October 11, 2013
Version 1.4

Training NeuroBayes® with the Code Generator

Phi-T Physics Information Technologies GmbH

The NeuroBayes® neural network can be trained either on Linux or Windows platforms by means of a code generator and a steering file. A script allows to automatize the procedure of generating the code, compiling, executing the training and perform a graphical evaluation of the results.
The packaging of NeuroBayes® with the code generator includes a data sample of the Data Mining Cup 2005[1] and two steering files with solutions proposed by Phi-T.
The purpose of this document is to guide the user through the usage of the example and to give a reference for the keywords of the code generator. It is possible to

- generate C++ code and visualize the results with Root (ASCII interface)
- generate FORTRAN code and visualize the results with PAW (ASCII or HBOOK interface).

At this stage only the usage of the C++/ASCII interface is described, leaving the explanation of the FORTRAN/ASCII and FORTRAN/HBOOK interface to a later version of this document.

1 Running an example

1.1 The Teacher

The provided steering files, *codegen_first_cc* and *codegen_dmc_cc*, are both relative to the Data Mining Cup 2005 contest. The choice of input variables and pre-processing are different in the two files, being *codegen_dmc_cc* a more advanced example.
The CSV input file with data from the Data Mining Cup 2005, *train.csv*, is also given.

[1] http://www.data-mining-cup.de/2005/News/Aktuelles/1136971809/

1.2 Linux Systems

When working on Linux machines, make sure that the environmental variables `NEUROBAYES`, `CODEGEN` and `PHIT_LICENCE_PATH` are properly set, that the NeuroBayes® license is valid and that the Root environment is setup[2].
Create a working directory, copy the steering files in it. Change the path of the input file in the steering file, e.g. *codegen_first_cc*, and set it to the actual path of *train.csv*.
Copy the batch script `$NEUROBAYES/tools/teacher.sh` in your working directory, or execute it from its original location. The training is started by typing

```
./teacher.sh codegen_first_cc
```

This command will:

- generate and compile the C++ code necessary to read in the input file;
- feed the Teacher with the variables specified in the steering file with the requested pre-processing flags;
- train the network;
- execute the post-training analysis.

The generated files and the results are saved in the time-stamped sub-folder `teacher_<date>_<time>`. The file `analysis.ps.gz` collects the post-training analysis graphs. For the meaning of the graphs, please refer to the NeuroBayes® C++ user guide [1].

1.3 Windows Systems

Before starting with the example provided with the NeuroBayes® package, make sure that you have a valid NeuroBayes® license in *C:\phitLicence\phitLicence.lic*, otherwise the Teacher and the Expert will not be able to start.
The *nbSandbox* folder is a working area which can be used to train the NeuroBayes® Teacher and to run the Expert. It contains a batch file, *teacher.bat*, and the two example steering files. The input file is stored in the nbSanbox folder as well. Change the path of the input file in the steering file, e.g. *codegen_first_cc*, and set it to the actual path of *train.csv*.
In order to get started, launch *StartNB* from your Start Menu. This will open a DOS shell window in which the environment variables needed to generate, compile and run programs using NeuroBayes® have been set.
You can launch a Teacher example by typing in the DOS shell:

```
teacher.bat codegen_first_cc
```

This command will:

- generate and compile the C++ code necessary to read in the input file;
- feed the Teacher with the variables specified in the steering file with the requested pre-processing flags;
- train the network;
- execute the post-training analysis.

The C++ code, the produced expertise and the Teacher output are stored in a "project folder" in *Projects*, which is marked by a timestamp (*teacher_<date>_<time>*). It is possible to define a custom name for this folder by passing a second command-line argument to *teacher.bat*, e.g.

```
teacher.bat codegen_first_cc FirstTest
```

[2] It is recommended to use Root version 4.04/02

will train the Teacher in the folder *Project\FirstTest*.
If Root is installed, the post-training distributions are plotted in the file *analysis.pdf*, which is stored in the project folder.

1.4 The Cross-Validation Training

In some cases the data sample available for a network training is very limited and training with low statistics might lead to over-training. The cross-validation is a technique to deal with low statistic samples and to check if a network has been over-trained. It consists in dividing the sample in N separated subsamples, performing N trainings, each with N-1 subsamples. Each of the N produced expertises is then applied to the subsample not used for the corresponding training.
In order to run a cross-validation, the corresponding option `CROSS_VALIDATION` has to be set in the steering file, specifying the number N of subsamples. The job is started then by typing simply:

Linux Systems: `./teacher.sh <codegen file>`

Windows Systems: `teacher.bat <codegen file>`

A single executable performs the following operations:

- trains the Teacher N times, producing N different expertise files;
- runs the Expert once applying the right expertise to each subsample;
- trains the Teacher once again with the complete sample and produce another expertise file;
- runs the Expert again on the full sample applying the expertise produced by the last training.

As a result, two output files are produced, one containing the Expert prediction with the cross-validation procedure and the other containing the prediction obtained with the complete training. In these files the sample index and the target value are also stored for each entry. A comparison between the predictions is shown in the file `outofsample.pdf`.
Since the Expert is run in the same job as the Teacher, the Expert script (`expert.sh` or `expert.bat`) is not generated in this case.

2 The steering file

In this section a reference for the keywords of the code generator is given. The steering file for the code generator contains **comments** and **keywords** followed by arguments. When a line starts with the character **#**, it is interpreted as a comment line. The keywords which can be used in a steering file allow either to set options for the code generation or to set the parameters for the Teacher or the Expert.
In the following the words after the keyword indicate if arguments should follow and their type (integer, floating point variable or string). Arguments shown in brackets are optional. Where it applies, it is indicated if the strings arguments have to be enclosed in quotes ("string") or in apices ('string').

2.1 Keywords specific for the code generator

The parameters set by the following keywords are used by the code generator:

ADDTARGET int : Switches the addition of the target and LCUT to the output on (1) and off (0).

APPEND "string" : the argument to pass is the name in quotes of a file to include in the generated code, either containing the definition of functions or a list of header files. When FORTRAN code is generated, the specified file is appended to every executable. In the C++ case, the file is included at the beginning of the file which contains the `INCLUDE_TEACHER` and the `INCLUDE_ALL` blocks.

APPEND_TO_LINKER string : the argument to pass is a path to an object file (without quotes), which will be linked to all the executables.

BOOST int int string string (string) : when this keyword is included in the codegen file, a boost training is performed on the sample.
The following parameters have to be given:

```
nBoost varset boostvar1 (boostvar2) (boostvarFinal) (target)
```

where :

- `nBoost` is an integer value indicating the number of sub-samples to be considered for the boost;
- `varset` is the set of variables that the user wishes to use for the boost training, it can also be equal to the varset index used with the `NETWORK` keyword;
- `boostVar1` is a string which specifies a name for the variables which contain the result of the first training (pre-boost);
- `boostVar2` is the name the variable containing the result of the boost for a classification training;
- `boostvarFinal` is only needed for a classification training and is the name given to the variable equal to the combination of the pre-boost and boost training.
- `taget` can be specified for classification training allowing for the generation of some plots comparing the boost training and the classification.

A multiboost training uses the same syntax as a single boost round. Up to ten boost rounds can be defined:

```
BOOST   nBoost varset  boostvar1 boostvar2
BOOST2  nBoost varset2 boostvar2 boostvar3
BOOST3  nBoost varset3 boostvar3 boostvar4
...
```

BOOSTDEF this keyword starts a block, which is ended with the keyword ENDBOOSTDEF, in which the weights W1 for the boost training are defined (similarly to an INCLUDE-block). The weights have to be written as elements of the array `weightArray`, which has two dimensions equal to the `number of boost rounds` and to the `nBoost` parameter, passed with the BOOST keyword:

```
weightArray[number of boost rounds][nBoost]
```

Internally it is checked that the sum of the elements of `weightArray` is equal to 1. By default when only one boost sample is used, the weight is set to 1 and the BOOSTDEF-block is not needed.

BOOSTTHRESHOLD int : sets the number of events in the "higher" boost samples.

CHARCONV_FILE string : specifies the file which has to be used to interpret character variables or dates, in case they are present in the input file (e.g. the variables Z_METHODE and B_GEBDATUM in the Data Mining Cup input file). The path can also be relative to the working directory.
Even if in the input file it is not needed to transform any variable, when generating C++ code charconv.cc should be edited leaving in the line `return atof(read);`.
If this file is not given, the Teacher and the Expert will not be correctly compiled.
If the function `convert` is not adapted to the project, the programs which use it will crash.

COMPILER string : specifies the language in which the code is going to be generated. The available options are g77 (default) and gcc. When gcc is chosen, C++ code will be generated and the code written by the user in the `INCLUDE` blocks (see below) has to be in C++ as well. Otherwise FORTRAN code is generated and expected in the `INCLUDE` blocks.

COMPILERFLAGS string : additional flags to to the compiler command in the makefile (only for the Linux version).

COST_FUNCTION float float float float : the four numbers which correspond to the elements of a 2×2 cost matrix. It is assumed that the elements are given in the following order:

decision	Signal	Background	truth
	1^{st}	2^{nd}	Signal
	3^{rd}	4^{th}	Background

When a cost function is given, the value which maximizes or minimizes the costs (optimal cut) is computed. The prediction for an event is the 1 if the Expert output is larger than the optimal cut and it is 0 otherwise.

CROSS_VALIDATION int (int) : specifies the number of sub-samples to generate for a cross-validation training. By default a training on the complete sample is executed as well. This can be avoided by adding an integer different from zero after the number of samples. See section 1.4 for more information about the cross-validation training.

DATAFILE string : absolute path of the input file(s). Up to 100 files can be given.

DATAFILE_EXPERT string string : similar to `DATAFILE` with the difference that the files specified here are not used to train the network but only opened by the Expert to write out the result. The prefix for the outputfile can be specified with the second parameter. If not specified, the files listed with the keyword `DATAFILE` are used by both the Teacher and the Expert. Otherwise the files listed with the keyword `DATAFILE` are only used by the Teacher and those listed with `DATAFILE_EXPERT` are only used by the Expert. Up to 100 files can be given.

DATAFORMAT_INPUT string : specifies the format of the input file. Valid options are `CSV` and `HBOOK` (only `CSV` is available for the Windows version).

DATAFORMAT_OUTPUT string : specifies the format of the output file. Valid options are `CSV` and `HBOOK` (only `CSV` is available for the Windows version).

DEBUG int : set the debug level for the Teacher and for the Expert according to the scale given in section C.1 of [2].

DELIMITER string : character indicating which is the delimiter between the fields in the input file for the `DATAFORMAT` option `CSV`.

EFFPUR (float) (float) ... (float) : the efficiency-purity graphs for the expert are shown in the file `nbDensity.pdf`. The option is only valid for a shape reconstruction. It accepts up to ten floating point variables with values in the interval $(0, 1)$, which correspond to the fraction of the inclusive distribution that have to be considered. If no arguments are given, the default array of values $0.1, 0.3, 0.5, 0.7, 0.9$ is used.

In case a cross-validation is performed, the graphs of the complete training are compared to those obtained with the split-sample training.

The operations needed to produce the efficiency-purity graphs for the expert output are rather time-consuming and extra output files are generated, therefore this option should be switched off when it is not necessary.

ENDBOOSTDEF indicates end of a `BOOSTDEF` block

ENDEXPERT_ACTION indicates end of an `EXPERT_ACTION` block

ENDINCLUDE indicates end of an INCLUDE block

ENDVARNAMES indicates end of a VARNAMES block

ENDVARSET indicates end of a VARSET block

EXCEL_DLL (Windows version only) indicates that the DLL needed to interface the Expert from Excel has to be compiled. No parameters are needed.

EXPERTONLY int : In order to use the Expert without the Teacher set to 1. Default value is 0.

EXPERT_ACTION this keyword starts a block in which a list of at most 20 expert actions and their parameters can be specified. When a parameter is not given for a certain action, it is set to 0 by default. The list is read out only when performing a density training.
The output for each action is written out to the CSV output file. The possible actions are referenced in section C.2 of [2]. For the INVQUANT action it is possible to specify as a parameter one of the variables in the input file or declared by the user in the INCLUDE_DATAF block. PERFORMANCE, i.e. the value of the target, and WEIGHT are valid parameters as well.
When the word FTMEAN is included in the block, the function nb_expert_ftmean() is called (see section C.2 of [2]). The output of nb_expert_ftmean() is the expectation value of a function defined by the user. This function has to be declared and defined in a file appended via the APPEND keyword. The declaration has to be:

FORTRAN: `double precision function ft_function(X)`

C++: `double ft_function(float *x)`

If the user wishes to pass a parameter to ft_function(), this can be achieved by

FORTRAN: using a common block

C++: declaring a global variable in the appended file.

The value of the parameter can be set in the INCLUDE_ALL block

HEADLINE int : specifies if a headline containing the variable names is present in the input file (true $\rightarrow$ 1; false $\rightarrow$ 0). If set to true, it has lower priority than the VARNAMES block.

HEADLINE_FILE string : name of a file containing the header of the CSV input file, if this is different from the input file.

IDTUP int : identifier of n-tuple (only for HBOOK).

INCLUDE_TEACHER_C or INCLUDE_TEACHER_D the code written between this keyword and ENDINCLUDE will be included in the Teacher program. Here the target variable (PERFORMANCE) should be set by writing

```
PERFORMANCE= <your target variable>
```

Eventually cuts can be applied to the input variables. The cuts specified in this block by means of the variable LCUT are only applied when the Teacher runs, not when the Expert runs. Therefore this block should contain the cuts which select the training sample. The weights are also typically defined in this block by setting the variable WEIGHT.

INCLUDE_DATAF in the block following this keyword new variables can be declared. The programming language corresponding to the given COMPILER option has to be used.

INCLUDE_ALL the code written in the following block is pasted as is in the Teacher and in the Expert program. Here cuts can be defined using the variable `LCUT`: if `LCUT` is true for a record, this record will be rejected and not be used to train the network (Teacher). Neither will this record get a prediction from the Expert, which will return the default value -2.
By setting the variable `WEIGHT`, the relative importance of the records is defined. This variable must be set, the default is to have it set to 1. See section 2.3.3 of [2] for more information about training with weighted events.
If you enclose some operations or cuts in an if-statement using the boolean variable `expertJob`, then the code will only be executed by the Expert. Conversely, if you write the following lines:

```
if(!expertJob) {
  // user code
}
```

the user code will only be executed by the Teacher.

NETWORK string int : the first parameter (`classify` or `density`) sets the type of training. The second parameter specifies which `VARSET` list has to be taken.

RECL int : record length of n-tuple (only for HBOOK).

SYSTEM string : specifies if the project is run under Windows (`win`) or Linux (`unix`, default).

TARGETMEDIAN int : sets the % of the targetparameter.

VARNAMES the list of the variable names in the input file should follow this keyword if `HEADLINE` has been set to 0. It is important to list the variable names in the order in which they appear in the input file to avoid confusion when the names of the variables are used to perform operations on the variables themselves (e.g. in the `INCLUDE_ALL` block). The keyword `ENDVARNAMES` marks the end of the list.

VARSET The number following this keyword is a tag for the list which follows. More than a list can be specified. The list of the variables which will be used to train the neural network is enclosed between `VARSET` and `ENDVARSET`.
The numbers following the variable names are the individual pre-processing flags (see section A.1.2 of [2] for further information about individual pre-processing).
The file *codegen_dmc_cc* used for the Data Mining Cup (see section 1.1) contains a more advanced example:

```
ANZ_BEST          29 2 2 4   # decorr:NEUKUNDE,Z_METHODE
                      a b c d
```

the number in the position `a` is the individual pre-processing flag; the number in `b` means that the variable should be de-correlated from 2 variables; `c` and `d` tell the ordinal numbers of the variables from which the de-correlation should be performed (the 2nd input node is `NEUKUNDE`, the 4th is `Z_METHODE`[3]). The target variable has to be excluded from this list. In general, the order in which the variables are given does not matter. A variable can be de-correlated only from variables preceding it in the list.

WORKDIR string : name of the folder in which the C++ code for the Teacher and for the Expert will be initially generated. The folder name is changed to that of the project folder in a second step. The default value is `temp`.

[3] Beware the FORTRAN numbering scheme: internally node n.1 is the target, node n.2 is the first input variable, etc.

2.2 Keywords specific for the Teacher and for the Expert

The following keywords allow to set the Teacher parameters. The arguments which they can accept and their effect corresponds to that of the functions NB_DEF_keyword.
For these keywords it is possible to append the characters _C or _D. When _C is appended it means that the setting is valid only for a classification training. Accordingly the keywords ending with _D specify parameters which are valid only for a density training.
If you want to perform a classification and a density training with the same input file, you might want to use the keywords with _C and _D in your steering file and switch between the training types by changing the NETWORK keyword. In this way a single steering file is enough to manage two types of training.
For further information, please consult section 2.1 of [2].

INITIALPRUNE int : the number following sets the maximum order of moments for the target distribution. This option can be set only when a density training is performed.

ITERATIONS int : number of iterations that the Teacher should perform. If this is set to 0, only the pre-processing will be executed. This is useful when the user would like to have a first look at the input variables and choose the preprocessing flags.

ITERATIONS_BOOST int : allows to set the number of iterations that the boost training should perform. When this parameter is not set, the boost training will go through as many iteration as the training in the first round, that is the number set by the ITERATIONS keyword. The default value for this parameter is 0.

LEARNDIAG int : include in the training error function an extra term depending on the distance of the signal purity from the diagonal. Valid choices for the parameter are 0 and 1.

LOSS 'string' : type of loss function, see section C.1 in [2] (reference for NB_DEF_LOSS) for the possible values.

LOSSWGT float : the given parameter which corresponds to the signal weight factor. This is an advanced option which can be used in a classification training to weight the signal differently than the background. This can be also achieved by setting the WEIGHT variable. The difference with respect to setting the WEIGHT variable is that the weight of the signal will not be affected for the pre-processing, but will be affected during the training. When WEIGHT is used, the signal weight changes for both the pre-processing and the training.

MAXLEARN float : limits the maximum learning speed.

METHOD 'string' : sets the training method to 'BFGS' or 'NOBFGS' (default). Information about the BFGS method can be found in [3].

NODE2 int : number of nodes in the intermediate layer. If this parameter is not specified, the Teacher will compute it. The keyword NODE2, without _C or _D, does not exist.

NODE3_D int : number of nodes in the output layer. The default value is 20. For a classification it cannot be set because it is always equal to 1.

PREPRO_C int : specifies the default preprocessing (12) for a classification. Please refer to section A.1.1 of [2] to learn how you can set a significance cut using this variable.

PREPRO_D int : specifies the default preprocessing (32) for a density training. Please refer to section A.1.1 of [2] to learn how you can set a significance cut using this variable.

QUANTILE_C float : sets a quantile of a continuous target distribution (between 0 and 1) to be used as threshold for a classification. The keyword QUANTILE_D does not exist.

RANSEED int : sets a random seed as a start value for the training.

REG 'string' : sets the regularization strategy, see C.1 in [2] (reference for NB_DEF_REG) for the possible options.

RELIMP float : sets the relative importance of the nodes, see C.1 in [2] (reference for NB_DEF_RELIMPORTANCE) for the meaning of this option. The keyword RELIMP, without _C or _D, does not exist.

RTRAIN float : fraction of the input sample which is used for the training. The remaining fraction is used for testing.

SHAPE 'string' : defines the shape treatment for a density training, see see C.1 in [2] (reference for NB_DEF_SHAPE) for the possible values.

SPEED float : sets the training speed.

SURRO float : specifies if a surrogate training should be performed. The argument is a random seed.

2.3 Example of a steering file

```
#WORKDIR temp
#SYSTEM win
COMPILER gcc

### data format
DATAFORMAT_INPUT CSV
DATAFORMAT_OUTPUT CSV

### Specify one or more data files. One DATAFILE statement for each file.
DATAFILE  /home/user/DMC/train.csv

### CSV delimiter
DELIMITER ';'

HEADLINE 1

### CSV character conversion
CHARCONV_FILE /home/user/DMC/charconv.cc

### NeuroBayes definitions
ITERATIONS 10
PREPRO_C 12
SPEED_C 10.

### NeuroBayes input variables
VARSET 1
NEUKUNDE          18
chk_3day          18
Z_METHODE         38 1 2     # decorr:NEUKUNDE
ANZ_BEST          29 1 2     # decorr:NEUKUNDE
ANZ_BEST          29 2 2 4   # decorr:NEUKUNDE,Z_METHODE
B_TELEFON
B_GEBDATUM        92
FLAG_LRIDENTISCH  8 2 2 4   # decorr:Z_METHODE
FLAG_NEWSLETTER
```

```
Z_CARD_VALID       34 2 2 4   # decorr:Z_METHODE
Z_LAST_NAME        38 2 2 4   # decorr:Z_METHODE
TIME_BEST          34
ANZ_BEST_GES       49 2 2 4   # decorr:NEUKUNDE,Z_METHODE
WERT_BEST_GES      34 2 2 4   # decorr:NEUKUNDE,Z_METHODE
DATUM_LBEST        34 2 2 4   # decorr:NEUKUNDE,Z_METHODE
MAHN_AKT           44 2 2 4   # decorr:NEUKUNDE,Z_METHODE
MAHN_HOECHST       44 2 2 4   # decorr:NEUKUNDE,Z_METHODE
ENDVARSET

### Network type
### classify [varset]
### density [varset]
NETWORK classify 1

EXPERT_ACTION
INCLDENSITY
MEDIAN
TRIM 0.3
LERROR
ENDEXPERT_ACTION

### Code to be included in the Teacher (_C: classify, _D: density)
### (PERFORMANCE, cuts (LCUT) etc.)

INCLUDE_TEACHER_C
PERFORMANCE = TARGET_BETRUG;
ENDINCLUDE

INCLUDE_TEACHER_D
ENDINCLUDE

INCLUDE_DATAF
float anum_01_1;
float chk_3day;
ENDINCLUDE

#SURRO_C 12345
#INITIALPRUNE_D 3

INCLUDE_ALL
anum_01_1 = (int)(ANUMMER_01/100000.);

chk_3day = 0.;
if (CHK_LADR == 1.||CHK_RADR == 1.||CHK_KTO == 1.
||CHK_CARD == 1.|| CHK_COOKIE == 1. || CHK_IP == 1.)
chk_3day = 1.;

// Delta-Functions --> -999.
if (ANZ_BEST_GES == 0.) ANZ_BEST_GES=-999.;
if (WERT_BEST_GES == 0.) WERT_BEST_GES=-999.;
```

```
// Cuts
LCUT = LCUT || false;

// Weights
WEIGHT = 1.;
LCUT = LCUT || (WEIGHT <= 0);

ENDINCLUDE
```

References

[1] "*The NeuroBayes® C++ Guide*", $NEUROBAYES/doc/HowToCpp.pdf

[2] "*The NeuroBayes® User's Guide*", $NEUROBAYES/doc/NeuroBayes-HowTo.pdf

[3] R.H. Byrd and P. Lu, J. Nocedal, C. Zhu, "*A Limited Memory Algorithm for Bound Constrained Optimization*" SIAM Journal on Scientific and Statistical Computing , (1995) 16, 5, pp. 1190-1208