blue yonder

Forward looking. Forward thinking.

The NeuroBayes® User's Guide

Version November 3, 2016

Contents

1 Introduction **5**
1.1 Getting NeuroBayes® 5
1.2 How NeuroBayes® works 5

2 NeuroBayes® Teacher: Training the network **7**
2.1 Setting up NeuroBayes®-Teacher in general 7
2.2 Tips and Tricks for setting up the Teacher 13
2.2.1 Training with low statistics 13
2.2.2 Training with high statistics 13
2.2.3 Training with weights 14
2.2.4 Surrogate training 14
2.2.5 Set up NeuroBayes® for density estimation 14
2.2.6 Set up NeuroBayes® for classification 15

3 NeuroBayes®-Expert: Using NeuroBayes® for analysis **17**
3.1 Setting up NeuroBayes®-Expert in general 17
3.1.1 Using the Expert for shape-reconstruction 20
3.1.2 Using the Expert for classification 21

A Technical details of the Teacher **23**
A.1 Preprocessing 23
A.1.1 Switches for the global preprocessing 24
A.1.2 Individual variable preprocessing 25
A.1.3 Preprocessing with orthogonal polynomials 28
A.1.4 Ranking of the input variables 29
A.2 Regularisation 31
A.3 Momentum 31
A.4 Pruning 32

B Technical details of the Expert **33**
B.1 Trimmed mean 33

C Reference to function calls **35**
C.1 Interface for the Teacher 35
C.2 Interface for the Expert 44

Chapter 1

Introduction

This section provides a step-by-step guide to set up a new network with the NeuroBayes® package (please refer to [Fei01] for details). The package is written in Fortran but a wrapper to C++ is available on github.com.

1.1 Getting NeuroBayes®

The NeuroBayes® neural network is available under the NeuroBayes® license from Blue Yonder GmbH. Please contact **"neurobayes@blue-yonder.com"** for general information and for information about the license. NeuroBayes® is available for the Linux operating system.

1.2 How NeuroBayes® works

The NeuroBayes® package consists of several libraries (which are located in the directory `$NEUROBAYES/lib`), several examples, wrappers allowing to call NeuroBayes® from different programming languages and some utilities.
The NeuroBayes® neural network is divided into a kernel- and an interface part. The kernel contains all functions needed by NeuroBayes® for training or analysis, whereas the interface contains all functions needed by the user to interact with the kernel.
In order to use NeuroBayes® for your own programs, the libraries have to be linked into your program.
The neural network is trained by calling the NeuroBayes®-Teacher: this will set up the network topology, the NeuroBayes® parameters and perform the actual training. At the end of the training, the trained network (called the "expertise") is written to a file with a filename chosen by the user (e.g. `myneurobayes.nb`). This file contains all information needed to run an

analysis, e.g. the network parameters and all weights. After the training, the NeuroBayes®-Expert is used for analysing unknown events.

Chapter 2

NeuroBayes® Teacher: Training the network

2.1 Setting up NeuroBayes®-Teacher in general

This section describes how to set up the Teacher using the Fortran functions directly. You may want to use the C++ Interface for better convenience. Figure 2.1 illustrates the concept.
Please note that your license may limit the maximum number of training patterns, the maximum number of nodes and the maximum number of layers which you are allowed to use. These limits are defined in the file `nb_param.hh` which is located in the directory `$NEUROBAYES/include` and are hard-coded into the libraries. Since these limits cannot be changed by the user, a new license is required if you wish to exceed the limits.

In a first step, the network topology and steering parameters are set up. Then the training patterns are read in and the actual training is performed. After the training is completed, the "network", i.e. an array holding all relevant information (called the "expertise") may be written to the disk for later use. Note that you may have several networks at the same time since each network is uniquely described by such a file.
As a first step, two arrays required for the training need to be declared, one for the training patterns and one holding the expertise. The array holding the training patterns is a two-dimensional real-valued array called "IN". Its length is `NB_MAXDIM` (which is the maximum number of nodes you are allowed to use plus three) for the first index and `NB_MAXPATTERN` (the maximum number of training patterns) for the second index. (When using C/C++, the two

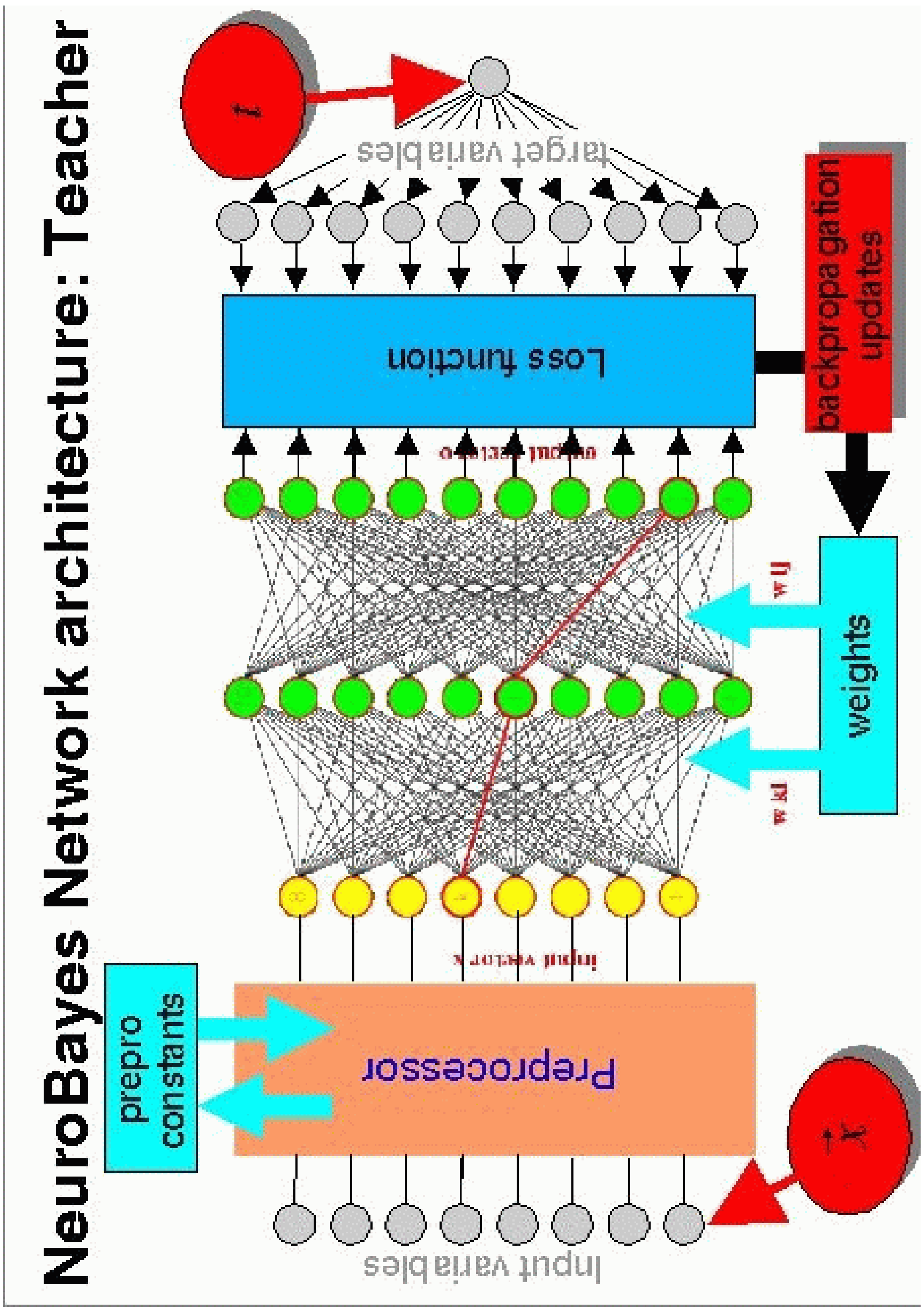

Figure 2.1: The NeuroBayes® Teacher architecture

indices have to be exchanged.)
The array holding the expertise is a one-dimensional real-valued array called ``EXPERTISE''. Its length is `NB_NEXPERTISE` which is calculated from the maximum number of layers/nodes that you are allowed to use.
The following steps are needed to setup the NeuroBayes® package:

1. Initialise NeuroBayes®-Teacher to default values: This is done via a call to the subroutine `NB_DEF`

2. Define the network task: This is done with a call to the subroutine `NB_DEF_TASK (CHTASK)` . The function expects one of the following two character-type arguments: `CLASSIFICATION` or `DENSITY`. Note that only the first three letters are actually checked.

3. Define the starting network architecture: This is done by calling a subroutine for each of the layers. Note that the maximum number of nodes and network layers is restricted by your license. The number of nodes in the input layer is defined by the number of variables you choose to train the network with plus one for the bias node. If `NVAR` is your number of variables, the input layer is set up via a call to the subroutine `NB_DEF_NODE1 (NVAR+1)` . The number of nodes in the hidden layer is set up by calling the subroutine `NB_DEF_NODE2 (nodes2)`.
   The third layer corresponds to the output layer, the number of nodes depends on the network task: In case of a binary classification (a yes/no distinction), the number of output nodes is one. In case of a density estimation, a number of nodes in the output layer of 20 is recommended. The number of nodes in the output layer is set up by a call to the subroutine `NB_DEF_NODE3(node3)`. Note that all arguments are integers.

The NeuroBayes® package is capable of pruning (see appendix A.4 for details), thus the choice of the number of nodes in the hidden layer is not very critical. However, a too small number of nodes may limit the learning capabilities of this network, whereas a too large number of intermediate nodes is not dangerous but training may take very long. On the other hand, if too many nodes are available, the network may learn certain features of the training-patterns by heart which limits the generalisation abilities of the trained network. In general, it is preferable to have a small network; the user should vary the number of nodes to find the optimal choice for the specific problem.
The network is now ready to run. However, several optional parameters may be modified by the user. The naming convention for the parameters is: 'chopt' is used for character-type variables, 'iopt' is used for integer-type variables and 'opt' is used for real-valued variables.

Type of regularisation: Possible choices are: `OFF`, `REG` (default), `ARD` ,`ASR`, `ALL`. The parameter is set up with a call to the subroutine `NB_DEF_REG (chopt)`, where the character-type argument is one of the choices above. Please refer to appendix A.2 for details.

Type of preprocessing: The default value is "12", see A.1 for details. This option is set up by a call to the subroutine `NB_DEF_PRE(iopt)` with the integer-valued parameter opt indicating the desired type of preprocessing. Preprocessing type 32 is recommended for density training. It is additionally possible to define the preprocessing for each input variable separately, see appendix A.1.2 for details.

Initial pruning This option is set up by call to the subroutine `NB_DEF_INITIALPRUNE (iopt)`. The number of remaining input variables after initial pruning are passed by the integer-valued argument. Note that this option is only useful for preprocessing-type 32 and shape-reconstruction. Please refer to appendix A.1 for details.

Type of loss-function: Possible choices are `ENTROPY` (default), `QUADRATIC` and `COMBINED`.
It is recommended to keep the default value since only here the error has a physical meaning. This parameter is set up by a call to the subroutine `NB_DEF_LOSS (chopt)` with the character-type argument type holding one of the choices above. Furthermore a term corresponding to the deviation of the signal purity, as a function of the network output, from the diagonal can be added to the loss-function. The user can switch this option by calling `NB_DEF_LEARNDIAG (opt)` with argument 1 (on) or 0 (off). The default is off.

Shape treatment: Possible choices are `OFF`, `INCL`, `MARGINAL`, `DIAG` and `TOTL` .
If you choose `INCL`, direct connections from the input to the output layer are set to describe the inclusive distribution. If you choose `TOTL`, direct connections from the input to the output layer are set to describe the linear density estimation. This option is set by a call to the subroutine `NB_DEF_SHAPE (chopt)` where the character-type argument is one of the options given above. When the option `DIAG` is chosen, at the end of the preprocessing procedure the network output is transformed so that the signal purity versus the network output is distributed along the diagonal.The option `MARGINAL` allows to substitute the network with a marginal sum method [MR04] and it is usable only for classification trainings.
It is recommended, to use the option `INCL` for shape-reconstruction.

Momentum: Optionally, a momentum can be specified for the training. Please refer to appendix A.3 for details. This parameter is set up by a call to the subroutine `NB_DEF_MOM (opt)` with the real-valued parameter opt. The momentum term may lie in the interval [0.0,1.0[. The default value is 0.0.

Weight update: Normally, the weights are updated every 200 events. If needed, this can be changed by a call to the subroutine `NB_DEF_EPOCH (iopt)` with the integer-valued parameter iopt specifying the number of events after which the weight update should be done.

Ratio train/test sample: As a default, the network uses all presented training patterns for training. This is very useful in case of low statistics. If sufficient statistics is available, a fraction of the presented training patterns may be used by the network for testing. This option can be set up by a call to the subroutine `NB_DEF_RTRAIN (opt)` with the real-valued parameter `opt` specifying the fraction of presented events used for training. The parameter opt has to lie in the interval [0.0, 1.0], where `opt=1.0` is the default.

Number of training iterations: This parameter defines the number of training iterations, i.e. the number of times all training patterns are presented. This option can be modified by a call to the function `NB_DEF_ITER (iopt)` with the integer - valued parameter iopt specifying the number of complete iterations. As a default, 100 iterations are performed. It is possible to perform 0 iterations, which means that the neural network does not run and the results of the preprocessing are saved into the expertise. In some cases, e.g. when training with the option `DIAG` for the shape, the results are meaningful and can be applied to new data.

Increase learning speed: A multiplicative factor may be set by a call to the function `NB_DEF_SPEED(opt)` by which the learning speed calculated by NeuroBayes® is multiplied (depending on the problem up to a factor of 1000). Thus, the network will learn faster but might not learn as well as with a low learning speed. It is recommended that only advanced users use this parameter. By default, a speed-factor of 1.0 (do not increase learning speed) is used.

Limit learning speed: The maximal learning speed may be limited by a call to the function `NB_DEF_MAXLEARN(opt)`. If the learning speed calculated by NeuroBayes® exceeds this limit, the user-provided limit is taken as the new learning speed. This option is useful if you manually increased the learning speed by a call to `NB_DEF_SPEED(opt)` or if you have very few training patterns. It is recommended that only advanced users use this feature. By default, the learning rate is limited to be smaller than 1.0

Training Method: It is possible to use the *BFGS algorithm* [BPL95] for the training of the neural network. The option can be switched on by calling `NB_DEF_METHOD(chopt)` with the argument `BFGS`. This choice can be reset by calling the same function with argument `NOBFGS`. By default BFGS is not used.

After the optional parameters have been set up, the network training can be started. First, all training samples have to be read into the `IN` array defined above. The first index holds the values of all input variables for one event, whereas the second index contains the number of the event considered. (NoPattern is the number of the current pattern):

- `IN(1, NoPattern)` = training target 1, e.g. from Monte Carlo truth information or historical database.
- `IN(2, NoPattern)` = value of first input variable
- `IN(3, NoPattern)` = value of second input variable
- ...
- `IN(NVAR+1, NoPattern)` = value of last input variable
- `IN(NB_MAXNODE +1, NoPattern)` = weight (default value is 1)
- `IN(NB_MAXNODE +2, NoPattern)` = training target 1 [1]
- `IN(NB_MAXNODE +3, NoPattern)` = training target 2

After all training events are read in the actual training is started by a call to the subroutine `NB_TEACHER (NSAMPLES,IN,EXPERTISE)`. The three parameters are: The number of teaching samples (`NSAMPLES`) determined from the routine reading in the training events, the two-dimensional input array `IN` and the one-dimensional array `EXPERTISE` . The first two parameters (`NSAMPLES` and `IN`) are input parameters to the Teacher, whilst the `EXPERTISE` is the output of the training. This array holds all relevant information about the fully trained network.

[1] has to be the same as `IN(1, NoPattern)` to keep backward compatibility

After the training is completed, the expertise should be written out to a file by a call to the subroutine `NB_SAVEEXPERTISE (chopt, EXPERTISE)` where the first character - type variable chopt is the name of the file (e.g. `myneurobayes.nb`) the expertise is written to and the array `EXPERTISE` is the training output.

2.2 Tips and Tricks for setting up the Teacher

This section is intended to give some pragmatic hints about how to set up the different options of the Teacher.

2.2.1 Training with low statistics

This discussion is meant for users with only a few thousand (let's say below 10'000) training patterns.
Training a neural network with low statistics is always problematic, thus you should try to increase the number of training patterns. If this is not possible, you can still try to train NeuroBayes® but you have to be very careful about the optional settings.
The most critical part is the *learning speed*: If it is too high, the network may run out of control, i.e. the Teacher doesn't find it's way to the minimum. In these cases, an inconsistent architecture often results which the Teacher tries to avoid by pruning away the inconsistency. However, the complete network might be destroyed in this way. Thus, you should limit the learning speed to a very low value, e.g. to 0.01 by a call to the subroutine `CALL NB_DEF_MAXLEARN(IOPT)`. Consequently, you should not increase the learning speed by a call to `CALL NB_DEF_Speed(IOPT)` with a number greater than 1.0. You might try a number smaller than 1.0, though, since the learning speed is multiplied by this number. Since now the learning speed is very low, you will have to train much longer since the minimum can now be reached only after many iterations.
Normally, a *weight update* is done every 200 events. Since you don't have many events, you might want to try to perform a weight update earlier. This can be done by a call to the subroutine `NB_DEF_EPOCH(IOPT)` where the integer-valued argument should be a number smaller than 200.

2.2.2 Training with high statistics

It is always advisable to train a neural network with as many input patterns as possible. To fully train NeuroBayes®, no special action is required.

However, since now the network learns much more in a single iteration as in the case of low or intermediate amount of input patterns, the learning speed computed by NeuroBayes® may be increased without harming the training. This can be done by a call to `CALL NB_DEF_Speed(IOPT)`. The argument may be as large as 1000; to be on the save side, you should limit the learning speed by a call to `CALL NB_DEF_MAXLEARN(IOPT)` to avoid a too large learning speed which could lead NeuroBayes® off its way to the minimum.

2.2.3 Training with weights

It is possible to assign a *weight* to an input pattern, i.e. to tell the network that the particular pattern should be treated differently from other patterns. The weights are assigned when the array `IN` is filled and are stored internally in `IN(NB_MAXNODE+1, event)`. A weight of 1.0 means that the pattern should be taken as it is, a weight of 0.0 means that the pattern should be completely ignored. Any (real-valued) number is allowed and represents the degree of "acceptance" you want to assign to the particular training pattern.
There is yet another scenario where using weights can make sense. If a correct classification of "signal" is more (or less) important than a correct classification of "background", you can weight the loss function for signal with a factor a by using `CALL NB_DEF_LOSSWGT(a)`. This way the preprocessing is not affected, but only the training process. Note that the network output n can't be interpreted as a probability p any more in this case, but the following relation applies:

$$p = \frac{1}{1 + a\left(\frac{1}{n} - 1\right)} \tag{2.1}$$

2.2.4 Surrogate training

In order to estimate the level of noise present in the input patterns, the method of "surrogate training" has been developed which can be activated by a call to the `SUBROUTINE NB_DEF_SURRO(Seed)`, where `Seed` is a real valued argument used as a seed for a random number generator. This method tries to estimate what amount of the network output is determined by statistically relevant features of the training sample and how much noise the network has picked up.

2.2.5 Set up NeuroBayes® for density estimation

In this mode, the network is trained to reconstruct the probability density function (PDF) of the given target value.

In the file `vardef.f` containing the variable definitions, the variable `PERFORMANCE` is used to tell the network the desired target value: performance = target value, e.g. Monte Carlo truth information. Example: `PERFORMANCE = vtrue`.
Additionally, it is possible to assign cuts and weights to the training events here. The logical variable `LCUT` is set true if the current event is not to be used by NeuroBayes® for training. The variable `WEIGHT` is used to assign a weight to the current training event. Note that all cuts based on Monte Carlo truth information have to be removed prior to using NeuroBayes® Expert if data is to be analysed. The real-valued array X holds the variables that NeuroBayes® should use. Note that the first element, X(1), is reserved for the target value and has therefore to remain blank. All user-variables have to start at the second element of X.
It is recommended to make use of the feature, that NeuroBayes® may already know the inclusive distribution, i.e. issue a call to the subroutine `NB_DEF_SHAPE (chopt)` with either `chopt='INC'` or `chopt='TOT'`.

2.2.6 Set up NeuroBayes® for classification

When NeuroBayes® is used to perform a classification , it is trained to distinguish if an event is of type A or B. The number of nodes in the output layer has to be one. In binary classification set `PERFORMANCE` to zero if it is of type A and to one if it is of type B. The real-valued array X is used in the same way as in the above case of density estimation and holds the variables NeuroBayes® should use for training. The variables `LCUT` and `WEIGHT` are used as in the above case, as well.

Chapter 3

NeuroBayes®-Expert: Using NeuroBayes® for analysis

3.1 Setting up NeuroBayes®-Expert in general

This section describes how to set up the Expert in general. Figure 3.1 illustrates the concept. The NeuroBayes® Expert uses the expertise created by the Teacher. This file is read in by the Expert and used to restore the network topology, i.e. the number of nodes and layers, all weights, etc. Then the data is analysed event by event by the NeuroBayes® neural network and the desired quantities are calculated. Note that the NeuroBayes® Expert buffers the results already calculated from the same expertise and event, i.e. if you wish to calculate several quantities for the same event and expertise, most of the calculations need not to be redone. However, once either the expertise or the event (i.e. the input array X) change, the buffer is cleared and filled with the calculations for the new event. A sample output for three events is shown in figure 3.2.

In order for the Expert to read in the expertise, a one-dimensional, real-valued array called `EXPERTISE` of the same length (`NB_NEXPERTISE`) has to be defined. Furthermore, the one-dimensional, real-valued array X holding the variables used for the analysis has to be defined as well. Its length is `NB_MAXDIM` . This array is the same as the one used for the variables in the training.
Then the expertise has to be read in. This is the file created by the Teacher at the end of the training process. Note that since the network is completely defined by the file holding the NeuroBayes® expertise, several networks using different files and arrays may be used at the same time.
After the expertise is read in, the number of input variables has to be de-

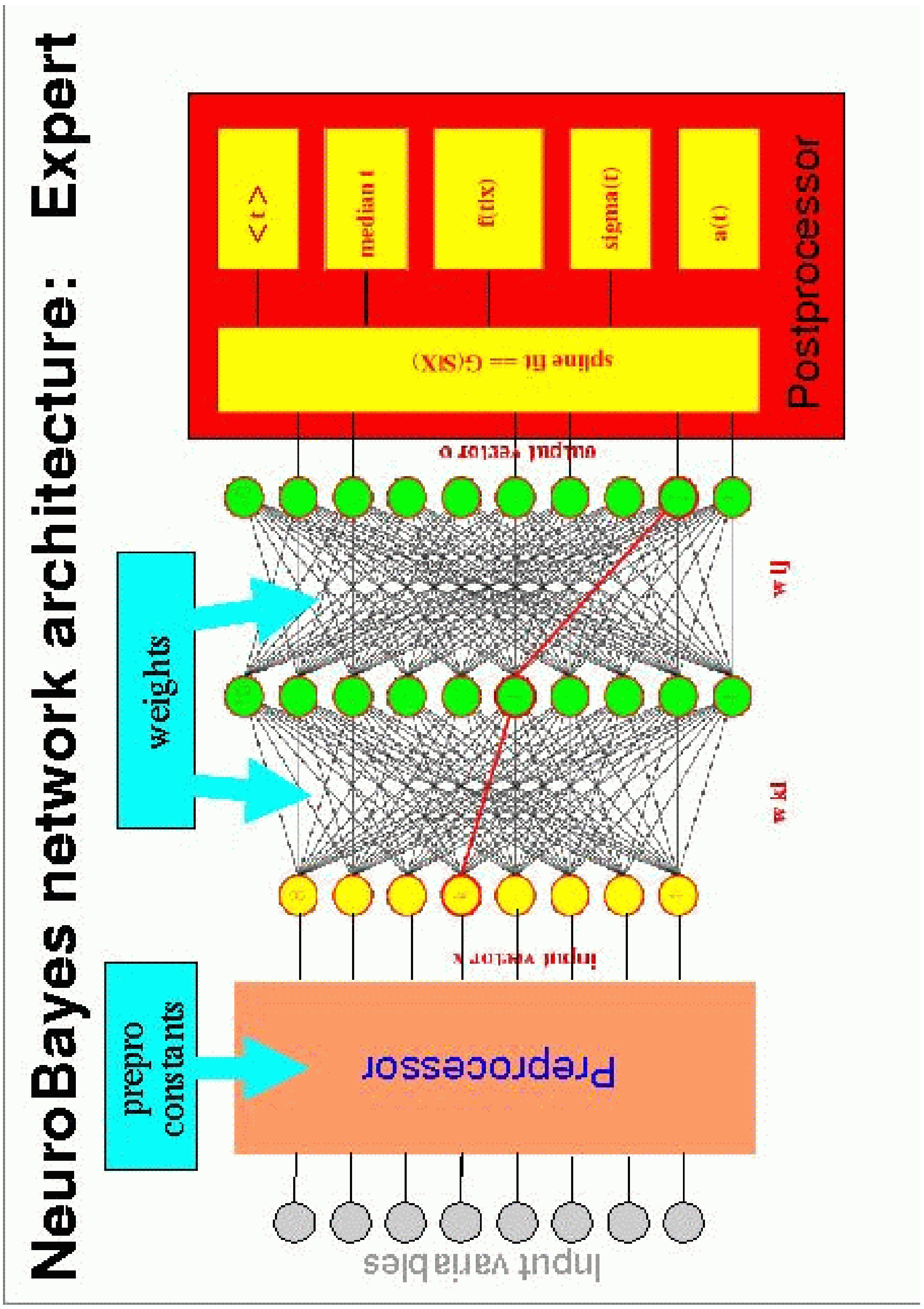

Figure 3.1: The NeuroBayes® Expert architecture

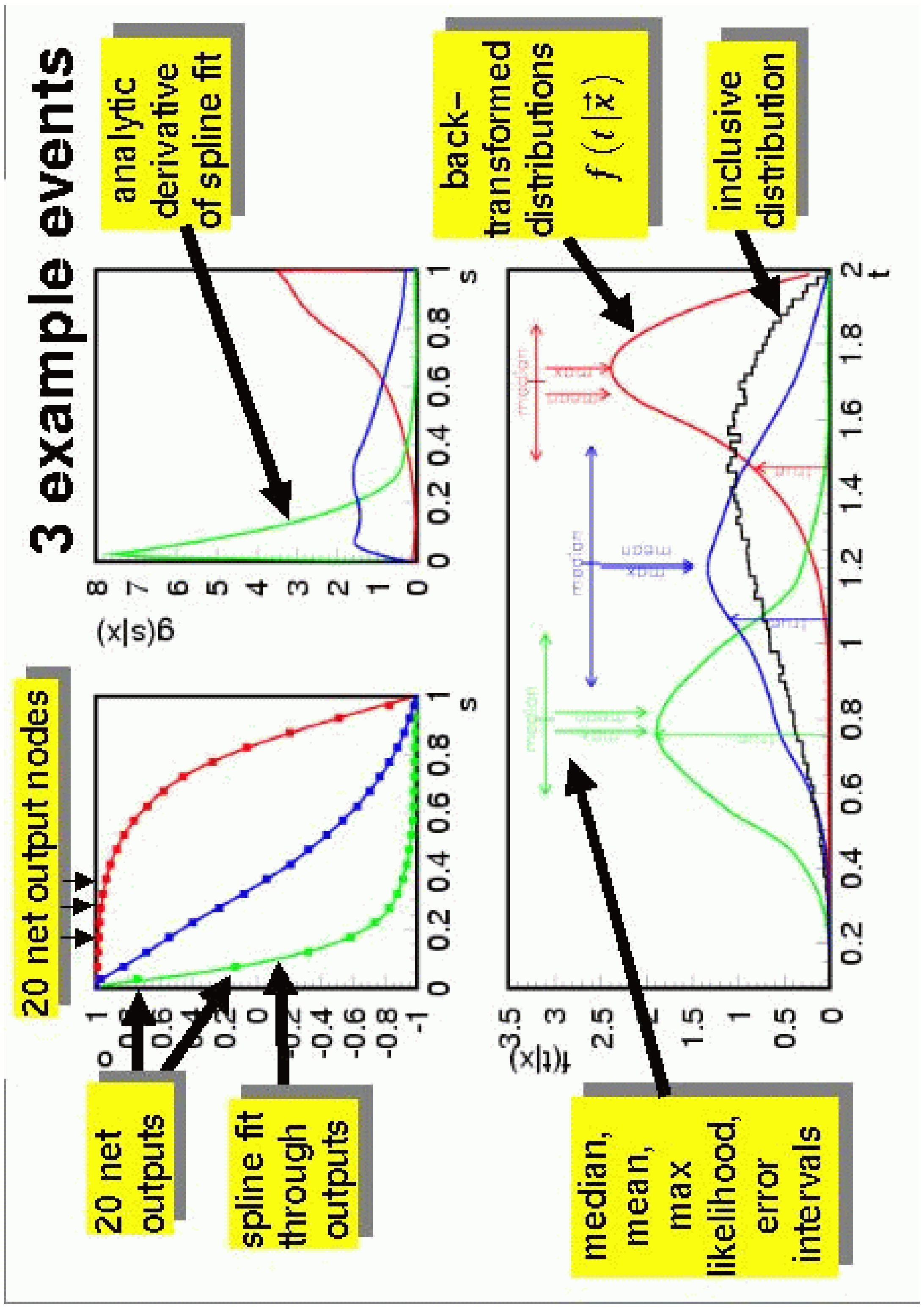

Figure 3.2: The NeuroBayes® Expert: sample output

termined from the expertise. This is done by assigning the return value (of type integer) of the function `NB_NVAREXPERTISE` to an integer-type variable, e.g. `NVAR = NB_NVAREXPERTISE(EXPERTISE)`.
Then you need to fill the array X with the values of the different variables NeuroBayes® should use to analyse the event. The array X has to be filled in the same way as for the training:

- `X(1)` = not used
- `X(2)` = value of first input variable
- `X(3)` = value of second input variable
- ...
- `X(NVAR+1)` = value of last input variable

Note that this array has to be filled for each event separately.
The actual analysis is done by calling the function nb_expert. This function takes as input arguments the name of the desired quantity, the expertise, the values of the input variables stored in the array X and a further argument which is needed for some quantities. Assuming the name of the value holding the output of the Expert (i.e. the value of the desired quantity) is named "output" (a real valued variable), the Expert is called by `output = nb_expert (action, expertise, X, T)`, where action is a character-type argument specifying the desired quantity to be calculated, expertise is the name of the array holding the expertise (i.e. `EXPERTISE` by default), X is the array holding the values of the input variables and T is a real-valued variable which is needed for some actions. If no further argument is required, a dummy variable has to be given.
Note that all possible actions are character-type variables and have to be passed in single quotes, e.g. the correct call for calculating the median is: `output = NB_EXPERT ('MEDIAN', EXPERTISE, X, 0.0) .`
When linked with the appropriate interface library to HBOOK, ASCII or ROOT, the Expert fills several control histograms which can be used to appraise the quality of the trained network. See section **??**.

3.1.1 Using the Expert for shape-reconstruction

NeuroBayes® is designed to estimate the *full* probability density function (PDF) of the analysed event.
Using the NeuroBayes® Expert (see section C.2 for details), quantities such

as the mean or the median (as well as an error estimate) of the distribution can be calculated. Note that since the shape estimate is not necessarily Gaussian, asymmetric errors may occur. Although the nomenclature σ corresponds to Gaussian distributions only, the terms have been used here for simplicity. A correct treatment can be found in [Fei01].

Extracting the full probability density function

The full probability density function (PDF) estimated by NeuroBayes® can be extracted on an event-by-event basis.
One way to access the full distribution is to fill a histogram using the provided action `PLOT` described in section C.2.

3.1.2 Using the Expert for classification

For binary classification problems (i.e. a yes/no question), the action `BINCLASS` is provided (depends on input vector X but not on argument T). The return value of the Expert lies in the interval $[-1.0, 1.0]$. Negative numbers indicate that the event does not belong to the desired class (the answer is "no"), whereas positive numbers indicate that the event belongs to the class (the answer is "yes"). The absolute value of the return value is a quality measure: The closer the value is to 1.0 or -1.0, the better the result is, i.e. in an ideal world the return value would only be either -1.0 or 1.0. If the network is perfectly trained, the probability that the answer is "yes" is (`NB_EXPERT()`$+1)/2$.

Appendix A

Technical details of the Teacher

A.1 Preprocessing

The preprocessing procedure prepares the input variables in a way that the network can handle them easily.
In a first step, the input variables are equalised: The original input variable may be distributed according to an arbitrary probability density function. This distribution is transformed to a flat distribution by a nonlinear transformation. This has the advantage that the user does not have to think about the properties of the input variables: If they are thought to be useful from a physical point of view, they can be directly put into the network without having any network-related constrictions in mind. In a second step, the flat distribution is transformed into a Gaussian with mean zero and $\sigma = 1$. At this point the variables are ranked according to the significance of their correlation to the target. This procedure is described in section A.1.4. The further preprocessing procedure de-correlates the N input variables from each other.
This procedure is called **global preprocessing** and it is applied to all variables. For single variables the procedure executed before the ranking and the decorrelation can be altered by the user by means of the **individual variable preprocessing** (see A.1.2 for details).

A.1.1 Switches for the global preprocessing

The global preprocessing is controlled by a flag composed as a three-digit integer *preproc* $= kij$[1]. The user sets a three-digit number, but in reality three different options are set. Therefore each digit has an own meaning. The meaning of i is:

- $i = 0$: do not perform de-correlation
- $i = 1$: de-correlate input variables and normalise
- $i = 2$: de-correlate input variables and rotate all linear dependence with target to the first new input variable, i.e. X(2)
- $i = 3$: de-correlate input variables and rotate according to correlation to moments of performance

and j means:

- $j = 0$: no preprocessing
- $j = 1$: flatten input variables
- $j = 2$: transform input variables to Gaussian distribution

The integer k switches on the *automatic variable selection* option, i.e. the user specifies a long list of possible input variables and the NeuroBayes® Teacher automatically decides which variables are taken for the training. The decision is based on the statistical significance of the input variable, which is computed as described in section A.1.4. This decision can be influenced by the user: Via the parameter k, the cut in terms of $\frac{1}{2}\sigma$ can be specified above which the variable is kept. If you do not want to use this feature, it is sufficient to treat the global preprocessing flag as a two-digit number, i.e. *preproc* $= ij$. Variables which are preprocessed by taking their correlation to the width of the target (see section A.1.2) are an exception to this rule and they are never excluded from the input set.
In detail, the value of k means:

- $k = 1$: keep variables which significance is at least 0.5σ
- $k = 2$: keep variables which significance is at least 1.0σ

[1]Starting from version `20021025`. In earlier version the flag is a two-digit integer number *preproc* $= ij$.

- ...
- $k = 9$: keep variables which significance is at least 4.5σ

Example: To keep only variables which are significant to at least $4\sigma (k = 8)$, de-correlate input variables and rotate according to correlation to moments of performance ($i = 3$) and transform input variables to Gaussian distribution ($j = 2$), choose $preproc = 832$.
The recommended setting for shape-reconstruction (i.e. the network learns the distribution of the target variable) is preproc=32.

A.1.2 Individual variable preprocessing

It is often useful to treat different input variables with different pre-processing flags. For this purpose a preprocessing flag and up to `NB_MaxPreproPar` preprocessing parameters can optionally be defined for each input variable separately. This is done via calls to `NB_DEF_PREPROFLAG` and `NB_DEF_PREPROPAR`[2].

If you are using the HBOOK or the ASCII interface, you have to define the individual preprocessing along with the network input variable definitions in the file `vardef.f`:

`X(NoVariable)` = variable name
`PreproFlag(NoVariable)` = preprocessing flag
`PreproPar(NoVariable,1)` = first parameter
`PreproPar(NoVariable,2)` = second parameter
...
`PreproPar(NoVariable,NB_MaxPreproPar)` = last parameter

The individual pre-processing flag is a three-digit integer number[3], `PreproFlag` = kij. Similarly to the global pre-processing flag, each digit steers different procedures of the variable transformation.

The digit j Possible values for j are:

- $j = 1$: no transformation[4]
- $j = 2$: transform to Gaussian[5]

[2]In versions prior to 20101109, this information had to be coded into the last (`NB_MaxPreproPar+1`) "events" of the `IN` array

[3]In versions of NeuroBayes earlier than 20060217, the flag is a two-digit integer number `PreproFlag` = ij. Starting from version 20060217 the flag is a three-digit integer number.

[4]forbidden.

[5]the only legal combinations with j are 12 and 92.

- $j = 3$: transform to flat distribution[6]
- $j = 4$: use result of regularised fit to mean values of target
- $j = 5$: use result of regularised monotonous fit to mean values of target
- $j = 8$: use regularised mean values of target for unordered classes
- $j = 9$: use regularised mean values of target for ordered classes

The digit i The digit i has been introduced to modify the action defined in digit j: In general it is very useful to flatten a distribution before performing a fit. Occasionally the user might however prefer the original distribution to be fitted. This is possible, but keep in mind that a variable often has to be pre-treated regarding its range, extreme values, etc. before it behaves well. Furthermore, distributions may contain δ-functions. This happens e.g. when the value of a variable is not known for each event. In the current version of NeuroBayes®, the user can demand a special treatment for *one* δ-function. Its value must be set to -999 beforehand. Since -999 is a special value for NeuroBayes®, the program will abort in case an input variable has values very close (± 0.5), but not identical to the value of the δ-function.
Another option is to correlate input variables not with the mean target, but with the width of the target distribution. This is interesting especially for quality-type variables. Allowed values for i are:

- $i = 1$: mean target, flatten the distribution, no δ-function
- $i = 2$: mean target, use original distribution, no δ-function
- $i = 3$: mean target, flatten the distribution, δ-function at -999
- $i = 4$: mean target, use original distribution, δ-function at -999
- $i = 5$: width of target, flatten the distribution, no δ-function
- $i = 6$: width of target, use original distribution, no δ-function
- $i = 7$: width of target, flatten the distribution, δ-function at -999
- $i = 8$: width of target, use original distribution, δ-function at -999
- $i = 9$: mean target, flatten the distribution, δ-function at -999.

[6]the only legal combinations with j are 23 and 93.

The flag $i = 9$ is similar to $i = 3$, with the exception that in the transformation the δ-function is set exactly to 0 and the distribution, except the δ-function, is transformed to have null mean and unit width.
Please note that for class-type variables ($j = 8, 9$) there is no point in flattening and thus NeuroBayes® makes no difference between e.g. $i = 1$ and $i = 2$.
It is important to note that the variables which are preprocessed with the correlation to the width of the target behave differently with respect to the significance and the automatic variable selection of the global preprocessing.These are never excluded from the input set, even if their significance falls below the cut set by the user via the global preprocessing flag. The significance is still computed as described in section A.1.4. However, since the significance is relative to a different property of the target distribution, it does not have the same meaning as for variables preprocessed with the correlation to the mean value of the target.

The digit k The digit k has only one possible value, $k = 1$. It can be used in case one has N times the same value in input. In order to treat the errors correctly when performing fits, the N can be given as a pre-processing parameter. The effect is for fits a scaling of error bars by $\sqrt{N}$, for class-type pre-processing ($j = 8, 9$) the number of class members is divided by N.

Individual preprocessing parameters have been invented in order to give users the possibility to pass additional information for the desired preprocessing. Parameters are recognised only for regularised spline fits ($j = 4$, $j = 5$) and (un)ordered classes ($j = 8$, $j = 9$). In these cases, one can optionally take into account previous variables (which as well have to be preprocessed with either $j = 4$, $j = 5$, $j = 8$ or $j = 9$) to be independent of correlations. The first parameter denotes the number n_p of variables to be taken into account. The following n_p parameters contain the identifiers of these variables (remember that only previous variables may be taken into account). For example (`vardef.f`, HBOOK interface):

```
...
X(3) = variable_name_3
PreproFlag(3) = 18
...
X(5) = variable_name_5
PreproFlag(5) = 35
PreproPar(5,1) = 1
PreproPar(5,2) = 3
```

```
...
X(8) = variable_name_8
PreproFlag(8) = 14
PreproPar(8,1) = 2
PreproPar(8,2) = 3
PreproPar(8,3) = 5
```

This would mean unordered class preprocessing for variable 3. For variable 5 a regularised monotonous spline fit is then applied, taking already into account the correlation to the target of variable 3 and using a special treatment for a δ-function at -999. If variable 3 and 5 were 100% correlated, the fit result would be a flat line since there would be no additional independent correlation to the target in variable 5. If they were 0% correlated, the fit result would be the same as if variable 3 was not accounted for in the fit. Finally, variable 8 is preprocessed with a regularised spline fit, where the correlation of variable 3 and the additional independent correlation of variable 5 are accounted for.
The pre-processing flag k is set by giving as first preprocessing parameter still the number of variables to take into account for de-correlation. If no de-correlation is intended, the first parameter has to be set to zero. The last parameter is N. For example, to set $N = 10$ for variable 3 and variable 5, the previous example has to be modified in the following way:

```
...
X(3) = variable_name_3
PreproFlag(3) = 118
PreproPar(3,1) = 0
PreproPar(3,2) = 10
...
X(5) = variable_name_5
PreproFlag(5) = 135
PreproPar(5,1) = 1
PreproPar(5,2) = 3
PreproPar(5,3) = 10
```

A.1.3 Preprocessing with orthogonal polynomials

The idea is that already the preprocessing gives a good linear estimate of the desired output. The network then only has to learn the details of the distribution and nonlinear corrections to the initial estimate. This corresponds

to having direct connections between the input- and the output-layer of the network. Note that this option is only useful for shape-reconstruction.
This type of preprocessing is done by expanding the training target (i.e. the truth information) in orthogonal polynomials.

Initial Pruning

After the (user defined) input variables have been preprocessed with orthogonal polynomials, the new input variables have new meaning: In the first variable, all correlation to the first orthogonal polynomial is stored, in the second, all correlation to the second orthogonal polynomial, etc. Since the higher-oder polynomials strongly oscillate, the corresponding (transformed) input variables might make network training difficult. Thus, it might be an advantage to prune away the input variables of the higher order polynomials. This can be done by a call to the subroutine `NB_DEF_INITIALPRUNE(IOPT)`, where the integer-valued argument gives the number of (transformed) input variables which should be kept, i.e. up to which order the polynomials should be used. Note that you will lose information from your original (not preprocessed) input variables, since you decrease the number of transformed input nodes.

A.1.4 Ranking of the input variables

The ranking of the input variables on the base of their significance is one of the most useful features of NeuroBayes®.
The correlation matrix of the N input variables and the total correlation of the input set to the target are computed after the variables have been preprocessed. If no individual preprocessing is requested, or if a monotonous fit is performed (see section A.1.2 for details), the correlation of the variable to the target is expected to be similar to that of the original variable, otherwise the correlation might be rather different.
After the correlation matrix is computed, one variable at at the time is removed from the input set and the correlation to the target is computed again. The variables are then sorted according to the loss of correlation to target caused by their exclusion. The variable causing the least loss of information, i.e. the least significant variable, is discarded. The correlation matrix is computed again and the procedure of removing one variable at the time is then repeated with $N-1$ variables. After the second least significant variable is removed, the procedure is repeated with $N-2$ variables and so on, until only one variable, i.e. the most significant one, remains. For what concerns the ranking, the significance of a variable is equal to the loss of correlation

caused by the removal of this variable at the relevant point in the procedure described here, multiplied by $\sqrt{n}$, where n is the sample size.
In NeuroBayes® there are different quantities describing the importance of an input variable. These quantities are typically printed out by NeuroBayes® at the end of the preprocessing. The quantities, given in units of σ, are explained in the following.

Additional significance: This is the significance computed for a variable with the iterative method described above. It is the quantity used for the ranking and for the pruning method, that is the cut on significance to retain only the most important variables. NeuroBayes® prints out the additional significance for each input variable after issuing the log message "`variables sorted by significance`".

Significance of this variable only: This quantity is the correlation of a variable to the target multiplied by $\sqrt{n}$, where n is the sample size. The computation does not take into account other variables. For the most significant variable this value is equal to the additional significance. The value of this quantity is printed for each variable after the log message "`correlations of single variables to target`".

Significance loss when the variable is removed: This is the loss of correlation multiplied by $\sqrt{n}$ when only this variable is removed from the input set and the total correlation to the target is re-computed with $N-1$ variables, in the first iteration of the method described above. Therefore for the least significant variable the significance loss and the additional significance have the same value. For each variable the significance loss is printed by NeuroBayes® after the message "`significance loss when removing single variables`".

Global correlation to other variables: This quantity is the correlation of a variable to all the others, computed with the complete $N \times N$ matrix. The global correlation is printed after the message "`global correlations between input variables`" is issued.

The user might choose to discard some input variables according to the additional significance. This is the figure of merit automatically chosen by NeuroBayes® when a significance cut has been requested via the global preprocessing flag (see section A.1.1 for instructions on how to set the cut).
Please note that a significance cut does not exclude variables which are preprocessed with the correlation to the width of the target, as mentioned in section A.1.2. These variables have to be removed explicitly by the user.

A.2 Regularisation

This paragraph briefly describes the different regularisation options available for the neural network. At the moment, four different regularisation types are available:

- `OFF` : no regularisation
- `REG` : Bayesian regularisation scheme (default)
- `ARD` : Automatic Relevance Detection (on inputs)
- `ASR` : Automatic Shape Regularisation (on outputs)
- `ALL` : ARD and ASR (on inputs and outputs)

In the *Bayesian regularisation* procedure, the weights of all nodes are divided into three different classes: One class for the bias node in the input layer, one class for all input nodes except the bias node and one class for all weights from the hidden layer to the output layer of the neural network.
In the *Automatic Relevance Detection (ARD)* all input nodes get their own regularisation constants which are independent from each other.
In the *Automatic Shape Regularisation (ASR)* procedure all output nodes get their own regularisation constants independent of each other– similar to ARD – .
The number of regularisation constants is summarised in table A.1.

Table A.1: number of regularisation constants

Layer	'standard'	ARD	ASR	ARD+ASR
bias → hidden	1	1	1	1
input → hidden	1	N_{input}	1	N_{input}
hidden → output	1	1	N_{output}	N_{output}

A.3 Momentum

Adding a momentum term helps the neural network to get out of local minima.This technique simply adds a fraction of the previous weight update to the current one. When the gradient keeps pointing in the same direction, this will increase the size of the steps taken towards the minimum. When the gradient keeps changing direction, the momentum term will smooth out the variations.

A.4 Pruning

The NeuroBayes®-network is optimising its structure during the learning process using pruning: The individual connections are multiplied by an exponentially small weight during the learning iterations (This is called “weight-decay’). If the network does not ’revive’ the connection, it will eventually become too small to contribute to the network. This connection is then removed completely (“pruned away”) from the network and the next learning iteration is done for the remaining smaller network. If a connection is removed, the network prints out a message like “kill weight from layer x knot y to knot z”.
The NeuroBayes® Teacher uses a scheme based on the current learn path for defining the pruning limit. The starting value can be set via a call to `NB_DEF_PRUNEMIN (value)` which should be set to a quite small number, e.g. 10^{-5}. The final value of the pruning scheme can be set `NB_DEF_PRUNEMAX (value)`. Here a quite large number should be chosen, e.g. 10^{-1}. The pruning algorithm then interpolates between these two numbers.
A further pruning option is to kill the network if its significance is below some specified cut. This cut can be set via a call to `NB_DEF_PRUNERESULT(sigma)` which sets the cut in terms of σ. Note that this feature is only useful when the inclusive distribution is fixed during the training.

Appendix B

Technical details of the Expert

B.1 Trimmed mean

A robust estimator is the trimmed mean, which cuts away a certain fraction of the tails of the distribution and computes the mean from the remaining distribution: If for example n measured points are available, the $(1-2r)n/2$ largest and smallest points are not considered, the mean is computed from the remaining $2rn$ points.
The trimmed mean depends on a (real-valued) parameter r ($r \in [0.0, 0.5]$). In the case of $r = 0.5$ the trimmed mean is identical to the "normal" mean, for $r \to 0$ the trimmed mean becomes the median. Figure B.1 shows the asymptotic efficiency of the trimmed mean as a function of the parameter r for several symmetric distributions (normal-distribution, Cauchy-distribution, double-exponential-distribution). The picture has been taken from [V.B98]. The trimmed mean can be used as a robust estimator if the actual distribu-

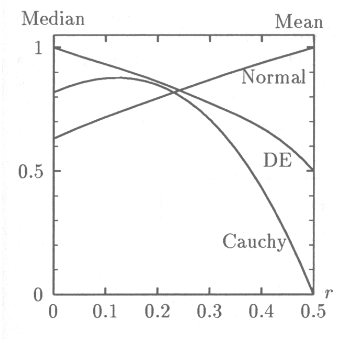

Figure B.1: Asymptotic efficiency of trimmed mean for several symmetric distributions

tion is not known to maximise the minimal possible efficiency.

Appendix C

Reference to function calls

C.1 Interface for the Teacher

In this section the functions which allow the user to interact with the Teacher are referenced. For each of them, the purpose, the needed input and the output are described and the corresponding function to call in the C++ interface is indicated. In some cases this indication is missing. This means that the C++ interface has a wrapper called exactly like the FORTRAN function, including the capitalisation of the characters, and accepting exactly the same arguments in input.

`SUBROUTINE NB_DEF()`

Purpose: Initialises the Teacher.

C++ equivalent: The method `NeuroBayesTeacher::NB_DEF (bool resetInput)` calls the FORTRAN function. When the default argument `resetInput` is set to `false`, the teacher input array is not initialised. This speeds up the initialisation, but it is not recommended since it may cause trouble, e.g. in a cross-validation training.

`SUBROUTINE NB_DEF_DEBUG (IDEBUG)`

Purpose: Sets the debugging flag of the Teacher.

Input: `IDEBUG` integer-valued variable setting the debugging verbosity level of the Teacher. Any integer between -2 and $+2$ is accepted.

parameter	meaning
-2	don't print anything
-1	print only header
0	print some information, no debug print-out
1	print calls to subroutines, value of certain parameters, ...
2	print arrays content at initialisation and at other stages

```
SUBROUTINE NB_DEF_EPOCH (NEPOCH)
```

Purpose: Defines the number of events sampled before a new weight update is done.

Input: `NEPOCH` integer-valued variable holding the number of epochs after which a weight update is done. Any value between 1 and the total number of events is valid.

```
SUBROUTINE NB_DEF_INITIALPRUNE (IOPT)
```

Purpose: Defines the number of remaining input variables after initial pruning. This option is meaningful only for shape reconstruction and preprocessing scheme 32. Further details con be found in section A.1.

Input: `IOPT` integer-valued argument to indicate the number of remaining input variables after initial pruning.

```
SUBROUTINE NB_DEF_ITER (NITER)
```

Purpose: Defines the number of complete iterations in the training, i.e. the number of times all training patterns are presented to the network.

Input: `NITER` integer-valued variable holding the number of training iterations. Any value larger or equal to 0 is valid. In NeuroBayes® versions earlier than `20060321`, `NITER=0` is not accepted and at least one iteration is always executed.

```
SUBROUTINE NB_DEF_LEARNDIAG (VALUE)
```

Purpose: Allows to include in the training error function a term corresponding to the distance of the signal purity from the diagonal.

Input: `VALUE` integer-valued variable. When the value 1 is passed, the extra term is included in the error function. When 0 is passed, the term is not added (default).

`SUBROUTINE NB_DEF_LOSS (CHLOSS)`

Purpose: Defines the loss function to be minimised.

Input: `CHLOSS` character-type array specifying the type of loss function. Valid choices are `'QUADRATIC'` and `'ENTROPY'`. Only the first three characters of the string are actually checked.

`SUBROUTINE NB_DEF_LOSSWGT (AWGT)`

Purpose: Sets the weight for the loss function for signal events.

Input: `AWGT` real-valued signal weight factor.

`SUBROUTINE NB_DEF_MAXLEARN (MAX)`

Purpose: Set an upper limit to the learning rate.

Input: `MAX` real-valued variable describing the upper limit of the learning rate. By default, an upper limit of 1.0 is used.

`SUBROUTINE NB_DEF_METHOD (CHMETHOD)`

Purpose: Allows to set the BFGS algorithms as training method. For more information, please see section 2.1 and [BPL95].

Input: `CHMETHOD` character variable corresponding to the training method. The possible choices are `'BFGS'` and `'NOBFGS'` (default).

`SUBROUTINE NB_DEF_MOM (AMOMENTUM)`

Purpose: Defines the momentum term used for the training. Please refer to section A.3 for details.

Input: `AMOMENTUM` real-valued variable specifying the momentum used for the training. Valid choices are values larger than 0 and smaller than 1.

`SUBROUTINE NB_DEF_NODE1 (NODE1)`

Purpose: Defines the number of nodes in the first layer (input layer).

Input:	`NODE1`	integer-valued variable specifying the number of nodes in the input layer. This should always be the number of input variables plus one (for the bias node), e.g. if the number of input variables is `NVAR` , the subroutine should be called with argument `NODE1 = NVAR + 1`

```
SUBROUTINE NB_DEF_NODE2 (NODE2)
```

Purpose: Defines the number of nodes in the intermediate layer (hidden layer).

Input:	`NODE2`	integer-valued variable specifying the number of nodes in the hidden layer. If too few hidden nodes are chosen, the network's learning ability may be limited, if too many hidden nodes are chosen, training will take a long time.

```
SUBROUTINE NB_DEF_NODE3 (NODE3)
```

Purpose: Defines the number of nodes in the output layer.

Input:	`NODE3`	integer-valued variable specifying the number of nodes in the output layer. The choice of the number depends on the task the network is trained to perform.

```
SUBROUTINE NB_DEF_PRE (IOPT)
```

Purpose: Defines the global preprocessing scheme. Further details are given in section A.1.

Input:	`IOPT`	integer-valued argument to indicate the preprocessing scheme. The default value is 12.

```
SUBROUTINE NB_DEF_PREPROFLAG (NODE,FLAG)
```

Purpose: Defines an individual preprocessing flag.

Input:	`NODE`	integer-valued input node number for which the flag is to be set.
	`FLAG`	integer-valued individual preprocessing flag for the given node. Preprocessing flags are described in appendix A.1.2.

```
SUBROUTINE NB_DEF_PREPROPAR (NODE,PAR,VALUE)
```

Purpose: Defines an individual preprocessing parameter.

Input:	`NODE`	integer-valued input node number for which the parameter is to be set.
	`PAR`	integer-valued individual preprocessing parameter number to be set.
	`VALUE`	real-valued individual preprocessing parameter value for the given node and parameter number. Preprocessing parameters are described in appendix A.1.2.

```
SUBROUTINE NB_DEF_PRUNEMAX (VALUE)
```

Purpose: Sets the final value for network pruning.

Input:	`VALUE`	real-valued variable setting the final value for pruning network connections. A quite high value should be chosen, e.g. 10^{-1}.

```
SUBROUTINE NB_DEF_PRUNEMIN (VALUE)
```

Purpose: Sets the starting value for network pruning.

Input:	`VALUE`	real-valued variable setting the starting value for pruning network connections. A quite low value should be chosen, e.g. 10^{-5}.

```
SUBROUTINE NB_DEF_PRUNERESULT (SIGMA)
```

Purpose: Kills an insignificant network. This feature is meaningful only when the inclusive shape is fixed.

Input:	`SIGMA`	real-valued variable specifying the cut below which the network is pruned away completely. The cut has to be specified in terms of σ.

```
SUBROUTINE NB_DEF_QUANTILE (VALUE)
```

Purpose: Defines a quantile of a continuous target distribution to be used as threshold for a classification. This is useful to compare how a classification training performs in comparison with a certain output node of a density training.

Input: `VALUE` real-valued variable corresponding to the requested quantile. Values between 0 and 1 are allowed.

```
SUBROUTINE NB_DEF_RANSEED (ISEED)
```

Purpose: Sets the random seed for the training.

Input: `ISEED` integer-valued seed.

```
SUBROUTINE NB_DEF_REG (CHREG)
```

Purpose: Defines the regularisation scheme used during the training. Please refer to section A.2 for details.

Input: `CHREG` character-valued array specifying the type of regularisation. Valid choices are `'OFF'`, `'REG'`, `'ARD'`, `'ASR'` and `'ALL'` .

```
SUBROUTINE NB_DEF_RELIMPORTANCE (RELIMPO)
```

Purpose: Sets the relative weight of the output nodes in the error function.

Input: `RELIMPO` real-valued variable describing the relative importance of the output nodes, `RELIMPO` $\in [0.0, 1.0]$. By default, `RELIMPO` = 0.0 is used (all output nodes have the same relative importance). Setting `RELIMPO` to 1.0 (outside nodes get larger weights) has been observed to give good results in high resolution samples.

```
SUBROUTINE NB_DEF_RTRAIN (RTRAIN)
```

Purpose: Defines the fraction of events that is used for the actual training. If the fraction is set to a value smaller than 1.0, the remaining patterns will be used for testing.

Input: `RTRAIN` real-valued variable holding the fraction of events used for training. Any number larger than 0.0 and smaller or equal to 1.0 is valid.

```
SUBROUTINE NB_DEF_SHAPE (CHSHAPE)
```

Purpose: Defines if direct connections between the input and the output layer are established.

Input:	`CHSHAPE`	character-type array defining the behaviour of direct connections. `CSHAPE` can take one of the following values:

`'OFF'`	no direct connections between input and output layer.
`'INC'`	direct connections between input and output layer are established to describe the inclusive distribution.
`'TOT'`	direct connections between input and output layer are established to describe the linear density estimation.
`'DIAG'`	a spline fit to the output node result is performed so that the signal purity versus the network output is distributed along the diagonal after the preprocessing and before the training. This option can be used only with the global preprocessing options 22 (classification), 32 and 42 (shape reconstruction).
`'MARGINAL'`	a binomial marginal sum method [MR04] is substituted to the neural network. This method is not suited for problems with several input variables and correlated variables.

All values are legal for a density estimation, except for `MARGINAL`. Legal values for classification are `OFF, INC, DIAG, MARGINAL`.

```
SUBROUTINE NB_DEF_SPEED (SPEED)
```

Purpose: Defines a factor by which the learning speed is multiplied. This results in faster learning but the network may not learn as well.

Input:	`SPEED`	real-valued variable by which the learning rate is multiplied. By default, a value of `SPEED` $= 1.0$ is used.

```
SUBROUTINE NB_DEF_SURRO (SEED)
```

Purpose: Set the surrogate training mode to estimate statistical bias of preprocessing and neural network. Different analyses with different seeds can be used to observe the stability of the error estimate (cf. histograms $300 + i$).

Input:	`SEED`	real-valued variable setting the seed for the random number generator.

```
SUBROUTINE NB_DEF_TASK (CHTASK)
```

Purpose: Defines the type of task the the Teacher will perform.

Input: `CHTASK` character-type array holding the names of the different tasks. Valid choices for `CHTASK` are

`'CLASSIFICATION'`	the teacher learns to distinguish two classes of events
`'DENSITY'`	the teacher learns to reconstruct a probability density function
`'REGRESSION'`	obsolete

Only the first three elements of the character array are actually checked.

```
SUBROUTINE NB_PREPRO_AND_NETWORK (INUM,IN,EXPERTISE)
```

Purpose: Performs preprocessing and trains the NeuroBayes® neural network.

Input:	`INUM`	integer-valued variable specifying the total number of training patterns presented to the Teacher
	`IN`	real-valued array of size `(NB_MAXDIM, INUM)` holding the input variables for each event
Output:	`EXPERTISE`	real-valued array of length `NB_NEXPERTISE` containing the network topology, preprocessing constants and neural network weights.

C++ equivalent: This function is called by the method `TrainNet()` of the `NeuroBayesTeacher` class, which initialises and saves the control histograms as well.

```
SUBROUTINE NB_SAVEASCARRAY (FILENAME,EXPERTISE)
```

Purpose: Saves the expertise as a C-array in a file, which is meant to be included in the user's code, for example to load the expertise explicitly.

Input:	`FILENAME`	string specifying the name of the output file.
	`EXPERTISE`	real-valued array of length `NB_NEXPERTISE`, which is filled by the subroutine `NB_TEACHER`.

C++ equivalent: This function is called by the method `TrainNet()` of the `NeuroBayesTeacher` class, when a name for the resulting file has been passed via the `NeuroBayesTeacher::SetCArrayFile()` method.

`SUBROUTINE NB_SAVEEXPERTISE (FILENAME,EXPERTISE)`

Purpose: Saves the expertise in an ASCII file (the extension `.nb` is typically used).

Input:		
	`FILENAME`	string specifying the name of the output file.
	`EXPERTISE`	real-valued array of length `NB_NEXPERTISE`, filled by the subroutine `NB_TEACHER`.

C.2 Interface for the Expert

This section gives a reference for the functions through which the user can interact with the Expert. In the C++ interface, the `Expert` class has an equivalent method for most of the functions listed here.

The FORTRAN function `NB_EXPERT` and its entries have been removed from the FORTRAN interface. They are listed here only for illustration purposes for *C++ users.*

```
REAL FUNCTION NB_EXPERT (ACTION,EXPERTISE,X,T)
```

Purpose: Uses the trained network to analyse events.

Input:	`ACTION`	character-type variable specifying the desired quantity to be computed.
	`EXPERTISE`	real-valued array of length `NB_EXPERTISE`. This array hold all relevant information about the network (network topology, weights, ...)
	X	real-valued array of length `NB_NDIM` holding the input variables for the specific event to be analysed.
	T	real-valued variable, needed for some actions. If the variable is not needed for the desired action, a dummy value (e.g. 0.0) has to be given.

Output: This is the return value of the function as a real-valued variable.

Possible actions:

1. quantities depending on neither input array X nor on argument T:

`RNDINCL`	random number distributed according to the inclusive PDF
`TMAX`	maximum value of inclusive distribution
`TMIN`	minimum value of inclusive distribution

2. quantities not depending on input array X but on argument T:

`INCLDENSITY`	probability density value of inclusive distribution at argument T
`INVQINCL`	returns percentage of probability mass having a lower value than T for the inclusive distribution

3. quantities depending on input array X but not on argument T:

`BINCLASS`	the Expert returns a real number in the interval $[-1.0, 1.0]$ The return value is positive if the event belongs to the desired class (i.e. the answer is "yes") or negative if the event does not belong to the desired class (the answer is "no"). The closer the value is to the extreme values, the better the network estimate is.
`MEAN`	mean value of the estimated PDF
`MEDIAN`	median of the estimated PDF
`LERROR`	median -1σ of the estimated PDF [1]
`REGR`	obsolete option
`RERROR`	median $+1\sigma$ of the estimated PDF[1]
`RNDCOND`	random number distributed according to the conditional PDF

4. quantities depending on both input array X and argument T:

`CONDDENSITY`	conditional density at argument T
`QUANTILE`	quantile at argument T ($T \in [0.0, 1.0]$)
`INVQUANT`	returns percentage of probability mass having a lower value than T for the conditional probability density function (this is the inverse operation to `QUANTILE` described above.)
`PLOT`	the reconstructed PDF is plotted into histogram T (where T is the number of the desired histogram)
`TRIM`	trimmed mean of the distribution with parameter T ($T \in [0.0, 0.5]$). The mean of the distribution starting from the quantile $0.5-T$ and up to the quantile $0.5+T$ is computed.

The only meaningful action for a classification is `BINCLASS`.

Example: In order to calculate the median for a given event with input array X, with an expertise array called `EXPERTISE`, the following lines of code suffice:

```
REAL MED
...
MED = NB_EXPERT('MEDIAN',EXPERTISE,X,0.0)
```

C++ Equivalent: `nb_expert(ACTION key,float* X,float T)`, where `ACTION` is an enumeration with elements listed under "*Possible actions*". The `EXPERTISE` array does not need to be passed since it is a member of the `Expert` class and it is initialised in the constructor.

`REAL FUNCTION NB_EXPERT_FTMEAN (`F`)`

Purpose: Calculates expectation value of function F. This function has to be called after `NB_EXPERT` has been called at least once, otherwise some useful arrays are not yet correctly filled and the program is aborted.

Input:	F	Function of which the expectation value should be computed. F has to be declared in double precision and has to have one float argument.

C++ Equivalent: the method `NB_EXPERT_FTMEAN(double (*f)(float*))` of the `Expert` class is the equivalent of this function in the C++ interface.

```
REAL FUNCTION NB_EXPERT_GETPINP (XPREPRO)
```

Purpose: Fills an array with the values of the input variables after the preprocessing.

Output:	`XPREPRO`	real-valued array `XPREPRO` of dimension `NB_MAXDIM`, containing the preprocessed input set.

C++ Equivalent: the method `NB_EXPERT_GETPINP(float* XPREPRO)` of the `Expert` class is the equivalent of this function in the C++ interface. It has to be called after `nb_expert` to obtain a meaningful result.

```
REAL FUNCTION NB_FLATTOCOND (RNFLAT,TABXS,SC)
```

Purpose: Transforms a random number distributed uniformly in $[0.0, 1.0]$ to a random number which follows the conditional probability density of the considered event. This is the same as calling `NB_EXPERT` with the action `RNDCOND`.

Input:	`RNFLAT`	real-valued random number distributed uniformly in the interval [0,1]
	`TABXS`	smooth inclusive distribution (from `NB_FILLTABXS`)
	`SC`	g(s|x) spline coefficients (from `NB_SPLINECOEFF`)

C++ Equivalent: does not exist.

```
SUBROUTINE NB_DEF_DEBUGEXPERT(IDEBUG)
```

Purpose: Sets the debugging flag of the Expert.

Input: `IDEBUG` integer-valued variable corresponding to the debugging verbosity level of the Expert. Any integer between -2 and $+2$ is accepted.

parameter	meaning
-2	don't print anything
-1	print only header
0	normal output
1	write calls to subroutines, values of certain parameters, ...
2	most verbose, write arrays content at different stages

C++ Equivalent: The debug flag can be changed by specifying the second argument in the constructor of the `Expert` class. The default is 0. The debug flag can be set to a value larger than -1 only if a valid license is present.

```
SUBROUTINE NB_DEFGSPLINE (ModeIn,NIn,RegIn)
```

Purpose: Makes spline fit steerable when it is set before `NB_EXPERT` is called for the first time.

Input:

`ModeIn`	integer-valued variable switching between automatic and manual spline fit. `ModeIn = 0 :` automatic spline fit `ModeIn = 1 :` manual spline fit
`NIn`	integer-valued variable which determines the number of spline coefficients used in the fit (note that a equidistant binning in the interval [0,1] is used in the case of the manual spline fit). The number of spline-coefficients should not be larger than the number of nodes in the output-layer.
`RegIn`	real-valued variable determining the regularisation constant used for the spline fit.

C++ Equivalent: corresponds to the method `Expert::NB_EXPERT_DEFGSPLINE(int ModeIn, int NIn, float RegIn)`.

```
SUBROUTINE NB_FILLTABXS (EXPERTISE,TABXS)
```

Purpose: Fills the array `TABXS` used internally in the NeuroBayes® Expert. This array is needed when the function `NB_FLATTOCOND` is used.

Input: `EXPERTISE` real-valued array of length `NB_EXPERTISE`

Output: `TABXS` real-valued array of length `NB_NVALUE+20` which holds the internally used array `TABXS` and describes the smooth inclusive distribution.

C++ Equivalent: corresponds to the method `Expert::NB_EXPERT_FILLTABXS(float* TABXS)`.

```
SUBROUTINE NB_READEXPERTISE(FILENAME,EXPERTISE)
```

Purpose: Sets up the array containing the complete set of weights and parameters written out by the training procedure. This array is used internally to extract a prediction for each given input set.

Input: `FILENAME` character-type array containing the name of the expertise file that has to be read in

Output: `EXPERTISE` real-valued array of length `NB_NEXPERTISE`, holding the result of the training.

C++ Equivalent: a private wrapper of this function exists, which is called when an `Expert` object is constructed.

```
SUBROUTINE NB_SPLINECOEFF (SC)
```

Purpose: Fills an array with the spline coefficients used to describe the full Probability Density Function estimated by NeuroBayes® (in the transformed variable $g(s|x)$).

Output: `SC` real-valued array holding the spline coefficients

C++ Equivalent: does not exist.

Bibliography

[BPL95] R.H. Byrd and C. Zhu P. Lu, J. Nocedal. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific and Statistical Computing*, 16(5):1190–1208, 1995.

[Fei01] Michael Feindt. Neurobayes - a neural bayesian estimator for conditional probability densities. Technical Report IEKP-KA/01-1, Institut für experimentelle Kernphysik, Universität Karlsruhe, January 2001.

[MR04] K. D. Schmidt M. Radtke. *Handbuch zur Schadenreservierung.* Verlag Versicherungswirtschaft GmbH, 2004.

[V.B98] E.Lohrman V.Blobel. *Statistische und numerische Methoden der Datenanalyse.* Teubner Studienbücher, 1998.