October 11, 2013
Version 1.2

The NeuroBayes® C++ Guide

Phi-T Physics Information Technologies GmbH

1 Introduction

NeuroBayes® is a very sophisticated algorithm that is much more than just a normal feed-forward neural network. It helps in selecting relevant variables, can handle arbitrarily distributed variables including missing input, avoids over-training by implementing several regularization and pruning schemes. In addition to simple classification, NeuroBayes® can provide reconstruction of the probability density function of real-valued variables.
This document describes how to use the neural network package with C++. It is only intended as a complement to the NeuroBayes® user's guide [1]. The description is relative to the libraries compiled after Dec., 1^{st} 2005.
The NeuroBayes® libraries are currently available for two platforms: Linux and Windows.
The training of a neural network and the call to an expertise are only possible on machines for which the NeuroBayes® license is available. The license can be obtained upon request from Phi-T by sending an e-mail to `licence@phi-t.de`. When working with Linux, the environment variable `PHIT_LICENCE_PATH` has to be set to the to the directory containing the license file. In the Windows version the license has to be stored in the file `C:\phitLicence\phitLicence.lic`.
For what concerns the Linux version of the libraries, two sets exist for the Teacher and for the Expert: a shared library and a statically built library.

N.B.: The statically linked libraries require that the chosen interface is linked to the executable:

- `libNeuroBayesInterfaceASCII.a`: Control plots are written out to a text file

- `libNeuroBayesInterfaceDummy.a`: Control plots are not written out
- `libNeuroBayesHBOOK.a`: Control plots are written out in the `HBOOK` format to be used with `PAW`

In the Windows version, two dynamically linked libraries (DLL) contain the FORTRAN core of the Teacher and of the Expert and the interface:

- `nbascii.dll`: Teacher and Expert code linked with the ASCII interface, that is the control plots are written out to a text file
- `nbdummy.dll`: Teacher and Expert code linked with the dummy interface, that is the control plots are not written out

The `lib` files corresponding to these DLL are also available.

The C++ interface to the FORTRAN DLLs is compiled with Visual C++ and is in `NeuroBayesVC++.lib`. In the following it is assumed that the environment variable `NEUROBAYES` has been set to the directory in which NeuroBayes® is installed.

The `$NEUROBAYES` directory contains

- **Windows version only:** the DLLs and some helper programs in `bin/`
- the libraries in the subdirectory `lib/`
- the header of the Expert and Teacher class and other helper headers in `include/`
- an analysis macro to evaluate the quality of the teaching process in `cpptools/`
- example code for C++ programming in `examples/`
- the user's manual and this document in `doc/`

The user interface to the NeuroBayes® routines follows closely the description in the user's guide [1].

2 Network training

In C++, the NeuroBayes® Teacher and Expert are classes which have to be instantiated to be used. Due to the internal structure, the Teacher class is a singleton, therefore only one Teacher can be used in a session. There is no restriction on the number of instances for the NeuroBayes® Expert, several instances of this class can be run, e.g. to allow an analysis to use multiple networks in one session.

2.1 General remarks

Although the user interface to NeuroBayes® follows the description in [1], a few subtleties have to be taken into account. Since the Teacher is a singleton class, an instance of this class has to be created:

```
NeuroBayesTeacher* nb = NeuroBayesTeacher::Instance()
```

All further Teacher commands are then executed using this class, for example

```
nb -> DEF_NODE1(10);
```

In general, commands operating on NeuroBayes® directly are in all upper case, whereas commands handled by the C++ interface are in mixed case. These commands are in detail:

- `nb->SetOutputFile(const char*)`: sets a name for the expertise file.
- `nb->SetCArrayFile(const char* thisName)`: sets the name of the file in which the expertise is saved as a C-array.
- `nb->SetRootFile(char*)`: sets a name for the Root output file. This filename is ignored when the ASCII interface is used.
- `nb->SetWeight(float)`: defines the weight of an event. The default is 1.0.
- `nb->SetIndividualPreproFlag(int i,int flag)`: sets the preprocessing flag for the `i`-th variable. The possible values for `flag` and a description of their meaning can be found in Appendix A of [1].
- `nb->SetIndividualPreproParameter(int i,int j,int par)`: sets the `j`-th parameter for the `i`-th input variable.
- `nb->SetTarget(float)`: passes the target variable. For classification problems, the target variable defines if an event is signal ($1.$) or background ($0.$ or $-1.$). For density reconstruction the target is a continuous float variable.
- `nb->SetNextInput(int n,float var[n])`: for each event, it passes to the network an array of n input variables.
- `nb->TrainNet()`: it actually performs the training and writes out the result in the expertise file

2.1.1 Memory allocation

The memory to keep training events is dynamically allocated when `nb->SetNextInput` is called. When training with large datasets, dynamic reallocation of memory can be slow and may temporarily even use more memory than physically available. If the number of events to be used in the training is known in advance, it can be set via:

- `nb->SetNEvents(int)`: sets the number of training events to make sure enough memory is allocated to keep the whole dataset

Up to the given number of events, no reallocation is necessary. Please note that despite dynamic allocation there is still a hard upper limit defined by the library at `NB_MAXPATTERN`.

2.2 Training in a compiled C++ executable

In order to use the NeuroBayes® Teacher in a compiled executable, the header file NeuroBayesTeacher.hh has to be included. An instance of the Teacher class is then created via:

```
NeuroBayesTeacher* nb = NeuroBayesTeacher::Instance()
```

After filling the input variables and using the NeuroBayes® commands to define the network properties, the network can be trained. Remember to link in the appropriate interface library for the control plots.

Linux In your Makefile, add the option

```
-I$(NEUROBAYES)/include
```

to the compiling command to include the needed headers, and the option

```
-L$(NEUROBAYES)/lib -lNeuroBayesTeacherCPP
```

to the linking command.

Windows/Visual C++ Add the option

```
/I "%NEUROBAYES%\include\''
```

to the compiling command to include the needed header, and the list of needed libraries

```
"%NEUROBAYES%\lib\NeuroBayesVC++.lib"
"%NEUROBAYES%\lib\nbascii.lib"
```

to the linking command if, for example you want to use either the Teacher in conjuction with the ASCII interface. Make sure that the folder %NEUROBAYES%\bin is correctly included in the path, so that the DLLs can be found at runtime.

2.3 The Teacher text output

When the Teacher is started, at first all events are read in. Then the network is set up and the input variables are preprocessed. For each variable the preprocessor builds a table that contains the edges of bins with 1% of the variable distribution. For example, for a continuous variable one can have

```
Transdef: Tab for variable  4   -3.8429842 -0.782660127 -0.505313218
 -0.280599266 -0.150258929 -0.0792960748 -0.0474978536 -0.0328703821
 -0.0269424133 -0.0238223225  ...
```

The integral of the variable distribution between each pair of consecutive numbers in the table is 0.01. Such a table can help to discover bad behavior of input variables and to choose different preprocessing strategies.

The preprocessor computes the correlation matrix of the input variables. For a set of 7 variables one can have for example:

```
COVARIANCE MATRIX (IN PERCENT)

 0    1.0    2.0    3.0    4.0    5.0    6.0    7.0    8.0
 0    0.8    1.0    1.0    1.0    1.0    1.0    1.0    1.0
 1  100.0   -0.5   21.4   30.9  -15.8  -28.6  -12.0  -14.2
 2   -0.5  100.0   -0.7   -5.5   -0.2   -1.2    1.3   -0.7
 3   21.4   -0.7  100.0    6.2  -11.4   21.6   -3.6   -4.0
 4   30.9   -5.5    6.2  100.0   -5.5  -15.4   -2.9   -2.2
 5  -15.8   -0.2  -11.4   -5.5  100.0    0.3   -5.4    3.1
 6  -28.6   -1.2   21.6  -15.4    0.3  100.0    3.2    5.1
 7  -12.0    1.3   -3.6   -2.9   -5.4    3.2  100.0    0.4
 8  -14.2   -0.7   -4.0   -2.2    3.1    5.1    0.4  100.0
```

The second line in this table gives the variance for each variable. The first colum of the matrix is always for the target variable. In the example given one can see that the fourth variable[1] has the highest correlation with the signal and the second variable has the least correlation. Variable n° 6 has the highest anti-correlation to the target.
The preprocessor computes the significance of the variables and sorts them according to their importance (for details about how the significance is computed, please see section A.1.4 of [1]):

```
1 most relevant variable  6 corr  40.2332764 in sigma:  52.6241379I=  7
2 most relevant variable  4 corr  26.4777851 in sigma:  34.6322937I=  6
3 most relevant variable  3 corr  15.3113966 in sigma:  20.0269318I=  5
4 most relevant variable  8 corr  12.180047 in sigma:   15.9312029I=  4
5 most relevant variable  7 corr  11.1776428 in sigma:  14.6200829I=  3
6 most relevant variable  5 corr  8.89108562 in sigma:  11.6293221I=  2
7 most relevant variable  2 corr  0.537851393 in sigma: 0.703496397I= 1
```

The chart suggests that one variable could be excluded from the input set, having a significance of less than 3 sigma. The method NB_DEF_PRE(int) sets an automatic variable selection according to a minimal significance decided by the user (for the correct usage of this method see Appendix A of [1]).
Finally, the correlation matrix is diagonalized and the variables used for the training are computed as a combination of the input variables.
At this point the iteration over the sample starts. According to the user's settings, a fraction of the sample is used for the training and the rest is used for test purposes. This is achieved by using the method NB_DEF_RTRAIN(float ratio). By default ratio = 1.0, so no test sample is available.
The training results are stored in the expertise file (e.g. myneurobayes.nb). The results obtained by running on the test samples are written out to a file in the format specified by the interface chosen. This file contains also the

[1] The variable numbering scheme in the Teacher output is shifted by two units with respect to the numbering in C++, since in FORTRAN the first vector index is 1 and the first element of the variable vector is always the target variable.

histograms produced during the preprocessing and the training. The description of each histogram can be found in Appendix E of [1].

2.4 The analysis macro

The analysis macro takes as input the Teacher histogram file and gives an overview of the learning performance. The macro $NEUROBAYES/macros/analysis.C is to be used with Root[2]. The Teacher text output has first to be converted into the Root format by means of the macro asciiToRoot.C, which resides in the same folder as analysis.C.
The analysis macro can be called with the following arguments in the given order:

- Teacher histogram file:
  - pass a Root file (was created by asciiToRoot.C beforehand)
  - pass the Teacher text output (.txt) to the macro → asciiToRoot.C is automatically called to create a Root file

  default value is ‘‘TeacherHistos.root’’
- name of file finally containing the output plots:
  - *filename*.ps → a PostScript file is created
  - *filename*.pdf → a Portable Document Format file is created

  default value is ‘‘analysis.ps’’
- input variable sorting flag (only valid if correl_signi file is passed as well):
  - 0: sort by input array index
  - 1: sort by relevance
  - 2: sort by ‘‘only this’’

  default value is 0
- name of file containting the correlations and significances, which was created by correl_signi_cmap;
  if this file is passed to the script, the significances, preprocessing, and ranking of each input variable are shown as well;
  default is ‘‘correl_signi.txt’’.

The graphs shown in the output file are:

- the error, that is the difference between the network output and the target, for the learning sample and for the test sample (if any) for each training iteration (unless zero iterations training);

[2] It is recommended to use Root version 4.00/08 or newer.

- the network response for signal and for background, for each output node;
- the purity as a function of the network output, for each output node;
- the efficiency/purity graph for each output node;
- the signal efficiency/efficiency graph (for classification only);
- mean target versus target distribution (after preprocessing) with spline fit for each output node (if option 'DIAG');
- the target distribution (for density training only);
- the correlation matrix for the input set;
- for each input variable
  - distribution for the signal and for the background transformed into a flat distribution in the interval [0,1];
  - mean target in bins of the input variable (if performed, additionally spline fit or class prepro);
  - distribution after the transformation, that is the distribution of the variable as it enters into the neural network;
  - efficiency/purity graph if only this variable is used to separate signal and background; the efficiency/purity graph for the whole network is superimposed (for a classification training, the graph relative to one of the central output nodes is shown);
- network architecture (unless zero iterations training);

3 Using the NeuroBayes® Expert

3.1 Calling the Expert in a C++ program

The Expert can be called in a C++ program. In order to do that, the header of the Expert class, Expert.hh, has to be included into the client code. Then a pointer to the Expert has to be defined and initialized to an existing expertise file. The expertise itself can be called in the source file for a given set of input variables. The program which calls the Expert class has to be linked to the NeuroBayes® libraries.
The user has to make sure that he/she is using the expertise which fits to the input, because the Expert does not complain if, for example, the number of input variables is different than expected and in such a case the network output would be invalid. Because of this, the user has also the chance to save the expertise a C-array (see method NeuroBayesTeacher::SetCArrayFile()), which can be then included in the source code, and to convert an expertise from the text format to a C-array (method Expert::convertToArray(char *ExpertiseFileNa

The program which calls the Expert class has to be linked to the NeuroBayes® libraries.
For example, assume that the expertise in "myneurobayes.nb" performs a binary classification and it is stored in the file NBExpertise. The client code could look like this:

- **Header file**: include the header file and define a pointer to the Expert

```
#include "Expert.hh"
...
Expert* Net;
...
```

- **Source file**: initialize the Expert, define the input variables and feed them to the Expert

```
...
Net = new Expert("myneurobayes.nb");

// loop over the events
  ...
  double var[nvar];
  var[0] = var0;
  ...
  var[nvar] = varN;

  double output = Net->nb_expert(var,Expert::BINCLASS,0.0);
...
```

In order to compile this code against NeuroBayes® libraries, the compiler options have to be set as explained in section 2.2. The object file has to be linked to the NeuroBayesExpertCPP when working with Linux.

References

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