TMultiLayerPerceptron

UCL

Designing and using Multi-Layer Perceptrons with ROOT.

Christophe Delaere

FNRS Research Fellow

UCL – Belgium

Outlook

UCL

- Introduction
- Multi-layer perceptrons
- Learning methods
- Implementation
- Examples
  - mlpHiggs.C
  - fitting a function
- Timing
- Conclusions

Introduction

UCL

Neural Networks are more and more used in various fields for data analysis and classification, both for research and commercial institutions.

- Image analysis
- Financial movements predictions and analysis
- Sales forecast and product shipping optimisation
- In particles physics: mainly for classification tasks (signal over background discrimination)

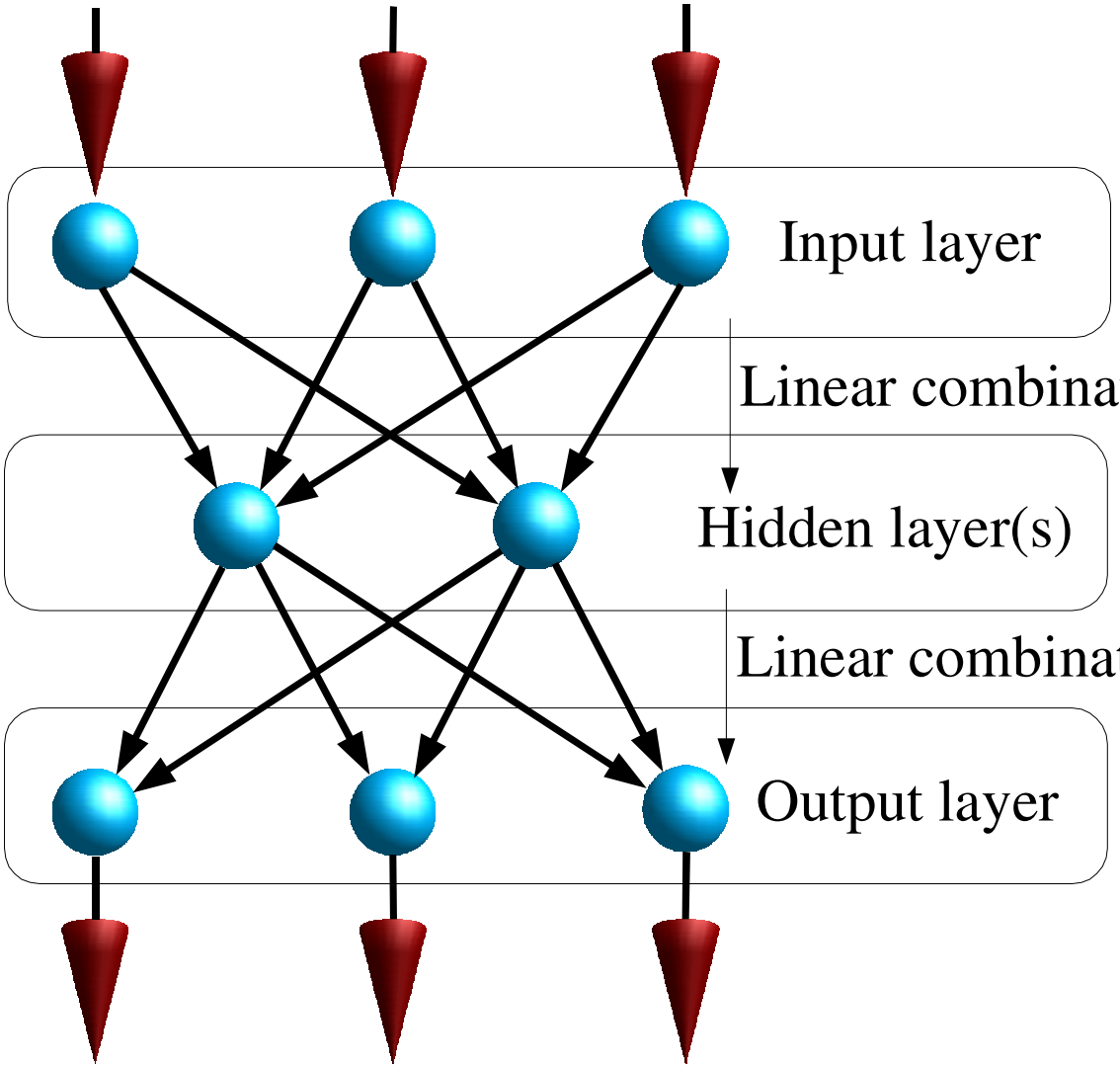
Several tools: for Matlab, or in various programming languages.
MLPfit: fast and powerful, already ported to paw.
(Jerome Schwindling, *http://schwind.home.cern.ch/schwind/MLPfit.html*)

Existing solutions do implement powerfull learning methods, are evolutive tools for research on neural networks, but are generally *not suited to the large samples* ROOT is used to manipulate.

Something new was needed...

A clear flexible Object Oriented implementation has been choosen, starting from *MLPfit*.

Multi-layer perceptrons (1)



Neuron

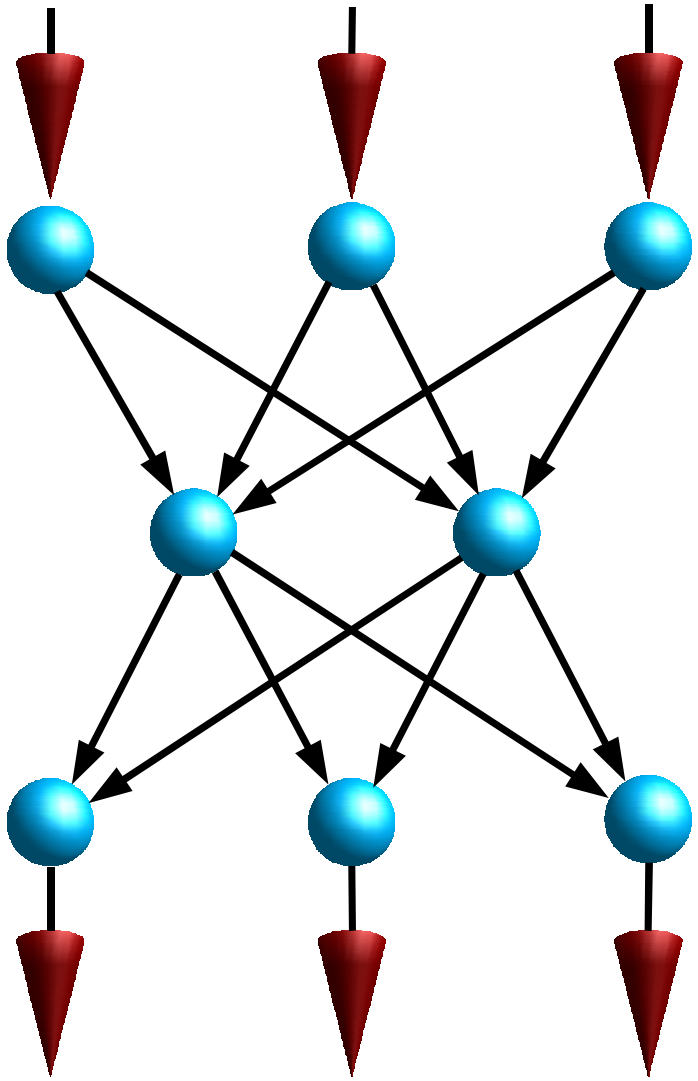
Synapse

Normalization

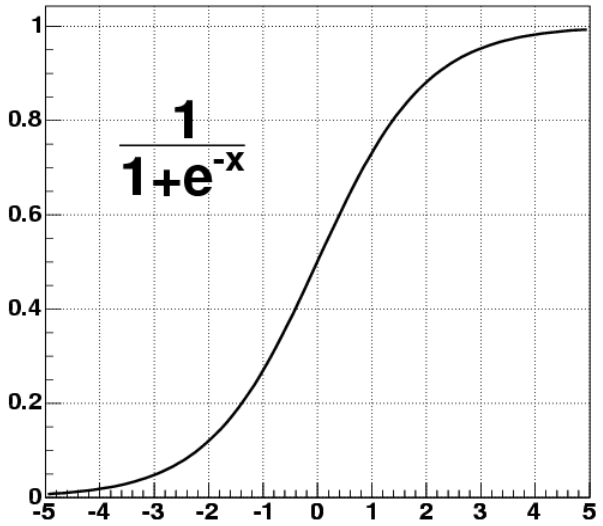
Evaluation of a function $f(x)$

Forward inputs ($f(x)=x$)
& computes the error

Multi-layer perceptrons (2)



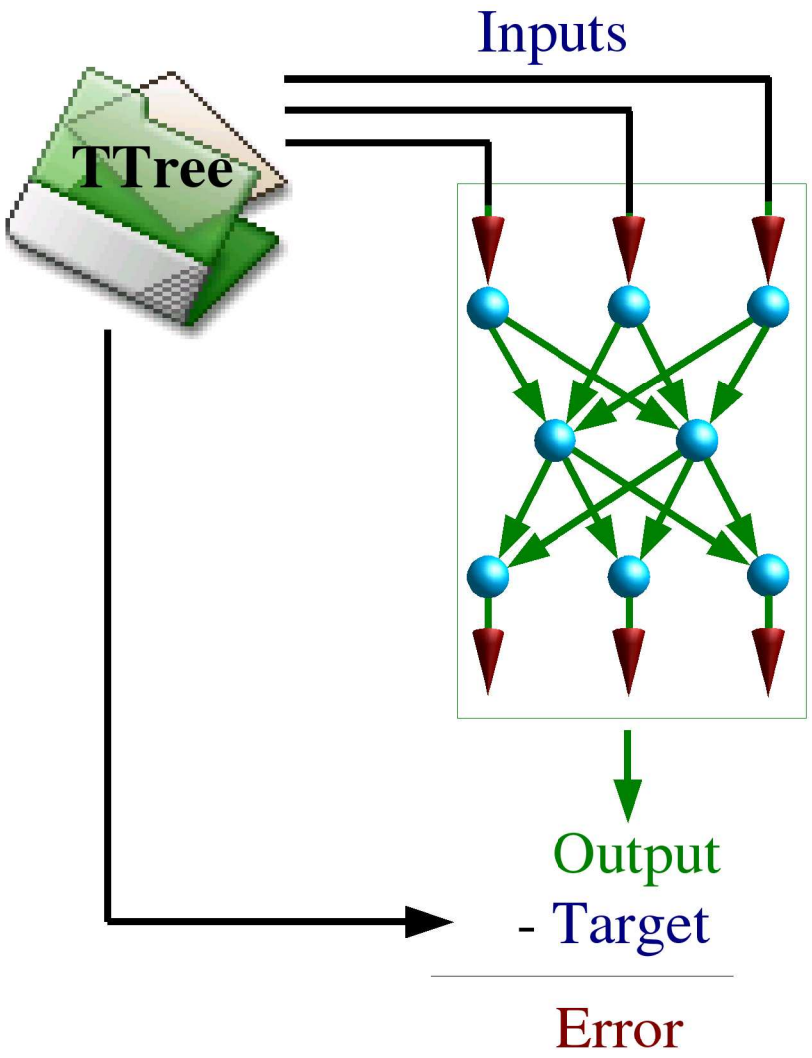
Hidden neurons are sigmoids.



A linear combination of sigmoids can approximate any continuous function.

Trained with output = 1 for the signal and 0 for the background, the approximated function of inputs X is the probability of signal, knowing X.

Learning (training) principle



The aim of all learning methods is to minimize the total error on a set of weighted examples. The total error is defined as

$$\Delta = \frac{1}{2}\sqrt{\sum_i \Delta_i^2}\ ,$$

Δ_i being the error on each individual output neuron.

The error is evaluated for each sample (each entry in the TTree). One can also compute the total error on the whole set of events.

So, in general, for a fit, target = function value, and for a discrimination, target is 1 for the signal, 0 for the background.

Learning methods

UCL

The most trivial learning method is the (Robbins-Monro) stochastic minimization:
The weights are updated after each example according to the formula:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

with

$$\Delta w_{ij}(t) = -\eta(\partial \Delta_p / \partial w_{ij} + \delta) + \epsilon \Delta w_{ij}(t-1)$$

- steps follow the gradient
- additionnal "flat-spot elimination factor" δ
- second-order term

5 other learning methods are implemented:

- Steepest descent with fixed step size,
- Steepest descent algorithm with line search,
- Conjugate gradients with the Polak-Ribiere updating formula,
- Conjugate gradients with the Fletcher-Reeves updating formula
- and the Broyden, Fletcher, Goldfarb, Shanno (BFGS) method.

Implementation

TNeuron class

This is a transfert function, an input or an output
and may be associated to a TTree branch or to a set of synapses.

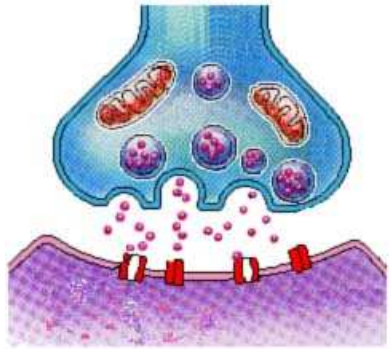
$$\frac{\partial \Delta_i}{\partial w_j} = \frac{\partial f}{\partial w_j} \left(\sum_{k \in out} \frac{\partial f}{\partial w_k} \right)$$

Other services: normalisation, output, error.

TSynapse class

This is a weighted bidirectionnal link between 2 neurons

$$\frac{\partial \Delta_i}{\partial w_j} = in \frac{\partial f_{out}}{\partial w_{out}}$$

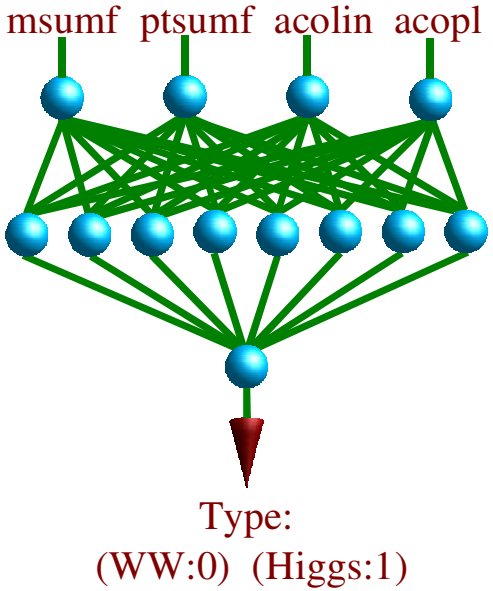


TMultiLayerPerceptron is a collection of neurons and synapses.

Services: configuration, training and USER INTERFACE

Example 1 : mlpHiggs.C

With Monte Carlo events simulated at LEP, a neural network is build to make the difference between WW events and events containing a Higgs boson.



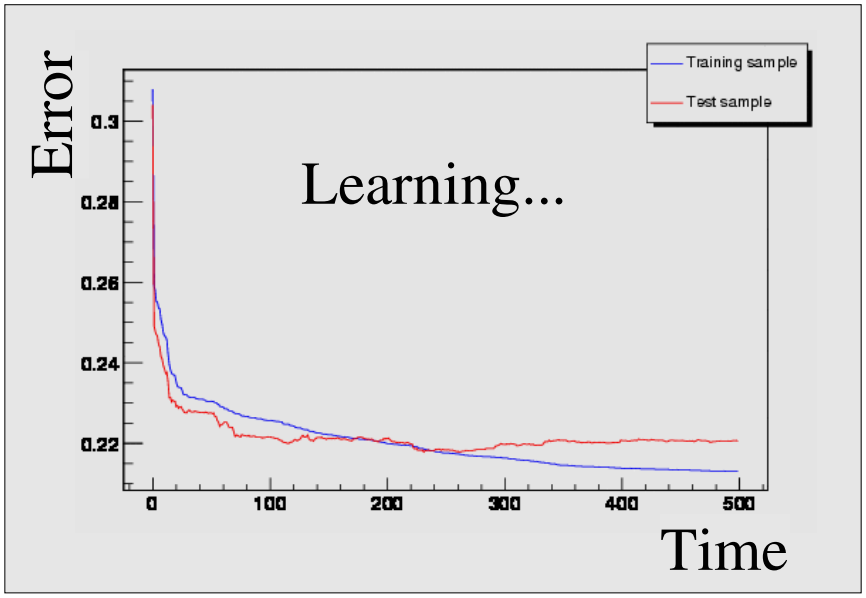
Starting with a TFile containing two TTrees, one for the signal, the other for the background, a simple script is used. Those 2 trees are merged into one, with an additionnal type branch.

```
TMultiLayerPerceptron *mlp =
 new  TMultiLayerPerceptron(
   "msumf,ptsumf,acolin,acopl:8:type", tree);
```

This network is then trained:

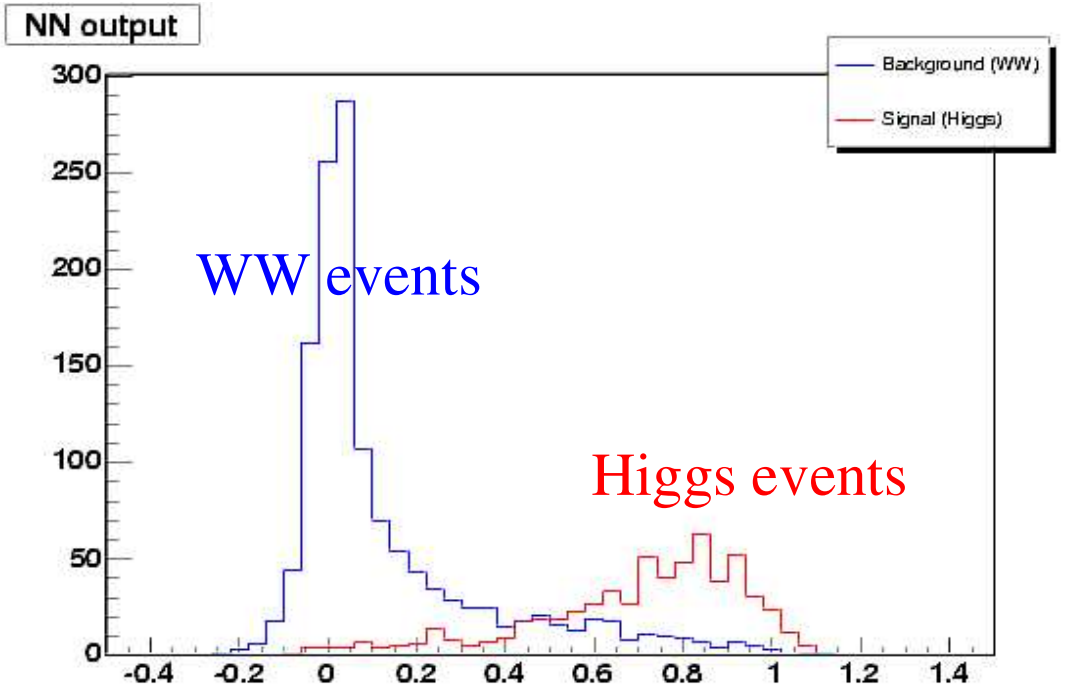
```
mlp->Train(500, "text,graph,update=10");
```

Example 1 : mlpHiggs.C



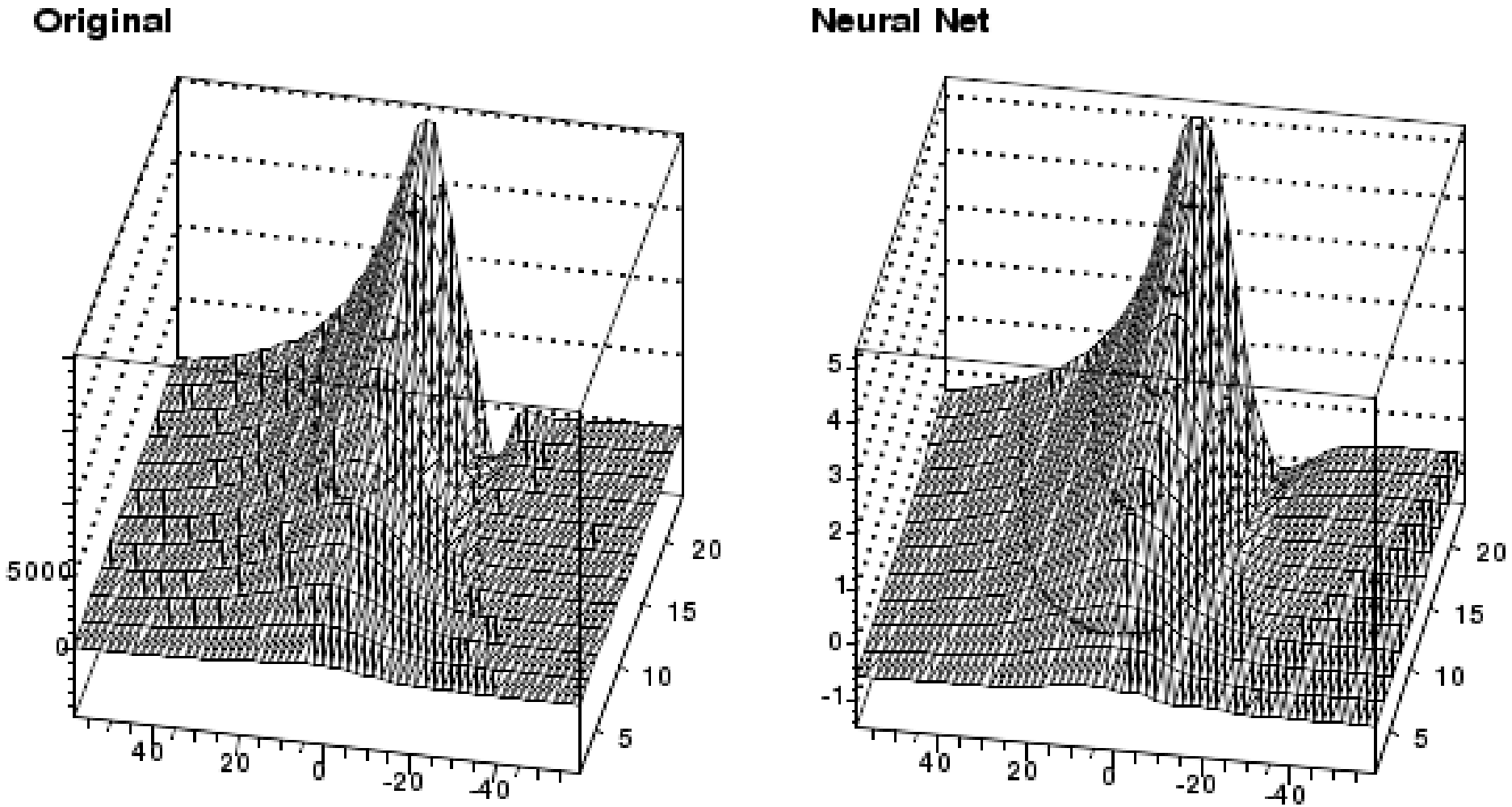
During the learning, one sees 2 curves: the sample has been divided into a training set and a test set.

The resulting NN distributions :



Example 2 : fitting a function

NN are also used to fit functions:

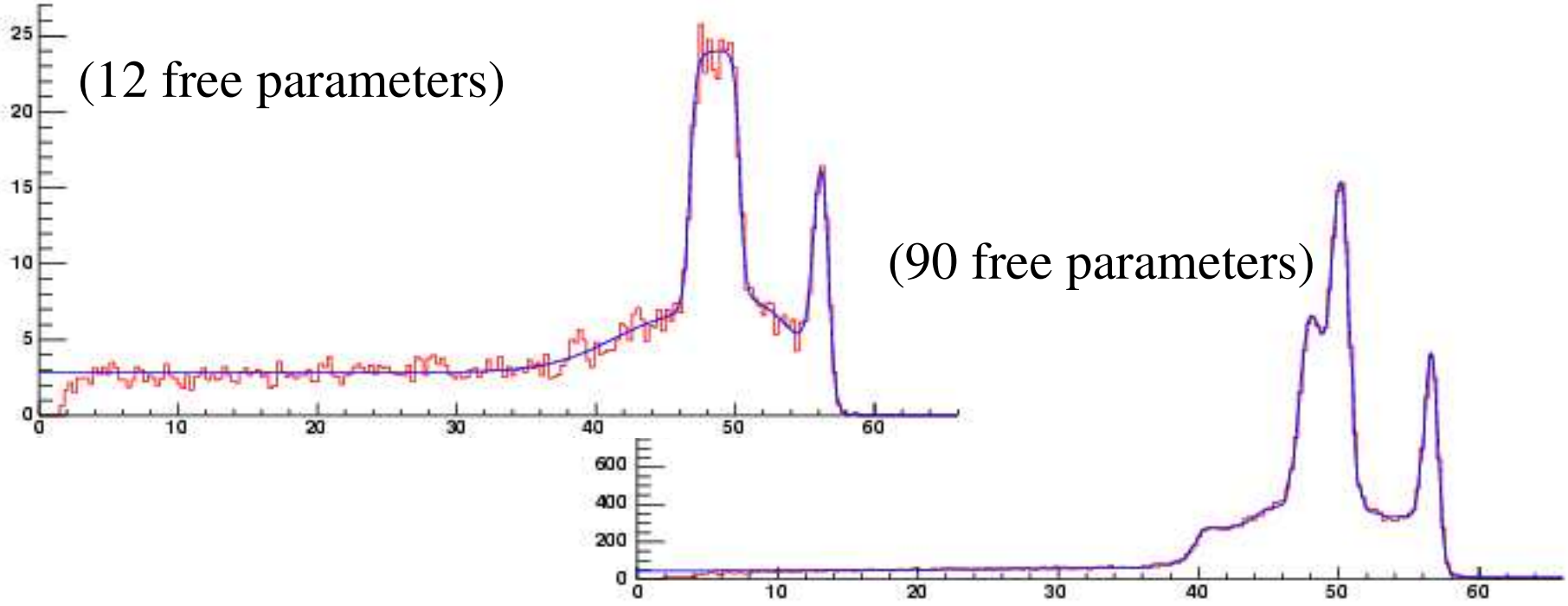


2-10-1 network. 90s learning time with the BFGS method.

Example 3 : fitting a 1D function

In nuclear physics, people handle complex spectrums difficult to fit. Such a fit is sometimes used to perform a subsequent background subtraction.

Christophe Dufauquez, from Louvain-la-Neuve, has used a NN as such:



Timing

UCL

Here are the results of the learning of:
the Higgs example above

- 979 TTree entries for learning
- 979 TTree entries for test
- 1000 epochs (iterations)

on a mobile AMD Athlon(tm) XP 1500+ (458.8 rootmarks)

	Standalone MLPfit	ROOT script	Factor
Start	0,02	0,34	
Stochastic	6,08	34,53	5
Fixed step size	5,88	33,61	5,71
Steepest descent	12,83	88,87	6,92
Ribiere-Polak	10,4	63,94	6,15
Fletcher-Reeves	10,95	63,46	5,79
BFGS	13,26	88,28	6,66

t (sec)

Only trivial optimization of the code has been achieved now.
This difference might be recovered.

Conclusions

UCL

- A multi-layer perceptron implementation is now released with ROOT since the version 3.10.1

- The flexible implementation should allow to extend the code to other networks.

- Some timing studies hows that MLPfit is faster by a factor $\leqslant 7$. Code optimization might allow to recover at least part of it.

- MLPs are fully working and easy to use. It has a growing user-base.

  More documentation, reference papers and examples can be found on the TMultiLayerPerceptron website:
  http://www.fynu.ucl.ac.be/users/c.delaere/MLP/