

Report about my work in CERN summer school

Peter Krachkov

9.08.13

In high-energy physics, with the search for ever smaller signals in ever larger data sets, it has become essential to extract a maximum of the available information from the data. Multivariate classification methods based on machine learning techniques have become a fundamental ingredient to most analyses. Also the multivariate classifiers themselves have significantly evolved in recent years. Statisticians have found new ways to tune and to combine classifiers to further gain in performance. Integrated into the analysis framework ROOT, TMVA is a toolkit which hosts a large variety of multivariate classification algorithms. Training, testing, performance evaluation and application of all available classifiers is carried out simultaneously via user-friendly interfaces. With version 4, TMVA has been extended to multivariate regression of a real-valued target vector. Regression is invoked through the same user interfaces as classification. TMVA 4 also features more flexible data handling allowing one to arbitrarily form combined MVA methods. A generalised boosting method is the first realisation benefiting from the new framework.

But some new regression methods do not work correct. Neuron network is one type of machine learning technique. This method have been realized in ROOT TMVA with two different method: back propogation method and Broyden-Fletcher-Goldfarb-Shannon (BFGS) method. But neuron network with back propogation method does not work correctly. My goal was found reason bad working neuron network with this method. This method work very good and widely apply for classification problem because it work faster then BFGS method.

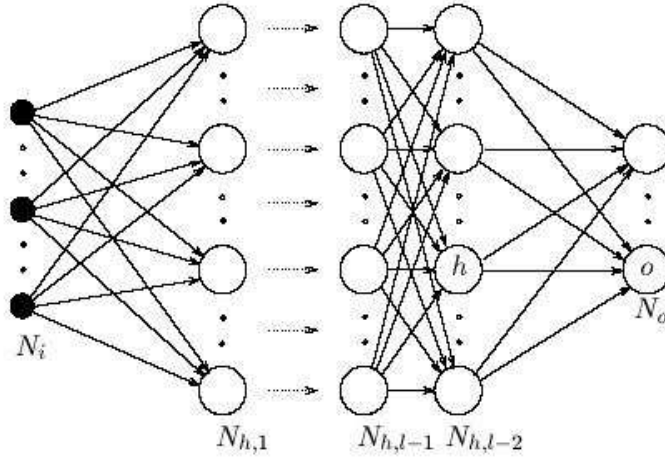
1 back propogation method

A feed-forward network has a layered structure. Each layer consists of units which receive their input from units from a layer directly below and

send their output to units in a layer directly above the unit. There are no connections within a layer. The N_i inputs are fed into the first layer of N_h hidden units. The input units are merely units no processing takes place in these units. The activation of a hidden unit is a function F_i of the weighted inputs plus a bias, as given in eq

$$y_k(t+1) = F_k(s) = F_k\left(\sum_j \omega_{jk} y_j(t) + \theta_j(t)\right)$$

The output of the hidden units is distributed over the next layer of N_{h+1} hidden units, until the last layer of hidden units, of which the outputs are fed into a layer of N_o output units .



Although backpropagation can be applied to networks with any number of layers, just as for networks with binary units it has been shown (Hornik, Stinchcombe, White, 1989; Funahashi, 1989; Cybenko, 1989; Hartman, Keeler, Kowalski, 1990) that only one layer of hidden units suffices to approximate any function with finitely many discontinuities to arbitrary precision, provided the activation functions of the hidden units are non-linear (the universal approximation theorem). In most applications a feed-forward network with a single layer of hidden units is used with a sigmoid activation function for the units.

The error measure E_p is defined as the total quadratic error for pattern p at the output units:

$$E_p = \frac{1}{2} \sum_o (d_o^p - y_o^p)^2$$

where d_o^p is the desired output for unit o when pattern p is clamped. We further set $E = \sum_p E_p$ as the summed squared error. We can write

$\frac{\partial E^p}{\partial \omega_{jk}} = \frac{\partial E^p}{\partial s_k^p} y_j^p$ When we define $\delta_k^p = -\frac{\partial E^p}{\partial s_k^p}$ The trick is to figure out what δ_k^p should be for each unit k in the network. The interesting result, which we now derive, is that there is a simple recursive computation of these δ_k^p which can be implemented by propagating error signals backward through the network.

To compute δ_k^p we apply the chain rule to write this partial derivative as the product of two factors, one factor reflecting the change in error as a function of the output of the unit and one reflecting the change in the output as a function of changes in the input. Thus, we have

$$\delta_k^p = -\frac{\partial E^p}{\partial s_k^p} = -\frac{\partial E^p}{\partial y_k^p} \frac{\partial y_k^p}{\partial s_k^p}$$

Let us compute the second factor. By equation $y_k^p = F(s_k^p)$ which is the same result as we obtained with the standard delta rule. Substituting this and $\frac{\partial y_k^p}{\partial s_k^p} = F'(s_k^p)$, we get $\delta_o^p = (d_o^p - y_o^p)F'(s_o^p)$ for any output unit o. Secondly, if k is not an output unit but a hidden unit $k = h$, we do not readily know the contribution of the unit to the output error of the network. However, the error measure can be written as a function of the net inputs from hidden to output layer and we use the chain rule to write

$$\frac{\partial E^p}{\partial s_h^p} = -\sum_{o=1}^{N_0} \delta_o^p \omega_{ho}$$

These equations give a recursive procedure for computing the δ_k^p for all units in the network, which are then used to compute the weight changes according to equation. This procedure constitutes the generalised delta rule for a feed-forward network of non-linear units.

2 My work

I tried to use TMVA: multivariate regression techniques for fit function. Back propagation method was applied, but it didn't work. Fig 1 displays fit linear function.

I changed different options (neuron type, number of hidden layer, number neuron in each layer and e.t.c) but it didn't help. After that I wrote own simple neuron network with back propagation method. My network has the opportunity to change type activation function, number of hidden layers and number of neuron in each layer. I think that in TMVA: back propagation method has some bug. For example fit by my neuron network and TMVA network with the same options is shown in Fig 2, 3.

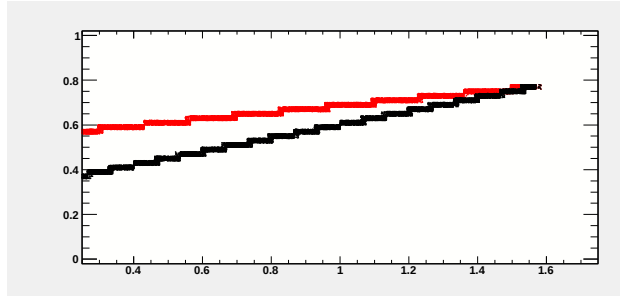


Рис. 1: Fit linear function, red line-true function, black line fitting function

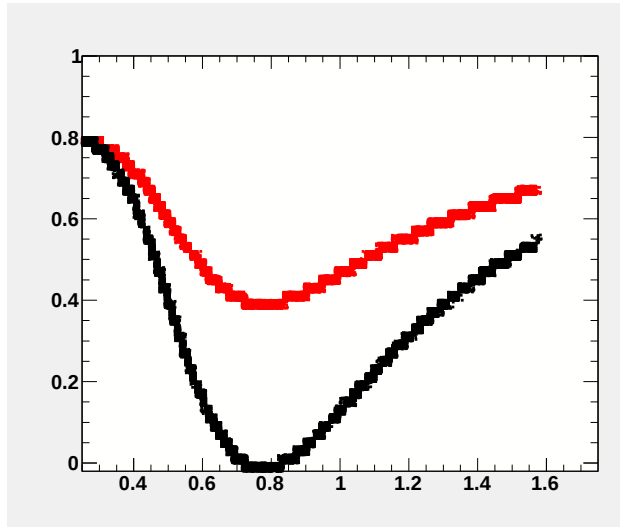


Рис. 2: Fit some function with help TMVA neural network, red line-true function, black line fitting function

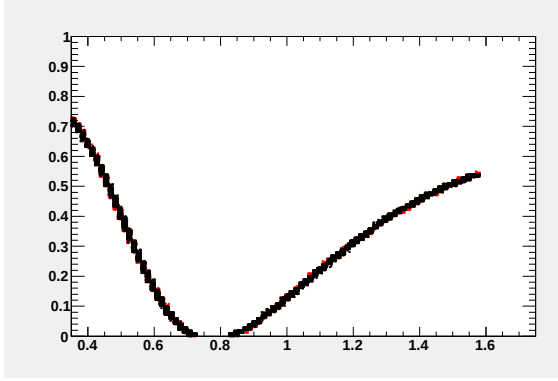


Рис. 3: Fit some function with help my neural network, red line-true function, black line fitting function

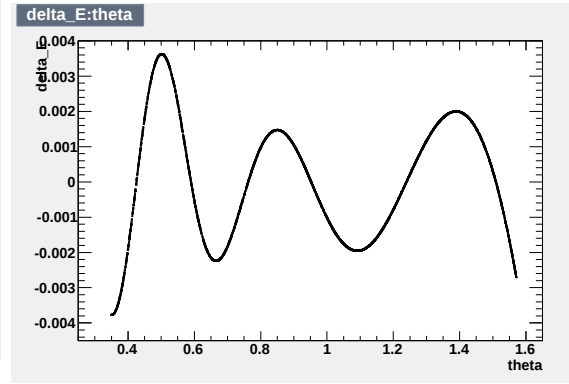


Рис. 4: Difference between fitting value and true value

This neuron network was used one hidden layer and ten neurons with sigmoid activation function. TMVA has been realized so:

```
factory->BookMethod(TMVA::Types::kMLP,"MLPold"!H:!V:VarTransform=Norm:
NeuronType=sigmoid:NCycles=2000:HiddenLayers=10:TrainingMethod=BP:Sampling=0.3:
SamplingEpoch=0.8:ConvergenceImprove=1e-6:ConvergenceTests=15");
```

I used 10^6 monte carlo events given this formula 2 with sigma equal zero.

$$y = (0.8 - 4.5 \text{Landau}(5x - 4, 0.1, 1., 0))$$

Landau() means Landau distribution probability density function.

After that I tested the other artificial intelligence method is used in TMVA regression and try to understand why some method did not work in different case.