Second Differentials in Arbitrary Feed-Forward Neural Networks*

Fabrice ROSSI†
THOMSON-CSF/AIRSYS
7-9, rue des Mathurins
92221 BAGNEUX, FRANCE
e-mail : rossi@ceremade.dauphine.fr

Abstract

We extend here a general mathematical model for feed-forward neural networks. Such a network is represented as a vectorial function f of two variables, x (the input of the network) and w (the weight vector). We have already shown that the differential of f can be computed with an extended back-propagation algorithm as well as with a direct method. In this paper, we show that the second differentials of f can also be computed with several different algorithms. Evaluating the theoretical complexities of these methods allow to choose the fastest algorithm for a particular architecture. This will allow us to handle arbitrary feed-forward neural network learning with the help of recent training and analysis techniques based on the Hessian matrix of the error.

1 Introduction

Léon Bottou and Patrick Gallinari have proposed in [2] a general framework for the cooperation of neural modules. In a former paper [6] we have extended this model in order to derive a general feed-forward neural network model that easily adapts to neural network simulation software design [4]. The key idea of this model is to generalize the notion of neuron and to allow an arbitrary feed-forward connection graph. The main limitation of the classic MLP and of all its derivatives is the strong relationship between the communication structure (the graph) that allows neurons to talk to each other and the controlling structure (the weights) that allows a training algorithm to modify the local computation performed by a neuron. Our generalization breaks this link: we use an arbitrary non cyclic graph for the communication structure and a control vector for each neuron. This weight vector does not control communications as a connection weight: the relationship between the input of the neuron, its control vector and its output is only modeled by a vectorial function which can be of any type. Moreover, the output of a neuron is a vectorial value: this allows to handle for instance a layer of MLP neurons as a complex neuron ans includes the traditionnal case of single output neurons. This model is mathematically described in section 2. The main problem with such a model is of course to train it. Gradient based optimization algorithms are directly usable for such a training, as the gradient of the error made by a given network can be computed with two algorithms (a direct algorithm and an extended back-propagation). These results demonstrated in [5, 6] are recalled in 3.

Computing the Hessian matrix of the error made by a neural network is very useful for practical purpose such as post training analysis or second order training (see [3] for a review of such methods). In this paper, we show that, unlike what was stated in [1], it is always possible to compute exactly the second differential of the output of any neuron of the network. We also extend the results about second derivatives presented in [3]: the authors use a model close to Bottou and Gallinari's model and do not study the complexity of the computation.

Due to space limitation, the proofs are omitted in this paper. All the necessary details can be found in technical reports ([5, 7]).

*This work was performed on Mrs Kim K. PHAM's responsibility, at THOMSON-CSF/AIRSYS.
Published in ICNN 96.
Available at http://apiacoa.org/publications/1996/icnn96.pdf

†Up to date contact informations for Fabrice Rossi are avaible at http://apiacoa.org/

2 The general model

2.1 The neuron

In our model, a neuron is a vectorial function of several variables:

Definition 1 *Let n be a positive integer and let $I_1, \ldots, I_n$, W and O be $n+2$ vectorial spaces on $\mathbb{R}$ of finite dimensions. A n-**input neuron** is a C^2 function from $I_1 \times I_2 \times \ldots \times I_n \times W$ to O.*

If N is such a neuron, we write $dN_w = \frac{\partial N}{\partial w}$ its partial differential with respect to its $(n+1)$-th variable and $dN_{i_k} = \frac{\partial N}{\partial o^k}$ its partial differential with respect to its k-th variable. N_i is the i-th coordinate of the output of the neuron. $\frac{\partial N_i}{\partial w_j}$ is the (i,j) term of the Jacobian matrix $\frac{\partial N}{\partial w}$ (a similar notation is available for $\frac{\partial N}{\partial o^k}$). $\frac{\partial^2 N_i}{\partial o_t^j \partial w_u}$ is the differential with respect to the u-th coordinate of N $(n+1)$-th variable of the differential of N_i with respect to the t-th coordinate of its j-th variable ($\frac{\partial^2 N_i}{\partial o_t^j \partial o_u^m}$ and $\frac{\partial^2 N_i}{\partial w_t \partial w_u}$ have similar definitions).

In this definition, W, I_k and O are respectively the weight space of the neuron, its k-th input space and its output space.

2.2 The neural net

The structure of our generalized neural network is based on a graph and in order to simplify the rest of the paper we introduce here some notations: $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is a graph with exactly n nodes ($\mathcal{N}$ is its node set and $\mathcal{E}$ its edge set); $N^1, \ldots, N^n$ is the sequence of the graph nodes; $P(N^k) = P(k)$ is the set of the predecessors of N^k; $S(N^k) = S(k)$ is the set of the successors of N^k. We assume that node N^k has exactly p^k predecessors and s^k successors, and that we call $P(k)_1, \ldots, P(k)_{p^k}$ the sequence of the predecessors of N^k. In general, superscripts correspond to node numbers and subscripts correspond to input or output numbers. Furthermore, we will not make any distinction between a node N^k and its rank k.

We also need to generalize the notion of predecessor: for an arbitrary node N, we define $P^0(N)$ as $\{N\}$ and recursively $P^k(N)$ as $\{M \in \mathcal{N} \mid \exists Q \in P^{k-1}(N)$ so that $(M,Q) \in \mathcal{E}\}$. We have therefore $P^1(N) = P(N)$. We call also $P^+(N)$ the set $\cup_{k=1}^{\infty} P^k(N)$ and $P^*(N) = P^0(N) \cup P^+(N)$. Similar sets can be defined in order to generalize the notion of successor.

We can now define a neural network:

Definition 2 *A **feed-forward neural network** is a **non cyclic** graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ which fulfills the following conditions:*

1. *The elements of $\mathcal{N}$ are **neurons**. The output space of N^k is O^k and its weight space is W^k.*
2. *If $p^k > 0$ then neuron N^k is a p^k-input neuron. The input spaces of N^k are $I_1^k, \ldots, I_{p^k}^k$.*
3. *If $p^k = 0$ then neuron N^k is a 1-input neuron with input space I^k.*
4. *If $p^k > 0$ then for each input i of the neuron N^k the following condition holds: $\dim I_i^k = \dim O^{P(k)_i}$.*

Let us now introduce some additional definitions related to the neural network.

$In = \{N \in \mathcal{N} \mid P(N) = \emptyset\} = \{In_1, \ldots, In_{in}\}$ is the *subset of $\mathcal{N}$ which elements have no predecessor*. An element of In is an input node. As we make no distinction between a node and its rank, we can call for instance O^{In_k} the output space of the node $N^j = In_k$, which is in fact O^j.

$Out = \{N \in \mathcal{N} \mid S(N) = \emptyset\} = \{Out_1, \ldots, Out_{out}\}$ is the *subset of $\mathcal{N}$ which elements have no successor*. An element of Out is an output node.

The neural network has for *input space* $I = \prod_{k=1}^{in} I^{In_k}$, for *output space* $O = \prod_{k=1}^{out} O^{Out_k}$ and for *weight space* $W = \prod_{k=1}^{n} W^k$.

2.3 Computing the output

We define the output of the network like this:

Definition 3 *Let $\mathcal{G}$ be a feed-forward neural network. Let $x = (x_1, \ldots, x_{in}) \in I$ be an input vector and let $w = (w^1, \ldots, w^n) \in W$ be a weight vector. For each l, $1 \leq l \leq n$, $o^l(x, w)$, the output of the neuron N^l, and $E^l(x, w)$, the generalized input of N^l, are computed with the help of the following recursive construction:*

- $o^l(x, w) = N^l(E^l(x, w), w^l)$
- *If $N^l \in In$, we have $N^l = In_k$. Then $E^l(x, w) = x_k$.*
- *If $N^l \notin In$: $E^l(x, w) = \left(o^{P(l)_1}(x, w), \ldots, o^{P(l)_{p^l}}(x, w)\right)$.*

Then the output of the network is $G(x, w) = \left(o^{Out_1}(x, w), \ldots, o^{Out_{out}}(x, w)\right)$.

Of course this definition is correct only because the underlying graph is non cyclic.

2.4 Computing the differential

2.4.1 Direct method

As a composed function, G is C^2 (since the N^i are C^2). The first method to compute its differential is to use the standard derivation rule of composed functions. This is the *direct method*. The general formula is:

$$\frac{\partial o^l}{\partial w^j}(x, w) = \sum_{k=1}^{p^l} dN^l_{i_k}\left(E^l(x, w)\right) \frac{\partial o^{P(l)_k}}{\partial w^j}(x, w) \tag{1}$$

2.4.2 Back-propagation

The key idea of the back-propagation algorithm is to consider $o^l(x, w)$, the output of neuron N^l, as a function of $o^j(x, w)$, the output of another neuron N^j. We define in fact $o^{l \to j}(x, w, f^j)$ which is the output of node N^l when o^j is "free" from the network constraints and can take arbitrary values (represented by f^j). We have of course $o^{l \to j}(x, w, o^j(x, w)) = o^l(x, w)$ and the following equation:

$$\frac{\partial o^l}{\partial w^j}(x, w) = \frac{\partial o^{l \to j}}{\partial o^j}\left(x, w, o^j(x, w)\right) dN^j_w\left(E^j(x, w)\right) \tag{2}$$

The back-propagation algorithm provides us with a recursive method to compute $\frac{\partial o^{l \to j}}{\partial o^j}$ (This differential can also be computed with a direct method using an equation close to equation 1). We first need an additional definition:

Definition 4 *Let $\mathcal{G}$ be a graph and let N^k and N^l be two nodes of $\mathcal{G}$. $r(k, l)$ is the rank of N^k in the predecessor set of N^l, i.e., N^k is the $r(k, l)$-th predecessor of N^l.*

Then, we have the following general formula:

$$\frac{\partial o^{l \to j}}{\partial o^j}\left(x, w, o^j(x, w)\right) = \sum_{N^k \in S(j)} \frac{\partial o^{l \to k}}{\partial o^k}\left(x, w, o^k(x, w)\right) dN^k_{i_{r(j,k)}}(I^k(x, w)) \tag{3}$$

In order to simplify the rest of the paper, we will use the following slightly incorrect notation: $\frac{\partial o^l}{\partial o^j}(x, w) = \frac{\partial o^{l \to j}}{\partial o^j}\left(x, w, o^j(x, w)\right)$

2.4.3 Error function

Computing the first differential of the error made by $\mathcal{G}$ with respect to some given pattern can also be done with the direct method or with the back-propagation. We assume that $\mathcal{E}$, the error function, is a function from $O^{Out_1} \times \ldots \times O^{Out_{out}}$ (the output space of the network) to $\mathbb{R}$. $\frac{\partial \mathcal{E}}{\partial o^k}$ is the partial differential of $\mathcal{E}$ with respect to the output of O^{Out_k}. Moreover, $\mathcal{E}^{\to l}$ is the error of the network as a function of the output of node N^l.

• the direct method gives:

$$\frac{\partial \mathcal{E}}{\partial w^j}(x,w) = \sum_{k=1}^{out} \frac{\partial \mathcal{E}}{\partial o^k}(x,w) \frac{\partial o^{Out_k}}{\partial w^j}(x,w) \tag{4}$$

• the back-propagation method gives:

$$\frac{\partial \mathcal{E}^{\to j}}{\partial o^j}\left(x,w,o^j(x,w)\right) = \sum_{N^k \in S(j)} \frac{\partial \mathcal{E}^{\to k}}{\partial o^k}\left(x,w,o^k(x,w)\right) \frac{\partial N^k}{\partial o^{r(j,k)}}(I^k(x,w)) \tag{5}$$

An equation similar to equation 2 allows to compute $\frac{\partial \mathcal{E}}{\partial w^l}$ with the help of $\frac{\partial \mathcal{E}^{\to l}}{\partial o^l}$.

3 The second differentials

Computing the Hessian matrix of the error made by a neural network is a quite difficult task because the simplest approach (i.e., applying the chain rule) gives time consuming methods and we must therefore study extended differentiation methods which give better results but are more difficult to study. In this section, we summarize our general results.

3.1 The direct method

This method is a simple application of the chain rule and gives for the general case (i.e., when $m \neq l$ and $j \neq l$):

$$\frac{\partial^2 o_i^l}{\partial w_t^j \partial w_u^m} = \sum_{k=1}^{p^l} \sum_{q=1}^{n^{P(l)_k}} \left(\frac{\partial o_q^{P(l)_k}}{\partial w_t^j}(x,w) \sum_{r=1}^{p^l} \frac{\partial^2 N_i^l}{\partial o_q^k \partial o^r} \frac{\partial o^{P(l)_r}}{\partial w_u^m} + \frac{\partial N_i^l}{\partial o_q^k} \frac{\partial^2 o_q^{P(l)_k}}{\partial w_t^j \partial w_u^m} \right) \tag{6}$$

If $j = l$ and $m \neq l$, we have:

$$\frac{\partial^2 o_i^l}{\partial w_t^l \partial w_u^m} = \sum_{r=1}^{p^l} \frac{\partial^2 N_i^l}{\partial w_t^l \partial o^r} \frac{\partial o^{P(l)_r}}{\partial w_u^m} \tag{7}$$

When $j = l = m$, $\frac{\partial^2 o_i^l}{\partial w_t^l \partial w_u^l}$ is simply a local Hessian matrix.

Similar formulae are fulfilled for $\frac{\partial^2 \mathcal{E}}{\partial w_t^j \partial w_u^m}$. The key point for these formulae is that in order to compute the Hessian of o_i^l, we need the Hessian of o_s^k for all $k \in P(l)$ (and for all s). Recursively this implies that in order to compute the Hessian matrix of the error, we need the Hessian matrix for the output functions of every node of the network.

3.2 The mixed method

Another method for computing the second differentials is to differentiate the back-propagation formulae (i.e., equations 2 and 3). We obtain the following results (similar formulae are fulfilled for $\mathcal{E}$):

• For the local equation:

$$\text{when } m \neq j, \quad \frac{\partial^2 o_i^l}{\partial w_t^j \partial w_u^m} = \sum_{q=1}^{n^j} \left(\frac{\partial o_i^l}{\partial o_q^j} \sum_{r=1}^{p^j} \frac{\partial^2 N_q^j}{\partial w_t^j \partial o^r} \frac{\partial o^{P(j)_r}}{\partial w_u^m} + \frac{\partial^2 o_i^l}{\partial o_q^j \partial w_u^m} \frac{\partial N_q^j}{\partial w_t^j} \right) \tag{8}$$

$$\text{when } m = j, \quad \frac{\partial^2 o_i^l}{\partial w_t^j \partial w_u^j} = \sum_{q=1}^{n^j} \left(\frac{\partial^2 o_i^l}{\partial o_t^j \partial w_u^j} \frac{\partial N_q^j}{\partial w_t^j} + \frac{\partial o_i^l}{\partial o_q^j} \frac{\partial^2 N_q^j}{\partial w_t^j \partial w_u^j} \right) \tag{9}$$

• When $N^m \notin S(j)$, we have a recursive equation, when $j \neq l$:

$$\frac{\partial^2 o_i^l}{\partial o_t^j \partial w_u^m} = \sum_{N^k \in S(j)} \sum_{q=1}^{n^k} \left(\frac{\partial o_i^l}{\partial o_q^k} \sum_{r=1}^{p^k} \frac{\partial^2 N_q^k}{\partial o_t^{r(j,k)} \partial o^r} \frac{\partial o^{P(k)_r}}{\partial w_u^m} + \frac{\partial^2 o_i^l}{\partial o_q^k \partial w_u^m} \frac{\partial N_q^k}{\partial o_t^{r(j,k)}} \right), \tag{10}$$

and when $j = l$:

$$\frac{\partial^2 o_i^l}{\partial o_t^l \partial w_u^m} = 0 \tag{11}$$

• When $N^m \in S(j)$, the recursive equation is:

$$\begin{aligned} \frac{\partial^2 o_i^l}{\partial o_t^j \partial w_u^m} &= \sum_{N^k \in S(j), k \neq m} \sum_{q=1}^{n^k} \left(\frac{\partial o_i^l}{\partial o_q^k} \sum_{r=1}^{p^k} \frac{\partial^2 N_q^k}{\partial o_t^{r(j,k)} \partial o^r} \frac{\partial o^{P(k)_r}}{\partial w_u^m} + \frac{\partial^2 o_i^l}{\partial o_q^k \partial w_u^m} \frac{\partial N_q^k}{\partial o_t^{r(j,k)}} \right) \\ &+ \sum_{q=1}^{n^m} \left(\frac{\partial^2 o_i^l}{\partial o_q^m \partial w_u^m} \frac{\partial N_q^k}{\partial o_t^{r(j,m)}} + \frac{\partial o^l}{\partial o_q^m} \frac{\partial^2 N_q^m}{\partial o_t^{r(j,m)} \partial w_u^m} \right) \end{aligned} \tag{12}$$

This method is very similar to the back-propagation method: in order to compute $\frac{\partial^2 o_i^l}{\partial w_t^j \partial w_u^m}$, we need $\frac{\partial^2 o_i^l}{\partial o_s^j \partial w_u^j}$ (for all s) and in order to obtain $\frac{\partial^2 o_i^l}{\partial o_s^j \partial w_u^j}$, the recursive method needs $\frac{\partial^2 o_i^l}{\partial o_q^k \partial w_u^m}$, for each $N^k \in S(j)$ and therefore recursively for each $N^k \in S^+(j)$. The efficiency of the method comes from the fact that $\frac{\partial^2 o_i^l}{\partial o_q^k \partial w_u^m}$ can also be used to compute $\frac{\partial^2 o_i^l}{\partial w_q^k \partial w_u^m}$ which is needed in order to obtain the complete Hessian matrix.

The main problem of this method is that it is not symmetric: when a differentiation order is chosen for (N^j, N^m), we must keep it for all (N^k, N^m), where $N^k \in S^+(N^j)$. See [7] for a complete discussion on the drawbacks of this method.

3.3 The back-propagation method

The key idea of the back-propagation method is to compute $\frac{\partial^2 o_i^l}{\partial o_t^j \partial w_u^j}$ with a method close to the first order back-propagation. We define first a "free" first order differential $\left(\frac{\partial o^l}{\partial o^j}\right)^{\to m}$, a function from $I \times W \times O^m$ to $L(O^j, O^l)$ (the matricial space of linear functions from O^j to O^l):

• if $N^j = N^l$: $\left(\frac{\partial o^l}{\partial o^j}\right)^{\to m}(x, w, f^m) = Id_{O^j}$.

• if $N^j \notin P^*(l)$: $\left(\frac{\partial o^l}{\partial o^j}\right)^{\to m}(x, w, f^m) = 0_{O^j, O^l}$

• if $N^j \in P^+(l)$:

$$\left(\frac{\partial o^l}{\partial o^j}\right)^{\to m}(x, w, f^m) = \sum_{N^k \in S(j)} \left(\frac{\partial o^l}{\partial o^k}\right)^{\to m}(x, w, f^m) \frac{\partial N^k}{\partial o^{r(j,k)}} \left(o^{P(k)_1 \to m}(x, w, f^m), \ldots, w^k\right) \tag{13}$$

Then we have the following powerful result if $N^m \notin S^+(j)$ [7]:

$$\frac{\partial^2 o_i^l}{\partial o_t^j \partial w^m}(x, w) = \frac{\partial}{\partial o^m} \left(\frac{\partial o_i^l}{\partial o_t^j}\right)^{\to m}(x, w, o^m(x, w)) \frac{\partial N^m}{\partial w}(E^m(x, w), w^m) \tag{14}$$

Therefore in order to obtain $\frac{\partial^2 o_i^l}{\partial o_t^j \partial w^m}$, we just need to compute $\frac{\partial}{\partial o^m}\left(\frac{\partial o_i^l}{\partial o_t^j}\right)^{\to m}(x, w, o^m(x, w))$.

• if $N^j \in P^+(l)$:

$$\frac{\partial}{\partial o_v^m}\left(\frac{\partial o_i^l}{\partial o_t^j}\right)^{\to m} = \sum_{N^k \in S(j)} \sum_{q=1}^{n^k} \left(\frac{\partial o_i^l}{\partial o_q^k} \sum_{r=1}^{p^k} \frac{\partial^2 N_q^k}{\partial o_t^{r(j,k)} \partial o^r} \frac{\partial o^r}{\partial o_v^m} + \frac{\partial}{\partial o_v^m}\left(\frac{\partial o_i^l}{\partial o_q^k}\right)^{\to m} \frac{\partial N_q^k}{\partial o_t^{r(j,k)}} \right) \tag{15}$$

• if $N^j \notin P^+(l)$:

$$\frac{\partial}{\partial o_v^m}\left(\frac{\partial o_i^l}{\partial o_t^j}\right)^{\rightarrow m} = 0_{O^m,\mathbb{R}} \tag{16}$$

The main advantage of this method is its symmetry:

As explained in the previous section, when we choose to compute $\frac{\partial^2 o_i^l}{\partial w_t^j \partial w_u^m}$, we need $\frac{\partial}{\partial o_u^m}\left(\frac{\partial o_i^l}{\partial o_s^k}\right)^{\rightarrow m}$ for all $N^k \in S^*(j)$. For efficiency reasons, this means that we will compute $\frac{\partial^2 o_i^l}{\partial w_s^k \partial w_u^m}$ with this differentiation order. But as the calculation formulae are not symmetric, this might be longer than using the alternate order. Both orders are available for the pair (N^k, N^m) only if $N^m \notin S^+(N^k)$ and $N^j \notin S^+(N^m)$. A quite complex proof leads to the following fundamental property (demonstrated in [7]):

Property *If $N^m \notin S^+(N^k)$ and $N^k \notin S^+(N^m)$:*

$$\frac{\partial}{\partial o_u^m}\left(\frac{\partial o_i^l}{\partial o_s^k}\right)^{\rightarrow m}(x, w, o^m(x, w)) = \frac{\partial}{\partial o_s^k}\left(\frac{\partial o_i^l}{\partial o_u^m}\right)^{\rightarrow k}(x, w, o^k(x, w)) \tag{17}$$

Therefore, each time the differentiation order can be chosen, the results are equivalent and therefore, choosing the order w^j before w^m does not imply anything for the successors of N^j (see [7] for details). This result is really important and does not appear if we use an inaccurate notation such as $\frac{\partial^2 o_i^l}{\partial o_s^k \partial o_u^m}$ (used in [3]). The simple meaning of this notation is in fact $\frac{\partial^2 o_i^{l\rightarrow k,m}}{\partial o_s^k \partial o_u^m}$ where the output of o^l depends on the "free" outputs of o^k and o^l. But in general, $\frac{\partial^2 o_i^{l\rightarrow k,m}}{\partial o_s^k \partial o_u^m} \neq \frac{\partial}{\partial o_u^m}\left(\frac{\partial o_i^l}{\partial o_s^k}\right)^{\rightarrow m}$.

In summary, the back-propagation method applies three formulae: equation 8 (or 9) which connects $\frac{\partial^2 o_i^l}{\partial w_t^j \partial w^m}$ to $\frac{\partial^2 o_i^l}{\partial o_t^j \partial w^m}$; equation 14 in order to obtain $\frac{\partial^2 o_i^l}{\partial o_t^j \partial w^m}$ from $\frac{\partial}{\partial o_v^m}\left(\frac{\partial o_i^l}{\partial o_t^j}\right)^{\rightarrow m}$ and equation 15 (or 16) in order to recursively compute $\frac{\partial}{\partial o_v^m}\left(\frac{\partial o_i^l}{\partial o_s^k}\right)^{\rightarrow m}$ for all $N^k \in S^*(j)$.

The case of the Hessian matrix of the error is almost the same. The main difference is for the particular cases which use other formulae (see [7]).

3.4 Complexity

The theoretical complexity of the second differential calculation is fully studied in [7]. We give here a simple summary.

3.4.1 The pure second order part

This part of the complexity takes only into account the calculation needed to compute the Hessian matrix assuming that every needed local first and second differentials, and every non-local first differentials have been already computed.

The obtained complexity formulae are really complex and in fact they are not directly comparable. On one hand, for the simple case of a multi-layer perceptron (MLP), the back-propagation method is the best algorithm. On the other hand, for very particular architectures, the direct method can be better than the back-propagation.

In fact, the problem is close to the first order differential problem explained in [6]. The theoretical complexity must be computed for the used architecture in order to decide which algorithm to use.

3.4.2 The first order part

We can provethat in order to compute the Hessian matrix of the error, we need the first order differential $\frac{\partial o^l}{\partial o^k}$ for all node pairs in the network [7]. Once again, we can compute the theoretical time needed to compute these values with the first order direct method or with the first order back-propagation method [5, 7]. For a classic MLP with a decreasing number of neurons (i.e. the number of neurons in layer k is less or equal to the number of neurons in layer $k-1$ for $k > 1$), the direct method is faster than the back-propagation and in spite of what is proposed in [3], the fastest way to compute the gradient of the error (as long as we want to compute simultaneously the

Hessian) is the direct method. For less common networks (such as compression networks) with a bottleneck, the back-propagation algorithm might be faster.

4 Conclusion

In this paper, we have extended results presented in [6]. We have described a general mathematical model for feed-forward neural networks in which first and second differentials can be efficiently computed. This model is used at present as the theoretical basis of a general neural networks simulator software [4]. The results presented here allow to use for very complex and uncommon neural structures recent training and analysis techniques based on the Hessian matrix of the error. More precise results available in [7] can be use in order to choose the fastest Hessian evaluation algorithm for a given architecture.

References

[1] Léon Bottou. *Une Approche théorique de l'Apprentissage Connexioniste ; Applications à la reconnaissance de la Parole*. Thèse de doctorat, Université d'Orsay, 1991.

[2] Léon Bottou and Patrick Gallinari. A Framework for the Cooperation of Learning Algorithms. In R.P. Lippmann, J.E. Moody, and D.S. Touretzky, editors, *Neural Information Processing Systems*, volume 3, pages 781–788. Morgan Kauffman, 1991.

[3] Wray L. Buntine and Andreas S. Weigend. Computing Second Derivatives in Feed-Forward Networks : A Review. *IEEE Trans. on Neural Networks*, 5(3):480–488, May 1994.

[4] Cédric Gégout, Bernard Girau, and Fabrice Rossi. NSK, an Object-Oriented Simulator Kernel for Arbitrary Feedforward Neural Networks. In *Int. Conf. on Tools with Artificial Intelligence*, pages 93–104, New Orleans (Louisiana), November 1994. IEEE.

[5] Cédric Gégout, Bernard Girau, and Fabrice Rossi. A General Feed-Forward Neural Network Model. Technical report NC-TR-95-041, NeuroCOLT, Royal Holloway, University of London, May 1995. Available at `http://apiacoa.org/publications/1995/neurocolt1995.pdf`.

[6] Cédric Gégout, Bernard Girau, and Fabrice Rossi. Generic Back-Propagation in Arbitrary Feedforward Neural Networks. In D. W. Pearson, N. C. Steele, and R. F. Albrecht, editors, *Int. Conf. on Artificial Neural Nets and Genetic Algorithms*, pages 168–171, Alès, April 1995. Springer Verlag.

[7] Fabrice Rossi. Second Differentials in Arbitrary Feed-Forward Neural Networks. Technical report THOMSON-CSF/AIRSYS/RDTE-594/96, THOMSON-CSF/AIRSYS, September 1996. Available at `http://apiacoa.org/publications/1996/thomson1996second.pdf`.