Exact Calculus of First and Second Derivatives of Arbitrary Feed-Forward Neural Networks

Serge Iovleff*

December 7, 2001

Abstract

In this article we show how to calculate general feed-forward neural networks first and second derivatives and all order derivatives of a multilayer perceptron. Our approach is simple and founded on an idea first discovered by Vapnick [9]. This approach allows to integer in an evident way a penality or added constraints on the parameters. We show that the Hessian is bloc diagonal and that the computation of the first and second derivatives can be done with only one back-propagation. Morever, in the perceptron case it appears clearly there is a significant reduction of the number of data to handle in second order estimation methods.

1 Introduction

Standard learning algorithm of the multilayer perceptron uses a back-propagation algorithm to evaluate the first derivatives of the objective function. The calculus of the first derivative allows to use a gradient algorithm (first order method) whose defaults are well-known : low convergence and tricks of the local minima or by the almost flat spot of the objective function.

To accelerate convergency, second order methods are frequently used : essentially the conjugate gradient method or the quasi-Newton methods like BFGS and its variants approaching the Hessian during the iterations [5, 8]. Methods based on the exact Hessian such as Newton-Raphson or truth region methods [4] do not seems to have been implemented. This is due to the fact it is said that for large networks, the effective computation of the full Hessian is time-consuming [2]. Our aim is thus, in a first part, to obtain an efficient method of calculation of the first and second derivatives for an arbitrary feed-forward neural network. Secondly, to develop theses formulas for the main feed-forward neural networks used : the multilayer perceptron (for which we obtain derivatives at all order) and the radial basis function (RBF) networks. We obtain that the Hessian is bloc diagonal for a general feed-forward neural network and can be effectively

*Serge iovleff is with the IUP of Statistics and Computation, University of South Britain, 56000 Vannes, France. E-mail : Serge.Iovleff@univ-ubs.fr

computed with only one back-propagation. Moreover, in the case of the perceptron the number of non nul elements is greatly reduced so that the Hessian could be computed in an extremely reasonnable time. As far as we know, this fact has not been noticed until now. Remind that exact calculus of the second derivatives of a multilayer perceptron has been obtained by Bishop [1] and for an arbitrary feed-forward neural networks by W.L Buntine and A.S. Weigend [2] (see also [6]). In both cases the authors use the "chain-rule" to derivate and obtain an algorithm requiering $2h + 2$ back-propagations to compute the Hessian (where h is the number of hidden units). At the moment, to our mind, no explicit formula for the higher order derivatives exists in the perceptron case.

Our approach is different in two ways :

1. For the mathematical modelisation : we use the layer disposition of the feed-forward neural networks and do not work with a directed aciclyc graph.

2. For the calculus of the first and second derivatives we do not use the "chain-rule". We follow an idea of Vapnick [9] and work with an under constraint optimization problem.

This approach clarify a bit the notations and sheds new light on the back-propagation algorithm : it is just the computation of the Lagrange multiplier.

This document is organized in the following way. In a first time we introduce the notations we will use, and we describe the working way of a general feed-forward neural networks. Then we describe the optimization problem that we have to solve. We calculate the first and second derivatives of an arbitrary feed-forward neural network and obtain the back-propagation algorithm to compute them effectively. Results are then detailed for the two main feed-forward networks actually used : the multilayer perceptron and the RBF networks.

2 Preliminaries

Let's define first the vocabulary. The neural networks are described in a functional point of view by three aspects:

1. An *architecture* which describes the relationship between the neurons with an oriented graph (The nodes represent the neurons and the arrows the communication direction between them).

2. A *propagation* method which describes how the neurons communicate between themselves through the arrows.

3. An *activation* which represents the treatment realized in an individual way by the neurons.

The feed-forward Neural networks are characterized by a layer disposition of the neurons which does not permit a layer to communicate with the neurons of

the layers behind them. However, a neuron of a layer is allowed to communicate with any neuron of any layer ahead him.

A neural networks is set up by a group of unknown parameters that we have to estimate (learning step) using a sample (the examples). The parameters are estimated by minimizing a cost function (or objective function).

2.1 Notations

For the dimensions, we assume that we have $m+1$ layers and that for each layer we have $s_0, s_1, \ldots, s_m$ units and $t_1, t_2, \ldots, t_m$ parameters.

For the index, we always use the subscript i to number the examples, the subscript j to number the units in a layer, the uperscript k to number the layers and the subscript l to number the parameters.

Generally we try to use capital letters $X, H, E, \ldots$ for vectorial quantities and the letters $x, h, e, \ldots$ for scalar quantities.

In order to get compact notations, we use the following conventions :

- For every $1 \leq k \leq m$, the sets $\mathbb{W}^k$ represent $\mathbb{R}^{t_1} \times \ldots \times \mathbb{R}^{t_k}$, and the special set $\mathbb{W} = \mathbb{W}^m$ is called the parameter space.
- W^k denote an element of $\mathbb{R}^{t_k}$ and $\mathbf{W}^k$ an element of $\mathbb{W}^k$. An element $\mathbf{W}^m = \mathbf{W} \in \mathbb{W}$ is called a parametrization of the Neural Network.

In a similar way :

- for every $0 \leq k \leq m$, the sets $\mathbb{X}^k$ represent $\mathbb{R}^{s_0} \times \ldots \times \mathbb{R}^{s_k}$ and the special set $\mathbb{X} = \mathbb{X}^m$ is called the configuration space.
- X^k denote an element of $\mathbb{R}^{s_k}$, $\mathbf{X}^k$ denote an element of $\mathbb{X}^k$. An element $\mathbf{X}^m = \mathbf{X} \in \mathbb{X}^m$ is called a configuration of the Neural Network.

Concerning the activation and propagation functions, we use the following conventions :

- We note h_j^k the *activation* function of the j-st neuron of the layer k ($k > 0$) and we assume that h_j^k is class $\mathcal{C}^p$ where p is the derivation order that we want to calculate. The activation functions do not depend of the parameters.

  We note H^k the vector of the s_k functions $(h_j^k)_{j=1}^{s_k}$.

- In a similar way we note $h_j^{k(p)}$ the p-st derivative of the activation function and $H^{k(p)}$ the vector of the p-st derivatives.

- We note P^k the *propagation* function of the neural networks of the layers 0, 1,...,$k-1$ to the layer k. This function depend on a group of parameters W^k

$$\begin{array}{rccc} P^k & : & \mathbb{X}^{k-1} \times \mathbb{R}^{t_k} & \longrightarrow & \mathbb{R}^{s_k} \\ & & (\mathbf{X}^{k-1}, W^k) & \longrightarrow & P^k(\mathbf{X}^{k-1}; W^k) \end{array}$$

Thus $P^k = (p_j^k)_{j=1}^{s_k}$ is a vector of s_k functions. We assume that the p_j^k are class $\mathcal{C}^p$ reports to W^k and are class $\mathcal{C}^1$ reports to $\mathbf{X}^{k-1}$. We note P^k the vector of the s_k functions $(p_j^k)_{j=1}^{s_k}$.

- Finally, we denote by $D^1_{X^{k_1}} P^k$ the matrix of the first derivatives $\left(\partial p_j^k / \partial x_j^{k_1}\right)$, by $D^1_{W^k} p_j^k$ the vector of the first derivatives $\left(\partial p_j^k / \partial w_l^k\right)$ and by $D^2_{W^k} p_j^k$ the matrix of the second derivatives $\left(\partial^2 p_j^k / \partial w_{l_1}^k \partial w_{l_2}^k\right)$

2.2 Propagation equations

A parametrization $\mathbf{W}$ fixed and a vector $\mathbf{X}^0$ taken as input, we propagate it using the recurrence formula

$$X^k = H^k(P^k(\mathbf{X}^{k-1}; W^k)) \tag{1}$$

This equation is called the propagation equation. The vector X^m is called the *calculated* output of the networks.

Later, we shall also use the following notation

$$X^{k(p)} = H^{k(p)}(P^k(\mathbf{X}^{k-1}; W^k)) \tag{2}$$

Examples of usual neural networks are postponed in section 4.

3 Estimation of the parameters (Learning step)

We assume to have in hand a training set of the form $(\mathbf{X}_i^0, Y_i)$, for $i = 1, \ldots, n$. For a given parametrization $\mathbf{W}$, we can compute the output of the neural net $(X_1^m, \ldots, X_n^m)$ using the propagation equation (1). We give ourselves a loss function $g(X_i^m, Y_i)$ and we assume that g is class $\mathcal{C}^1$ reports to X_i^m (generally the loss function is taken as quadratic, i.e. $g(X_i^m, Y_i) = 1/2 \left\| Y_i - X_i^m \right\|^2$ but many other choices are possible). We give also a penalisation function pen(.) on the parameters (we can take for example $\text{pen}(\mathbf{W}) = \lambda \sum_{k=1}^m \left\| W^k \right\|$ where λ is an *a priori* fixed parameter) and we assume pen() is class $\mathcal{C}^p$ reports to $\mathbf{W}$.

The estimation of the parameters is done by minimizing the objective function

$$\sum_{i=1}^{n} g(X_i^m, Y_i) + \text{pen}(\mathbf{W}). \tag{3}$$

Note that g depends on the parameters $\mathbf{W}$ through X_i^m and that in this form, the minimization problem (3) is not correctly specified. It can be completely specified in two ways. The usual way consists to consider X_i^k as a function of $\mathbf{W}^k$, and in solving the following optimization problem on $\mathbb{W}$

$$\begin{array}{l} \min \mathcal{E}(\mathbf{W}) = \sum_{i=1}^n g(X_i^m(\mathbf{W}), \mathbf{Y}_i) + \text{pen}(\mathbf{W}) \text{ where} \\ \left\{ \begin{array}{lcl} X_i^m(\mathbf{W}) & = & H^m(P^m(X_i^0, X_i^1(\mathbf{W}^1) \ldots, X_i^{m-1}(\mathbf{W}^{m-1}); W^m)) \\ & \vdots & \\ X_i^2(\mathbf{W}^2) & = & H^2(P^2(X_i^0, X_i^1(\mathbf{W}^1); W^2)) \\ X_i^1(\mathbf{W}^1) & = & H^1(P^1(X_i^0; W^1)) \end{array} \right. \end{array} \tag{4}$$

and then using the chain-rule for derivatives to compute the derivatives. Observe that when the propagations functions P^k are not very simple, the calculus of the first and second derivative can be very tiedous.

The second way consists to integrate directly the propagation equation (1) as a constrain in the optimization problem and then to solve on $\mathbb{W} \times \mathbb{X}^n$

$$\begin{cases} \min \sum_{i=1}^{n} g(X_i^m, Y_i) + \text{pen}(\mathbf{W}) \\ \text{u.c. } X_i^k = H^k(P^k(\mathbf{X}_i^{k-1}; W^k)) \quad 1 \leq i \leq n \quad 1 \leq k \leq m. \end{cases} \tag{5}$$

Thus, we want to estimate simulteanously the parametrization and the n configurations of the feed-forward neural network.

We transform the constrained minimization problem (5) in an unconstrained minimization problem by introducing the Lagrange multiplier. We define the line vectors $E_i^k = (e_{i1}^k, \ldots, e_{is_k}^k)$ for $i = 1, \ldots, n$ and $k = 1, \ldots, m$, and we now want to solve the following problem on $\mathbb{W} \times \mathbb{X}^n \times \mathbb{X}^n$

$$\min \mathcal{L} = \sum_{i=1}^{n} g(X_i^m, Y_i) + \text{pen}(\mathbf{W}) + \sum_{i=1}^{n} \sum_{k=1}^{m} E_i^k (X_i^k - H^k(P^k(\mathbf{X}_i^{k-1}; W^k))). \tag{6}$$

This approach has been introduced by Vapnick [9]. The problems (4) and (5) look very similar. In order to show that both approach are equivalent, we point out that if f_1 and f_2 are two real functions of p and q variables with $p < q$, and morever for every $(x_1, \ldots, x_p)$, the derivatives $\partial f_2 / \partial x_k$, $k \geq p+1$, can be anulated at a unique point $(x_{p+1}(x_1, \ldots, x_p), \ldots, x_q(x_1, \ldots, x_p))$ then, if

$$f_1(x_1, \ldots, x_p) = f_2(x_1, \ldots, x_p, x_{p+1}(x_1, \ldots, x_p), \ldots, x_q(x_1, \ldots, x_p))$$

every extrema of f_1 can be expressed as an extrema of f_2 and vice-versa.

Derivating $\mathcal{L}$ in respect to E_i^k and annulating these derivatives we get the propagation equation (1). Inserting it in the problem (6), the variables E_i^k disapear, and we get the problem (4). This show the equivalence of both approach.

3.1 Derivatives calculus

To solve the problem (6) and apply the standard optimization algorithms, we have to calculate the first and second derivatives of $\mathcal{L}$ in report to the parameters $\mathbf{W}$ and the first derivative in report to the variables $\mathbf{E}$ and $\mathbf{X}$. Also we will see that it is possible to get an explicit solution to annulate the gradient with respect to these variables. In order to simplify our notations, we drop the index i from the equations.

The derivatives in respect to the Lagrange multipliers e_j^k are easely found

$$\frac{\partial \mathcal{L}}{\partial e_j^k} = x_j^k - h_j^k(p_j^k(\mathbf{X}^{k-1}; W^k)).$$

Annulating theses equations we get the propagation equation (1).

Derivating in respect to the variables x_j^k, we have to distinguish while $k = m$ and while $k \neq m$. In the first case we have

$$\frac{\partial \mathcal{L}}{\partial x_j^m} = \frac{\partial g}{\partial x_j^m} + e_j^m$$

and when $1 \leq k < m$, we have

$$\frac{\partial \mathcal{L}}{\partial x_j^k} = e_j^k - \sum_{k_1=k+1}^{m} \sum_{j_1=1}^{s_{k_1}} e_{j_1}^{k_1} x_j^{k(1)} \frac{\partial p_{j_1}^{k_1}}{\partial x_j^k}(\mathbf{X}^{k-1}; W^k)$$

where $x_j^{k(1)}$ has been defined in equation (2). Annulating these equations, we get the backpropagation equations

$$e_j^m = -\frac{\partial g}{\partial x_j^m} \tag{7}$$

$$e_j^k = \sum_{k_1=k+1}^{m} \sum_{j_1=1}^{s_{k_1}} e_j^{k_1} x_j^{k_1(1)} \frac{\partial p_{j_1}^{k_1}}{\partial x_j^k}(\mathbf{X}^{k-1}; W^k) \tag{8}$$

Finally, we calculate the derivatives in respect to the parameters w_l^k, for the first derivative we have

$$\frac{\partial \mathcal{L}}{\partial w_l^k} = -\sum_{j=1}^{s_k} e_j^k x_j^{k(1)} \frac{\partial p_j^k}{\partial w_l^k}(\mathbf{X}^{k-1}; W^k) + \frac{\partial \text{pen}}{\partial w_l^k} \qquad l = 1, \ldots, t_k \tag{9}$$

For the second derivative we have

$$\begin{aligned}
\frac{\partial^2 \mathcal{L}}{\partial w_{l_1}^{k_1} \partial w_{l_2}^{k_2}} &= \frac{\partial^2 \text{pen}}{\partial w_{l_1}^{k_1} \partial w_{l_2}^{k_2}} \qquad \text{if } k_1 \neq k2, \\
\frac{\partial^2 \mathcal{L}}{\partial w_{l_1}^{k} \partial w_{l_2}^{k}} &= \frac{\partial^2 \text{pen}}{\partial w_{l_1}^{k} \partial w_{l_2}^{k}} - \sum_{j=1}^{s_k} e_j^k x_j^{k(1)} \frac{\partial^2 p_j^k}{\partial w_{l_1}^k \partial w_{l_2}^k}(\mathbf{X}^{k-1}; W^k) \\
&\quad - \sum_{j=1}^{s_k} e_j^k x_j^{k(2)} \frac{\partial p_j^k}{\partial w_{l_1}^k}(\mathbf{X}^{k-1}; W^k) \frac{\partial p_j^k}{\partial w_{l_2}^k}(\mathbf{X}^{k-1}; W^k) \qquad (10) \\
&\quad \text{otherwise.}
\end{aligned}$$

3.2 Implementation

Let us reintroduce the index i and assume that $\frac{\partial^2 \text{pen}}{\partial w_{l_1}^{k_1} \partial w_{l_2}^{k_2}} = 0$ when $k^1 \neq k_2$. In this case, by equation (10), the Hessian is bloc diagonal and the number of non null second derivatives are at most $t_1^2 + t_2^2 + \ldots + t_m^2$.

The main steps to compute the first and second derivatives consist firstly in propagating all the exemples X_i^0 and for $k = 1$ to m to compute the quantities

$$\begin{aligned}
X_i^k &= H^k(P^k(\mathbf{X}_i^{k-1}; W^k)) \\
X_i^{k(1)} &= H^{k(1)}(P^k(\mathbf{X}_i^{k-1}; W^k)) \\
X_i^{k(2)} &= H^{k(2)}(P^k(\mathbf{X}_i^{k-1}; W^k)).
\end{aligned}$$

The second step consists in back-propagating the errors and to compute the Lagrange multiplier

$$E_i^m = D^1_{X_i^m} g(X_i^m, Y_i)$$

and for $k = m-1$ to 1

$$E_i^k = \sum_{k_1=k+1}^{m} E_i^{k_1} \mathrm{Diag}(X_i^{k(1)}) D^1_{X^k} P^{k_1}(\mathbf{X}_i^{k_1-1}; W^{k_1}).$$

The final step consists in computing the vectors of the first derivatives

$$D^1_{W^k}\mathcal{L} = -\sum_{i=1}^{n}\sum_{j=1}^{s_k} e_{ij}^k x_{ij}^{k(1)} D^1_{W^k} p_j^k(\mathbf{X}_i^{k-1}; W^k) + D^1_{W^k}\mathrm{pen}(\mathbf{W})$$

and the matrices of the second derivatives

$$D^2_{W^k}\mathcal{L} = -\sum_{i=1}^{n}\sum_{j=1}^{s_k} e_{ij}^k x_{ij}^{k(2)} D^2_{W^k} p_j^k(\mathbf{X}_i^{k-1}; W^k) + D^2_{W^k}\mathrm{pen}(\mathbf{W})$$

4 Examples

4.1 A completely connected m-layers perceptron

The usual perceptron perform a linear propagation from the layer $k-1$ to the layer k. In our framework this can be generalized in the following way : the propagation functions P^k are linear functions performing the transformations

$$P^k(\mathbf{X}^{k-1}; W^k) = W^{0,k}X^0 + \ldots + W^{k-1,k}X^{k-1}$$

The parameters of the layer k are the weight matrix $W^{k_1,k}$ with $0 \leq k_1 < k \leq m$. It is possible to introduce a bias by agregating an element always equal to 1 to the vectors X_i^0.

The first and the second derivatives of this propagation function are easily computed. For every $0 \leq k_1, k_2 < k \leq m$, every $1 \leq j \leq s_{k_1}$, and every $1 \leq j_1, j_2 \leq s_k$, we have

$$\frac{\partial p_j^k}{\partial w_{jj_1}^{k_1 k}} = x_{j_1}^{k_1} \qquad \text{and} \qquad \frac{\partial^2 p_j^k}{\partial w_{jj_1}^{k_1 k}\partial w_{jj_2}^{k_2 k}} = 0 \tag{11}$$

and for every $0 \leq k_1 < k \leq m$, we have

$$D^1_{X_i^{k_1}} P^k(\mathbf{X}_i^{k-1}; W^k) = W^{k_1,k}.$$

The back-propagation equation is thus

$$E_i^k = \sum_{k_1=k+1}^{m} E_i^{k_1} \mathrm{Diag}(X_i^{k_1(1)}) W^{k,k_1}.$$

For the first derivatives, we compute the gradient vector corresponding to the j-st line of the matrix W^{k_1,k_2}, $(k_1 < k_2)$

$$\frac{\partial \mathcal{L}}{\partial W_j^{k_1,k_2}} = -\sum_{i=1}^{n} e_{ij}^{k_2} x_{ij}^{k_2(1)} X_i^{k_1}.$$

For the second derivatives, by (11) we just have to compute the Hessian corresponding to each lines of W^{k_1,k_2} as all other derivatives are null. For the j-st line, we have

$$\frac{\partial^2 \mathcal{L}}{\partial \left(W_j^{k_1,k_2}\right)^2} = -\sum_{i=1}^{n} e_{ij}^{k_2} x_{ij}^{k_2(2)} X_i^{k_1} X_i^{k_1\prime}.$$

The number of non-null second derivatives of a perceptron is thus $\sum_{k=1}^{m} s_k \sum_{k_1=0}^{k-1} s_{k_1}^2$. In the case of the usual perceptron (only connexion between contiguous layer are allowed), this number is reduced to $\sum_{k=1}^{m} s_k s_{k-1}^2$. This is much less than the usual $\sum_{k=1}^{m} (s_k s_{k-1})^2$ storage needed by second order method.

4.1.1 Higher order derivative calculus

The calculus of the higher order derivatives is easy, we give as an example the p-st order derivative, which is

$$\frac{\partial^p \mathcal{L}}{\partial w_{j_1 j_2}^{k_1,k_2} \ldots \partial w_{j_1 j_p}^{k_1,k_2}} = -e_{ij_1}^{k_2} x_{ij_1}^{k_2(p)} x_{ij_2}^{k_1} \ldots x_{ij_p}^{k_1} \tag{12}$$

The derivative is reduced to zero as soon as either the line index of two weights or the layer number index are different.

4.2 The Radial Basis Function (RBF) Neural Networks

For the first layer, the propagation functions p_j^1 are some quadratic functions

$$p_j^1(X^0; \mu_j, M_j) = (X^0 - \mu_j)' M_j (X^0 - \mu_j)$$

where the M_j are symetric positive definite matrices. The networks parameters for the first layer are the centers and the ponderation matrices

$$W^1 = (\mu_j, M_j)_{j=1}^{s_1}.$$

That makes an important number of parameters, that can be reduced by assuming the hypothesis either $M_j = \mathrm{Diag}(1/\sigma_{1j}^2, \ldots, 1/\sigma_{s_0 j}^2)$ or $M_j = 1/\sigma_j^2 Id$ (isotropic case). The activation functions for the first layer consist of radial functions, and in general are taken as the exponential function : $h_j^1(x) = e^{-\frac{1}{2}x}$. For the following layers ($k = 2, \ldots, m$), the propagation functions are some linear functions as for the perceptron.

As in the perceptron case, it is useful to add to the vectors X_i^0 a component always equals to 1. With this convention, the propagation function p_j^1 can be written as

$$p_j^1(X^0; W^1) = X^{0\prime} W_j X^0$$

where each W_j is a square matrix of size $(s_0+1)^2$ with a generic element $w_{j_1 j_2 j}$.

In regards to the RBF neural networks, only the first layer take part with a quadratic propagation, and the back-propagation equation is the same as for a perceptron. Only the calculus of the parameters derivative taking part in the propagation from layer 0 to layer 1 are different. The derivatives with respect to the parameters $w_{j_1 j_2 j}$ are

$$\frac{\partial \mathcal{L}}{\partial w_{j_1 j_2 j}} = -e_j^1 x_j^{1(1)} x_{j_1}^0 x_{j_2}^0$$

whereas the second derivatives with respect to the parameters $w_{j_1 j_2 j}$ and $w_{j_3 j_4 j}$ are

$$\frac{\partial^2 \mathcal{L}}{\partial w_{j_1 j_2 j} \partial w_{j_3 j_4 j}} = -e_j^1 x_j^{1(2)} x_{j_1}^0 x_{j_2}^0 x_{j_3}^0 x_{j_4}^0 .$$

5 Conclusion

We obtain a back-propagation algorithm to evaluate in an exact manner the first and second derivatives of a completely general feed-forward neural network : arbitrary propagation functions, direct connections between non contiguous layer and general objective function that can include a regularization term on the parameters.

We show that in the perceptron case the Hessian is bloc diagonal. Therefore, a significant increase in speed and efficiency in the estimation parameters methods currently used, should be realized by implementing a method using the effective calculus of the second derivative to estimate the parameters of a feed-forward neural network. Morever, from a statistical point of view, the Hessian is an estimate of the information matrix of the model. It can thus be used for testing hypothesis of nullity of the weights in a backward pruning method as implemented by [3] or by [7]. Those points will be the aim of our further investigations.

References

[1] C. Bishop, "Exact calculation of the hessian matrix for the multilayer Perceptron", *Neural Computation*, vol. 4, pp. 494-501, 1992.

[2] W.L. Buntine, A.S. Weigend, "Computing 2nd derivatives in feed-forward networks. A review.", *IEEE transactions on Neural Networks*, 5 (3), pp. 480-488, 1995.

[3] Y. Le Cun, J.S. Denker, and S.A. Solla, "Optimal brain damage", in *Advances in Neural Information Processing systems 2 (NIPS*89)*, D.S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, pp. 598-605, 1990.

[4] J. Nocedal, S.J. Wright, *Numerical Optimization*, Springer Verlag, 1999.

[5] W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, *Numerical Recipes: The Art of Scientific Programming*, Cambridge University Press, 1986.

[6] F. Rossi, "Second Differentials in Arbitrary Feed-Forward Neural Networks", *Proceeding of the IEEE International Conference on Neural Networks*, Washington DC (USA), pp. 418-423, 1996.

[7] J. Rynkiewicz, "Estimation et identification de modèles autorégressifs nonlinéaires multidimensionnels", Technical Report Samos 125, Paris 1 University, 2000.

[8] A.J. Shepherd, *Second-Order Methods for Neural Networks*, Springer Verlag, 1997.

[9] Vapnick V. N. *The Nature of Statistical Learning Theory*, Springer Verlag, 1995.