SNNS network file,
`<C-filename>`	is the name of the output file
`<function-name>`	is the name of the procedure in the application.

This tool compiles an SNNS network file into an executable C source. It reads a network file `<network.net>` and generates a C source named `<C-filename>`. The network can be called now as a function named `<function-name>`. If the parameter `<function-name>` is missing, the name of `<C-filename>` is taken without the ending "*.c". If this parameter is also missing, the name of the network file is chosen and fitted with a new ending for the output file. This name without ending is also used for the function name.

It is not possible to train the generated net, SNNS has to be used for this purpose. After completion of network training with SNNS, the tool SNNS2C is used to integrate the trained network as a C function into a separate application.

This program is also an example how to use the SNNS kernel interface for loading a net and changing the loaded net into another format. All data and all SNNS functions – except the activation functions – are placed in a single C function.

Note:
SNNS2C does not support sites. Any networks created with SNNS that make use of the site feature can not be converted to C source by this tool. Output functions are not supported, either.

The program can translate the following network-types:

- Feedforward networks trained with Backpropagation and all variants of it like Quickprop, RPROP etc.
- Radial Basis Functions
- Partially-recurrent Elman and Jordan networks
- Time Delay Neural Networks (TDNN)
- Dynamic Learning Vector Quantisation (DLVQ)
- Backpropagation Through Time (BPTT, QPTT, BBPTT)
- Counterpropagation Networks

While the use of SNNS or any parts of it in commercial applications requires a special agreement/licensing from the developers, the use of trained networks generated with

SNNS2C is hereby granted without any fees for any purpose, provided proper academic credit to the SNNS team is given in the documentation of the application.

14.12.1 Program Flow

Because the compilation of very large nets may require some time, the program outputs messages, indicating which state of compilation is passed at the moment.

loading net... the network file is loaded with the function offered by the kernel user interface.

dividing net into layers ... all units are grouped into layers where all units have the same type and the same activation function. There must not exist any dependencies between the units of the layers except the connections of SPECIAL HIDDEN units to themselves in Elman and Jordan networks or the links of the BPTT-networks.

sorting layers... these layers are sorted in topological order, e.g. first the input layer, then the hidden layers followed by the output layers and at last the special hidden layers. A layer which has sources in another layer of the same type is updated later as the source layer.

writing net... selects the needed activation functions and writes them to the C-source file. After that, the procedure for pattern propagation is written.

14.12.2 Including the Compiled Network in the Own Application

Interfaces: All generated networks may be called as C functions. This functions have the form:

`int`*`function-name`*`(float *in, float *out, int init)`

where `in` and `out` are pointers to the input and output arrays of the network. The init flag is needed by some network types and it's special meaning is explained in 14.12.3. The function normally returns the value 0 (OK). Other return values are explained in section 14.12.3.

The generated C-source can be compiled separately. To use the network it's necessary to include the generated header file (*.h) which is also written by SNNS2C. This header file contains a prototype of the generated function and a record which contains the number of input and output units also.

Example: If a trained network, was saved as "myNetwork.net" and compiled with

```
snns2c myNetwork.net
```

then the generated Network can be compiled with

```
gcc -c myNetwork.c
```

To include the network in your own application the header file must be included. There should also two arrays being provided, one for the input and one for the output of the network. The number of inputs and outputs can be derived from a record in the header file. This struct is named like the function which contains the compiled network and has the suffix REC to mark the record. So the number of input units is determined with `myNetworkREC.NoOfInput` and the number of outputs with `myNetworkREC.NoOfOutput` in this example. Hence, your own application should contain:

```
⋮

#include "myNetwork.h"

⋮

float *netInput, *netOutput; /* Input and Output arrays of the Network */
netInput  = malloc(myNetworkREC.NoOfInput  * sizeof(float));
netOutput = malloc(myNetworkREC.NoOfOutput * sizeof(float));

⋮

myNetwork(netInput, netOutput, 0)

⋮
```

Don't forget to link the object code of the network to your application

14.12.3 Special Network Architectures

Normally, the architecture of the network and the numbers of the units are kept. Therefore a dummy unit with the number 0 is inserted in the array which contains the units. Some architectures are translated with other special features.

TDNN: Generally, a layer in a time delay neural network consists of feature units and their delay units. SNNS2C generates code only containing the feature units. The delay units are only additional activations in the feature unit. This is possible because every delay unit has the same link weights to it's corresponding source units as its feature unit.

So the input layer consists only of its prototype units, too. Therefore it's not possible to present the whole input pattern to the network. This is not necessary because it can be presented step by step to the inputs. This is useful for a real-time application, with the newest feature units as inputs. To mark a new sequence the *init*-flag (parameter of the function) can be set to 1. After this, the delays are filled when the init flag is set to 0 again. To avoid meaningless outputs the function returns NOT_VALID until the delays are filled again.

There is a new variable in the record of the header file for TDNNs. It is called "MinDelay" and is the minimum number of time steps which are needed to get a valid output after the *init*-flag was set.

CPN: Counterpropagation doesn't need the output layer. The output is calculated as a weighted sum of the activations of the hidden units. Because only one hidden unit has the activation 1 and all others the activation 0, the output can be calculated with the winner unit, using the weights from this unit to the output.

DLVQ: Here no output units are needed either. The output is calculated as the bias of the winner unit.

BPTT: If all inputs are set to zero, the net is not initialized. This feature can be chosen by setting the *init*-flag.

14.12.4 Activation Functions

Supported Activation Functions: Following activation functions are implemented in SNNS2C

Supported Activation Functions	
Act_Logistic	Act_StepFunc
Act_Elliott	Act_Identity
Act_BSB	Act_IdentityPlusBias
Act_TanH	Act_RBF_Gaussian
Act_TanHPlusBias	Act_RBF_MultiQuadratic
Act_TanH_Xdiv2	Act_RBF_ThinPlateSpline
Act_Perceptron	Act_TD_Logistic
Act_Signum	Act_TD_Elliott
Act_Signum0	

Including Own Activation Functions: The file "tools/sources/functions.h" contains two arrays: One array with the function names (ACT_FUNC_NAMES) and one for the macros which represent the function (ACT_FUNCTIONS). These macros are realized as character strings so they can be written to the generated C-source.

The easiest way to include an own activation function is to write the two necessary entries in the first position of the arrays. After that the constant "ActRbfNumber" should be increased. If a new Radial Basis function should be included, the entries should be appended at the end without increasing ActRbfNumber. An empty string ("") should still be the last entry of the array Act_FUNC_NAMES because this is the flag for the end of the array.

14.12.5 Error Messages

Here is the list of possible error messages and their brief description.

not enough memory: a dynamic memory allocation failed.

Note:
The C-code generated by snns2c that describes the network (units and links) defines

one unit type which is used for all units of the network. Therefore this unit type allocates as many link weights as necessary for the network unit with the most input connections. Since this type is used for every unit, the necessary memory space depends on the number of units times size of the biggest unit.

In most cases this is no problem when you use networks with moderate sized layers. But if you use a network with a very large input layer, your computer memory may be to small. For example, an $n - m - k$ network with $n > m > k$ needs about $n * (n + m + k) * (sizeof(float) + sizeof(void*))$ bytes of memory, so the necessary space is of $O(n^2)$ where n is the number of units of the biggest layer.

We are aware of this problem and will post an improved version of snns2c as soon as possible.

can't load file: SNNS network-file wasn't found.

can't open file: same as can't load or disk full.

wrong parameters: wrong kind or number of parameters.

net contains illegal cycles: there are several possibilities:

- a connection from a unit which is not a SPECIAL HIDDEN unit to itself.
- two layers are connected to each other when not exactly one of them is SPECIAL HIDDEN.
- cycles over more than two layers which don't match the architecture of Jordan networks.

BPTT-networks have no restrictions concerning links.

can't find the function *actfunc*: the activation function *actfunc* is not supported.

net is not a CounterPropagation network: Counterpropagation-networks need a special architecture: one input, one output and one hidden layer which are fully connected.

net is not a Time Delay Neural Network: The SNNS TDNNs have a very special architecture. They should be generated by the BIGNET-tool. In other cases there is no guaranty for successful compilation.

not supported network type: There are several network types which can't be compiled with the SNNS2C. The SNNS2C-tool will be maintained and so new network types will be implemented.

unspecified Error: should not occur, it's only a feature for user defined updates.

unknown error code : *number*: must not occur, please report to the SNNS-Team.

Chapter 15

Kernel Function Interface

15.1 Overview

The simulator kernel offers a variety of functions for the creation and manipulation of networks. These can roughly be grouped into the following categories:

- functions to manipulate the network
- functions to determine the structure of the network
- functions to define and manipulate cell prototypes
- functions to propagate the network
- learning functions
- functions to manipulate patterns
- functions to load and save the network and pattern files
- functions for error treatment, search functions for names, functions to change default values etc.

The following paragraphs explains the interface functions in detail. All functions of this interface between the kernel and the user interface carry the prefix `krui_`...(kernel user interface functions).

Additionally there are some interface functions which are useful to build applications for ART networks. These functions carry the prefix `artui_`...(ART user interface functions).

15.2 Unit Functions

The following functions are available for manipulation of the cells and their components:

```
krui_getNoOfUnits()
krui_getNoOfSpecialUnits()
```

```
krui_getFirstUnit()
krui_getNextUnit()
krui_setCurrentUnit( int  UnitNo )
krui_getCurrentUnit()
krui_getUnitName( int  UnitNo )
krui_setUnitName( int  UnitNo, char *unit_name )
krui_searchUnitName( char  *unit_name )
krui_searchNextUnitName( void )
krui_getNoOfTTypeUnits()
krui_getUnitOutFuncName( int  UnitNo )
krui_setUnitOutFunc( int  UnitNo, char *unitOutFuncName )
krui_getUnitActFuncName( int  UnitNo )
krui_setUnitActFunc( int  UnitNo, char *unitActFuncName )
krui_getUnitFTypeName( int UnitNo )
krui_getUnitActivation( int  UnitNo )
krui_setUnitActivation( int  UnitNo, FlintType  unit_activation )
krui_getUnitInitialActivation( int  UnitNo )
krui_setUnitInitialActivation( int  UnitNo, FlintType unit_i_activation)
krui_getUnitOutput( int  UnitNo )
krui_setUnitOutput( int  UnitNo, FlintType  unit_output )
krui_getUnitBias( int  UnitNo )
krui_setUnitBias( int  UnitNo, FlintType  unit_bias )
krui_getUnitSubnetNo( int UnitNo )
krui_setUnitSubnetNo( int UnitNo, int  subnet_no)
krui_getUnitLayerNo( int UnitNo )
krui_setUnitLayerNo( int UnitNo, unsigned short layer_bitField )
krui_getUnitPosition( int  UnitNo, struct PosType  *position )
krui_setUnitPosition( int  UnitNo, struct PosType  *position )
krui_getUnitNoAtPosition( struct PosType  *position, int subnet_no )
krui_getUnitNoNearPosition( struct PosType  *position, int subnet_no,
                            int range, int gridWidth )
krui_getXYTransTable( struct TransTable  * *xy_trans_tbl_ptr )
krui_getUnitCenters( int unit_no, int center_no,
                     struct PositionVector  * *unit_center )
krui_setUnitCenters( int unit_no, int center_no,
                     struct PositionVector  *unit_center )
krui_getUnitTType( int  UnitNo )
krui_setUnitTType( int  UnitNo, int  UnitTType )
krui_freezeUnit( int  UnitNo )
krui_unfreezeUnit( int  UnitNo )
krui_isUnitFrozen( int  UnitNo )
krui_getUnitInputType( UnitNo )
krui_getUnitValueA(int UnitNo)
krui_setUnitValueA(int UnitNo, FlintTypeParam unit_valueA)
krui_createDefaultUnit()
krui_createUnit( char  *unit_name, char  *out_func_name,
                 char  *act_func_name, FlintType  act,
```

```
                  FlintType  i_act, FlintType  out,
                  FlintType  bias)
krui_createFTypeUnit( char  *FType_name)
krui_setUnitFType( int  UnitNo, char *FTypeName )
krui_copyUnit( int  UnitNo, int copy_mode )
krui_deleteUnitList( int no_of_units, int unit_list[] )
```

Unit Enquiry and Manipulation Functions

`int  krui_getNoOfUnits()`
determines the number of units in the neural net.

`int  krui_getNoOfSpecialUnits()`
determines the number of special units in the neural net.

`int  krui_getFirstUnit()`
Many interface functions refer to a current unit or site. `krui_getFirstUnit()` selects the (chronological) first unit of the network and makes it current. If this unit has sites, the chronological first site becomes current. The function returns 0 if no units are defined.

`int  krui_getNextUnit()`
selects the next unit in the net, as well as its first site (if present); returns 0 if no more units exist.

`krui_err  krui_setCurrentUnit( int  UnitNo )`
makes the unit with number *UnitNo* current unit; returns an error code if no unit with the specified number exists.

`int  krui_getCurrentUnit()`
determines the number of the current unit (0 if not defined)

```
char     *krui_getUnitName( int  UnitNo )
krui_err  krui_setUnitName( int  UnitNo, char *unit_name )
```
determines/sets the name of the unit. `krui_setUnitName` returns NULL if no unit with the specified number exists.

`int  krui_searchUnitName( char  *unit_name )`
searches for a unit with the given name. Returns the first unit number if a unit with the given name was found, 0 otherwise.

`int  krui_searchNextUnitName( void )`
searches for the next unit with the given name. Returns the next unit number if a unit with the given name was found, 0 otherwise. `krui_searchUnitName( unit_name )` has to be called before at least once, to confirm the unit name. Returns error code if no units are defined.

```
char  *krui_getUnitOutFuncName( int  UnitNo )
char  *krui_getUnitActFuncName( int  UnitNo )
```
determines the output function resp. activation function of the unit.

```
krui_err  krui_setUnitOutFunc( int  UnitNo, char *unitOutFuncName )
krui_err  krui_setUnitActFunc( int  UnitNo, char *unitActFuncName )
```
sets the output function resp. activation function of the unit. Returns an error code if the function name is unknown, i.e. if the name does not appear in the function table as output or activation function. The f-type of the unit is deleted.

```
char  *krui_getUnitFTypeName( int UnitNo )
```
yields the f-type of the unit; returns NULL if the unit has no prototype.

```
FlintType  krui_getUnitActivation( int UnitNo )
krui_err   krui_setUnitActivation( int UnitNo,
                                   FlintTypeParam unit_activation )
```
returns/sets the activation of the unit.

```
FlintType krui_getUnitInitialActivation( int  UnitNo )
void      krui_setUnitInitialActivation( int  UnitNo,
                                         FlintType unit_i_activation )
```
returns/sets the initial activation of the unit, i.e. the activation after loading the net. See also `krui_resetNet()`.

```
FlintType  krui_getUnitOutput( int  UnitNo )
krui_err   krui_setUnitOutput( int  unit_no, FlintTypeParam  unit_output )
```
returns/sets the output value of the unit.

```
FlintType  krui_getUnitBias( int  UnitNo )
void  krui_setUnitBias( int  UnitNo, FlintType  unit_bias )
```
returns/sets the bias (threshold) of the unit.

```
int   krui_getUnitSubnetNo( int UnitNo )
void  krui_setUnitSubnetNo( int UnitNo, int  subnet_no)
```
returns/sets the subnet number of the unit (the range of subnet numbers is -32736 to +32735).

```
unsigned short  krui_getUnitLayerNo( int UnitNo )
void            krui_setUnitLayerNo( int UnitNo,int layer_no)
```
returns/sets the layer number (16 Bit integer).

```
void  krui_getUnitPosition( int  UnitNo, struct PosType  *position )
void  krui_setUnitPosition( int  UnitNo, struct PosType  *position )
```
determines/sets the (graphical) position of the unit. See also include file `glob_typ.h` for the definition of PosType.

```
int  krui_getUnitNoAtPosition( struct PosType  *position, int subnet_no )
```
yields the unit number of a unit with the given position and subnet number; returns 0 if no such unit exists.

```
int  krui_getUnitNoNearPosition( struct PosType  *position,
                                 int subnet_no, int range,
                                 int gridWidth )
```
yields a unit in the surrounding (defined by *range*) of the given position with the given graphic resolution *gridWidth*; otherwise like `krui_getUnitNoAtPosition(...)`.

```
krui_err krui_getUnitCenters( int unit_no, int center_no,
                              struct PositionVector  * *unit_center )
```
returns the 3D-transformation center of the specified unit and center number. Function has no effect on the current unit. Returns error number if unit or center number is invalid or if the SNNS-kernel isn't a 3D-kernel.

```
krui_err krui_setUnitCenters( int unit_no, int center_no,
                              struct PositionVector  *unit_center )
```
sets the 3D-transformation center and center number of the specified unit. Function has no effect on the current unit. Returns error number if unit or center number is invalid or if the SNNS-kernel isn't a 3D-kernel.

```
krui_err krui_getXYTransTable( dummy )
```
returns the base address of the XY-translation table. Returns error code if the SNNS-kernel isn't a 3D-kernel.

```
int       krui_getUnitTType( int  UnitNo )
krui_err  krui_setUnitTType( int  UnitNo, int  UnitTType )
```
gets/sets the IO-type[1] (i.e. input, output, hidden) of the unit. See include file `glob_typ.h` for IO-type constants. `Set` yields an error code if the IO-type is invalid.

```
krui_err  krui_freezeUnit( int  unit_no )
```
freezes the output and the activation value of the unit, i.e. these values are not updated anymore.

```
krui_err  krui_unfreezeUnit( int  unit_no )
```
switches the computation of output and activation values on again.

```
bool  krui_isUnitFrozen( int  unit_no )
```
yields TRUE if the unit is frozen, else FALSE.

```
int  krui_getUnitInputType( UnitNo )
```
yields the input type. There are three kinds of input types:

`NO_INPUTS`: the unit doesn't have inputs (yet).

`SITES`: the unit has one or more sites (and therefore no direct inputs).

`DIRECT_LINKS`: the unit has direct inputs (and no sites).

See also file `glob_typ.h`.

```
FlintType krui_getUnitValueA(int UnitNo)
void      krui_setUnitValueA(int UnitNo, FlintTypeParam unit_valueA)
```
returns and sets the Value A field of the unit structure.

[1]The term T-type was changed to IO-type after completion of the kernel

Unit Definition Functions

`int krui_createDefaultUnit()`
creates a unit with the properties of the (definable) default values of the kernel. The default unit has the following properties:

- standard activation and output function
- standard activation and bias
- standard position-, subnet-, and layer number
- default IO type
- no unit prototype
- no sites
- no inputs or outputs
- no unit name

Returns the number of the new unit or a (negative) error code. See also include file `kr_def.h`.

```
int  krui_createUnit( char  *unit_name, char  *out_func_name,
                      char  *act_func_name, FlintTypeParam i_act,
                      FlintTypeParam  bias)
```
creates a unit with selectable properties; otherwise like `krui_createDefaultUnit()`. There are the following defaults:

- standard position-, subnet-, and layer number
- default IO type
- no unit prototype
- no sites
- no inputs or outputs

Returns the number of the new unit or a (negative) error code. See also include file `kr_def.h`.

`int krui_createFTypeUnit( char *FType_name)`
creates a unit with the properties of the (previously defined) prototype. It has the following default properties:

- standard position number, subnet number and layer number
- no inputs or outputs

The function returns the number of the new unit or a (negative) error code.

`krui_err krui_setUnitFType( int UnitNo, char *FTypeName )`
changes the structure of the unit to the intersection of the current type of the unit with the prototype; returns an error code if this operation has been failed.

`int krui_copyUnit( int UnitNo, int copy_mode)`
copies a unit according to the copy mode. Four different copy modes are available:

- copy unit with all input and output connections
- copy only input connections

- copy only output connections
- copy only the unit, no connections

Returns the number of the new unit or a (negative) error code. See `glob_typ.h` for reference of the definition of constants for the copy modes.

`krui_err krui_deleteUnitList( int no_of_units, int unit_list[] )`
deletes 'no_of_units' from the network. The numbers of the units that have to be deleted are listed up in an array of integers beginning with index 0. This array is passed to parameter 'unit_list'. Removes all links to and from these units.

15.3 Site Functions

Before input functions (sites) can be set for units, they first have to be defined. To define it, each site is assigned a name by the user. Sites can be selected by using this name. For the definition of sites, the following functions are available:

```
krui_createSiteTableEntry( char  *site_name, char  *site_func )
krui_changeSiteTableEntry( char  *old_site_name, char  *new_site_name,
                           char  *new_site_func )
krui_deleteSiteTableEntry( char *site_name )
krui_getFirstSiteTableEntry( char * *site_name, char * *site_func )
krui_getNextSiteTableEntry( char * *site_name, char * *site_func )
krui_getSiteTableFuncName( char  *site_name )
krui_setFirstSite( void )
krui_setNextSite( void )
krui_setSite( char  *site_name )
krui_getSiteValue()
krui_getSiteName()
krui_setSiteName( char  *site_name )
krui_getSiteFuncName()
krui_addSite( char *site_name )
krui_deleteSite()
```

Functions for the Definition of Sites

`krui_err  krui_createSiteTableEntry( char  *site_name, char  *site_func )`
defines the correspondence between site function and name of the site. Error codes are generated for site names already used, invalid site functions, or problems with the memory allocation.

```
krui_err  krui_changeSiteTableEntry( char  *old_site_name,
                                     char  *new_site_name,
                                     char  *new_site_func )
```
changes the correspondence between site function and name of the site. All sites in the network with the name `old_site_name` change their name and function. Error codes are

generated for already defined site names, invalid new site function, or problems with the memory allocation.

`krui_err  krui_deleteSiteTableEntry( char *site_name )`
deletes a site in the site table. This is possible only if there exist no sites in the network with that name. Returns an error code if there are still sites with this name in the net.

`bool  krui_getFirstSiteTableEntry( char * *site_name, char * *site_func )`
`bool  krui_getNextSiteTableEntry ( char * *site_name, char * *site_func )`
returns the first/next pair of site name and site function. The return code is TRUE if there is (still) an entry in the site table, else FALSE.

`char  *krui_getSiteTableFuncName( char  *site_name )`
returns the name of the site function assigned to the site. If no site with this name exists, a pointer to NULL is returned.

Functions for the Manipulation of Sites

`bool  krui_setFirstSite( void )`
initializes the first site of the current unit, i.e. the first site of the current unit becomes current site. If the current unit doesn't have sites, FALSE is returned, else TRUE.

`bool  krui_setNextSite( void )`
initializes the next site of the current unit. If the unit doesn't have more sites, FALSE is returned.

`krui_err  krui_setSite( char  *site_name )`
initializes the given site of the current unit. An error code is generated if the unit doesn't have sites, the site name is invalid, or the unit doesn't have a site with that name.

`FlintType  krui_getSiteValue()`
`char      *krui_getSiteFuncName()`
returns the name/value of the site function of the current site.

`char  *krui_getSiteName()`
returns the name of the current site.

`krui_err  krui_setSiteName( char  *site_name )`
changes the name (and thereby also the site function) of the current site. An error code is returned if the site name is unknown. The f-type of the unit is erased.

`krui_err krui_addSite( char *site_name )`
adds a new site to the current unit. The new site is inserted in front, i.e. it becomes the first site of the unit. Therefore it is possible to make the new site current by a call to `krui_setFirstSite()`. `krui_addSite(...)` has no effect on the current site! Error codes are generated if the unit has direct input connections, the site name is invalid, or problems with the memory allocation occurred. The functionality type of the unit will be cleared.

`bool krui_deleteSite()`
deletes the current site of the current unit and all input connections to that site. The functionality type of the unit is also erased. `krui_setFirstSite()` or `krui_setNextSite()` has to be called before at least once, to confirm the current site/unit. After the deletion the next available site becomes current. The return code is TRUE if further sites exist, else FALSE. The following program is sufficient to delete all sites of a unit:

```
if ( krui_setFirstSite() )
  while ( krui_deleteSite() ) { }
```

15.4 Link Functions

The following functions are available to define or determine the topology of the network:

```
krui_getFirstPredUnit( FlintType  *strength )
krui_getNextPredUnit( FlintType   *strength )
krui_getCurrentPredUnit( FlintType  *strength )
krui_getFirstSuccUnit( int  UnitNo, FlintType  *strength )
krui_getNextSuccUnit( FlintType  *strength )
krui_isConnected( int  source_unit_no )
krui_areConnected( int source_unit_no, int target_unit_no,
                   FlintType  *weight )
krui_getLinkWeight()
krui_setLinkWeight( FlintTypeParam  weight )
krui_createLink( int source_unit_no, FlintTypeParam  weight )
krui_deleteLink()
krui_deleteAllInputLinks()
krui_deleteAllOutputLinks()
krui_jogWeights( FlintTypeParam  minus, FlintTypeParam  plus)
```

`int krui_getFirstPredUnit( FlintType *strength )`
determines the unit number of the predecessor unit of the current unit and site; returns 0 if no such unit exists, i.e. if the current unit has no inputs. If a predecessor unit exists, the connection between the two units becomes current and its strength is returned.

`int krui_getNextPredUnit( FlintType *strength )`
gets another predecessor unit of the current unit/site (returns 0 if no more exist). Otherwise like `krui_getFirstPredUnit(...)`.

`int krui_getCurrentPredUnit( FlintType *strength )`
yields the current predecessor unit.

`int krui_getFirstSuccUnit( int UnitNo, FlintType *strength )`
yields unit number and connection strength of the first successor unit to the current unit. The return code is 0 if no such unit exists, i.e. the current unit has no outputs. If a successor unit exists, the connection between the two units becomes current. If the successor unit has sites, the site connected with this link becomes current site. The function is slow, because

the units are connected only backwards (lookup time is proportional to the number of connections in the net).

`int  krui_getNextSuccUnit( FlintType  *strength )`
gets another successor unit of the current unit (returns 0 if no other successors exist). Otherwise like `krui_getFirstSuccUnit(...)`. The function is slow, because the units are only connected backwards.

`bool  krui_isConnected( int  source_unit_no )`
checks, whether there is a connection between the current unit and the source unit. If this is true, this link becomes current and TRUE is returned.

```
bool  krui_areConnected( int source_unit_no, int target_unit_no,
                         FlintType  *weight )
```
checks, whether there is a connection between the source unit and the target unit. In contrast to `krui_isConnected(...)` this function traverses sites during the search. If there is such a connection, this link becomes current and TRUE is returned.

```
FlintType  krui_getLinkWeight()
void       krui_setLinkWeight( FlintTypeParam  weight )
```
determines/sets the connection weight of the current link.

`krui_err  krui_createLink( int source_unit_no, FlintTypeParam weight )`
creates a new link between the current unit/site and the source unit. An error code is generated if a link between these two units already exists, or if the source unit does not exist.

`krui_err  krui_deleteLink()`
deletes the current link. To delete a connection between the current unit/site and the source unit a sequence of `krui_isConnected( source_unit_no )` and `krui_deleteLink()` is ideal.

```
krui_err  krui_deleteAllInputLinks()
krui_err  krui_deleteAllOutputLinks()
```
deletes all inputs/outputs at the current unit/site.

`void krui_jogWeights( FlintTypeParam minus, FlintTypeParam plus)`
adds uniform distributed random values to the connection weights of the network. Minus must be less then plus. See also `krui_setSeedNo(...)`.

15.5 Functions for the Manipulation of Prototypes

By describing the characteristic properties of units, like activation/output function and sites, the user can define unit prototypes (called f-types in SNNS). Thereby the user can create a library of units. It is a big advantage that each change to the prototypes in the library affects all units of this f-type in the whole network. This means, that all units of a certain type are updated with a change in the library. With the following functions prototypes can be defined and manipulated:

```
krui_setFirstFTypeEntry()
krui_setNextFTypeEntry()
krui_setFTypeEntry( char  *Ftype_symbol )
krui_getFTypeName()
krui_setFTypeName( char  *unitFType_name )
krui_getFTypeActFuncName()
krui_setFTypeActFunc( char  *act_func_name )
krui_getFTypeOutFuncName()
krui_setFTypeOutFunc( char  *out_func_name )
krui_setFirstFTypeSite()
krui_setNextFTypeSite()
krui_getFTypeSiteName()
krui_setFTypeSiteName( char  *FType_site_name )
krui_createFTypeEntry( char *FType_symbol,char *act_func,
                       char *out_func, int no_of_sites,
                       char * *array_of_site_names )
krui_deleteFTypeEntry( char *  FType_symbol )
```

`bool  krui_setFirstFTypeEntry()`
`bool  krui_setNextFTypeEntry()`
initializes the first/next prototype and makes it current. The return code is FALSE if no unit types are defined, otherwise TRUE.

`bool  krui_setFTypeEntry( char *Ftype_symbol )`
selects a prototype by a name and returns TRUE if the name exists.

`char  *krui_getFTypeName()`
determines the name of the current prototype.

`krui_err  krui_setFTypeName( char  *unitFType_name )`
changes the name of the current prototype. The name has to be unambiguous, i.e. all names have to be different. If the name is ambiguous, or if memory allocation failed, an error code is returned.

`char  *krui_getFTypeActFuncName()`
determines the name of the activation function of the current prototype.

`krui_err  krui_setFTypeActFunc( char * act_func_name )`
changes the activation function of the current prototype; returns an error code if the given function is not a valid activation function. All units of the net that are derived from this prototype change their activation function.

`char  *krui_getFTypeOutFuncName()`
determines the name of the output function of the current prototype.

`krui_err  krui_setFTypeOutFunc( char  *out_func_name )`
changes the output function of the current prototype; returns an error code if the given function is not a valid output function. All units of the net that are derived from this prototype change their output function.

`bool  krui_setFirstFTypeSite()`
selects the first site of the prototype (this site becomes current prototype site); returns TRUE if the prototype has sites.

`bool  krui_setNextFTypeSite()`
selects the next site of the prototype (this site becomes current prototype site); returns TRUE if the prototype has more sites.

`char  *krui_getFTypeSiteName()`
determines the name of the current prototype site.

`krui_err  krui_setFTypeSiteName( char *FType_site_name )`
changes the name (and also the site function) of the current prototype site. All units of the net that are derived from this prototype change their site names or site functions. An error code is generated if the new site name is not yet defined.

```
krui_err krui_createFTypeEntry( char *FType_symbol,char *act_func,
                                char *out_func, int no_of_sites,
                                char * *array_of_site_names )
```
defines the prototype of a unit. If the prototype is supposed to have sites, i.e. `no_of_sites > 0`, an array of pointers to strings is needed to define the sites. These pointers must point to strings containing the names of the sites. The number of pointers in the arrays must match `no_of_sites`. An error code is generated if the site names are ambiguous, one or several site names are unknown, the prototype name is ambiguous, or a memory allocation error occurred.

`void  krui_deleteFTypeEntry( char  *FType_symbol )`
deletes the specified prototype. If there are still units in the net which are derived from this prototype, they loose their unit type.

15.6 Functions to Read the Function Table

`int  krui_getNoOfFunctions()`
determines the number of available functions (unit, site, and learning functions).

```
void  krui_getFuncInfo( int  func_no,
                        char * *func_name,
                        int  *func_type )
```
determines the name and type (activation, output, site, or learning function) from the function table. See include file `glob_typ.h` for the definition of the function types.

`bool  krui_isFunction( char  *func_name, int  func_type )`
returns TRUE if the specified function is a valid function.

```
bool krui_getFuncParamInfo( char *func_name, int  func_type,
                            int *no_of_input_params,
                            int *no_of_output_params )
```
returns the number of input and output parameters of the given learning, update or initialization function. Returns TRUE if the given function exists, FALSE otherwise.

15.7 Network Initialization Functions

`krui_err  krui_setInitialisationFunc( char  *init_func )`
changes the initialization function; returns an error code if the initialization function is unknown.

`char  *krui_getInitialisationFunc( void )`
returns the current initialization function. The default initialization function is *'Randomize_Weights'* (see also `kr_def.h`).

`krui_err  krui_initializeNet( float *parameterArray, int  NoOfParams )`
initializes the network with the current initialization function.

15.8 Functions for Activation Propagation in the Network

`krui_err  krui_updateSingleUnit( int  UnitNo )`
evaluates the net input, the activation and the output value of the specified unit; returns an error code if the unit doesn't exist. `krui_updateSingleUnit(...)` also evaluates 'frozen' units.

`char  *krui_getUpdateFunc( void )`
returns the current update function. The default update function is *'Serial_Order'* (see also `kr_def.h`).

`krui_err  krui_setUpdateFunc( char  *update_func )`
Changes the current update function; returns an error code if the update function is invalid.

`krui_err  krui_updateNet( float *parameterArray, int  NoOfParams )`
updates the network according to the update function. The network should be a feed-forward type if one wants to update the network with the topological update function, otherwise the function returns a warning message. To propagate a pattern through the network the use of the following function calls is recommended:

```
krui_setPatternNo( pat_no );
krui_showPattern( OUTPUT_NOTHING );
krui_updateNet( parameterArray, NoOfParams );
```

See also `krui_setSeedNo` for initializing the pseudo random number generator. The function returns an error code if an error occurred. The following update functions are available:

- synchronous firing: the units of the network all change their activation at the same time.
- chronological order: the user defines the order in which the units change their activations.
- random order: the activations are computed in random order. It may happen that some units are updated several times while others are not updated at all.
- random permutation: the activations of all units are computed exactly once, but in a random order.
- topological order: the units change their activations according to their topological order. This mode should be selected only with nets that are free of cycles (feedforward nets).

The topological order propagation method computes the stable activation pattern of the net in just one cycle. It is therefore the method of choice in cycle free nets. In other modes, depending upon the number of layers in the network, several cycles are required to reach a stable activation pattern (if this is possible at all).

15.9 Learning and Pruning Functions

```
krui_err  krui_setLearnFunc( char *learning_function)
krui_err  krui_setFFLearnFunc( char *FF_learning_function)
krui_err  krui_setPrunnFunc( char *pruning_function)
```

selects the learning / subordinate learning / pruning function; returns an error code if the given function is unknown.

```
char  *krui_getLearnFunc( void )
char  *krui_getFFLearnFunc( void )
char  *krui_getPrunFunc( void )
```

returns the name of the current learning / subordinate learning / pruning function. The default learning / subordinate learning function is *'Std_Backpropagation'*, the default pruning function is *'MagPruning'* (see also `kr_def.h`).

```
krui_err  krui_learnAllPatterns( float *parameterInArray,
                                 int  NoOfInParams,
                                 float  * *parameterOutArray,
                                 int *NoOfOutParams )
krui_err  krui_learnAllPatternsFF( float *parameterInArray,
                                   int  NoOfInParams,
                                   float  * *parameterOutArray,
                                   int *NoOfOutParams )
```

learns all patterns (each consisting of an input/output pair) using the current learning /

subordinate learning function. parameterInArray contains the learning parameter(s). NoOfInParams stores the number of learning parameters. parameterOutArray returns the results of the learning function. This array is a static array defined in the learning function. *NoOfOutParams points to an integer value that contains the number of output parameters from the current learning function. The function returns an error code if memory allocation has failed or if the learning has failed. Patterns must be loaded before calling this function.

```
krui_err  krui_learnSinglePattern( int  pattern_no,
                                   float   *parameterInArray,
                                   int NoOfInParams,
                                   float  * *parameterOutArray,
                                   int *NoOfOutParams )
krui_err  krui_learnSinglePatternFF( int  pattern_no,
                                     float   *parameterInArray,
                                     int NoOfInParams,
                                     float  * *parameterOutArray,
                                     int *NoOfOutParams )
```

same as krui_learnAllPatterns(...) or krui_learnAllPatternsFF(...) respectively, but teaches only the current pattern.

15.10 Functions for the Manipulation of Patterns

`krui_err  krui_setPatternNo( int  patter_no )`
sets the current pattern; returns an error code if the pattern number is invalid.

`krui_err  krui_deletePattern( void )`
deletes the current pattern.

`krui_err  krui_modifyPattern( void )`
modifies the current pattern. Sets the pattern to the current activation of the units.

`krui_err  krui_showPattern( int  mode )`
outputs a pattern on the activation or output values of the input/output units. The following modes are possible:

- OUTPUT_NOTHING: stores the input pattern in the activation of the input units.
- OUTPUT_ACT: like OUTPUT_NOTHING, but stores also the output pattern in the activation of the output units.
- OUTPUT_OUT: like OUTPUT_ACT, additionally a new output value of the output units is computed.

`krui_showPattern(...)`
draws pattern on the display. Generates an error code if the number of input and output units does not correspond with the previously loaded pattern. The constants of the various modes are defined in glob_typ.h.

`krui_err  krui_newPattern( void )`
creates a new pattern (an input/output pair). A pattern can be created by modifying the activation value of the input/output units. The function returns an error code if there is insufficient memory or the number of input/output units is incompatible with the pattern.

Note: `krui_newPattern()` switches pattern shuffling off. For shuffling the new patterns call:

```
krui_newPattern(...)
krui_shufflePatterns( TRUE )
```

`void  krui_deleteAllPatterns()`
deletes all previously defined patterns in main memory.

`krui_err krui_shufflePatterns( bool on_or_off )`
shuffles the order of the patterns if `on_or_off` is true. If `on_or_off` is false the original order can be restored. See also `krui_setSeedNo(...)`.

`krui_err  krui_shuffleSubPatterns(bool on_or_off)`
shuffles sub pattern pairs by using pseudo random generator.
`krui_shuffleSubPatterns(TRUE)` switches shuffeling of sub patterns on,
`krui_shuffleSubPatterns(FALSE)` switches shuffeling of sub patterns off.
The default presetting is `krui_shuffleSubPatterns (FALSE)`.

`int  krui_getNoOfPatterns( void )`
returns the actual number of patterns (0 if no patterns have been loaded).

`int  krui_getTotalNoOfSubPatterns( void )`
returns the total number of subpatterns contained in all patterns of the current pattern set (0 if no patterns have been loaded).

`krui_err  krui_allocNewPatternSet(int *set_no)`
returns the number of the allocated pattern set. In case of an error the error code will be returned.

`krui_err krui_setCurrPatSet(int number)`
chooses the number of the current pattern set. The number ranges from 0 to n-1.

`krui_err  krui_deletePatSet(int number)`
deletes all patterns of the pattern set with the given number.

```
krui_err  krui_GetPatInfo(pattern_set_info *set_info,
                          pattern_descriptor *pat_info)
```
gets all available information concerning the current pattern set and the current pattern.

```
krui_err  kruiDefShowSubPat(int *insize,int *outsize,int,
                            int *inpos,int *outpos)
```
defines the sub pattern that will be shown with the next call of `krui_showPattern`.

```
krui_err  krui_DefTrainSubPat(int *insize,int *outsize,int *instep,
                              int *outstep, int *max_n_pos)
```
defines how sub pattern should be generated during training. `krui_DefTrainSubPat()` has to be called before any training can take place.

```
krui_err   krui_AlignSubPat(int *inpos,int *outpos,int *no)
```
aligns the given sub pattern position to a valid position which fits the defined sub pattern training scheme (`krui_DefTrainSubPat`).

```
krui_err  krui_GetShapeOfSubPattern(int *insize,int *outsize,int *inpos,
                                    int *outpos,int *n_pos)
```
gets the shape of the sub pattern by using the current set,current pattern and the current train scheme (defined with `krui_DefTrainSubPat`).

15.11 File I/O Functions

```
krui_err  krui_loadNet( char  *filename, char  * *netname )
krui_err  krui_saveNet( char  *filename, char  *netname)
```
loads/saves a network from/to disk and generates an internal network structure.

```
krui_err krui_loadNewPatterns(char *filename,int *set_no)
krui_err  krui_saveNewPatterns(char *filename,int set_no)
```
saves and loads a pattern file with new style conventions.
The pattern set with the given number will be saved and loaded.

```
krui_err krui_saveResultParam(char *filename,bool create,int startpattern,
                       int endpattern,bool includeinput,bool includeoutput,
                       float *Update_param_array, int NoOfUpdateParam)
```
Saves the network result which depends on the loaded patterns. If create is false,the new file will be appended to an existing file. Startpattern and endpattern determines the range of patterns to use. The input patterns and the teaching output patterns can be included.

An error code is returned if an error occurred during the execution of the I/O functions.

15.12 Functions to Search the Symbol Table

```
bool  krui_getFirstSymbolTableEntry( char * *symbol_name,
                                    int *symbol_type )
bool  krui_getNextSymbolTableEntry( char * *symbol_name,
                                    int *symbol_type )
```
determines the name and type of the first/next entry in the symbol table and returns TRUE if another entry exists.

```
bool  krui_symbolSearch( char *symbol, int  symbol_type)
```
returns TRUE if the given symbol exists in the symbol table.

15.13 Miscelaneous other Interface Functions

```
char  *krui_getVersion()
```
determines the version number of the SNNS kernel.

```
void  krui_getNetInfo( int  *no_of_sites, int  *no_of_links,
                       int  *no_of_STable_entries,
                       int  *no_of_FTable_entries )
```
gathers various information about the network.

```
void  krui_getUnitDefaults( FlintType  *act, FlintType  *bias,
                            int *io_type, int *subnet_no, int *layer_no,
                            char * *act_func, char * *out_func)
```
determines the default values for generating units. See also `krui_createDefaultUnit()` and `krui_createFTypeUnit(...)`.

```
krui_err  krui_setUnitDefaults( FlintTypeParam act, FlintTypeParam bias,
                                int io_type, int subnet_no, int layer_no,
                                char *act_func, char *out_func)
```
changes the default values; returns an error code if the IO-type or the activation/output function is unknown.

```
void  krui_setSeedNo( long seed )
```
initializes the random number generator. If `seed = 0`, the random number generator is re-initialized (this time really at random!).

```
int  krui_getNoOfInputUnits()
int  krui_getNoOfOutputUnits()
```
return the number of input/output units

```
int  krui_getNoOfSpecialInputUnits()
int  krui_getNoOfSpecialOutputUnits()
```
return the number of special input/output units

```
void  krui_resetNet()
```
sets the activation values of all units to their respective defaults.

15.14 Memory Management Functions

```
krui_err  krui_allocateUnits( int  number )
```
reserves <number> units in memory. Additional units can be requested by multiple calls to that function. This function doesn't have to be called, since the SNNS kernel always reserves enough memory for units, sites and links. If a large amount of units is needed, however, a call to `krui_allocateUnits(...)` eases the administration of system resources. If `krui_allocateUnits(...)` is never called, units are always requested in blocks of size <UNIT_BLOCK>. See also `kr_def.h`.

```
void krui_getMemoryManagerInfo(int *unit_bytes, int *site_bytes,
                               int *link_bytes, int *NTable_bytes,
                               int *STable_bytes, int *FTable_bytes)
```
determines the number of allocated (not the number of used) bytes for the various components.

`void krui_deleteNet()`
deletes the network and frees the whole memory used for the representation of the data structures.

15.15 ART Interface Functions

The functions described in this paragraph are only useful if you use one of the ART models ART1, ART2 or ARTMAP. They are additional functions, not replacing any of the kernel interface functions described above. To use them, you have to include the files `art_ui.h` and `art_typ.h` in your application.

`krui_err artui_getClassifiedStatus( art_cl_status *status )`
returns the actual classification state of an ART network in parameter `status`. Type `art_cl_status` is described above. An SNNS error code is returned as function value. The function can be used for ART1, ART2 and ARTMAP models.

`krui_err artui_getClassNo( int *class_no )`
returns a number between 1 and M in parameter `class_no`, which is either the index of the actual winner unit in the F_2 layer of an ART1 or ART2 network or the one of the activated MAP unit in an ARTMAP network. It will return -1 as `class_no` if no actual class is active. An SNNS error code is returned as function value.

`krui_err  artui_getN (int *N )`
determines the number of F1-units in an ART1 or ART2 network.

`krui_err  artui_getM (int *M )`
determines the number of F2-units in an ART1 or ART2 network.

`krui_err  artui_getNa (int *Na )`
determines the number of F1a-units in an ARTMAP network.

`krui_err  artui_getMa (int *Ma )`
determines the number of F2a-units in an ARTMAP network

`krui_err  artui_getNb (int *Nb)`
determines the number of F1b-units in an ARTMAP network.

`krui_err  artui_getMb (int *Mb)`
determines the number of F2b-units in an ARTMAP network.

15.16 Error Messages of the Simulator Kernel

Most interface functions return an error code if the parameters contradict each other, or if an error occurred during execution. If no error occurred during execution of a kernel interface function, the function returns the code: KRERR_NO_ERROR. KRERR_NO_ERROR is equal to 0. The simulator kernel can generate 54 different error messages. The error code constants are defined in glob_typ.h. There is also a function to translate an error code into text:

```
char  *krui_error( int  error_code )
```

converts an error code to a string. The following error messages are used:

KRERR_NO_ERROR:	No Error
KRERR_INSUFFICIENT_MEM:	Insufficient memory
KRERR_UNIT_NO:	Invalid unit number
KRERR_OUTFUNC:	Invalid unit output function
KRERR_ACTFUNC:	Invalid unit activation function
KRERR_SITEFUNC:	Invalid site function
KRERR_CREATE_SITE:	Creation of sites is not permitted because unit has direct input links
KRERR_ALREADY_CONNECTED:	Creation of a link is not permitted because there exists already a link between these units
KRERR_CRITICAL_MALLOC:	allocation failed in critical operation.
KRERR_FTYPE_NAME:	Ftype name is not definite
KRERR_FTYPE_ENTRY:	Current Ftype entry is not defined
KRERR_COPYMODE:	Invalid copy mode
KRERR_NO_SITES:	Current unit does not have sites
KRERR_FROZEN:	Can't update unit because unit is frozen
KRERR_REDEF_SITE_NAME:	Redefinition of site name not permitted (site name already exists)
KRERR_UNDEF_SITE_NAME:	Site name is not defined
KRERR_NOT_3D:	Invalid Function: Not a 3D-Kernel
KRERR_DUPLICATED_SITE:	Unit has already a site with this name
KRERR_INUSE_SITE:	Can not delete site table entry because site is in use
KRERR_FTYPE_SITE:	Current Ftype site is not defined
KRERR_FTYPE_SYMBOL:	Given symbol is not defined in the symbol table
KRERR_IO:	Physical I/O error
KRERR_SAVE_LINE_LEN:	Creation of output file failed (line length, limit exceeded)
KRERR_NET_DEPTH:	The depth of the network does not fit the learning function
KRERR_NO_UNITS:	No Units defined
KRERR_EOF:	Unexpected EOF
KRERR_LINE_LENGTH:	Line length exceeded

KRERR_FILE_FORMAT:	Incompatible file format
KRERR_FILE_OPEN:	Can not open file
KRERR_FILE_SYNTAX:	Syntax error at line
KRERR_MALLOC1:	Memory allocation error 1
KRERR_TTYPE:	Topologic type invalid
KRERR_SYMBOL:	Symbol pattern invalid (must match [A-Za-z]^[|,]*)
KRERR_NO_SUCH_SITE:	Current unit does not have a site with this name
KRERR_NO_HIDDEN_UNITS:	No hidden units defined
KRERR_CYCLES:	cycle(s):
KRERR_DEAD_UNITS:	dead unit(s):
KRERR_INPUT_PATTERNS:	Pattern file contains not the same number of input units as the network
KRERR_OUTPUT_PATTERNS:	Pattern file contains not the same number of output units as the network
KRERR_CHANGED_I_UNITS:	Number of input units has changed
KRERR_CHANGED_O_UNITS:	Number of output units has changed
KRERR_NO_INPUT_UNITS:	No input units defined
KRERR_NO_OUTPUT_UNITS:	No output units defined
KRERR_NO_PATTERNS:	No patterns defined
KRERR_INCORE_PATTERNS:	Incore patterns incompatible with current network (remove loaded patterns before loading network)
KRERR_PATTERN_NO:	Invalid pattern number
KRERR_LEARNING_FUNC:	Invalid learning function
KRERR_PARAMETERS:	Invalid parameters
KRERR_UPDATE_FUNC:	Invalid update function
KRERR_INIT_FUNC:	Invalid initialization function
KRERR_DERIV_FUNC:	Derivation function of the activation function does not exist
KRERR_I_UNITS_CONNECT:	input unit(s) with input connections to other units
KRERR_O_UNITS_CONNECT:	output unit(s) with output connections to other units
KRERR_TOPOMODE:	Invalid topological sorting mode
KRERR_LEARNING_SITES:	Learning function doesn't support sites
KRERR_SITES_NO_SUPPORT:	Sites are not supported
KRERR_NO_MASPAR_KERNEL:	This isn't a MasPar Kernel
KRERR_NOT_NEIGHBOUR_LAYER:	Connection(s) between unit(s) in non-neighbour layers are not supported
KRERR_MUCH_LAYERS:	The number of layers is too high
KRERR_NOT_FULLY_CONNECTED:	The network layers aren't fully connected
KRERR_MODE_FF1_INVALID_OP:	This operation is not allowed in parallel kernel mode
KRERR_NET_TRANSFORM:	Change of network type isn't possible in parallel kernel mode

KRERR_NO_CURRENT_LINK:	No current link defined
KRERR_NO_CURRENT_UNIT:	No current unit defined
KRERR_UNIT_NO_INPUT:	Current unit does't have any inputs
KRERR_TOPO_DEFINITION:	Invalid parameter in topologic definition section
KRERR_BAD_CONNECTION:	Creation of link between these units is not permitted
KRERR_MASPAR_NO_RESPOND:	MasPar doesn't respond
KRERR_NOT_IMPLEMENTED_YET:	This function isn't implemented yet
KRERR_NOT_PARRALLEL_MODE:	Kernel isn't in parallel mode
KRERR_MASPAR_OUT_OF_PE_MEM:	MasPar ran out of memory
KRERR_MASPAR_COMMUNICATE:	MasPar communication error
KRERR_MASPAR_OUT_OF_PE:	MasPar ran out of processors
KRERR_MISSING_DEFAULT_FUNC:	Missing default function (check function table)
KRERR_MASPAR_MULTI_OUT_FUNC:	MasPar kernel does not support multiple unit output functions
KRERR_MASPAR_MULTI_ACT_FUNC:	MasPar kernel doesn't support multiple unit activation functions
KRERR_NET_DEPTH:	The depth of the network does not fit to the learning function
KRERR_NO_OF_UNITS_IN_LAYER:	Wrong no. of units in layer
KRERR_UNIT_MISSING:	Unit missing or not correctly connected
KRERR_UNDETERMINED_UNIT:	Unit does not belong to a defined layer in the network
KRERR_ACT_FUNC:	Unit has wrong activation function
KRERR_OUT_FUNC:	Unit has wrong output function
KRERR_SITE_FUNC:	Unexpected site function at unit
KRERR_UNEXPECTED_SITES:	Unit is not expected to have sites
KRERR_UNEXPECTED_DIRECT_INPUTS:	Unit is expected to have sites
KRERR_SITE_MISSING:	Site missing at unit
KRERR_UNEXPECTED_LINK:	Unexpected link
KRERR_LINK_MISSING:	Missing link(s) to unit
KRERR_LINK_TO_WRONG_SITE:	Link ends at wrong site of target unit
KRERR_TOPOLOGY:	The network is not fitting the required topology
KRERR_PARAM_BETA:	Wrong beta parameter in unit bias value
KRERR_CC_ERROR1:	Topo_ptr_array is sorted in the wrong way
KRERR_CC_ERROR2:	There is no memory allocated
KRERR_CC_ERROR3:	Not enough memory to run Casscade
KRERR_CC_ERROR4:	Hidden unit with more than one recurent link detected
KRERR_CC_ERROR5:	Hidden unit without recurrent link found
KRERR_CC_ERROR6:	Hidden layer does not fit the required topology
KRERR_CC_ERROR7:	The first link of a hidden unit is not a recurent link

KRERR_CC_ERROR8:	Special unit with more than one recurent link detected
KRERR_CC_ERROR9:	First link of a special unit is not a recurent link
KRERR_CC_ERROR10:	Wrong update function
KRERR_CC_ERROR11:	Wrong init function
DLVQ_ERROR1:	There are empty classes
DLVQ_ERROR2:	There is a class lower than zero
DLVQ_ERROR3:	Wrong number of output units.
DLVQ_ERROR4:	This network is not fitting the required topology
DLVQ_ERROR5:	There isn't a unit for every class
KRERR_NP_NO_MORE_ENTRIES:	No more free pattern sets available
KRERR_NP_NO_SUCH_PATTERN_SET:	No such Pattern set defined
KRERR_NP_CURRENT_PATTERN:	No current pattern defined
KRERR_NP_DIMENSION:	Specified sub pattern does not fit into pattern
KRERR_NP_NO_SUCH_PATTERN:	No such pattern available
KRERR_NP_NO_CURRENT_PATTERN_SET:	No current pattern set defined
KRERR_NP_DOES_NOT_FIT:	Pattern (sub pattern) does not fit the network
KRERR_NP_NO_TRAIN_SCHEME:	No sub pattern shifting scheme defined
KRERR_NP_NO_OUTPUT_PATTERN:	Pattern contains no output information
KRERR_NP_INCOMPATIBLE_NEW:	New Pattern does not fit to existing set
KRERR_NP_PARAM_ZERO:	Illegal parameter value <= 0 specified
KRERR_IP_ISNOTINITED:	Paragon kernel is not initialized
KRERR_IP_ISINITED:	Paragon kernel is already initialized
KRERR_IP_SENDMSGFAILED:	Sending a message fails
KRERR_IP_MSGSYNTAX:	Syntax error in received message
KRERR_IP_UNKNOWENCMD:	Receive unknown command
KRERR_IP_LESSPAT:	There are less patterns than allocated nodes
KRERR_IP_WEIGHTUPDATE:	Weight update with global sum fails
KRERR_IP_LEARNFNOTPAR:	Learning function is not parallelized

Chapter 16

Transfer Functions

16.1 Predefined Transfer Functions

The following site, activation, and output functions are already predefined. Future releases of the kernel will have additional transfer functions.

Site Functions:

Function	Formula
`Linear`	$net_j(t) = \sum_i w_{ij} o_i$
`Produkt`	$net_j(t) = \prod_i w_{ij} o_i$
`PI`	$net_j(t) = \prod_i o_i$
`Max`	$net_j(t) = \max_i w_{ij} o_i$
`Min`	$net_j(t) = \min_i w_{ij} o_i$
`At_least_2`	$net_j(t) = \begin{cases} 0 & \text{if } \sum_i w_{ij} o_i < 2 \\ 1 & \text{else.} \end{cases}$

Several other site functions have been implemented for the ART models in SNNS: `At_least_2`, `At_least_1`, `At_most_0`, `Reciprocal`. These functions normally are not useful for other networks. So they are mentioned here, but not described in detail. For more information please refer to the section about the corresponding ART model in SNNS.

Activation functions:

Function	Formula
BAM	$a_j(t) = \begin{cases} 1 & \text{if } net_j(t) > 0 \\ a_j(t-1) & \text{if } net_j(t) = 0 \\ -1 & \text{if } net_j(t) < 0 \end{cases}$
BSB	$a_j(t) = net_j(t) * \theta_j$
Elliott	$a_j(t) = \frac{net_j(t)+\theta_j}{1+\lvert net_j(t)+\theta_j \rvert}$
Identity	$a_j(t) = net_j(t)$
IdentityPlusBias	$a_j(t) = net_j(t) + \theta_j$
Logistic	$a_j(t) = \frac{1}{1+e^{-(net_j(t)+\theta_j)}}$
Logistic_notInhibit	like Logistic, but skip input from units named "Inhibit"
Logistic_Tbl	like Logistic, but with table lookup instead of computation
MinOutPlusWeight	$a_j(t) = \min_i(w_{ij} + o_i)$
Perceptron	$a_j(t) = \begin{cases} 1 & \text{if } net_j(t) \geq \theta_j \\ 0 & \text{if } net_j(t) < \theta_j \end{cases}$
Product	$a_j(t) = \prod_i w_{ij} o_i$
RBF_Gaussian	see the chapter about RBFs in the user manual
RBF_MultiQuadratic	see the chapter about RBFs in the user manual
RBF_ThinPlateSpline	see the chapter about RBFs in the user manual
RM	$a_j(t) = \begin{cases} 0.85 * a_j(t-1) + \\ 0.15 * net_j(t) * \\ (1 - a_j(t-1)) & \text{if } net_j(t) > 0 \\ 0.85 * a_j(t-1) + \\ 0.15 * net_j(t) * \\ (1 + a_j(t-1)) & \text{if } net_j(t) \leq 0 \end{cases}$
Signum	$a_j(t) = \begin{cases} 1 & \text{if } net_j(t) > 0 \\ -1 & \text{if } net_j(t) \leq 0 \end{cases}$
Signum0	$a_j(t) = \begin{cases} 1 & \text{if } net_j(t) > 0 \\ 0 & \text{if } net_j(t) = 0 \\ -1 & \text{if } net_j(t) < 0 \end{cases}$
StepFunc	$a_j(t) = \begin{cases} 1 & \text{if } net_j(t) > 0 \\ 0 & \text{if } net_j(t) \leq 0 \end{cases}$
TanH	$a_j(t) = \tanh(net_j(t) + \theta_j)$
TanH_Xdiv2	$a_j(t) = \tanh(net_j(t)/2 + \theta_j)$

Several other activation functions have been implemented for the ART models in SNNS: `Less_than_0`, `At_most_0`, `At_least_2`, `At_least_1`, `Exactly_1`, `ART1_NC`, `ART2_Identity`, `ART2_Rec`, `ART2_NormP`, `ART2_NormV`, `ART2_NormW`, `ART2_NormIP`, `ART2_Rst`, `ARTMAP_NCa`, `ARTMAP_NCb`, `ARTMAP_DRho`.

These functions normally are not useful for other networks. So they are mentioned here, but not described in detail. For cascade correlation and time delay networks the following modified versions of regular activation functions have been implemented: `RCC_logistic`, `RCC_LogisticSym`, `RCC_Tanh`, `RCC_Linear`, `TD_Logistic`, `TD_Elliott`. They behave like the ordinary functions with the same name body.

Output Functions:

Function	**Formula**
`Identity`	$o_j(t) = a_j(t)$
`Clip_0_1`	$o_j(t) = \begin{cases} 0 & \text{if } a_j(t) \leq 0 \\ a_j(t) & \text{if } a_j(t) < 1 \\ 1 & \text{if } a_j(t) \geq 1 \end{cases}$
`Clip_1_1`	$o_j(t) = \begin{cases} -1 & \text{if } a_j(t) \leq -1 \\ a_j(t) & \text{if } -1 < a_j(t) < 1 \\ 1 & \text{if } a_j(t) \geq 1 \end{cases}$
`Threshold_0.5`	$o_j(t) = \begin{cases} 0 & \text{if } a_j(t) \leq 0.5 \\ 1 & \text{if } a_j(t) > 0.5 \end{cases}$

Two other output functions have been implemented for ART2 in SNNS: `ART2_Noise_PLin` and `ART2_Noise_ContDiff` These functions are only useful for the ART2 implementation So they are mentioned here, but not described in detail.

16.2 User Defined Transfer Functions

The group of transfer functions can be exented arbitrarily by the user. In order to make them available inside SNNS the following steps have to be performed:

1. In file "`.../SNNSv4.0/kernel/func_tbl.c` the name of the function has to be inserted in the function table. An example entry for an activation function would be
   "Act_MyFunction",ACT_FUNC, 0, 0, (FunctionPtr) MyFancyFunction,

   Notice that the second entry defines the type (activation, initialization, etc.) of the new function!

   If the new function is an activation function, the corresponding derivation function has also to be inserted in the function table. E.g.:
   "Act_MyFunction",ACT_DERIV_FUNC, 0, 0, (FunctionPtr)ACT_DERIV_MFF,

   This entry has to be given, even if no such derivation function exists in the mathematical sense. In that case `''ACT_DERIV_Dummy''` has to be specified as name of the derivation function.

   If the function exists, it has to be declared and implemented just as the activation function.

Please note, that activation and derivation function have to have the same name suffix (here: "MyFunction")!

2. The functions must be implemented as C programs in following files:

   activation functions in "`.../SNNSv4.0/kernel/trans_f.c`"
   output functions in "`.../SNNSv4.0/kernel/trans_f.c`"
   site functions in "`.../SNNSv4.0/kernel/trans_f.c`"
   initialization functions in "`.../SNNSv4.0/kernel/init_f.c`"
   learning functions in "`.../SNNSv4.0/kernel/learn_f.c`"
   update functions in "`.../SNNSv4.0/kernel/update_f.c`"

   The name of the implemented function has to match the name specified in the function table!

3. Function prototypes have to be defined in the corresponding header (.h) and private header files (.ph).

4. Go to the SNNSv4.0 root directory and build (using the command 'build') new executables. Remember that if you don't recompile, the modification to the kernel won't take any effect.

The new function should be available now in the user interface together with all predefined functions.

Appendix A

Kernel File Interface

A.1 The ASCII Network File Format

The ASCII representation of a network consists of the following parts:

- a header, which contains information about the net
- the definition of the teaching function
- the definition of the sites
- the definition of cell types
- the definition of the default values for cells
- the enumeration of the cells with all their characteristics
- the list of connections
- a list of subnet numbers
- a list of layer numbers

All parts, except the header and the enumeration of the cells, may be omitted. Each part may also be empty. It then consists only of the part title, the header line and the boarder marks (e.g.: `----|---|------`).

Entries in the *site definition section* do not contain any empty columns. The only empty column in the *type definition section* may be the *sites* column, in which case the cells of this type do not have sites.

Entries in the *unit definition section* have at least the columns *no.* (cell number) and *position* filled. The entries (rows) are sorted by increasing cell number. If column *typeName* is filled, the columns *act func*, *out func*, and *sites* remain empty.

Entries in the *connection definition section* have all columns filled. The respective cell does not have a site, if the column *site* is empty. The entries are sorted by increasing number of the target cell (column *target*). Each entry may have multiple entries in the column *sources*. In this case, the entries (number of the source cell and the connection strength) are separated by a comma and a blank, or by a comma and a newline (see example in the Appendix B).

The file may contain comment lines. Each line beginning with # is skipped by the SNNS-kernel.

A.2 Form of the Network File Entries

Columns are separated by the string ␣|␣. A row never exceeds 250 characters.

Strings may have arbitrary length. The compiler determines the length of each row containing strings (maximum string length + 2). Within the columns, the strings are stored left adjusted. Strings may not contain blanks, but all special characters except `|`. The first character of a string has to be a letter.

Integers may have an arbitrary number of digits. Cell numbers are always positive and not zero. Position coordinates may be positive or negative. The compiler determines the length of each row containing integers (maximum digit number + 2). Within the columns, the numbers are stored right adjusted.

Floats are always stored in fixed length with the format `Vx.yyyyy`, where **V** is the sign (+, - or blank), **x** is 0 or 1 and **y** is the rational part (5 digits behind the decimal point).

Rows containing floats are therefore always 10 characters long (8 + 1 blank on each side).

If a row contains several sites in the *type* or *unit definition section*, they are written below each other. They are separated in the following way: Directly after the first site follows a comma and a newline (`\n`). The next line starts with an arbitrary number of blanks or tabs in front of the next site.

The source of a connection is described by a pair, the cell number and the strength of the connection. It always has the format `nnn:Vx.yyyyy` with the following meaning:

- `nnn` Number of the source
- `Vx.yyyyy` Strength of the connection as a float value (format as described above)

The compiler determines the width of the column **nnn** by the highest cell number present. The cell numbers are written into the column right adjusted (according to the rules for integers). The column `Vx.yyyyy` has fixed width. Several source pairs in an entry to the *connection definition section* are separated by a comma and a blank. If the list of source pairs exceeds the length of one line, the line has to be parted after the following rule:

- Separation is always between pairs, never within them.
- The comma between pairs is always directly behind the last pair, i.e. remains in the old line.
- After a newline (`\n`) an arbitrary number of blanks or tabs may precede the next pair.

A.3 Grammar of the Network Files

A.3.1 Conventions

A.3.1.1 Lexical Elements of the Grammar

The lexical elements of the grammar which defines network files are listed as regular expresions. The first column lists the name of the symbol, the second the regular expresion defining it. The third column may contain comments.

- All terminals (characters) are put between "...".
- Elements of sets are put between square brackets. Within the brackets, the characters represent themselves (even without "), and - defines a range of values. The class of digits is defined, e.g. as `["0"-"9"]`.
- Characters can be combined into groups with parenteses `()`.
- `x*` means, that the character or group x can occur zero or more times.
- `x+` means, that the character or group x must occur at least once, but may occur several times.
- `x?` means, that x can be omitted.
- `x{n}` means, that x has to occur exactly n times.
- `x|y` means, that either x or y has to occur.
- `*`, `+` and `{}` bind strongest, `?` is second, `|` binds weakest.
- Groups or classes of characters are treated like a single character with respect to priority.

A.3.1.2 Definition of the Grammar

The Grammar defining the interface is listed in a special form of EBNF.

- Parts between square brackets `[]` are facultative.
- `|` separates alternatives (like with terminal symbols).
- `{x}` means, that x may occur zero or more times.
- CSTRING is everything that is recognized as string by the C programming language.

A.3.2 Terminal Symbols

```
WHITESPACE              {" "|"\n"|"\t"}                    /* whitespaces */
BLANKS_TABS             {" "|"\t"}                         /* only blanks or tabs */
W_COL_SEP               (" "|"\n"|"\t") {" "|"\n"|"\t"} "|" {" "|"\n"|"\t"}
                        /* at least one blank and the column separation */
COL_SEP                 {" "|"\n"|"\t"} "|" {" "|"\n"|"\t"}  /* column separation */
COMMA                   {" "|"\n"|"\t"} "," {" "|"\n"|"\t"}  /* at least the comma */
EOL                     {" "|"\n"|"\t"} "\n" {" "|"\n"|"\t"} /* at least "\n" */
CUT                     {" "|"\n"|"\t"} (" "|"\n"|"\t") {" "|"\n"|"\t"}
                        /* at least a blank, "\t", or "\n" */
COLON                   ":"


/* separation lines for different tables */

TWO_COLUMN_LINE         "-"+"|""-"+
THREE_COLUMN_LINE       "-"+"|""-"+"|""-"+
FOUR_COLUMN_LINE        "-"+"|""-"+"|""-"+"|""-"+
SIX_COLUMN_LINE         "-"+"|""-"+"|""-"+"|""-"+"|""-"+"|""-"+
SEVEN_COLUMN_LINE       "-"+"|""-"+"|""-"+"|""-"+"|""-"+"|""-"+"|""-"+
TEN_COLUMN_LINE         "-"+"|""-"+"|""-"+"|""-"+"|""-"+"|""-"+"|""-"+
                        "|""-"+"|""-"+"|""-"+

COMMENT                 {{" "|"\n"|"\t"} "#" CSTRING "\n" {" "|"\n"|"\t"}}

VERSION                 "V1.4-3D" | "V2.1" | "V3.0" /* version of SNNS */

SNNS                    "SNNS network definition file"  /* output file header */


/* eleven different headers */

GENERATED_AT            "generated at"
NETWORK_NAME            "network name :"
SOURCE_FILES            "source files"
NO.OF_UNITES            "no. of unites :"
NO.OF_CONNECTIONS       "no. of connections :"
NO.OF_UNIT_TYPES        "no. of unit types :"
NO.OF_SITE_TYPES        "no. of site types :"
LEARNING_FUNCTION       "learning function :"
PRUNING_FUNCTION        "pruning function :"
FF_LEARNING_FUNCTION    "subordinate learning function :"
UPDATE_FUNCTION         "update function :"


/* titles of the different sections */

UNIT_SECTION_TITLE        "unit definition section"
DEFAULT_SECTION_TITLE     "unit default section"
SITE_SECTION_TITLE        "site definition section"
TYPE_SECTION_TITLE        "type definition section"
CONNECTION_SECTION_TITLE  "connection definition section"
LAYER_SECTION_TITLE       "layer definition section"
SUBNET_SECTION_TITLE      "subnet definition section"
TRANSLATION_SECTION_TITLE "3D translation section"
TIME_DELAY_SECTION_TITLE  "time delay section"
```

```
/* column-titles of the different tables */

NO                        "no."
TYPE_NAME                 "type name"
UNIT_NAME                 "unit name"
ACT                       "act"
BIAS                      "bias"
ST                        "st"
POSITION                  "position"
SUBNET                    "subnet"
LAYER                     "layer"
ACT_FUNC                  "act func"
OUT_FUNC                  "out func"
SITES                     "sites"
SITE_NAME                 "site name"
SITE_FUNCTION             "site function"
NAME                      "name"
TARGET                    "target"
SITE                      "site"
SOURCE:WEIGHT             "source:weight"
UNIT_NO                   "unitNo."
DELTA_X                   "delta x"
DELTA_Y                   "delta y"
Z                         "z"
LLN                       "LLN"
LUN                       "LUN"
TROFF                     "Troff"
SOFF                      "Soff"
CTYPE                     "Ctype"

INTEGER                   ["0"-"9"]+                                    /*integer */
SFLOAT                    ["+" | " " | "-"] ["1" | "0"] "." ["0"-"9"]{5} /*signed float*/
STRING                    ("A"-"Z" | "a"-"z" | "|")+                    /*string */
```

A.3.3 Grammar:

```
out_file                ::= file_header sections

file_header             ::= WHITESPACE  COMMENT h_snns EOL COMMENT h_generated_at EOL
                            COMMENT h_network_name EOL COMMENT h_source_files EOL
                            COMMENT h_no.of_unites EOL COMMENT h_no.of_connections EOL
                            COMMENT h_no.of_unit_types EOL COMMENT h_no.of_site_types EOL
                            COMMENT h_learning_function EOL COMMENT h_update_function EOL
                            COMMENT h_pruning_function EOL
                            COMMENT ff_learning_function EOL


/* parts of the file-header */

h_snns                  ::= SNNS BLANKS_TABS VERSION
h_generated_at          ::= GENERATED_AT BLANKS_TABS CSTRING
h_network_name          ::= NETWORK_NAME BLANKS_TABS STRING
h_source_files          ::= SOURCE_FILES [BLANKS_TABS COLON BLANKS_TABS CSTRING]
```

```
h_no.of_unites          ::= NO.OF_UNITES BLANKS_TABS INTEGER
h_no.of_connections     ::= NO.OF_CONNECTIONS BLANKS_TABS INTEGER
h_no.of_unit_types      ::= NO.OF_UNIT_TYPES BLANKS_TABS INTEGER
h_no.of_site_types      ::= NO.OF_SITE_TYPES BLANKS_TABS INTEGER
h_learning_function     ::= LEARNING_FUNCTION BLANKS_TABS STRING
h_pruning_function      ::= PRUNING_FUNCTION BLANKS_TABS STRING
h_ff_learning_function  ::= FF_LEARNING_FUNCTION BLANKS_TABS STRING
h_update_function       ::= UPDATE_FUNCTION BLANKS_TABS STRING

sections                ::= COMMENT unit_section [COMMENT default_section]
                            [COMMENT site_section] [COMMENT type_section]
                            [COMMENT subnet_section] [COMMENT conn_section]
                            [COMMENT layer_section] [COMMENT trans_section]
                            [COMMENT time_delay_section] COMMENT


/* unit default section */

default_section         ::= DEFAULT_SECTION_TITLE CUT COMMENT WHITESPACE  default_block
default_block           ::= default_header SEVEN_COLUMN_LINE EOL
                            {COMMENT default_def} SEVEN_COLUMN_LINE EOL
default_header          ::= ACT COL_SEP BIAS COL_SEP ST COL_SEP SUBNET COL_SEP
                            LAYER COL_SEP ACT_FUNC COL_SEP OUT_FUNC CUT
default_def             ::= SFLOAT W_COL_SEP SFLOAT W_COL_SEP STRING W_COL_SEP
                            INTEGER W_COL_SEP INTEGER W_COL_SEP STRING W_COL_SEP
                            STRING CUT


/* site definition section */

site_section            ::= SITE_SECTION_TITLE CUT COMMENT WHITESPACE  site_block
site_block              ::= site_header TWO_COLUMN_LINE EOL {COMMENT site_def}
                            TWO_COLUMN_LINE EOL
site_header             ::= SITE_NAME SITE_FUNCTION CUT
site_def                ::= STRING W_COL_SEP STRING CUT


/* type definition section */

type_section            ::= TYPE_SECTION_TITLE CUT COMMENT WHITESPACE  type_block
type_block              ::= type_header FOUR_COLUMN_LINE EOL {COMMENT type_def}
                            FOUR_COLUMN_LINE EOL
type_header             ::= NAME COL_SEP ACT_FUNC COL_SEP OUT_FUNC COL_SEP SITES CUT
type_def                ::= STRING W_COL_SEP STRING W_COL_SEP STRING W_COL_SEP
                            [{STRING COMMA} STRING] CUT


/* subnet definition section */

subnet_section          ::= SUBNET_SECTION_TITLE CUT COMMENT WHITESPACE  subnet_block
subnet_block            ::= subnet_header TWO_COLUMN_LINE EOL {COMMENT subnet_def}
                            TWO_COLUMN_LINE EOL
subnet_header           ::= SUBNET COL_SEP UNIT_NO CUT
subnet_def              ::= INTEGER W_COL_SEP {INTEGER COMMA} INTEGER CUT
```

```
/* unit definition section /*

unit_section          ::= UNIT_SECTION_TITLE CUT COMMENT WHITESPACE  unit_block
unit_block            ::= unit_header TEN_COLUMN_LINE EOL {COMMENT unit_def}
                          TEN_COLUMN_LINE EOL

unit_header           ::= NO COL_SEP TYPE_NAME COL_SEP UNIT_NAME COL_SEP
                          ACT COL_SEP BIAS COL_SEP ST COL_SEP POSITION COL_SEP
                          ACT_FUNC COL_SEP OUT_FUNC COL_SEP SITES CUT
unit_def              ::= INTEGER W_COL_SEP ((STRING W_COL_SEP) | COL_SEP)
                          ((STRING W_COL_SEP) | COL_SEP) ((SFLOAT W_COL_SEP) | COL_SEP)
                          ((SFLOAT W_COL_SEP) | COL_SEP) ((STRING W_COL_SEP) | COL_SEP)
                          INTEGER COMMENT INTEGER COMMENT INTEGER W_COL_SEP
                          ((STRING W_COL_SEP) | COL_SEP) ((STRING W_COL_SEP) | COL_SEP)
                          [{STRING COMMA} STRING]

/* connection definition section */

connection_section    ::= CONNECTION_SECTION_TITLE CUT
                          COMMENT WHITESPACE connection_block
connection_block      ::= connection_header THREE_COLUMN_LINE EOL
                          {COMMENT connection_def} THREE_COLUMN_LINE EOL
connection_header     ::= TARGET COL_SEP SITE COL_SEP SOURCE:WEIGHT CUT
connection_def        ::= ((INTEGER W_COL_SEP) | COL_SEP) STRING W_COL_SEP
                          {INTEGER WHITESPACE  COLON WHITESPACE  SFLOAT COMMA}
                          INTEGER WHITESPACE  COLON WHITESPACE  SFLOAT CUT


/* layer definition section */

layer_section         ::= LAYER_SECTION_TITLE CUT COMMENT WHITESPACE  layer_block
layer_block           ::= layer_header TWO_COLUMN_LINE EOL {COMMENT layer_def}
                          TWO_COLUMN_LINE EOL
layer_header          ::= LAYER COL_SEP UNIT_NO CUT
layer_def             ::= INTEGER W_COL_SEP {INTEGER COMMENT} INTEGER CUT


/* 3D translation section */

translation_section   ::= TRANSLATION_SECTION_TITLE CUT
                          COMMENT WHITESPACE  translation_block
translation_block     ::= translation_header THREE_COLUMN_LINE EOL
                          {COMMENT translation_def} THREE_COLUMN_LINE EOL
translation_header    ::= DELTA_X COL_SEP DELTA_Y COL_SEP Z CUT
translation_def       ::= INTEGER W_COL_SEP INTEGER W_COL_SEP INTEGER


/* time delay section */

td_section            ::= TIME_DELAY_SECTION_TITLE CUT COMMENT WHITESPACE  td_block
td_block              ::= td_header SIX_COLUMN_LINE EOL {COMMENT td_def}
                          SIX_COLUMN_LINE EOL
td_header             ::= NO COL_SEP LLN COL_SEP LUN COL_SEP
                          TROFF COL_SEP SOFF COL_SEP CTYPE CUT
td_def                ::= INTEGER W_COL_SEP INTEGER W_COL_SEP INTEGER W_COL_SEP
                          INTEGER W_COL_SEP INTEGER W_COL_SEP INTEGER W_COL_SEP
```

A.4 Grammar of the Pattern Files

The typographic conventions used for the pattern file grammar are the same as for the network file grammar (see section A.3.1.1).

A.4.1 Terminal Symbols

```
WHITE              {" " | "\t"}
FREE               {^"\n"}                     /* anything up to EOL */
COMMENT            "#" FREE "\n"
L_BRACKET          "["
R_BRACKET          "]"
V_NUMBER           [Vv]{INT}+"."{INT}+          /* version number */
INT                ["0"-"9"]
NUMBER             [-+]?{{INT}+ | {INT}+{EXP} |{INT}+"."{INT}*({EXP})? |
                   {INT}*"."{INT}+({EXP})?}
EXP                [Ee][-+]?{INT}+
VERSION_HEADER     "SNNS pattern definition file"
GENERATED_AT       "generated at" {FREE}* "\n"
NO_OF_PATTERN      "No. of patterns" {WHITE}* ":"
NO_OF_INPUT        "No. of input units" {WHITE}* ":"
NO_OF_OUTPUT       "No. of output units" {WHITE}* ":"
NO_OF_VAR_IDIM     "No. of variable input dimensions" {WHITE}* ":"
NO_OF_VAR_ODIM     "No. of variable output dimensions" {WHITE}* ":"
MAXIMUM_IDIM       "Maximum input dimensions" {WHITE}* ":"
MAXIMUM_ODIM       "Maximum output dimensions" {WHITE}* ":"
```

A.4.2 Grammar

```
pattern_file       ::= header pattern_list
header             ::= VERSION_HEADER V_NUMBER GENERATED_AT NO_OF_PATTERN NUMBER
                       i_head [o_head] [vi_head] [vo_head]
i_head             ::= [Comment] NO_OF_INPUT NUMBER
o_head             ::= [Comment] NO_OF_OUTPUT NUMBER
vi_head            ::= [Comment] NO_OF_VAR_IDIM NUMBER MAXIMUM_IDIM actual_dim
vo_head            ::= [Comment] NO_OF_VAR_ODIM NUMBER MAXIMUM_ODIM actual_dim
actual_dim         ::= (L_BRACKET actual_dim_rest R_BRACKET) |
                       (L_BRACKET R_BRACKET)
actual_dim_rest    ::= dim_entry | (actual_dim_rest dim_entry)
dim_entry          ::= INT
pattern_list       ::= pattern | (pattern_list pattern)
pattern            ::= pattern_start pattern_body
pattern_start      ::= [Comment] [actual_dim]
pattern_body       ::= pattern_entry | (pattern_body pattern_entry)
pattern_entry      ::= NUMBER
```

Appendix B

Example Network Files

The lines in the connection definition section have been truncated to 80 characters per line for printing purposes.

B.1 Example 1:

```
SNNS network definition file V3.0
generated at Fri Aug  3 00:28:44 1992

network name : klass
source files :
no. of units : 71
no. of connections : 610
no. of unit types : 0
no. of site types : 0


learning function : Std_Backpropagation
update function   : Topological_Order


unit default section :

act      | bias     | st | subnet | layer | act func     | out func
---------|----------|----|--------|-------|--------------|-------------
 0.00000 |  0.00000 | h  |      0 |     1 | Act_Logistic | Out_Identity
---------|----------|----|--------|-------|--------------|-------------


unit definition section :

no. | typeName | unitName | act      | bias     | st | position | act func | out func | sites
----|----------|----------|----------|----------|----|----------|----------|----------|-------
  1 |          | u11      |  1.00000 |  0.00000 | i  |  1, 1, 0 |||
  2 |          | u12      |  0.00000 |  0.00000 | i  |  2, 1, 0 |||
  3 |          | u13      |  0.00000 |  0.00000 | i  |  3, 1, 0 |||
  4 |          | u14      |  0.00000 |  0.00000 | i  |  4, 1, 0 |||
  5 |          | u15      |  1.00000 |  0.00000 | i  |  5, 1, 0 |||
  6 |          | u21      |  1.00000 |  0.00000 | i  |  1, 2, 0 |||
  7 |          | u22      |  1.00000 |  0.00000 | i  |  2, 2, 0 |||
  8 |          | u23      |  0.00000 |  0.00000 | i  |  3, 2, 0 |||
  9 |          | u24      |  1.00000 |  0.00000 | i  |  4, 2, 0 |||
 10 |          | u25      |  1.00000 |  0.00000 | i  |  5, 2, 0 |||
```

```
 11 |          | u31      |  1.00000 |  0.00000 | i  |  1, 3, 0 |||
 12 |          | u32      |  0.00000 |  0.00000 | i  |  2, 3, 0 |||
 13 |          | u33      |  1.00000 |  0.00000 | i  |  3, 3, 0 |||
 14 |          | u34      |  0.00000 |  0.00000 | i  |  4, 3, 0 |||
 15 |          | u35      |  1.00000 |  0.00000 | i  |  5, 3, 0 |||
 16 |          | u41      |  1.00000 |  0.00000 | i  |  1, 4, 0 |||
 17 |          | u42      |  0.00000 |  0.00000 | i  |  2, 4, 0 |||
 18 |          | u43      |  0.00000 |  0.00000 | i  |  3, 4, 0 |||
 19 |          | u44      |  0.00000 |  0.00000 | i  |  4, 4, 0 |||
 20 |          | u45      |  1.00000 |  0.00000 | i  |  5, 4, 0 |||
 21 |          | u51      |  1.00000 |  0.00000 | i  |  1, 5, 0 |||
 22 |          | u52      |  0.00000 |  0.00000 | i  |  2, 5, 0 |||
 23 |          | u53      |  0.00000 |  0.00000 | i  |  3, 5, 0 |||
 24 |          | u54      |  0.00000 |  0.00000 | i  |  4, 5, 0 |||
 25 |          | u55      |  1.00000 |  0.00000 | i  |  5, 5, 0 |||
 26 |          | u61      |  1.00000 |  0.00000 | i  |  1, 6, 0 |||
 27 |          | u62      |  0.00000 |  0.00000 | i  |  2, 6, 0 |||
 28 |          | u63      |  0.00000 |  0.00000 | i  |  3, 6, 0 |||
 29 |          | u64      |  0.00000 |  0.00000 | i  |  4, 6, 0 |||
 30 |          | u65      |  1.00000 |  0.00000 | i  |  5, 6, 0 |||
 31 |          | u71      |  1.00000 |  0.00000 | i  |  1, 7, 0 |||
 32 |          | u72      |  0.00000 |  0.00000 | i  |  2, 7, 0 |||
 33 |          | u73      |  0.00000 |  0.00000 | i  |  3, 7, 0 |||
 34 |          | u74      |  0.00000 |  0.00000 | i  |  4, 7, 0 |||
 35 |          | u75      |  1.00000 |  0.00000 | i  |  5, 7, 0 |||
 36 |          | h1       |  0.99999 |  0.77763 | h  |  8, 0, 0 |||
 37 |          | h2       |  0.19389 |  2.17683 | h  |  8, 1, 0 |||
 38 |          | h3       |  1.00000 |  0.63820 | h  |  8, 2, 0 |||
 39 |          | h4       |  0.99997 | -1.39519 | h  |  8, 3, 0 |||
 40 |          | h5       |  0.00076 |  0.88637 | h  |  8, 4, 0 |||
 41 |          | h6       |  1.00000 | -0.23139 | h  |  8, 5, 0 |||
 42 |          | h7       |  0.94903 |  0.18078 | h  |  8, 6, 0 |||
 43 |          | h8       |  0.00000 |  1.37368 | h  |  8, 7, 0 |||
 44 |          | h9       |  0.99991 |  0.82651 | h  |  8, 8, 0 |||
 45 |          | h10      |  0.00000 |  1.76282 | h  |  8, 9, 0 |||
 46 |          | A        |  0.00972 | -1.66540 | o  | 11, 1, 0 |||
 47 |          | B        |  0.00072 | -0.29800 | o  | 12, 1, 0 |||
 48 |          | C        |  0.00007 | -2.24918 | o  | 13, 1, 0 |||
 49 |          | D        |  0.02159 | -5.85148 | o  | 14, 1, 0 |||
 50 |          | E        |  0.00225 | -2.33176 | o  | 11, 2, 0 |||
 51 |          | F        |  0.00052 | -1.34881 | o  | 12, 2, 0 |||
 52 |          | G        |  0.00082 | -1.92413 | o  | 13, 2, 0 |||
 53 |          | H        |  0.00766 | -1.82425 | o  | 14, 2, 0 |||
 54 |          | I        |  0.00038 | -1.83376 | o  | 11, 3, 0 |||
 55 |          | J        |  0.00001 | -0.87552 | o  | 12, 3, 0 |||
 56 |          | K        |  0.01608 | -2.20737 | o  | 13, 3, 0 |||
 57 |          | L        |  0.01430 | -1.28561 | o  | 14, 3, 0 |||
 58 |          | M        |  0.92158 | -1.86763 | o  | 11, 4, 0 |||
 59 |          | N        |  0.05265 | -3.52717 | o  | 12, 4, 0 |||
 60 |          | O        |  0.00024 | -1.82485 | o  | 13, 4, 0 |||
 61 |          | P        |  0.00031 | -0.20401 | o  | 14, 4, 0 |||
 62 |          | Q        |  0.00025 | -1.78383 | o  | 11, 5, 0 |||
 63 |          | R        |  0.00000 | -1.61928 | o  | 12, 5, 0 |||
 64 |          | S        |  0.00000 | -1.59970 | o  | 13, 5, 0 |||
 65 |          | T        |  0.00006 | -1.67939 | o  | 14, 5, 0 |||
 66 |          | U        |  0.01808 | -1.66126 | o  | 11, 6, 0 |||
 67 |          | V        |  0.00025 | -1.53883 | o  | 12, 6, 0 |||
 68 |          | W        |  0.01146 | -2.78012 | o  | 13, 6, 0 |||
 69 |          | X        |  0.00082 | -2.21905 | o  | 14, 6, 0 |||
 70 |          | Y        |  0.00007 | -2.31156 | o  | 11, 7, 0 |||
 71 |          | Z        |  0.00002 | -2.88812 | o  | 12, 7, 0 |||
----|----------|----------|----------|----------|----|----------|----------|----------|-------
```

```
connection definition section :

target | site | source:weight
-------|------|----------------------------------------------------------------
    36 |      |  1: 0.95093,  6: 3.83328, 11: 1.54422, 16: 4.18840, 21: 4.59526
                12: 2.30336, 17:-3.28721, 22:-0.43977, 27: 1.19506, 32:-0.84080
                23:-4.97246, 28:-3.30117, 33: 3.26851,  4:-0.19479,  9: 1.33412
                34:-1.02822,  5:-2.79300, 10:-1.97733, 15:-0.45209, 20:-0.61497
    37 |      |  1:-0.93678,  6: 0.68963, 11:-0.94478, 16:-1.06968, 21:-0.47616
                12: 2.62854, 17: 5.05391, 22:-0.37275, 27: 0.12598, 32: 0.27619
                23:-1.45917, 28:-1.97934, 33: 1.01118,  4: 4.39595,  9:-2.78858
                34:-0.14939,  5: 1.80792, 10: 3.66679, 15: 2.53150, 20:-1.07000
    38 |      |  1: 2.44151,  6: 0.41693, 11: 2.12043, 16: 1.40761, 21: 1.83566
                12:-0.55002, 17: 2.08524, 22: 0.63304, 27: 0.27301, 32:-2.49952
                23: 3.14177, 28:-1.25889, 33:-6.35069,  4:-5.25082,  9: 0.01774
                34:-3.66092,  5: 3.24401, 10: 1.88082, 15: 6.44985, 20: 3.24165
    39 |      |  1: 5.17748,  6:-4.45709, 11:-0.65733, 16:-2.26190, 21:-2.69957
                12:-1.43420, 17: 0.33409, 22:-0.74423, 27:-1.38010, 32: 3.08174
                23:-4.42961, 28:-1.09858, 33: 2.09879,  4:-1.30835,  9: 0.79940
                34: 1.99276,  5: 2.61433, 10:-3.56919, 15: 1.00952, 20: 2.86899
    40 |      |  1: 3.03612,  6: 0.05247, 11:-3.20839, 16:-4.03382, 21:-3.55648
                12: 0.23398, 17: 1.33895, 22: 6.03206, 27:-0.01723, 32: 0.09160
                23:-1.07894, 28:-1.77930, 33: 1.59529,  4:-1.57236,  9: 0.74423
                34:-0.13875,  5: 5.30719, 10: 2.13168, 15:-2.34832, 20:-5.00616
    41 |      |  1:-4.41380,  6:-1.48152, 11:-2.62748, 16:-1.00557, 21:-0.06430
                12: 3.93844, 17:-4.01591, 22: 0.76102, 27:-0.36823, 32: 0.54661
                23: 4.15954, 28: 2.96118, 33:-3.30219,  4:-0.24202,  9: 1.56077
                34:-0.20287,  5:-1.46062, 10: 1.79490, 15: 1.96920, 20: 3.72459
    42 |      |  1: 1.97383,  6: 2.53253, 11: 2.04922, 16: 1.13969, 21: 1.81064
                12: 0.32565, 17: 4.64358, 22: 1.02883, 27:-1.05720, 32:-0.71916
                23:-1.00499, 28:-1.10925, 33:-3.18685,  4: 2.12575,  9: 0.36763
                34:-0.18372,  5:-4.93490, 10: 0.26375, 15:-2.02860, 20:-5.43881
    43 |      |  1: 0.07183,  6:-2.69081, 11:-1.24533, 16:-2.01347, 21:-1.36689
                12:-2.11356, 17: 1.24788, 22: 1.23107, 27: 0.27674, 32:-2.45891
                23: 5.17387, 28: 1.68170, 33:-2.30420,  4: 2.17011,  9: 0.86340
                34:-2.23131,  5:-0.11916, 10:-4.39609, 15:-2.92706, 20:-5.43783
    44 |      |  1:-1.27907,  6: 1.89325, 11:-0.60419, 16: 3.60368, 21: 4.24280
                12:-2.77766, 17: 1.01698, 22:-1.97236, 27: 1.38773, 32:-2.55429
                23: 1.95344, 28: 2.85157, 33:-0.55796,  4:-0.64082,  9: 1.92937
                34:-2.71524,  5: 5.31087, 10:-2.08897, 15:-5.75332, 20: 2.43438
    45 |      |  1:-1.22455,  6: 0.92594, 11: 1.13199, 16:-1.65062, 21:-1.41481
                12:-3.04575, 17:-3.21280, 22:-0.23726, 27: 2.11836, 32: 2.23237
                23: 5.96261, 28: 2.00822, 33: 2.97409,  4: 3.90943,  9: 1.54990
                34: 2.42877,  5:-3.58017, 10: 2.31309, 15:-4.01833, 20: 0.28834
    46 |      | 45:-3.97560, 44: 0.45729, 43:-1.16526, 42: 0.38939, 41: 2.80876
                36:-4.04184
    47 |      | 45:-4.88750, 44:-3.33955, 43:-1.72110, 42: 0.94756, 41:-2.24993
                36: 0.14327
    48 |      | 45: 1.02597, 44:-1.82773, 43:-1.04974, 42: 2.09881, 41:-0.53220
                36: 2.75042
    49 |      | 45: 1.58579, 44:-3.38572, 43:-0.89166, 42: 2.86233, 41: 2.25429
                36: 1.92163
    50 |      | 45:-2.95134, 44: 2.39376, 43:-2.95486, 42:-0.11771, 41:-2.41775
                36:-0.73749
    51 |      | 45:-4.16732, 44: 2.19092, 43: 3.46879, 42: 0.44175, 41:-2.47295
                36:-0.40437
    52 |      | 45: 1.78256, 44: 4.64443, 43:-2.50408, 42: 0.65889, 41:-2.52796
                36:-1.73887
    53 |      | 45:-3.64449, 44: 2.60025, 43:-1.57915, 42:-0.18638, 41:-4.14214
                36:-4.29717
    54 |      | 45:-0.45205, 44:-1.44890, 43: 5.23345, 42:-0.35289, 41: 2.43160
                36:-1.99719
    55 |      | 45: 0.46855, 44:-2.84431, 43:-1.80938, 42:-4.49606, 41: 1.16736
                36:-4.07946
```

```
   56 |       | 45:-1.27961, 44: 0.81393, 43: 2.66027, 42:-1.05007, 41: 0.47655
                36: 0.72057
   57 |       | 45:-1.21679, 44:-3.13145, 43:-0.69538, 42: 0.05268, 41:-3.12564
                36: 2.16523
   58 |       | 45:-3.44107, 44: 2.18362, 43:-1.60547, 42:-0.50213, 41: 2.47751
                36: 1.05544
   59 |       | 45:-5.06022, 44:-5.08796, 43:-0.74552, 42:-0.67009, 41: 2.90942
                36: 1.10170
   60 |       | 45:-0.03838, 44:-2.59148, 43:-0.98185, 42:-4.97460, 41: 1.03714
                36: 4.49324
   61 |       | 45:-3.65178, 44:-2.61389, 43:-1.76429, 42: 1.01853, 41:-3.90286
                36: 0.46651
   62 |       | 45: 1.17767, 44:-3.70962, 43:-1.11298, 42:-2.77810, 41: 2.22540
                36:-0.65937
   63 |       | 45: 1.52270, 44:-2.42352, 43: 2.59287, 42: 0.19606, 41:-3.00602
                36:-3.14334
   64 |       | 45: 3.00298, 44:-3.65709, 43:-1.65150, 42: 1.44547, 41:-3.98775
                36:-4.02590
   65 |       | 45: 0.56820, 44: 2.37122, 43: 2.89143, 42:-4.22975, 41:-2.32045
                36:-2.17370
   66 |       | 45:-4.13007, 44:-0.30654, 43:-0.63380, 42:-5.63405, 41:-1.78425
                36: 1.46460
   67 |       | 45: 1.66534, 44: 1.99220, 43:-1.22676, 42:-4.09076, 41:-3.58451
                36: 1.40745
   68 |       | 45: 0.60032, 44: 2.75100, 43:-1.17663, 42:-4.25699, 41: 1.60600
                36:-4.23531
   69 |       | 45:-4.31415, 44:-3.41717, 43:-0.77030, 42:-1.36290, 41: 1.92319
                36: 0.28689
   70 |       | 45: 0.54085, 44: 1.80036, 43:-2.19014, 42:-1.20720, 41: 1.54519
                36:-2.79769
   71 |       | 45: 0.00018, 44:-4.11360, 43: 0.85345, 42: 1.19947, 41: 1.22067
                36:-3.29634
-------|------|----------------------------------------------------------------
```

B.2 Example 2:

```
SNNS network definition file V3.0
generated at Fri Aug  3 00:25:42 1992

network name : xor
source files :
no. of units : 4
no. of connections : 5
no. of unit types : 2
no. of site types : 2


learning function : Quickprop
update function   : Topological_Order


site definition section :

 site name | site function
-----------|------------------
 inhibit   | Site_Pi
 excite    | Site_WeightedSum
-----------|------------------
```

```
type definition section :

name          | act func     | out func     |sites
--------------|--------------|--------------|---------
outType       | Act_Logistic | Out_Identity |
LongeroutType | Act_Logistic | Out_Identity |
--------------|--------------|--------------|---------


unit default section :

act      | bias     | st | subnet | layer | act func     | out func
---------|----------|----|--------|-------|--------------|-------------
 0.00000 |  0.00000 | h  |      0 |     1 | Act_Logistic | Out_Identity
---------|----------|----|--------|-------|--------------|-------------


unit definition section :

no. | typeName       | unitName | act      | bias     | st | position | act func
----|----------------|----------|----------|----------|----|----------|---------
  1 |                | in_1     |  1.00000 |  0.00000 | i  | 3,5,0    |||
  2 |                | in_2     |  1.00000 |  0.00000 | i  | 9,5,0    |||
  3 |                | hidden   |  0.04728 | -3.08885 | h  | 6,3,0    |||
  4 |                | result   |  0.10377 | -2.54932 | o  | 6,0,0    |||
----|----------------|----------|----------|----------|----|----------|---------


connection definition section :

target | site    | source:weight
-------|---------|------------------------------------------------------------
     3 |         | 1: 4.92521, 2:-4.83963
     4 |         | 1:-4.67122, 2: 4.53903, 3:11.11523
-------|---------|------------------------------------------------------------
```

Bibliography

[Aka69] H. Akaike. *Annals of the institute of statistical mathematics, 21*, chapter Fitting autoregressive models for prediction. 1969.

[Ama89] Shun-Ichi Amari. *Characteristics of Encoded Associative Memory.* Springer Verlag 1989, 1989.

[BD95] Michael R. Berthold and Jay Diamond. Boosting the performance of rbf networks with dynamic decay adjustment. In G. Tesauro, D. S. Touretzky, and T.K. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, 1995.

[BH93] D. Stork B. Hassibi. Second order derivatives for network pruning: Optimal Brain Surgeon. In T. J. Sejnowski G. E. Hinton and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems (NIPS) 5*, pages 164–171, San Mateo, 1993. Morgan Kaufmann Publishers Inc.

[Bie94] J. Biedermann. Anwendungen Neuronaler Netze beim VLSI-CAD. Diplomarbeit, Institut für Numerische und Angewandte Mathematik, Georg-August-Universität Göttingen, 1994.

[CG87a] G. A. Carpenter and S. Grossberg. A Massively Parallel Architecture for a Selforganizing Neural Pattern Recognition Machine. *Computer Vision, Graphics and Image Processing*, 37:54–115, 1987.

[CG87b] G. A. Carpenter and S. Grossberg. Stable self-organization of pattern recognition codes for analog input patterns. *Applied Optics*, 26:4919–4930, 1987.

[CG91] G. A. Carpenter and S. Grossberg. ARTMAP: Supervised Real-Time Learning and Classification of Nonstationary Data by a Self-Organizing Neural Network. *Neural Networks*, 4:543–564, 1991.

[D.E93] D.Elliott. A better activation function for artificial neural networks. ISR Technical Report TR 93-8, University of Maryland, 1993.

[DH73] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* Wiley & Sons, Inc, 1973.

[Elm90] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[Fah88] Scott E. Fahlman. Faster-learning variations on back-propagation: An empirical study. In T. J. Sejnowski G. E. Hinton and D. S. Touretzky, editors, *1988 Connectionist Models Summer School*, San Mateo, CA, 1988. Morgan Kaufmann.

[Fah91] S. E. Fahlman. The recurrent cascade-correlation architecture. Technical Report CMU-CS-91-100, School of Computer Science, Carnegie Mellon University, 1991.

[FL91] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, August 1991.

[GFL86] N.H. Goddard, M.A. Fanty, and K. Lynne. The Rochester Connectionist Simulator. Technical Report 189, Computer Science Department, Univ. of Rochester, June 1986.

[GHW79] G. H. Golub, M. Heath, and G. Wahba. *Technometrics, 21*, chapter Generalized cross-calidation as a method for choosing ridge parameter. 1979.

[GLML89] N. Goddard, K.J. Lynne, T. Mintz, and L.Bukys. The Rochester Connectionist Simulator. Technical Report 233 (revised), Univ. of Rochester, NY, oct 1989.

[God87] N. Goddard. *The Rochester Connectionist Simulator: User Manual.* Univ. of Rochester, NY, 1987.

[Har83] S. Harrington. *Computer Graphics - A Programming Approach.* McGraw-Hill, 1983.

[Her92] K.-U. Herrmann. ART – Adaptive Resonance Theory – Architekturen, Implementierung und Anwendung. Diplomarbeit 929, IPVR, Universität Stuttgart, 1992.

[HF91] M. Hoefeld and S. E. Fahlman. Learning with limited numerical precision using the cascade-correlation algorithm. Technical Report CMU-CS-91-130, School of Computer Science, Carnegie Mellon University, 1991.

[Hil85] W.D. Hillis. *The Connection Machine.* MIT Press, 1985.

[HS86a] W.D. Hillis and G.L. Steele. Data parallel algorithms. *ACM*, 29(12):1170–1183, 1986.

[HS86b] W.D. Hillis and G.L. Steele. Massively parallel computers: The Connection Machine and NONVON. *Science*, 231(4741):975–978, 1986.

[Hüb92] R. Hübner. 3d–Visualiserung der Topologie und der Aktivität neuronaler Netze. Diplomarbeit 846, IPVR, Universität Stuttgart, 1992.

[Hud92] M.J. Hudak. Rce classifiers: Theory and practice. *Cybernetics and Systems*, pages 483–515, 1992.

[JGHL80] G. G. Judge, W. E. Griffiths, R. C. Hill, and T. Lee. *The theory and practice of econometrics.* John Wiley & sons, 1980.

[Jor86a] M. I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 531–546, Hillsdale NJ, 1986. Erlbaum.

[Jor86b] M. I. Jordan. Serial order: A parallel distributed processing approach. Technical Report Nr. 8604, Institute for Cognitive Science, University of California, San Diego, La Jolla, California, 1986.

[JS91] R. Dow J. Sietsma. Creating Artificial Neural Networks That Generalize. *Neural Networks*, 4(1):67–79, 1991.

[KKLT92] T. Kohonen, J. Kangas, J. Laaksoonen, and K. Torkkola. Lvq_pak learning vector quantization program package. Technical report, Laboratory of Computer and Information Science Rakentajanaukio 2 C, 1991 - 1992.

[KL90] J. Kindermann and A. Linden. Inversion of neural networks by gradient descent. *Parallel Computing*, 14:277–286, 1990.

[Koh88] T Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, 1988.

[Koh89] Teuvo Kohonen. *Self-Organization and Associative Memory, Third Edition.* Springer Verlag 1989, 1989.

[Kor89] T. Korb. Entwurf und Implementierung einer deklarativen Sprache zur Beschreibung neuronaler Netze. Studienarbeit 789, IPVR, Universität Stuttgart, 1989.

[Kub91] G. Kubiak. Vorhersage von Börsenkursen mit neuronalen Netzen. Diplomarbeit 822, IPVR, Universität Stuttgart, 1991.

[KZ89] T. Korb and A. Zell. A declarative neural network description language. In *Microprocessing and Microprogramming.* North-Holland, August 1989.

[Mac90] N. Mache. Entwurf und Realisierung eines effizienten Simulatorkerns für neuronale Netze. Studienarbeit 895, IPVR, Universität Stuttgart, 1990.

[Mam92] G. Mamier. Graphische Visualisierungs–Hilfsmittel für einen Simulator neuronaler Netze. Diplomarbeit 880, IPVR, Universität Stuttgart, 1992.

[MM89] P. Smolensky M. Mozer. Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems (NIPS) 1*, pages 107–115, San Mateo, 1989. Morgan Kaufmann Publishers Inc.

[Mol93] Martin Fodslette Moller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6:525–533, 1993.

[MP69] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry.* The MIT Press, Cambridge, Massachusetts, 1969.

[MR92] H.Braun M. Riedmiller. Rprop: A fast adaptive learning algorithm. In *Proc. of the Int. Symposium on Computer and Information Science VII*, 1992.

[MR93] H.Braun M. Riedmiller. Rprop: A fast and robust backpropagation learning strategy. In *Proc. of the ACNN*, 1993.

[P+88] William H. Press et al. *Numerical Recipes, The Art of Scientific Computing.* Cambridge University Press, 1988.

[Pet91] A. Petzold. Vergleich verschiedener Lernverfahren für neuronale Netze. Studienarbeit 940, IPVR, Universität Stuttgart, 1991.

[RB93] M Riedmiller and H Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks 1993 (ICNN 93)*, 1993.

[RCE82] D.L. Reilly, L.N. Cooper, and C. Elbaum. A neural model for category learning. *Biol. Cybernet.*, 45, 1982.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the microstructure of cognition; Vol. 1: Foundations*, Cambridge, Massachusetts, 1986. The MIT Press.

[Rie93] M Riedmiller. Untersuchungen zu konvergenz und generalisierungsverhalten überwachter lernverfahren mit dem SNNS. In *Proceedings of the SNNS 1993 workshop*, 1993.

[RM86] D.E. Rumelhart and J.L. McClelland. *Parallel Distributed Processing*, volume 1. MIT Press, 1986.

[Sch78] G. Schwarz. *Annals of statistics 6*, chapter Estimating the dimensions of a model. 1978.

[Sch91a] M. Schmalzl. Rotations- und translationsinvariante Erkennung von maschinengeschrieben Zeichen mit neuronalen Netzen. Studienarbeit 1011, IPVR, Universität Stuttgart, 1991.

[Sch91b] D. Schmidt. Anwendung neuronaler Netzwerkmodelle zur Erkennung und Klassifikation exogener und endogener Komponenten hirnelektrischer Potentiale. Studienarbeit 1010, IPVR, Universität Stuttgart, 1991.

[Sch94] T. Schreiner. Ausdünnungsverfahren für Neuronale Netze. Diplomarbeit 1140, IPVR, Universität Stuttgart, 1994.

[Shi68] R. Shibata. *Biometrika, 68*, chapter An optimal selection of Regression Variables. 1968.

[Sie91] J. Sienel. Kompensation von Störgeräuschen in Spracherkennungssystemen mittels neuronaler Netze. Studienarbeit 1037, IPVR, Universität Stuttgart, 1991.

[SK92] J. Schürmann and U. Kreßel. Mustererkennung mit statistischen Methoden. Technical report, Daimler-Benz AG, Forschungszentrum Ulm, Institut für Informatik, 1992.

[Som89] T. Sommer. Entwurf und Realisierung einer graphischen Benutzeroberfläche für einen Simulator konnektionistischer Netzwerke. Studienarbeit 746, IPVR, Universität Stuttgart, 1989.

[Soy93] T. Soyez. Prognose von Zeitreihen mit partiell recurrenten Netzen und Backpropagtion. Studienarbeit 1270, IPVR, University of Stuttgart, 1993.

[SUN86] SUN. Sunview user reference manual. Technical report, SUN microsystems, 1986.

[Vei91] A. Veigel. Rotations– und rotationsinvariante Erkennung handgeschriebener Zeichen mit neuronalen Netzwerken. Diplomarbeit 811, IPVR, Universität Stuttgart, 1991.

[Vog92] M. Vogt. Implementierung und Anwendung von 'Generalized Radial Basis Functions' in einem Simulator neuronaler Netze. Diplomarbeit 875, IPVR, Universität Stuttgart, 1992.

[Was89] Philip D. Wasserman. *Neural Computing*. Van Nostrand Reinhold, New York, 1989.

[Was95] Philip D. Wasserman. *Advanced Methods in Neural Computing*. Van Nostrand Reinhold, 1995.

[Weh94] C. Wehrfritz. Neuronale netze als statistische methode zur erklärung von klassifikationen. Master's thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, Lehrstuhl für Statistik 1, 1994.

[Wer88] P. Werbos. Backpropagation: Past and future. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 343–353. IEEE Press, 1988.

[WHH+89] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang. Phoneme Recognition Using Time Delay Neural Networks. *IEEE Transactions on Accoustics, Speech and Signal Processing*, 37:328–339, 1989.

[YLC90] S. Solla Y. Le Cun, J. Denker. Optimal Brain Damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems (NIPS) 2*, pages 598–605, San Mateo, 1990. Morgan Kaufmann Publishers Inc.

[You89] D.A. Young. *X-Window-System - Programming and Applications with Xt*. Prentice Hall, 1989.

[Zel94] Andreas Zell. *Simulation Neuronaler Netze*. Addison-Wesley, 1994. published in German.

[Zim91] P. Zimmerer. Translations– und rotationsinvariante Erkennung von Werkstücken mit neuronalen Netzwerken. Diplomarbeit 777, IPVR, Universität Stuttgart, 1991.

[Zip90] D. Zipser. Subgrouping reduces compexity and speeds up learning in recurrent networks. In D.S. Touretzky, editor, *Advances in Neural Information Processing systems II*, pages 638–641, San Mateo, California, 1990. Morgan Kaufmann.

[ZKSB89] A. Zell, T. Korb, T. Sommer, and R. Bayer. Netsim: Ein Simulator für neuronale Netze. In *GWAI-89*. Springer-Verlag (Informatik-Fachberichte), 1989.

[ZMS90] A. Zell, N. Mache, and T. Sommer. Applications of neural networks. In *Proc. Applications of Neural Networks Conf., SPIE*, volume 1469, pages 535–544, Orlando Florida, 1990. Aerospace Sensing Intl. Symposium.

[ZMSK91a] A. Zell, N. Mache, T. Sommer, and T. Korb. Design of the SNNS neural network simulator. In *Östreichische Artificial-Intelligence-Tagung*, pages 93–102, Wien, 1991. Informatik-Fachberichte 287, Springer Verlag.

[ZMSK91b] A. Zell, N. Mache, T. Sommer, and T. Korb. Recent Developments of the SNNS Neural Network Simulator. In *Proc. Applications of Neural Networks Conf., SPIE*, volume 1469, pages 708–719, Orlando Florida, 1991. Aerospace Sensing Intl. Symposium.

[ZMSK91c] A. Zell, N. Mache, T. Sommer, and T. Korb. The SNNS Neural Network Simulator. In *GWAI-91, 15. Fachtagung für künstliche Intelligenz*, pages 254–263. Informatik-Fachberichte 285, Springer Verlag, 1991.

[ZZ91] P. Zimmerer and A. Zell. Translations– und rotationsinvariante Erkennung von Werkstücken mit neuronalen Netzwerken. In *Informatik-Fachberichte 290*, pages 51–58, München, 1991. DAGM Symposium.

Index