To appear in the International Journal of Circuit Theory and Applications, 1991

A UNIFIED FRAMEWORK FOR GRADIENT ALGORITHMS USED FOR FILTER ADAPTATION AND NEURAL NETWORK TRAINING

Sylvie MARCOS, Odile MACCHI, fellow, IEEE, Christophe VIGNAT

GRECO TdSI and Laboratoire des Signaux et Systèmes, CNRS - ESE

Plateau de Moulon, 91192 Gif sur Yvette cedex, FRANCE

Gérard DREYFUS, Léon PERSONNAZ, Pierre ROUSSEL-RAGOT

Laboratoire d'Electronique, Ecole Supérieure de Physique et de Chimie Industrielles de la Ville de Paris, 10 rue Vauquelin, 75005 Paris, FRANCE

ABSTRACT

In this paper, we present in a unified framework the gradient algorithms employed in the adaptation of linear time filters (TF) and the supervised training of (non linear) neural networks (NN). The optimality criteria used to optimize the parameters H of the filter or network are the least squares (LS) and least mean squares (LMS) in both contexts. They respectively minimize the total or the mean squares of the error $e(k)$ between an (output) reference sequence $d(k)$ and the actual system output $y(k)$ corresponding to the input $X(k)$. Minimization is performed iteratively by a gradient algorithm. The index k, in (TF), is time; it runs indefinitely. Thus iterations start as soon as reception of $X(k)$ begins. The recursive algorithm for the adaptation $H(k-1) \rightarrow H(k)$ of the parameters is implemented each time a new input $X(k)$ is observed. When training a (NN) with a finite number of examples, the index k denotes the example : it is upperbounded. Iterative (block) algorithms wait until all the K examples are received to begin the network updating. However, K being frequently very large, recursive algorithms are also often preferred in (NN) training. But they raise the question of ordering the examples $X(k)$.

Except in the specific case of a transversal filter, there is no general recursive technique for optimizing the LS criterion. However, $X(k)$ is normally a random stationary sequence; thus LS and LMS are equivalent when k gets large. Moreover the LMS criterion can always be minimized recursively with the help of the stochastic LMS gradient algorithm, which has low computational complexity.

In (TF), $X(k)$ is a sliding window of (time) samples, whereas in the supervised training of (NN) with arbitrarily ordered examples, $X(k-1)$ and $X(k)$ have nothing to do with each other. When this (major) difference is rubbed out by plugging a time signal at the network input, the recursive algorithms recently developped for (NN) training become similar to those of adaptive filtering. In this context, the present paper displays the similarities between adaptive cascaded linear filters and trained multilayer networks. It is also shown that there is a close similarity between adaptive recursive filters and neural networks including feedback loops.

The classical filtering approach is to evaluate the gradient by "forward propagation" whereas the most popular (NN) training method uses a gradient backward propagation method. We show that when a linear (TF) problem is implemented by a (NN), the two approaches are equivalent. Yet, the

backward method can be used for more general (non linear) filtering problems. Conversely, new insights can be drawn in the (NN) context by the use of a gradient forward computation.

The advantage of the (NN) framework, and in particular of the gradient backward propagation approach, is evidently to have a much larger spectrum of applications than (TF), since (i) the inputs are arbitrary, and (ii) the (NN) can perform nonlinear (TF).

INTRODUCTION

The problem of optimizing the set H of parameters of a system h is raised in a great many areas of Sciences. For instance, the system h can be a time filter (TF) that is a linear application from the space of time functions into itself, or a formal neural network (NN) that is a highly connected set of elementary non linear neural cells. (TF) and (NN) can be investigated in continuous time or in discrete time. In the former case, the evolution of the system is governed by a differential equation, in the latter case, it is governed by difference equations. This paper is only concerned with the second approach of discrete samples.

In 1960, Widrow published his ADALINE [1] and gave rise to the development of many optimization algorithms that can match the parameters H of the system to its environnment, even if time-varying. Such is the case of the Widrow-Hoff rule for self learning automata and of the LMS algorithm for adaptive linear filtering. These algorithms have a common purpose which is to minimize, by a gradient technique, a cost function based on the quadratic output errors, either through their total squares (the least squares (LS) criterion) or through their average squares (the least mean squares (LMS) criterion).

However, these algorithms were developed with unequal success in the various scientific areas. On the one hand, linear adaptive filters enjoyed a wide and fast stride with numerous applications in communications, sonar, radar etc... On the other hand, the networks of neurons or automata did not really spread out at the beginning. A reason for this difference is the greater conceptual simplicity of (TF). They are linear and have few parameters. Consequently, the theoretical analysis and hardware implementation are relatively straightforward. Conversely, (NN) are nonlinear, have a larger number of elementary cells which, besides, are interconnected with one another. Therefore analysis is much more intricate and implementation is a lot more costly.

The recent tremendous burst in technology, especially with digital signal processors, is one fact which spurred the research on (NN) [2]. It is recognized that their high degree of parallelism, their interconnectivity and their learning ability allow them to perform recognition and classification tasks, and make them robust and flexible. Among the important advances is a supervised training algorithm, called the gradient backpropagation rule [3]-[5] which generalizes the Widrow-Hoff algorithm for multilayer networks.

Recently, some contributions have appeared to show that (NN) are able to solve in a novel way certain ancient problems. It appears that (NN) can also be used as an efficient tool for new problems for instance in nonlinear time-filtering. A classical example of non linear filter called the adaptive differential pulse code modulation (ADPCM) sytem, is given in fig. 0; it is used in digital transmission of speech at a reduced bit rate. The purpose of the present paper is to establish a common framework to compare the criteria and algorithms used in the adaptation of (TF) and in the supervised training of (NN). We are not concerned here with specific applications.

In both contexts, optimization through iterative techniques are especially appreciated, the vector $\tilde{H}$ of optimum parameters being reached by the sequence of iterations

$$H(p) = H(p-1) + \Delta(p-1) \qquad (1)$$

where Δ is an increment which is related to the gradient of the cost function to be minimized.

Our objective is to **compare the two approaches of iterative gradient optimization** that have been set up in both fields. In particular we shall put into perspective the (LMS) algorithms [6]-[10] (also known as stochastic gradient) used in (TF), with the gradient algorithms involving backpropagation used in the supervised training of (NN). Their spirit is the same, but they are applied to contexts with important basic differences.

At first glance, it could appear that the structure of a (TF) is but a particular case of (NN) and that the theory of adaptive filtering should be completely imbedded in the one of learning (NN). But this is not true because certain important specific features of (TF) do not hold, in general, for (NN). In particular (TF) deals with time signals $X(k)$, and the index k, being time, is naturally ordered and usually unbounded. In contrast, (NN) do not always take into account the role of time. For instance in the context of classification, the index k refers to a collection of inputs $X(k)$ which are given to the networks as typical examples. The facts that the ordering of examples is arbitrary and that their number is finite are important differences between both contexts. An interesting specificity of (TF) is to adapt the filter H at the very moment k when it is being used. In other words, the iteration index is $p = k$ in the gradient algorithm (1). The resulting filter $H(k)$ is called "adaptive" and is able to

deal with evolutive inputs $X(k)$ and even with non stationary sequences. On the other hand, the usual concept of training for (NN) is the following : a fixed (non evolutive) collection of examples $X(k)$, called "training set", is given to the net for optimizing H. There is a preliminary training period during which the network H is optimized. During this period the optimizing iterations $H(p)$ do not necessarily correspond to the arrivals of examples, which means that p and k can be different. After optimization, the network keeps being used in its (fixed) final state. In this spirit, static (non ordered) as well as dynamic (ordered) [2, 11, 13] patterns can be learnt. This does not correspond to the common concept of adaptivity. It can be shown, however, that (NN) can be used in a truly adaptive fashion, whereby the network undergoes continual training while it is being used [12, 14, 15].

The present work is made of five parts. In the first section, we describe in detail the differences between the specific contexts of time filters and of neural networks. The LS and LMS optimization criteria are recalled in Section II, with their respective features in the two contexts. Section III is devoted to the derivation of the major gradient algorithms which permit optimization. An important distinction is made between iterative and recursive algorithms. In the latter the parameters are updated each time a new time sample arrives.

Sections IV and V constitute the core of this paper. In these sections, we study the gradient algorithms for some important examples of (TF) and (NN), respectively. In order to emphasize the similarity between both fields, these sections are built up in the same way, with three parallel subsections.

The first kind of systems is concerned with modular structures. In the context of (TF), we can take the example of a global filter $\mathbf{h}$ which is a cascade of sub-filters $\mathbf{h}_p$ (cf subsection IV.2). The corresponding example, in (NN) is a network $\mathbf{h}$ which is made of successive layers $\mathbf{h}_p$ (cf subsection V.2). The problem of adapting (or training) these compound systems $\mathbf{h}$ is very similar in both fields. In particular, the standard (LMS) algorithm of (TF) coincides for linear filters with the (NN) approach in which the gradient is backpropagated from the last layer until the first one (if the neurons have a linear activation function).

The second example deals with systems whose structures involve some kind of feedback loop. In (TF), we can take the example of an infinite impulse response (IIR) filter (cf subsection IV.3). The

corresponding concept in (NN) is the feedback network which also includes a loop. If the input $X(k)$ is a time signal, the network becomes quite similar to an IIR filter. The corresponding comparison again concludes that, for linear filters, the standard (LMS) algorithm of (TF) coincides with the (NN) gradient backpropagation approach, if the neurons have a linear activation function.

Conclusive remarks are given in Section VI. Throughout this paper, N denotes the total number of parameters of the system, grouped in the vector H ; p denotes an iteration index.

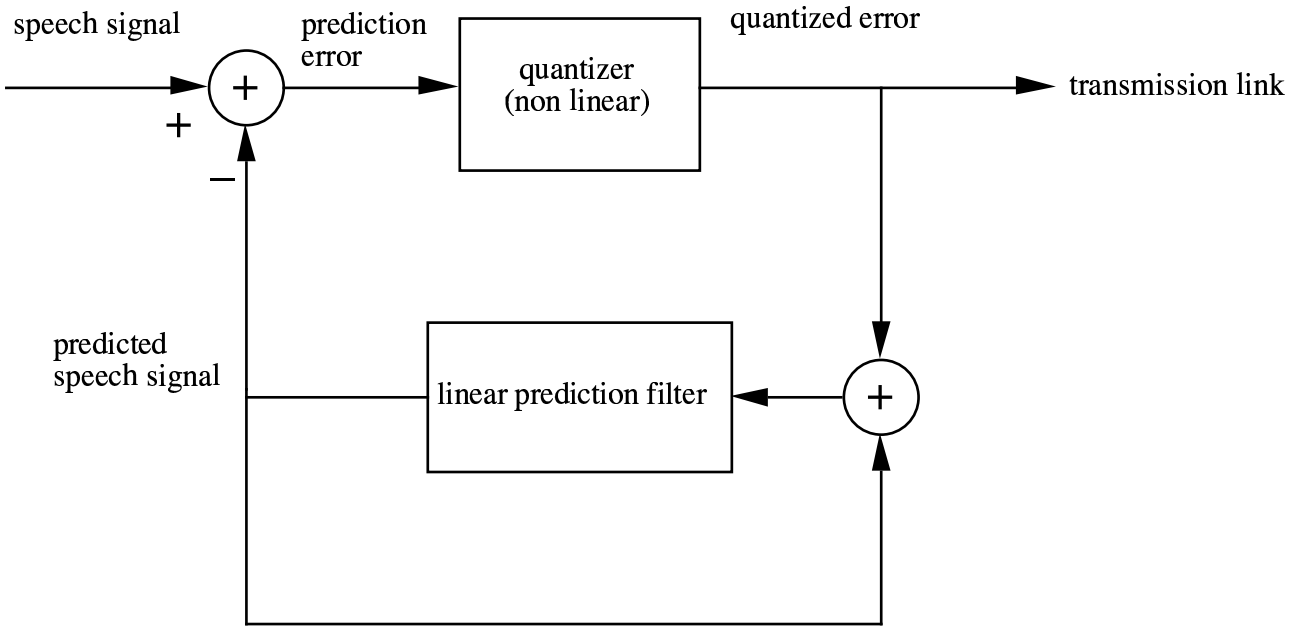

Fig. 0 : An example of non linear filter : the ADPCM system

I. DIFFERENCES BETWEEN (NN) AND (TF)

I.1.The context of trained (NN) (finite training set)

In the context of supervised training with a finite set of unordered examples, a (NN) is depicted in fig.1a. Supervised learning is a classical method to bring the network parameters H towards optimality. A (finite) collection of K examples is presented at the network input, together with the associated references. The examples can be images, speech segments, patterns, etc... The example # k consists of a collection of I real values $x_1(k), x_2(k), \ldots x_I(k)$, which may have different

physical meanings. For instance, x_1 is the size of a visual object, x_2 its light intensity, x_3 its orientation etc... This is why different inputs are represented by different marks in fig.1a.

The network executes certain (linear and nonlinear) calculations using the vectors

$$X(k) \triangleq (x_1(k), x_2(k), \ldots, x_I(k))^T, \tag{1.1}$$

$$H \triangleq (h_1, h_2, \ldots, h_N)^T, \tag{1.2}$$

where the second vector is built up with the N internal parameters of the network. The calculations use some nonlinear functions f such as the sigmoidal function

$$f(z) = A \tanh \rho z, \qquad \rho > 0. \tag{1.3}$$

In this way, for each example, the network produces a set $y_1(k), \ldots, y_O(k)$ of O outputs which constitute the output vector $Y(k) = (y_1(k), \ldots, y_O(k))^T$. Like for the input the variables inside $Y(k)$ may have different meanings. They are represented by different marks in fig. 1a. In this figure, the arrows indicate that the example presented at step k is removed before presentation of the next example.

It is clear that, in general, the input items $x_1(k-1)$ and $x_2(k)$ have nothing to do with each other; similarly $y_1(k-1)$ and $y_2(k)$ are not related. This is in contrast with the filtering problem (see below).

To train the network and bring its parameters H close to the optimal vector $\tilde{H}$, a finite collection $D(k), k = 1, \ldots, K$, of reference output vectors is available and the collection of errors

$$E(k) = D(k) - Y(k), \quad k \leq K, \tag{1.4}$$

is used to compute the increments $\Delta(p-1)$ in order to control the iterative search for $\tilde{H}$ in the algorithm (1).

I.2. The context of adaptive (TF)

The (TF) context is depicted in fig.1b. Then the iterative label k represents time. The situation is very similar to that of fig.1a, but with two additional properties :

(i) k increases indefinitely (no upperbound such as K in (1.4))

(ii) the input sequence is ordered and has the shifting property that

$$x_i(k) = x_{i-1}(k-1) \ . \tag{1.5}$$

The new input vector presented at time k shares its $I - 1$ last components $x_2(k), ..., x_I(k)$ with the old input vector presented at time $k - 1$. Thus the new information is fully contained in the first component

$$x(k) \triangleq x_1(k) \ . \tag{1.6}$$

Therefore all the coordinates of $X(k)$ have the same physical meaning. They are represented by identical marks in fig.1b. In this figure, the arrows indicate that the new data presented at time k is shifted one slot before the next time.

The filter executes certain linear calculations using the vector

$$X(k) = (x(k), x(k-1), ..., x(k-I+1))^T \cdot \tag{1.7}$$

At time k, it delivers a single (new) output $y_1(k) \triangleq y(k)$. The computation of $y(k)$ involves the filter internal parameters described by the H vector and may also involve past output values $y(k-1)$, $y(k-2)$, ... Note that the input and output marks are different in fig. 1b because the x and y variables have different physical meanings.

Fig.1b evidences the fact that all the information processing realized by the filter from time 1 until time k is summarized in the two input and output time sequences $x(1)$, $x(2)$, ... , $x(k)$ and $y(1)$, $y(2)$, ... , $y(k)$ respectively.

The adaptation of the filter is intended to match the output sequence $y(k)$ onto a reference sequence $d(k)$ and the indefinite sequence of (scalar) errors

$$e(k) = d(k) - y(k) \qquad k = 1, 2, ... \tag{1.8}$$

is used in the iterative increment $\Delta(p-1)$ in order to enforce the iterations (1) to approach the optimal parameter $\tilde{H}$.

A more general filtering situation occurs when there are several input sequences, say $x^1(k)$, $x^2(k)$ etc... and several output sequences as well, say $y_1(k)$, $y_2(k)$ etc... Conceptually this does not, however, bring a lot of new insights. This is why this paper is restricted to **a single input-single output (TF)**. For the sake of similarity, it will also be assumed that **the (NN) has only one output**

$$y(k) \triangleq y_1(k) . \tag{1.9}$$

In other words $\square = 1$ in (1.3). Correspondingly the reference outputs $D(k)$ are not vectors but scalar (say $d(k)$) and there is a collection of scalar errors $e(k)$ defined according to (1.8) as in the filtering case. Generalization to (NN) with several outputs is found in suitable papers [2]-[5]. In this paper, the emphasis is on similarities and dissimilarities between the supervised training of (NN) on one hand and the adaptation of (TF) on the other hand. Therefore, the restriction to a single output is not a loss of generality.

II. OPTIMIZATION CRITERIA

II.1. The LS criteria

For the time being, the system $\mathbf{h}$ under investigation is not specified. It can be a (TF) or a (NN). It can be linear or non-linear. It has N fixed parameters grouped in the vector H. At step k (k - th example or k-th time instant), the (single) output of $\mathbf{h}$ is

$$y(k) = F(H, X(k)) \tag{2.1}$$

where the vector $X(k)$ of inputs is given in (1.1) for (NN) and in (1.7) for (TF) ; the function F is linear or not versus the components or $X(k)$, depending whether the net or the filter is linear or not. In order to emphasize the dependency of $y(k)$ with respect to (w.r.t.) H, it is denoted $y_H(k)$. As noted below eq. (1.7), the computation of $y(k)$ may involve past outputs (or even past intermediate results).

Ideally each application should use a specific criterion to adequatly optimize the parameters H of $\mathbf{h}$ (error rate for pattern classification, bit error rate for digital transmission, signal to noise ratio for detection, listening test for speech transmission and recognition, etc...). Nevertheless, most of these criteria turn out to be mathematically intractable, except the criteria based on the errors

$$e_H(k) \triangleq d(k) - y_H(k) \tag{2.2}$$

between the reference and the system output.

The most popular criteria of that kind are known as "least squares" (LS). They consist in minimizing a cost J that is quadratic w.r.t. all the errors like (2.2) [16] .

We shall use the concept of "running cost", namely

$$J^k(H) = k^{-1} \sum_{n=1}^{k} e_H^2(n) \quad . \tag{2.3}$$

In (2.3), the pairs $(X(n), d(n))$ of inputs and references $n = 1, 2, \ldots, k$ are fixed. In this way, $J^k(H)$ depends only on the parameters H of the system. Eventually the running cost is also a function of the index k. This cost is running like time or examples, hence the name "running cost".

The running cost (2.3) is lowerbounded. Thus, under reasonably general assumptions of continuity for the function F in (2.1), it presents a minimum which, of course, depends on the index k . Let $\tilde{H}(k)$ be that minimum. We call it "running LS parameter". It will cancel the running cost gradient according to

$$\nabla J^k(H) = 0 \qquad \Big|_{H = \tilde{H}(k)} \tag{2.4}$$

where the gradient ∇ is calculated w.r.t. the parameter vector H.

Note the normalizing factor k^{-1} in (2.3). Evidently, the normalized running cost (2.3) and the unnormalized running cost

$$J_u^k(H) = \sum_{n=1}^{k} e_H^2(n) \quad . \tag{2.5}$$

are simultaneously minimized by the parameter $\tilde{H}(k)$. But $J_u^k(H)$ grows unbounded if k is allowed to increase indefinitely. Therefore $J^k(H)$ is more meaningful than $J_u^k(H)$, especially in the filtering context where the time k increases indefinitely.

Note also that the cost takes into consideration all the examples between 1 and k with the same importance.

In the context of adaptive (TF) where k has no upperbound, the usual concept is the running cost function $J^k(H)$ [*] . The latter can be evaluated from the very first step ($k = 1$). But then a

---

[*] It is implicit in the Signal Processing field that time is running. So the common terminology is simply "LS criterion" for $J^k(H)$ and "LS filter" for $\tilde{H}(k)$. But in the present comparison with (NN), we are obliged to make explicit the "running" idea in our terminology.

new question is raised : how does $\tilde{H}(k)$ varies with the step k ? There should indeed be some relationship between $\tilde{H}(k-1)$ and $\tilde{H}(k)$, and a procedure in order to derive $\tilde{H}(k)$ from $\tilde{H}(k-1)$.

Finally, we note that the LS criterion has been generalized in a number of ways in order to accommodate for nonstationarities in the sequences $\{X(k)\}$ and $d\{(k)\}$. In particular the "exponentially weighted LS" criterion used in (TF) has cost function [9]

$$J_{\nu}^{k}(H) = \sum_{n=1}^{k} (1-\nu)^{k-n} e_H^2(n) \quad , \tag{2.6}$$

where ν is a small positive quantity called forgetting rate. At the present time k, the weight associated to the past error $e_H(n)$ decreases as the delay $(k-n)$ increases. This permits to forget the influence of old errors and to track nonstationarities. Another way of achieving the same purpose is to use the "windowed LS" criterion which has cost function

$$J^{k,M}(H) = \frac{1}{M} \sum_{n=k-M+1}^{k} e_H(n)^2 \ . \tag{2.7}$$

This criterion has exact memory M and forgets completely the old errors $e_H(k-M)$, $e_H(k-M-1)$, and so on.

In the context of trained (NN), for instance in classification, the typical situation is to have a fixed collection of K examples $X(k), k = 1, 2, \ldots, K$ (the training set) with their associated references. Therefore, the usual cost function is the final or overall cost $J^K(H)$. The latter can be evaluated only after all examples have been presented to the network. We call "overall LS parameter" the corresponding (final) value $\tilde{H}(K)$ of the parameter. If all the examples do not play the same role, the most important ones can be suitably repeated. In this way, the importance of the various errors $e_H(n), n = 1, \ldots, k$, is naturally weighted inside the cost function (2.3). Therefore, in order to provide a proper comparison between (TF) and (NN), we can retain only the unweighted LS criterion associated to (2.3) or its windowed version (2.7). Like for (TF), it can be of interest to evaluate $\tilde{H}(K)$ for $k < K$, in order to prepare the calculation of the desired (final) parameter $\tilde{H}(K)$. This is done in Section III-5 below.

Evaluating $\tilde{H}(k)$. Since the gradient commutes with a sum, it follows from (2.3) that

$$\nabla J^k(H) = \frac{-2}{k} \sum_{n=1}^{k} e_H(n) \nabla_H y_H(n) \quad , \tag{2.8}$$

and similarly with the cost (2.7). In the right hand side (RHS) of (2.8), the quantities $y_H(n)$ are dependent on H and on $X(n)$. Thus, to avoid any ambiguity, the gradient has been written ∇_H.

For a known function F characteristic of the system (2.1), it is possible, at least in principle, to evaluate the gradients $\nabla_H y_H(n)$. Therefore, one possibility to calculate the optimum parameter $\tilde{H}(k)$ is the direct least squares procedure which follows:

first phase : for each possible candidate parameter H, we apply the system $\mathbf{h}$ to all the examples (or time samples) $X(n)$ so as to provide the $2k$ quantities $y_H(n)$, $\nabla_H y_H(n)$, for $n = 1, \ldots, k$. Hence the RHS of (2.8) as a function of H.

second phase : we search for the root(s) of the resulting function as required by (2.4). **Note that uniqueness of the root is not guaranteed.**

It is self evident that in the general case, the above procedure to derive the running LS parameter $\tilde{H}(k)$ is very heavy. In the filtering context (except in a very particular example, see the further subsection II.4) it cannot be implemented in real time, i.e., the computations do not stand within the delay between two time samples $x(k)$ and $x(k+1)$. The above procedure is not either used in the network context because of the excessive amount of required computational power. This explains why iterative procedures have gained so much favour in both fields. They solve phase 1 and phase 2 at the same time.

II.2. The LMS criterion

For any system $\mathbf{h}$, a prerequisite for the search of optimality of H to be meaningful is the existence of a statistical distribution for the pair $(X(k), d(k))$. In fact, if there were no statistical rule controlling the pairs $(X(k), d(k))$, there would be no advantage in learning H on the basis of a set of reference outputs.

In the (TF) context, and more generally in Signal Processing, this property corresponds to the assumption that the time sequence $(X(k), d(k))$ is stationary in the sense that the joint statistical properties of several pairs are invariant under an arbitrary time-shift. In the (NN) context, e.g., when the network is used for classification purposes, the very idea of training relies on the property that the training set items are representative of the items that the network will have to classify after training is completed. All these items have a common probability distribution.

This is why **in the rest of this paper, it is assumed that $(X(k), d(k))$ constitutes a stationary random sequence.**

Then, the very popular "least mean square" (LMS) criterion can be used to optimize the parameters H of system $\mathbf{h}$ [13]. It consists in searching for the minimum of the mean square cost

$$J(H) = E(e_H^2(k)) \quad . \tag{2.9}$$

The expectation symbol in (2.9) shows that the LMS criterion implements a statistical averaging over random trials, instead of the averaging over examples (or times) performed by the LS criterion as given in (2.3). Like $J^k(H)$, $J(H)$ is lowerbounded. Thus, it does present a minimum, denoted $\tilde{H}$. We call $\tilde{H}$ the "LMS parameter". Thanks to the stationary character of the random sequence $(X(k), d(k))$, this cost is a function of H which is independent of the step k. This is an advantage, over the LS cost. The corresponding optimum system does not depend on the running step k. It will cancel the cost gradient according to

$$\nabla J(H) = 0 \quad \Big|_{H = \tilde{H}} \quad . \tag{2.10}$$

Evaluating $\tilde{H}$. Since the gradient commutes with an expectation, it follows from (2.9) that

$$\nabla J(H) = -2\, E[\, e_H(k)\, \nabla_H\, y_H(k)] \quad . \tag{2.11}$$

Like for the LS optimum system, it is possible, in principle, to evaluate $\tilde{H}$ in two distinct phases. In the first phase, for each candidate parameter H, we apply the system $\mathbf{h}$ to a large number of (randomly sampled) input vectors $X(k)$ in order to produce the quantities $y_H(k)$ and $\nabla_H\, y_H(k)$ and to perform the averaging of these random trials. This phase will provide an estimate of the RHS of (2.11) as function of H. The second phase consists in finding the root (or the roots) of the resulting function as required by (2.10).

Again the above procedure is too heavy because of the huge amount of computations. Iterative algorithms are a good alternative because they have lower computational requirements. Moreover, they jointly solve phase 1 and phase 2.

II.3. Asymptotic equivalence between the LS and LMS criteria.

Consider the case of random inputs, when the pairs $(X(k),\ d(k))$ are strictly stationary and ergodic. Ergodism is the fact that the sampling realized by the index k is asymptotically equivalent to a sampling governed by the probability distribution of the random variable $(X,\ d)$. Therefore, the averaging over k of a (deterministic) function G of the random variable $(X,\ d)$ is asymptotically equivalent to the expectation of this G function, that is

$$\frac{1}{k}\sum_{n=1}^{k} G\,(X(k),\, d(k)) \to E[G(X(k),\, d(k))]\ , \qquad k \to \infty \quad . (2.12)$$

Applying this property to the function F of the system $\mathbf{h}$, it follows from definitions (2.3) and (2.9) that **the running LS cost tends towards the MS cost if k is actually allowed to tend to infinity** :

$$J^k(H) \to J(H), \qquad k \to \infty \quad . \qquad (2.13)$$

The infinite increase of k does happen in the filtering context (k = time). It could also happen in the training of (NN), in particular when the number K of examples in the training set is very large.

Then the LS and LMS criteria become equivalent and the corresponding optimum parameters satisfy

$$\tilde{H}(k) \rightarrow \tilde{H} \quad , \qquad k \rightarrow \infty \ . \tag{2.14}$$

The above discussion also means that the LMS and LS criteria are significantly different only during a transient initial period (k small). Otherwise (k large) we can choose indifferently any of the two criteria.

II.4 The transversal linear time filter case□

In subsection II.1, the difficulty involved in the evaluation of the running LS optimum parameter $\tilde{H}(k)$ was emphasized. There is, however, a case where it is possible to write it down with a relatively simple iterative formula similar to (1), i.e.

$$\tilde{H}(k) = \tilde{H}(k-1) + \tilde{\Delta}(k-1) \ , \tag{2.15}$$

the computational complexity involved in the increment $\tilde{\Delta}(k-1)$ being acceptable. This case is concerned with time filters when **the output is jointly linear versus the input vector $X(k)$ and versus the parameter vector H**. The corresponding linear filter, depicted in fig. 2, is called "transversal" or "finite impulse response" (FIR) ; it calculates the output

$$y_H(k) = X(k)^{\mathrm{T}} H \ . \tag{2.16}$$

Based on this equation, and on the shifting property (1.7) that is specific of (TF), very interesting results can be derived. In particular, it can be shown for the optimum LS parameter $\tilde{H}$, that the exact increment in (2.15) is given by

$$\tilde{\Delta}(k-1) = A(k)\, X(k)\, e_H(k) \ \Big|_{H = \tilde{H}(k-1)} \tag{2.17}$$

where there is a recursive formula for the N x N matrix $A(k)$:

$$A(k) = A(k-1) - \frac{A(k-1)X(k)X(k)^T A(k-1)}{1+X(k)^T A(k-1)X(k)}. \qquad (2.18)$$

These formulae are the so-called "recursive least squares" (RLS) [9]. Indeed, algorithm (2.15) is recursive in the sense of the following definition :

Definition : An iterative algorithm such as (1) is called recursive if there is a one-to-one correspondence between the occurrence of a new time input (or new example) k and the operation of a new iteration p. In other words p = k .

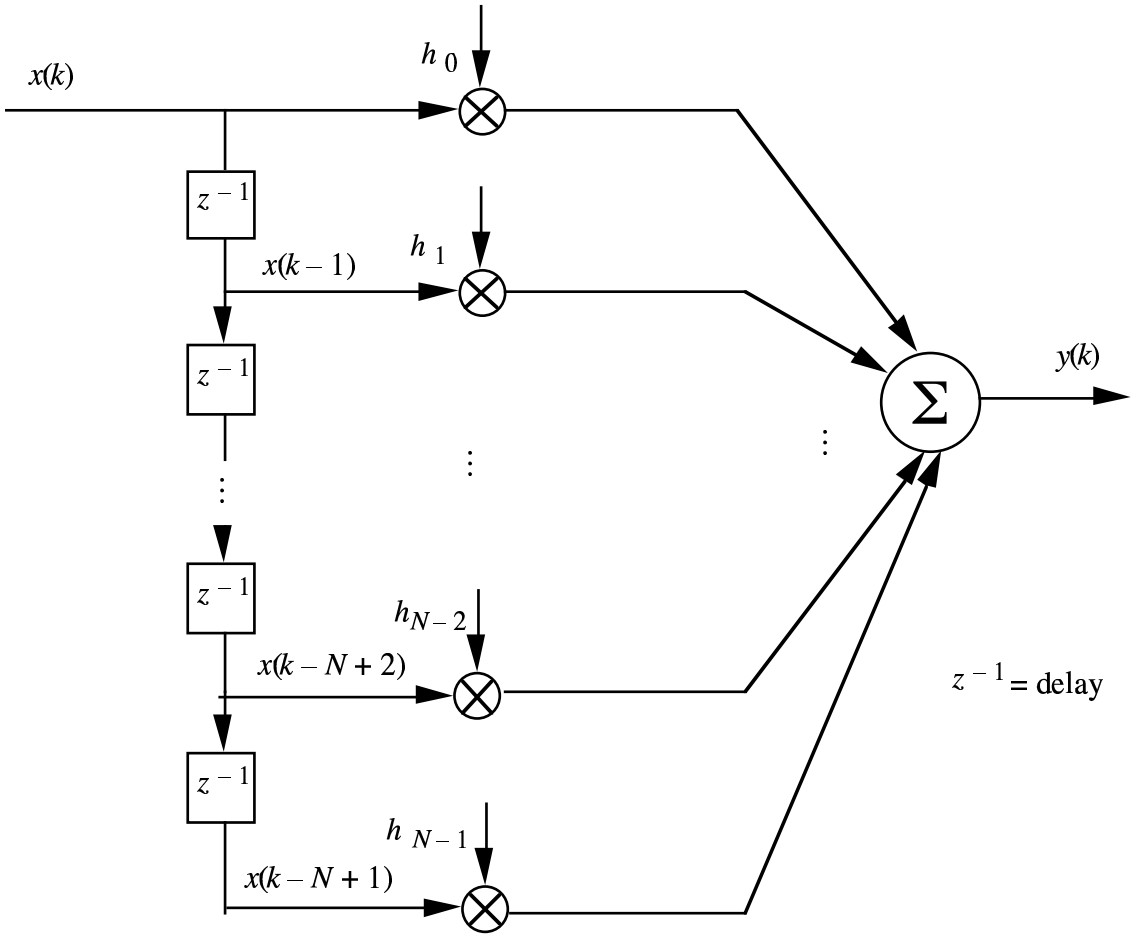

Fig. 2 : A Transversal time - filter

The RLS approach in (2.18) involves a number of multiplications proportional to N^2 at each iteration. This is a very small computational load as compared with the direct approach of subsection

II.1. In fact, the amount of computations can be further reduced, using the so-called "fast RLS" approach which avoids calculating the matrix $A(k)$ and directly evaluates the so-called said "Kalman-Gain", i.e., the N-coordinate vector

$$G(k) = A(k)\ X(k)\ . \tag{2.19}$$

Based on an elaborate theory of prediction, $G(k)$ itself can be computed recursively at the price of approximately 8 N multiplications per iteration k. See e.g. [9] for details.The RLS and fast RLS approaches can also take into account the weighting factor $(1-\nu)$ when the LS criterion is generalized according to (2.6). The same thing can be done for the window of length M in the cost (2.7).

The RLS algorithm is an important specificity of adaptive (TF) as compared to trained (NN). In the latter case, successive examples do not exhibit the shifting property (1.7). Moreover, the system equation (2.1) always involves some nonlinearity, such as the sigmoid function (1.3). Therefore, the RLS approach is not applicable. A fortiori, the associated fast RLS algorithms cannot be used. This means that there is no hope to exactly evaluate the LS optimum parameter $\tilde{H}\ (k)$, at least with a reasonable amount of computations. Therefore, the goal of any algorithm must be limited to an approximate evaluation. For instance, the iterative procedure (1) could be aimed at generating an approximately optimum parameter $H(k)$ such that (when all the examples have been presented to the net)

$$\left|\ H(K) - \tilde{H}(K)\ \right| \leq \varepsilon\ , \tag{2.20}$$

or (when enough examples have been presented)

$$\left|\ H(k) - \tilde{H}(k)\ \right| \leq \varepsilon \quad \text{for } k \geq k_0\ . \tag{2.21}$$

The first property is concerned with the case of a finite training set in (NN). The second property is useful in both contexts of (NN) and of (TF).

The present subsection was a way of introduction to the next section about iterative procedures.

III. ITERATIVE AND RECURSIVE GRADIENT ALGORITHMS

III.1. The objective

In the rest of this paper, it is assumed that the exact minimum $\tilde{H}(k)$ of the LS criterion cannot be calculated. One is satisfied with an approximate vector $H(k)$ satisfying (2.20) or (2.21) or having asymptotic optimality in the sense that

$$| H(k) - \tilde{H}(k) | \to 0 \ , \quad k \to \infty \ . \tag{3.1}$$

Similarly, minimization of the LMS cost (2.9) is seeked for, but only in an asymptotic way, either

$$| H(K) - \tilde{H} | < \varepsilon' \ , \tag{3.2}$$

or

$$| H(k) - \tilde{H} | < \varepsilon' \quad \text{for } k \geq k_0 \ , \tag{3.3}$$

or

$$| H(k) - \tilde{H} | \to 0 \ , \quad k \to \infty \ . \tag{3.4}$$

It follows from the asymptotic property (2.14) that the requirements (2.20), (2.21) or (3.1) about the LS criterion are respectively equivalent to the above requirements (3.2), (3.3) or (3.4) about the LMS criterion.

Gradient iterative procedures, also named steepest descent methods [6]-[8], are among the major and simplest means to achieve such properties.

III.2. The steepest descent method

If a known cost function $\mathcal{E}(H)$ presents a minimum $\hat{H}$, the gradient algorithm permits to approach $\hat{H}$ as illustrated on fig.3 with a 2-D vector. In this figure, H is plotted in the horizontal plane and $\mathcal{E}(H)$ is measured on the vertical axis. This gives a "bowl" surface of points $M = (H, \mathcal{E}(H))$ in the 3-D space, whose minimum is located at $H = \hat{H}$. The vertical half-plane that includes M and the horizontal vector $-\nabla \mathcal{E}(H)$ intersects the bowl along a steepest descent line. Hence the point

$$H' = H - \mu' \nabla \mathcal{E}(H) , \quad \mu' > 0 \tag{3.5}$$

is closer than H to the minimum $\hat{H}$, provided μ is not too large (to avoid an overshoot where H' climbs up on the opposite side of the bowl). This is the basis for the well-known iterative gradient algorithm

$$H(p) = H(p-1) - \mu' \nabla \mathcal{E}(H) \Big|_{H = H(p-1)} \tag{3.6}$$

Notice the label p for the iteration index. If $p \rightarrow \infty$, the vector $H(p)$ converges towards $\hat{H}$ provided the function $\mathcal{E}(H)$ is well-behaved and if $H(0)$ stands in the domain of attraction of the minimum $\hat{H}$ □.

III.3. The iterative block-LS gradient algorithm

Let us return to the problem of minimizing some LS cost function. In this subsection, we consider particularly the supervised training of a (NN), when the training set is arbitrarily ordered and **the total number K of examples is finite**. The relevant cost function is the overall cost

$$\mathcal{E}(H) \triangleq J^K(H) \quad . \tag{3.7}$$

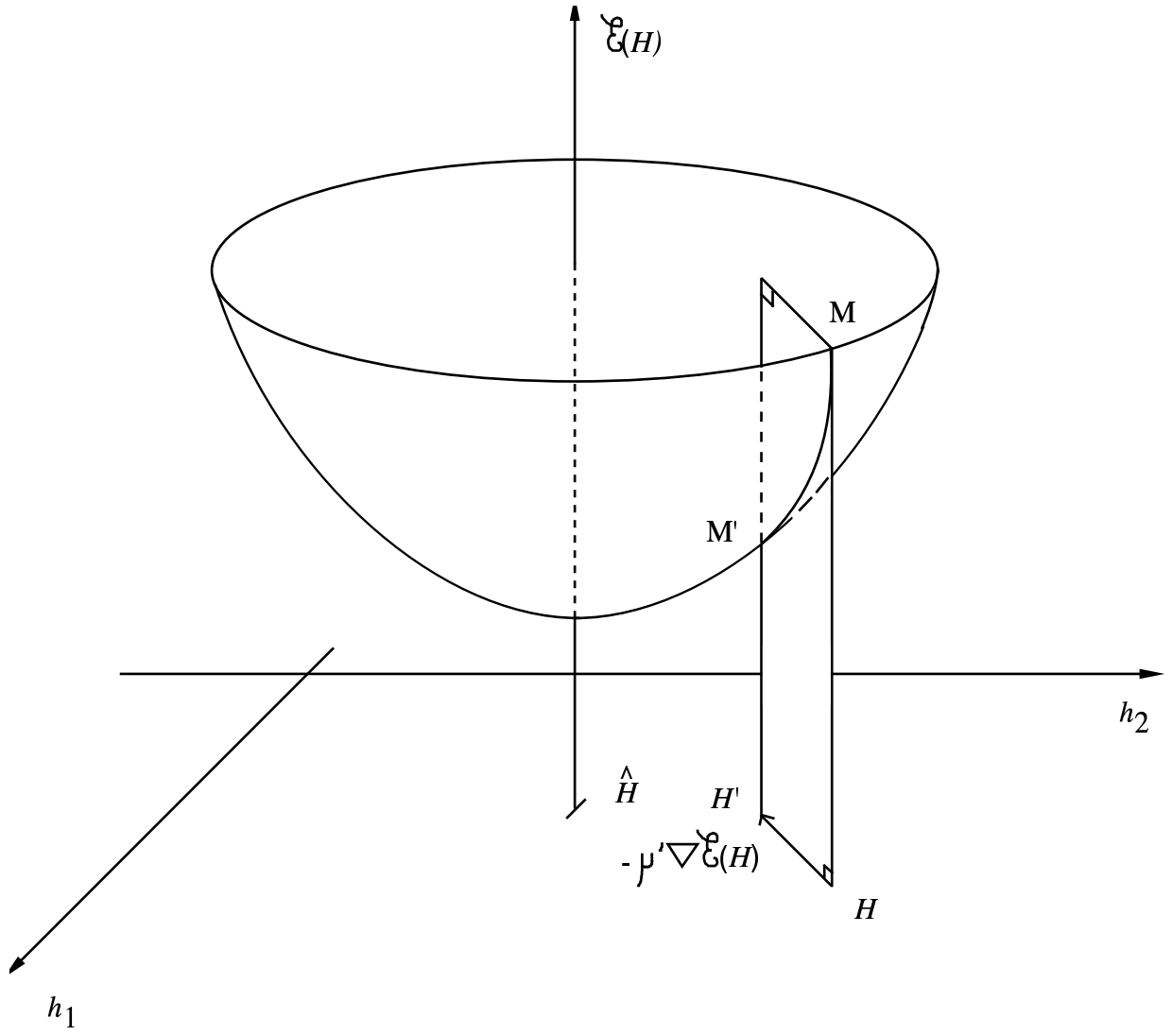

Fig. 3. The steepest descent method

Its gradient is given by (2.8) with $k = K$. The associated iterative algorithm is thus

$$H^K(p) = H^K(p-1) + \frac{\mu}{K}\sum_{k=1}^{K} e_H(k)\,\nabla_H y_H(k) \Big|_{H = H^K(p-1)} . \qquad (3.8)$$

Subject to initialization in a suitable neighbourhood, it is intended to reach the overall LS optimum parameter

$$\hat{H} = \tilde{H}(K) . \qquad (3.9)$$

In (3.8) the constant $\mu = 2\,\mu'$ is called **the step-size.** It is positive. The upper index K indicates the (limited) total number of examples. As for the iteration index p, it should tend to infinity, in principle.

Equation (3.8) shows the necessity of calculating at each iteration the outputs which correspond to all the K examples :

$$y_H(k) \Big|_{H = H^K(p-1)} \triangleq F(H^K\ (p-1), X(k))\ ,\ k = 1, ..., K \qquad (3.10)$$

and their associated gradients. In other words, **the iterative algorithm (3.8) is not recursive**. We call it the "block LS gradient algorithm", abbreviated as (BLS). Yet it is simpler than the direct procedure in two phases detailed in subsection II.1 because the $2K$ quantities $y_H(k)$, $\nabla_H y_H(k)$, $k = 1, ..., K$ do not have to be calculated for any possible candidate vector H, but only for the sequence $H^K(p-1)$ of parameters. Note that the algorithm (3.8) remains unchanged if the K examples are presented to the system in a different order : examples need not be ordered.

III.4. The recursive block-LS gradient algorithms

The block iterative algorithm (3.8) remains a heavy procedure by its computional aspects. The first level of simplification is to design a gradient algorithm that is recursive. The iterations (p) run at the same time as the examples (k) :

$$p = k\ . \qquad (3.11)$$

Because the iterations are naturally ordered, the one-to-one correspondence between the examples and the iterations will then put an order in the collection of examples. In this way, the (NN) training context becomes very close to the (TF) adaptation context — where the input vectors $X(k)$ are naturally labelled by the time k and run accordingly. The last difference between the two contexts lies in the shifting property of the vector $X(k)$ that is valid in the (TF) context and not in the (NN) context.

But this property is not relevant to the steepest descent context we are investigating in this Section. Thus, in the rest of Section III, there will be no distinction between (NN) and (TF).

In brief, in order to simplify the block-LS gradient algorithm, we must treat the supervised (NN) training problem like a (TF) adaptation problem where the examples k are ordered. The iterations are now labelled by k, a new one being performed each time a new example k arrives. **Under the generic expression "step k", we thus designate indifferently "time k", "example k", "iteration k".** In (NN) with a finite training set, the total number of steps is limited by

$$k \leq K' = MK \quad , \tag{3.12}$$

where M is the number of presentations of the training set.

The recursive algorithm associated with the block algorithm (3.8) [10] is obtained when the information available at step k is the sequence $(X(n), d(n))$ with $n \leq k$, according to

$$H(k) = H(k-1) + \frac{\mu}{k} \sum_{n=1}^{k} e_H(n) \nabla_H y_H(n) \Big|_{H = H(k-1)} \tag{3.13}$$

where we have omitted the superscript K which is irrelevant provided (3.12) is satisfied. In the following, (3.13) is called the "causal block-LS gradient algorithm" and it is abbreviated (CLS). Compared with (3.8) it has indeed the causality property that future examples ($n > k$) are not used in the increment of iteration k.

It is important to emphasize that in (NN) training, the ordering of examples is arbitrary, for instance for a classification task. Then, it must be proved that convergence properties of recursive algorithms in (NN) are unaffected when the ordering is switched. This result is related to the stationarity of the (random) sequence $(X(k), d(k))$.

III.5. Interpretation of the causal block-LS gradient algorithm

In this subsection, it is shown that the causal block-LS gradient algorithm is an intelligent attempt to use a recursive algorithm in order to reach the minimum of the running LS cost function

$$\mathcal{C}(H) = J^k(H) \tag{3.14}$$

at each step k of the algorithm. The corresponding minimum

$$\hat{H} = \tilde{H}(k) \tag{3.15}$$

could be reached if we had an iterative gradient sub-algorithm implemented between each step k and the next step $(k + 1)$ in order to reach the minimum of (3.14). But now the bowl in which the steepest descent is performed changes at each step k. Similarly, the bottom of the bowl is changing with k. We denote $H_p(k)$ the corresponding gradient subsequence with k being fixed.

Obviously, it is impossible to perform an infinite number of iterations p at each step k. A first level of simplification is to assume that a finite number P_k of iterations is performed at each step k. When all the steps k are chained, which results in a compound iterative algorithm. The initial value $H_0(k)$ at step k must be the final value $H_{P_{k-1}}(k-1)$ of the previous step $k-1$. The estimates $H(k)$ of $\tilde{H}(k)$ are the ends of each iterative sub-algorithm. Hence the compound algorithm (which depends on the a priori choice of the sequence of lengths P_k) :

$$H_0(k) = H_{P_{k-1}}(k-1) \triangleq H(k-1), \tag{3.16}$$

$$H_p(k) = H_{p-1}(k) - \mu' \nabla J^k(H) \Big|_{H = H_{p-1}(k)} . \tag{3.17}$$

One can choose the same length P for the sub-algorithms. It is also possible to allow a greater length P_k to initial steps (k small) in order to speed up initial convergence of the compound algorithm.

A second level of simplification is to implement a single iteration per sub-algorithm ($P_k \equiv 1$). Then it is easy to check that both formulae (3.16), (3.17) combine into

$$H(k) = H(k-1) - \mu' \nabla J^k(H) \Big|_{H = H(k-1)} \quad . \qquad (3.18)$$

With the value (2.8) of the gradient, this formula is identical to the recursive algorithm (3.13). In other words, the causal block-LS gradient algorithm can be viewed as the chaining over k of (short) iterative sub-algorithms, each one intended to find the running LS parameter $\tilde{H}(k)$. The corresponding procedure is depicted in fig. 4 in the case of a scalar parameter H ($N = 1$). The dark broken line jumping from one curve $J^k(H)$ to the next curve $J^{k+1}(H)$ illustrates the way in which the sequence $H(k)$ attempts to track the changing minimum $\tilde{H}(k)$.

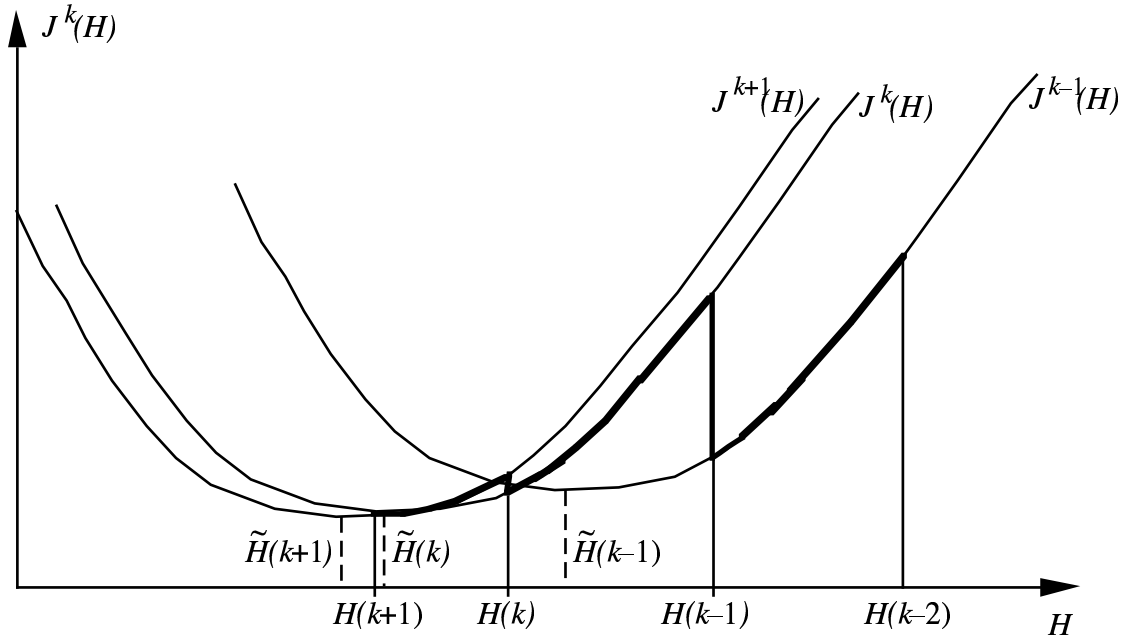

Fig. 4. Interpretation of the causal block-LS gradient algorithm (3.13)

The simplified algorithm (3.18) (or equivalently (3.13)) is not yet the familiar gradient algorithm because the cost function $J^k(H)$ changes at each iteration (step) k. However, the success of the (second level of) simplification relies on the fact that one iteration per step is enough to cover the (time) variation $\tilde{H}(k-1) \rightarrow \tilde{H}(k)$. This means that

(i) $\tilde{H}(k)$ varies slowly

(ii) $H(k)$ is close to $\tilde{H}(k)$.

In the stationary case where $J^k(H)$ has a limit for k large (see § II.3), property (i) is valid. Then property (ii) is valid when the algorithm (3.13) has reached a steady-state.

III.6. The sliding window LS gradient algorithm

Recursive implementation of the above detailed CLS gradient algorithm is possible. But it is awkward, especially in the (TF) context where k grows unbounded. Indeed, at step k where the system is in state $H(k-1)$, implementation of (3.13) involves the computation of all the quantities

$$y_H(n)\Big|_{H=H(k-1)} = F(H(k-1), X(n)), \quad n \le k \ , \tag{3.19}$$

together with their gradients. When k increases, this requires an indefinitely growing memory in order to store all the past inputs $X(1)$, $X(2\square)$,.... $X(n)$ and past references $d(1)$, $d(2)$, ... $d(n)$. In practice, this is feasible neither in (TF) adaptation nor in (NN) training when there is a large number K of examples.

Obviously a further simplification is to truncate the memory of the algorithm. If M is that memory, the increment in (3.13) will involve only the quantities

$$y_H(n)\Big|_{H=H(k-1)} \ , \quad k-M+1 \le n \le k \tag{3.20}$$

and their associated gradients. The calculation requires only the knowledge of the arguments

$$H(k-1), X(k), \ldots, X(k-M+1), d(k), \ldots, d(k-M+1) \ . \tag{3.21}$$

Hence the algorithm (3.13) is replaced by

$$H(k) = H(k-1) + \frac{\mu}{M} \sum_{n=k-M+1}^{k} e_H(n) \nabla_H y_H(n) \Big|_{H = H(k-1)} \quad (3.22)$$

In the following, (3.22) is called the "sliding window LS gradient algorithm" and it is abbreviated (SLS). When k is replaced by $(k+1)$, the time window used in the variables (3.21) will slide of one step ahead. This sliding effect is familiar in the (TF) context.

Note that algorithm (3.22) is intended to minimize the windowed LS criterion (2.7) in the same way and through the same approximations as algorithms (3.13) do for the basic LS criterion (2.3). It is always a feasible algorithm but its complexity remains high when M is large.

III.7. The stochastic LMS gradient algorithm

The final level of simplification is to set $M = 1$ in the recursive windowed algorithm (3.22), which yields

$$H(k) = H(k-1) + \mu \, e_H(k) \nabla_H y_H(k) \Big|_{H = H(k-1)}, \quad (3.23)$$

where μ is the (positive) step-size. This algorithm is widely known under the denomination of "LMS algorithm" [6]. In the following, we prefer the more precise name "stochastic LMS gradient algorithm" abbreviated in (STLMS). As appears below, (3.23) is a stochastic version of the iterative gradient algorithm associated with the least mean square criterion $J(H)$ namely

$$H(p) = H(p-1) - \mu' \nabla_H J(H) \Big|_{H = H(p-1)}. \quad (3.24)$$

According to (2.9), (2.11), this iterative algorithm reads

$$H(p) = H(p-1) + \mu \, E(e_H(k) \nabla_H y_H(k)) \Big|_{H = H(p-1)}. \quad (3.25)$$

Note that the k in the expectation is a dummy index.

The algorithm (3.23) generates a random vector $H(k)$ because the input $X(k)$ is random, and so is the output $y_H(k)$, as well as the associated quantities $e_H(k)$, $\nabla_H y_H(k)$ included in the increment. On the other hand, the increment of (3.25) is just the expectation of the former increment. This expectation depends on the (higher order) statistics of the sequences $X(k)$, $d(k)$, but not on their sample values $X(k)$, $d(k)$. Thus, **algorithm (3.25) generates a deterministic vector $H(p)$.** Apart from that difference, (3.23) is very close to (3.25), and it is easy to understand that the two algorithms will essentially reach the same limit, as evidenced by the following heuristic arguments. Let us iterate (3.23) P times :

$$H(k) = H(k-P) + \mu \sum_{n=k-P+1}^{k} e_H(n) \nabla_H y_H(n) \Big|_{H = H(n-1)} . \quad (3.26)$$

If μ is small enough, $H(k)$ is slowly varying, so that the parameter $H = H(n-1)$ does not change significantly when n scans the summation which is in the increment. Thus,

$$H(k) \approx H(k-P) + \mu P \Delta(H(k-P)), \quad (3.27)$$

where the increment

$$\Delta(H(k-P)) = \frac{1}{P} \sum_{n=k-P+1}^{k} e_H(n) \nabla_H y_H(n) \Big|_{H = H(k-P)} \quad (3.28)$$

uses the fixed parameter value $H = H(k-P)$. Ergodism of the sequence $(X(k), d(k))$ implies that

$$\Delta(H(k-P)) \approx E(e_H(n) \nabla_H y_H(n)) \Big|_{H = H(k-P)} \quad (3.29)$$

(provided P is large enough). The combination of (3.27) and (3.28) yields

$$H(k) \approx H(k-P) + \mu P E(e_H(n) \nabla_H y_H(n)) \Big|_{H = H(k-P)} . \quad (3.30)$$

This last algorithm is iterative and deterministic. It is identical to (3.25) except for a relabelling of the steps and a multiplication of the step size by P. We conclude that (3.23) and (3.25) have the same limit (up to the approximations made in (3.27) and (3.29)).

Provided the initial value is suitable, the iterative algorithm (3.25), or equivalently (3.24), reaches the minimum $\tilde{H}$ of the MS cost for which it was designed. As for the stochastic LMS gradient (3.23), it also converges towards $\tilde{H}$, but with some random fluctuations due to the randomness of $H(k)$.

The statistics of the fluctuations of $H(k)$ in the vicinity of $\tilde{H}$ when k is large, have been mathematically investigated in the (TF) context, for the case of transversal filters. It has been proved [6] [17] that

$$E(|H(k) - \tilde{H}|^2) \rightarrow \varepsilon(\mu\square) \quad , \quad k \rightarrow \infty \tag{3.31}$$

where the function $\varepsilon(\mu)$ is on the order of μ in the sense that

$$\varepsilon(\mu) \leq \alpha\,\mu \qquad \text{if} \quad \mu \leq \mu_0 \; . \tag{3.32}$$

In other cases, e.g. for IIR adaptive filters, results do not yet form such a well established theory and are sometimes heuristic. However, it is our conjecture that under suitable regularity assumptions, the results (3.31), (3.32) are extremely general in both contexts of (TF) adaptation and (NN) training.

III.8. Discussion/Comparison of algorithms

We have presented four major iterative algorithms

(BLS) : the block LS gradient algorithm (3.8)

(CLS) : the causal block LS gradient algorithm (3.13)

(SLS) : the sliding window LS gradient algorithm (3.22)

(STLMS) : the stochastic LMS gradient algorithm (3.23)

They are all intended to optimize iteratively the system $\mathbf{h}$ ((NN) or (TF)), i.e., to generate the vector H of parameters for which the outputs $y_H(k)$ of $\mathbf{h}$ are closest to the corresponding references $d(k)$. More precisely, all of them fulfill one or several among the objectives (2.20), (2.21), (3.1), (3.2), (3.3) or (3.4). Therefore, if stationarity and ergodism are assumed for the sequence $(X(k), d(k))$, the four algorithms are essentially equivalent in the sense that they asymptotically provide a good estimate of $\tilde{H}$ (provided initialization is good, in case of multiple minima) . But the accuracy of the result decreases as we switch the algorithm (BLS) into (CLS) then into (SLS) and finally into (STLMS). As a counterpart, implementation involves less and less computations. A more detailed comparison between these algorithms is given below.

(BLS) : for this iterative (non recursive) algorithm, a single finite but large collection of K items $(X(k), d(k))$ is repeatedly processed, say P times. If P is large enough, the initial objective (2.20) is fulfilled according to

$$\left| \square H^K(P) - \tilde{H}(K) \right| < \varepsilon \ , \qquad P \geq P_0 \ . \tag{3.33}$$

Because K is large, it follows from (2.14) that the objective (3.2) is fulfilled as well :

$$|\Box H^K(P) - \tilde{H}| < \varepsilon \ , \qquad P \geq P_0 \ . \tag{3.34}$$

The major drawback of (BLS) is to require a number of memory proportional to K, that is very high.

(CLS) : this algorithm is recursive, using one additional item ($X(k)$, $d(k)$) at each step. The interpretation given in subsection III.5 shows that the asymptotic objective

$$H(k) \to \tilde{H} \tag{3.4}$$

is fulfilled. When $\tilde{\Box} H(k)$ is slowly varying, this algorithm also reaches the objective

$$|\Box H(k) - \tilde{H}(k)| \leq \varepsilon \qquad \text{for } k \geq k_0. \tag{2.21}$$

This means that for intermediate indices k, when the LS optimum parameter $\tilde{H}(k)$ is no longer in its fast initial phase, but has not yet reached its asymptotic value $\tilde{H}$, the algorithm produces a vector $H(k)$ close to $\tilde{H}(k)$. This property is stronger than an asymptotic property.

The major drawback with (CLS) is the growing size of the required memory. Algorithms with infinite memory cannot deal with non strictly stationary sequences.

(STLMS) : this algorithm is recursive, using only the new data ($X(k)$, $d(k)$) at each step k. This is an important advantage, limiting the size of memories. As a result of (3.31), it achieves the objective

$$|H(k) - \tilde{H}| \leq \eta \ , \qquad \Box k \geq k_0 \ , \tag{3.3)'}$$

where η is small. It is essentially proportional to $\sqrt{\mu}$. This achievement is very close to (3.3).

The drawback with (STLMS) is that η cannot be chosen arbitrarily small since $H(k)$ is randomly fluctuating with a non zero asymptotic standard deviation. Thus, it does not achieve the objective (3.4).

(SLS) : this algorithm is intermediate between (CLS) and (STLMS). Although the objective (3.4) is not yet reached, the result (3.3)' is improved according to [9]

$$| H(k) - \tilde{H} \quad | \leq \eta \quad / \sqrt{M} \qquad , \quad k \quad \geq k_0 \quad : \qquad (3.3)''$$

the memory included in a window of length M improves the accuracy by a factor $\sqrt{M}$. Unfortunately, this improvement over (STLMS) has a counterpart in the increased memory size and increased computational amount.

Undoubtedly, the most widely used among these gradient algorithms is the (STLMS) one, because of its lower computational cost. This is true in the (NN) context [18] although the block-LS algorithm (3.8) is also in favour. This is equally true in (TF) although the sliding window LS one (3.22) is also implemented.

It is our feeling that in a number of applications — either in (TF) or in (NN) — the best possible benefit has not yet been obtained from the large available variety of gradient algorithms. This is left open for future research. In particular, the compound algorithm (3.16), (3.17) with a varying number P_k of iterations per step has not yet been sufficiently investigated under the aspect of its tracking capability when the LS optimum $\tilde{H}(k)$ has fast time variations.

In the rest of this paper, we restrict our investigations to the standard LMS algorithm (STLMS), given by eq. (3.23). Clearly, the first problem will be to evaluate the gradient

$$g(H, X(k)) \triangleq \nabla_H y_H(k) \quad . \qquad (3.35)$$

It depends heavily on the structure of system $\mathbf{h}$, whether cascaded, layered, with a feedback loop, and on the presence of nonlinear elements. Taking certain important specific examples, we shall calculate this gradient in further sections of this paper and point out the similarities between (TF) adaptation and (NN) training in these cases.

The second question is the theoretical investigation of convergence for the (STLMS) algorithm associated with each example. In each case, the conjecture that (3.31), (3.32) hold is very difficult to

prove. Moreover, this result remains subject to a good initial value $H(0)$ for the algorithm, because of the possible local (non global) minima of the MS cost $J(H)$. Convergence is also subject to the fact that the step-size μ is small. This condition is expressed by the upperbound μ_0 which appears in (3.32). In practice, there is however no explicit expression for μ_0. Thus, the meaning of the expression "μ is small" is quite vague. Usually this question is solved by trial and error, and the step-size is kept "very small" in a cautious manner.

IV. THE STOCHSTIC LMS ALGORITHM FOR ADAPTIVE FILTERING

Throughout this section, the index k is time and H is the parameter vector of a time-filter $\mathbf{h}$. This means that the output samples $y(k)$ are linear and time-invariant versus the input samples $x(k)$. When the parameter $H = H(k)$ obeys a recursive algorithm controlled by the output error $e_H(k)\Big|_{H = H(k-1)}$, the problem belongs to the generic category of "**adaptive filtering**". The entire Section IV is thus devoted to adaptive filtering, taking certain structures for the linear filter $\mathbf{h}$ and taking the particular STLMS algorithm to control the filter.

IV.1. The transversal filter

The simplest case corresponds to **transversal filtering,** depicted in Fig. 2 of Section II.4, where the output of system $\mathbf{h}$ is

$$y_H(k) = X(k)^T H \quad . \tag{4.1}$$

the vector $X(k)$ being a sliding window as in (1.7). In this case, as mentioned in subsection II.4, the running LS optimum parameter $\tilde{H}(k)$ can be exactly evaluated by the fast recursive algorithm which has 8 N multiplications per step. Unfortunately, this algorithm accumulates the numerical inaccuracies as the steps run indefinitely and it grows unbounded. The standard (non fast) recursive algorithms (2.17), (2.18) has no such drawback but involves more computations ($\mathbf{o}(N^2)$).

The LMS algorithm is thus a useful alternative to the LS approach. In a stationary and ergodic context, it will asymptotically provide the LMS optimum parameter $\tilde{H}$ minimizing the MS cost in (2.9). According to (4.1), this cost is

$$J(H) = P_d - 2\, H^{\mathrm{T}} R_{Xd} + H^{\mathrm{T}} R H \tag{4.2}$$

where P_d is the power of the reference signal and

$$R_{Xd} = E(X(k)\, d(k)) \tag{4.3}$$

is the input/reference cross-correlation vector and

$$R = E(X(k)\, X(k)^{\mathrm{T}}) \tag{4.4}$$

is the input covariance matrix. The cost (4.2) is quadratic w.r.t. H. This is an important feature because it precludes the possibility of local minima for $J(H)$: all the minima are global in the sense that they reach the minimum minimorum. The corresponding (iterative) gradient algorithm is thus insensitive to initial conditions. The minimum $\tilde{H}$ satisfies

$$R\, \tilde{H} = R_{Xd} \tag{4.5}$$

and it is unique when R is invertible. According to (4.1), the corresponding stochastic gradient is

$$\nabla_H y_H(k) = X(k) \quad . \tag{4.6}$$

Note that it is independent of the parameter state H. The STLMS algorithm is written with the very classical updating formula of adaptive filtering

$$H(k) = H(k-1) + \mu\, e_H(k)\, X(k) \quad \Big| \; H = H(k-1) \; . \tag{4.7}$$

The performance of this algorithm is expressed through formulae (3.31), (3.32) of the above subsection III.7. They show that (4.7) results in an estimation of $\tilde{H}$ having an arbitrary preassigned accuracy provided the step-size μ is small enough. We call this property "quasi-mean square convergence". However, this is at the price of a reduced convergence speed. Hence the well-known trade-off between speed and accurracy.

In a number of cases, the result (3.32) can be refined. Specifically, the residual function $\varepsilon(\mu)$ and the upperbound μ_0 on the step-size can be evaluated when $(X(k), d(k))$ is a zero-mean Gaussian variable [7]. Another case is when the random amplitude $|X(k)|$ is approximately constant. Then it is found that [19]

$$\varepsilon(\mu) = \frac{\mu N P_x}{2 - \mu N P_x} \; ; \; \mu_0 = \frac{2}{N P_x} \quad , \tag{4.8}$$

where

$$P_x = E(|x(k)|^2) \quad . \tag{4.9}$$

In Section III, the importance of the computational complexity associated to a recursive algorithm has been emphasized. Taking only the multiplications into account, it follows from (4.1) and (4.7) that for the adaptive FIR filter this complexity is reduced. The number of multiplications per step is

$$C(FIRF) = 2N + 1. \tag{4.10}$$

Let us now consider the cascaded implementation of such transversal filters.

IV.2. The cascade of transversal filters

When the number N of parameters of a transversal filter is high, it may be more convenient to implement the filtering in an indirect (but equivalent) cascaded mode as illustrated in fig. 5. Examples

of adaptive cascaded filters are found in Signal Processing for such applications as communications [20] and noise cancellation [21].

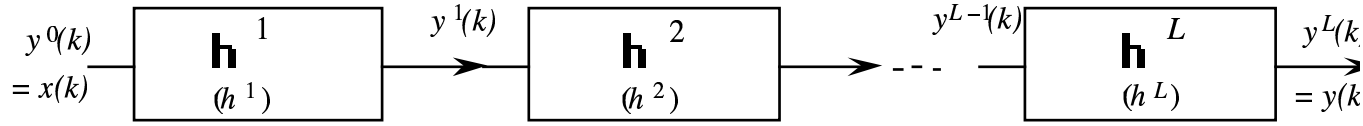

Fig. 5: A cascade of transversal filters

Let L be the number of cascaded transversal (sub) filters $\mathbf{h}^l$, and M_l be the number of parameters of the l-th cell. Then the global system

$$\mathbf{h} = \mathbf{h}^L \circ \ldots \circ \mathbf{h}^l \circ \ldots \circ \mathbf{h}^2 \circ \mathbf{h}^1 \tag{4.11}$$

is a linear filter in the sense that the final output

$$y(k) = y^L(k) \tag{4.12}$$

is linear and time invariant w.r.t. the input signal

$$x(k) = y^0(k) \quad . \tag{4.13}$$

Direct (transversal) structures and cascaded structures are known to be equivalent in the sense that the final output $y(k)$ can be written in the direct form (4.1)

$$y(k) = X(k)^{\mathrm{T}} R \tag{4.14}$$

with a parameter vector

$$R = r\,(H^1, H^2, \ldots, H^L) \ , \tag{4.15}$$

that depends on the (sub) filters parameters. Conversely a transversal filter $\mathbf{h}$ can be split into a cascade of transversal (sub)-filters. The cascaded form has the implementation advantage of modularity. Moreover for the sake of adaptivity, it is easier to control several small (sub) filters $\mathbf{h}^l$ than a single large filter $\mathbf{h}$. This is particularly important when stability of the inverse filter $\mathbf{h}^{-1}$ is required.

The cost of these improvements is that the output $y(k)$ is no longer linear w.r.t. the set

$$H = (H^1, H^2, \ldots, H^L) \tag{4.16}$$

of all the parameters. The function r in (4.15) involves cross-products. Therefore the RLS and fast RLS approaches to derive the running LS parameters are no longer valid. Hence a renewed interest in the LMS gradient approach. Here the gradient is taken w.r.t. the vector H in (4.16). For the l-th filter let

$$H^l = [h_0^l \ , h_1^l \ , \ldots , h_{M_l - 1}^l]^{\mathrm{T}} \tag{4.17}$$

be the impulse response and

$$Y^{l-1}(k) = \ [y^{l-1}(k), y^{l-1}(k-1), \ldots , y^{l-1}(k - M_l + 1) \ ]^{\mathrm{T}} \ , \tag{4.18}$$

be the input vector, where M_l denotes the number of parameters of filter # l. Applying formula (4.1) L times, the final output is

$$y(k) = \sum_{i_L=0}^{M_L-1} \ldots \sum_{i_l=0}^{M_l-1} \ldots \sum_{i_1=0}^{M_1-1} h_{i_L}^L \ \ldots \ h_{i_l}^l \ \ldots h_{i_1}^1 \ \ x(k - i_1 - \ldots \ i_l \ - \ldots - \ i_L) \ . \tag{4.19}$$

Given the reference sequence $d(k)$ for the final output $y(k)$, implementation of the stochastic LMS gradient according to (3.23), (3.35) requires evaluation of the L sub-gradients

$$U^l(k) \triangleq \ \nabla_{H^l} y(k) \tag{4.20}$$

(note that this quantity depends on all the parameters in H, and not only on H^l). It follows from (4.19) that the i_l-th coordinate of (4.20) is

$$\frac{\partial y(k)}{\partial h^l_{i_l}} = \sum_{i_L=0}^{M_L-1} \cdots \sum_{i_{l+1}=0}^{M_{l+1}-1} \sum_{i_{l-1}=0}^{M_{l-1}-1} \cdots \sum_{i_1=0}^{M_1-1} h^L_{i_L} \cdots h^{l+1}_{i_{l+1}}$$

$$\times\ h^{l-1}_{i_{l-1}} \cdots h^1_{i_1}\ x(k - i_L - \ldots - i_l - \ldots - i_1) \quad . \tag{4.21}$$

Comparing (4.21) with (4.19), it appears that the sub-gradient w.r.t. the parameters of the l-th filter is

$$U^l(k) = [u^l(k), \ldots, u^l(k - i_l), \ldots, u^l(k - M_l + 1)]^T \quad . \tag{4.22a}$$

It is built up with the successive outputs

$$u^l(k - i) = \mathbf{g}^l(x(k - i)) \tag{4.22b}$$

of the filter

$$\mathbf{g}^l \triangleq \mathbf{h}^L \circ \ldots \circ \mathbf{h}^{l+1} \circ \mathbf{h}^{l-1} \circ \ldots \circ \mathbf{h}^1 , \tag{4.23}$$

when the input is $x(k)$. This filter involves all the cascaded sub-filters except $\mathbf{h}^l$, i.e., the one w.r.t. which the sub-gradient is calculated. Denoting successive samples by a capital letter as in (1.7), the stochastic LMS algorithm (3.23) reads

$$H^l(k) = H^l(k - 1) + \mu\, e(k)\, U^l(k) \Big|_{\mathbf{h} = \mathbf{h}(k-1)} . \tag{4.24}$$

The corresponding adaptive (TF) is depicted in fig.6. In the further subsection V.2. it is shown that this algorithm can be viewed as the stochastic LMS gradient algorithm for a multilayer network of linear neurons (using the standard back-propagation approach).

Although the MS cost $J(H)$ is non quadratic w.r.t. H (because $y(k)$ is non linear w.r.t. H) it can be shown that $J(H)$ has no local minimum [22] : all minima are global. They are all equivalent in the sense that they correspond to identical sets of values $\tilde{H}^1$, $\tilde{H}^2$, ... , $\tilde{H}^L$ except for an arbitrary permutation. This is because in a cascade of filters, the ordering does not influence the value of the output. As a result, the gradient algorithm (4.24) has a certain sensitivity w.r.t. initial conditions : all the vectors $H^1(0)$, $H^2(0)$, ...□,$H^L(0)$ must be different, otherwise the algorithms cannot assign a specific objective $\tilde{H}^L$ to a specific cell $H^L(k)$. Apart from this (minor) drawback, adaptive cascaded (TF) raise no major difficulty.

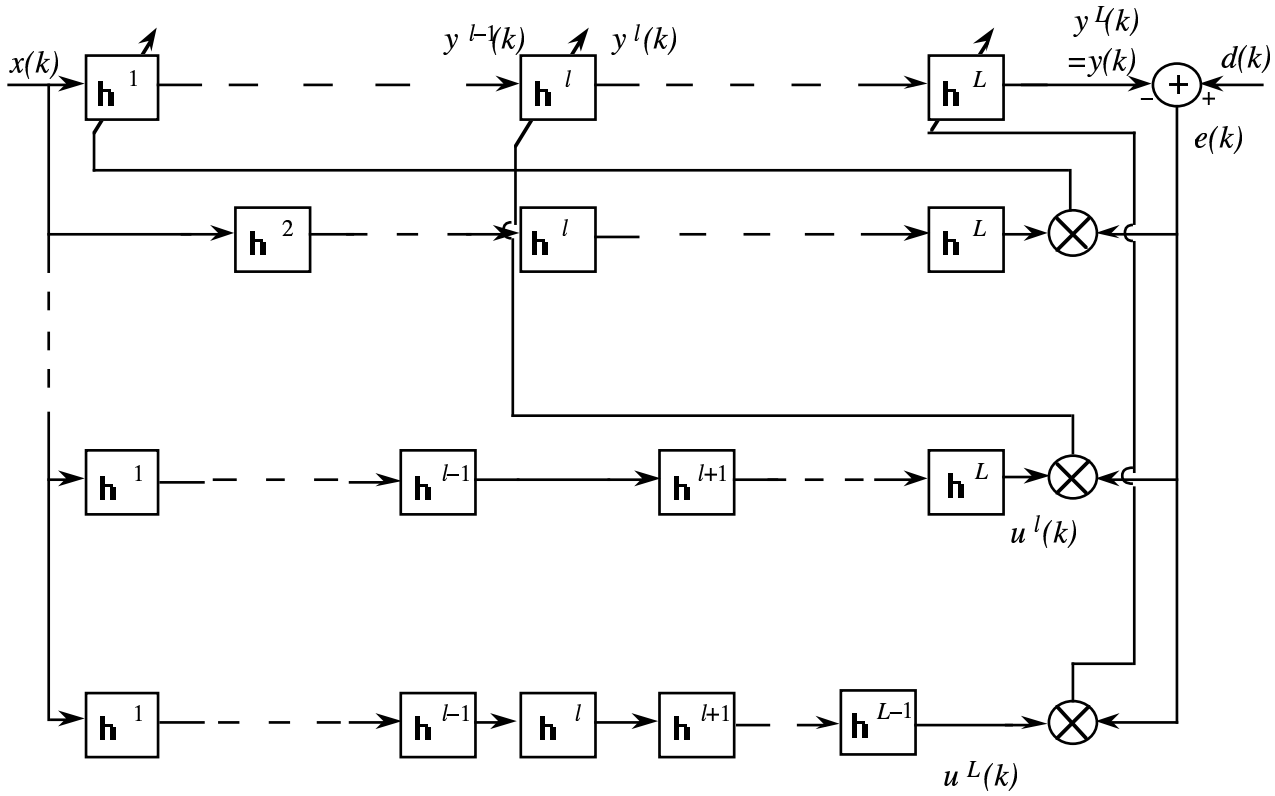

Fig. 6. Adaptation of a cascade of filters

Computational complexity. We denote

$$N = \sum_{l=1}^{L} M_l \qquad (4.25)$$

An evaluation of the r function in (4.15) shows that the equivalent filter $\mathbf{h}$ has not N parameters but $N' = N - L + 1$ parameters. Implementing the filter H^l requires M_l multiplications per step and per

output. Theoretically, it is required to evaluate $(M_l + 1)$ values of y^l to feed in the next filter H^{l+1}. However, assuming that $\mathbf{h}(k)$ changes slowly, it is allowed to take the approximation

$$y^l(k-j)\,\Big|_{\mathbf{h}=\mathbf{h}(k-1)} = y^l(k-j)\,\Big|_{\mathbf{h}=\mathbf{h}(k-j-1)} \tag{4.26}$$

which requires at step k only the computation of the new value $y^l(k)\,\Big|_{\mathbf{h}=\mathbf{h}(k-1)}$. The complexity associated with the filtering part of the algorithm is thus

$$C^f = N. \tag{4.27}$$

On the other hand, the additional complexity due to adaptation can be reduced by noting that the filter present in the l-th channel of adaptation is

$$\mathbf{g}^l = (\mathbf{h}^l)^{-1} \circ \mathbf{h} \tag{4.28}$$

(this follows from (4.11) and (4.23)). Hence the adaptation signal $u^l(k)$ obeys the recursion

$$\sum_{i=0}^{M_l-1} h_i^l \; u^l(k-i) = y(k) \quad . \tag{4.29}$$

Its calculation involves M_l multiplications. Using for $u^l(k-j)$ the same approximation as in (4.26), the vector $U^l(k)$ includes a sliding window. Thus, at each step k, only the new sample $u^l(k)$ must be calculated. Therefore, to implement the adaptation (4.24) in channel # l, the required number of multiplications is

$$C_l^a = 2M_l + 1 \quad . \tag{4.30}$$

In view of (4.27) and (4.30), the total number of multiplications per step is thus

$$\mathrm{C(CASCADE)} = 3N + 1, \tag{4.31}$$

$$C(\text{CASCADE}) = 3\,N' + 3\,L - 2\ . \qquad (4.32)$$

This complexity is slightly higher than the one of the direct FIR filter (see (4.10)).

In the next forthcoming example of (TF), the output $y(k)$ is again nonlinear w.r.t. the set H of parameters.

IV.3. The recursive filters

IV.3.1. The context

When the number of parameters of a transversal filter is too high, another convenient method is to replace the non recursive equation (4.1) by the recursive input-output filtering equation (the index H appearing in (4.1) is omitted in the following)

$$y(0), y(1), \ldots, y(m-1) \ \text{ fixed}, \qquad (4.33a)$$

$$y(k) = \sum_{j=1}^{m} b_j\, y(k-j) + \sum_{i=0}^{I-1} a_i\, x(k-i),\ k \geq m\ . \qquad (4.33b)$$

The corresponding filter $\mathbf{h}$ is depicted in fig. 7 with the help of two transversal filters A and B, both obeying the structure depicted in fig. 2. The filter B is included in a feedback loop and, for obvious causality reason, it presents a time delay of one step at the input (the box z^{-1} in fig. 7 denotes the delay).

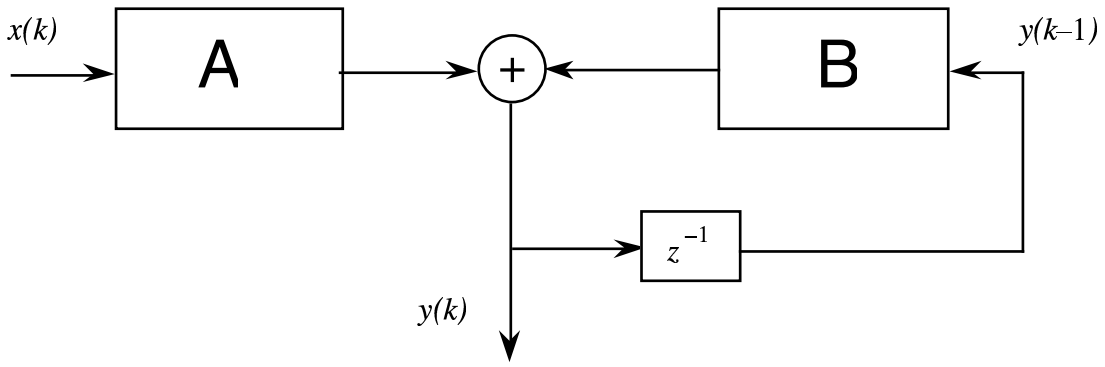

Fig. 7. A recursive time filter ☐

Examples of adaptive recursive filters are the control [23] of an unknown process, the prediction of a random signal [24], and the transmission of digital speech at a reduced bit rate (see fig.0) [25].

Equations (4.33) are equivalent to the transversal form (4.14) but with an infinite number of parameters in R. Hence the alternate name of infinite impulse response (IIR) filter. The advantage of (4.33) is the saving of a great many parameters. Moreover, when the input $x(k)$ is a random independent identically distributed (i.i.d.) sequence, formulae (4.33) generate the so-called very well-known ARMA model where the signal $y(k)$ is a correlated sequence.

The cost for having saved a large number of parameters is that $y(k)$ is no longer w.r.t. the set

$$H = (A^T, B^T)^T \tag{4.34}$$

of all the parameters a_j and b_i involved in (4.33) where

$$A^T \triangleq (a_0, a_1, \ldots, a_{I-1}) \quad ; \quad B^T \triangleq (b_1, b_2, \ldots, b_m) \quad . \tag{4.35}$$

Writing (4.33b) under the equivalent form

$$y(k) = A^T \, X(k) + B^T \, Y(k-1) \ , \tag{4.36}$$

it is indeed evident that changing B affects the vector $Y(k-1)$ in the RHS of (4.36) whose second term becomes non linear in B. Again a straightforward application of the RLS and fast RLS approaches is not possible. Hence the interest for the LMS gradient approach in order to optimize the filter **h**.

IV.3.2. The exact stochastic LMS gradient algorithm

In order to compute the gradient necessary for the recursive adaptation, let us introduce the auxiliary signals

$$u_i(k) \triangleq \frac{\partial y(k)}{\partial a_i} \quad , \quad i = 0, 1, \ldots, I-1 \quad , \tag{4.37a}$$

$$v_j(k) \triangleq \frac{\partial y(k)}{\partial b_j} \quad , \quad j = 1, 2, \ldots, m \quad . \tag{4.37b}$$

Differentiating (4.33) yields the two sets of equations

$$u_i(0) = \ldots = u_i(m-1) = 0 \quad , \quad i = 0, \ldots, I-1 \ , \tag{4.38a}$$

$$v_j(0) = \ldots = v_j(m-1) = 0 \quad , \quad j = 1, \ldots, m \ , \tag{4.39a}$$

$$u_i(k) = \sum_{n=1}^{m} b_n \, u_i(k-n) + x(k-i) \quad , k \geq m \ , \tag{4.38b}$$

$$v_j(k) = \sum_{n=1}^{m} b_n \, v_j(k-n) + y(k-j) \quad , k \geq m \ . \tag{4.39b}$$

These equations can be simplified into

$$u_i(k) = u_0(k-i) \triangleq u(k-i) \ , i = 0, \ldots, I-1 \quad , \quad k \geq m \ , \tag{4.40}$$

$$v_j(k) = v_1(k-j+1) \triangleq v(k-j)\ , j = 1, \ldots, m\ , k \geq m\ , \tag{4.41}$$

where the signals $u(k)$ and $v(k)$ are respectively derived from the input sequence $x(k)$ and from the output sequence $y(k)$ through a purely recursive filter associated with the same recursive part B as the initial filter h namely

$$u(k) = \sum_{j=1}^{m} b_j\, u(k-j) + x(k)\ , \quad k \geq m\ , \tag{4.42b}$$

$$v(k) = \sum_{j=1}^{m} b_j\, v(k-j) + y(k)\ , \quad k \geq m\ . \tag{4.43b}$$

It follows easily from (4.38a), (4.39a) that initial conditions are

$$u(0) = \ldots = u(m-1) = 0\ , \tag{4.42a}$$

$$v(0) = \ldots = v(m-1) = 0\ . \tag{4.43a}$$

Using vector notations for sliding windows of samples (as in eq. (1.7)) the two sub-gradient vectors then are

$$\nabla_A\, y(k) = U(k) \quad (I \text{ coordinates})\ , \tag{4.44}$$

$$\nabla_B\, y(k) = V(k-1) \quad (m \text{ coordinates})\ . \tag{4.45}$$

Although these formulae look very similar to the formula (4.6) for the gradient of a FIR filter, there are two major differences :

(i) the sliding vectors $U(k)$ and $V(k-1)$ cannot be evaluated knowing only the input $x(k)$ and output $y(k)$ of $\mathbf{h}$ at time k. The additional knowledge of the recursive parameter B is required to implement (4.42) and (4.43).

(ii) the three recurrences (4.33), (4.42) and (4.43) are initialized at times 0, 1, ... , $m-1$. Hence to implement them at the running time k, we need a memory of size k, which is growing indefinitely .

Points (i) and (ii) are important drawbacks directly related to the recursive nature of filter $\mathbf{h}$.

It follows from (3.23) (4.44) and (4.45) that the stochastic LMS algorithm reads

$$A(k) = A(k-1) + \mu\, e(k)\, U(k) \quad \Big|_{A = A(k-1)\ ;\, B = B(k-1)}\ , \tag{4.46}$$

$$B(k) = B(k-1) + \mu\, e(k)\, V(k-1) \quad \Big|_{A = A(k-1)\ ;\, B = B(k-1)}\ . \tag{4.47}$$

In these equations, the signal $y(k)$ (in the error $e(k)$) and the auxiliary signals $u(k)$, $u(k-1)$,.. , $v(k-1)$, $v(k-2)$ ... must be evaluated according to the full recurrences (4.33), (4.42), (4.43), **always taking the parameter (A, B) in the state $(A(k-1), B(k-1))$**. For instance, $y(k)$ must be computed along the set of equations

$$y^k(0)\ , y^k(1)\ , \ldots , y^k(m-1) \ \text{ fixed }\ , \tag{4.48a}$$

$$y^k(n) = \sum_{j=1}^{m} b_j(k-1)\ y^k(n-j) + \sum_{i=0}^{I-1} a_i(k-1)\, x(n-i)$$

$$\text{for } m \le n \le k\ . \tag{4.48b}$$

$$y(k) = y^k(k)\ ; \quad e(k) = d(k) - y(k). \tag{4.48c}$$

In $y^k(n)$, the upper index k emphasizes the dependency of the output at time $n \le k$ on the future filters parameters $A(k-1)$ and $B(k-1)$. Similarly with the calculation of the vector $U(k)$ taking the parameters $A(k-1), B(k-1)$ and the initial conditions in (4.42a). And similarly with the calculation of the vector $V(k-1)$.

The set of equations (4.48) clearly displays the difficulty (ii), that is the requirement for an indefinitely growing memory size. Thus it must be concluded that the above (STLMS) algorithm is not feasible. It will have to be simplified.

A third difficulty follows from the nonlinear character of $y(k)$ w.r.t. H. It is the fact that the MS cost $J(H)$ is non quadratic w.r.t. H. Unlike in the case of a cascade of transversal filters, this fact will often induce the presence of local minima [26], [27]. As a result the stochastic algorithm (4.46), (4.47) may have an undesirable sensitivity w.r.t. initial conditions : $H(k)$ may converge towards a local minimum $\tilde{H}$. This difficulty remains in the simplified version of the (STLSM) to be described in the following subsection.

IV.3.3. The finite-memory recursive LMS algorithm

A simplification required to make the above described algorithm feasible is to truncate the memory of the recurrence (4.48). This means that initial conditions fixed at times $0, 1, \ldots, m-1$ as in (4.48a) will be replaced by initial conditions fixed at times $k-M-m+1, k-M-m+2, \ldots, k-M$. In other words, there is a fixed number M of steps between the running step k and the late end $k-M$ of the window for the y initial values. In this way, at step k, algorithm (4.48) becomes

$$y^k(k-M-m+1),\ y^k(k-M-m+2), \ldots, y^k(k-M) \text{ fixed}, \tag{4.49a}$$

$$y^k(n) = \sum_{j=1}^{m} b_j(k-1)\ y^k(n-j) + \sum_{i=0}^{I-1} a_i(k-1)\ x(n-i)\ ,$$

$$k-M+1 \le n \le k\ . \tag{4.49b}$$

Similarly with a fixed number M' (resp.M'') of steps between the running step k and the late end $k-M'$ (resp. $k-M''$) of the window for the u (resp. v) initial values :

$$u^k(k-M'-m+1), u^k(k-M'-m+2), \ldots, u^k(k-M') \text{ fixed}\ , \tag{4.50a}$$

$$v^k(k-M''-m), v^k(k-M''-m+1), \ldots v^k(k-M''-1) \text{ fixed}, \tag{4.51a}$$

$$u^k(n) = \sum_{j=1}^{m} b_j(k-1)\; u^k(n-j) + x(n), \qquad k - M' + 1 \le n \le k\,, \quad (4.50b)$$

$$v^k(n) = \sum_{j=1}^{m} b_j(k-1)\; v^k(n-j) + y(n)\,, \qquad k - M'' \le n \le k-1\,. \quad (4.51b)$$

The last problem to implement the algorithm is to fix the initializing values in (4.49a, 4.50a, 4.51a). Because the step-size μ is small, $A(k)$, $B(k)$ should not be very far from $A(k-1)$, $B(k-1)$. Thus, the only meaningful choice when k is incremented to $k+1$ is to slide one step ahead the initializing windows and add the latest sample thanks to the recurrences (4.49b, 4.50b, 4.51b). This is illustrated for the y's on the diagram of Table 1.

	initial window (4 . 49 a)	recurrence (4 . 49 b)
time $k-1$	$y^{k-1}(k-M-m),\ y^{k-1}(k-M-m+1),\ \ldots\ , y^{k-1}(k-M-1)$	$y^{k-1}(k-M)$
time k	$y^{k}(k-M-m+1),\ \ldots\ , y^{k}(k-M-1),\ y^{k}(k-M)$	$y^{k}(k-M+1)$

Table 1. The initializing procedure for the (M-RLMS) algorithm

This table shows that the initial window at time k is given by the values

$$[y^{k-m}(k-M-m+1),\ \ldots\ , y^{k-2}(k-M-1)\,,\ y^{k-1}(k-M)]\,, \quad (4.49a')$$

i.e. the first samples respectively generated by (4.49b) at the previous steps $(k-m)$, $(k-m+1)$, $\ldots$, $(k-1)$. A similar procedure holds for the auxiliary signals u and v.

It is now possible to implement the updating equations (4.46) and (4.47) by plugging into the increments of A and B the values

$$e(k) = y^k(k) - d(k) \ , \tag{4.52}$$

$$U(k) = [u^k(k), u^k(k-1), \ldots , u^k(k-I+1)]^T, \tag{4.53}$$

$$V(k-1) = [v^k(k-1), v^k(k-2), \ldots , v^k(k-m)]^T, \tag{4.54}$$

that have been forwarded by the recurrences (4.49), (4.50), (4.51).

Hereafter, the corresponding update algorithm is called "(finite) memory recursive LMS", and abbreviated (M-RLMS). In this name, the word recursive is used to recall the recursive character of the filter, the recursive character of the algorithm being taken for granted and kept implicit. Note that it has been written using the same kind of approach as in subsection III.6 where the causal block-LS gradient algorithm was simplified into the sliding window LS gradient one.

The (M-RLMS) algorithm is always feasible, but its computational complexity and memory requirements are high when M, M' or M'' are large. Hence a further level of simplifications in the next subsection.

Computation complexity. By inspecting formulae (4.46), (4.47), (4.49b), (4.50b), (4.51b), it is easily shown that the total number of multiplications per step is

$$\text{C(M-RLMS)} = (M+1)\,N \ + (M' + M'')\,m \ + 2 \tag{4.55}$$

where $N = m + I$ is the total number of parameters, and m the length of the recursive part of the filter. For instance, for a purely recursive filter ($I = 1$) and for the same memory M on the y's, the u's and the v's, the complexity is

$$C^1(\text{M-RLMS}) = M(3N - 2) + (N + 2) \quad . \tag{4.56.a}$$

For a recursive filter of order 1 ($m = 1$), the complexity is

$$C_1(\text{M-RLMS}) = (M + 1)\,(N + 2) \quad . \tag{4.56.b}$$

In brief, introduction of the memory M in the algorithm is responsible for an increased computational burden, as well as for the requirement of additional hardware (RAM).

IV.3.4. The standard recursive LMS algorithm

The next level of simplification of the exact (STLMS) algorithm is to set $M = M' = M'' = 1$ in the above finite memory recursive LMS. Hence the so-called standard "recursive LMS" abbreviated in (RLMS) [28] whose formulae are given below

$$y(k) = \sum_{j=1}^{m} b_j(k-1)\, y(k-j) + \sum_{i=0}^{I-1} a_i(k-1)\, x(k-i), \tag{4.57}$$

$$u(k) = \sum_{j=1}^{m} b_j(k-1)\, u(k-j) + x(k), \tag{4.58}$$

$$v(k-1) = \sum_{j=1}^{m} b_j(k-1)\, v(k-j-1) + y(k-1), \tag{4.59}$$

$$A(k) = A(k-1) + \mu\, e(k)\, U(k), \tag{4.60}$$

$$B(k) = B(k-1) + \mu\, e(k)\, V(k-1). \tag{4.61}$$

The associated computational complexity is

$$C(\text{RLMS}) = 2(N + m + 1) \; . \tag{4.62}$$

IV.3.5. The extended LMS algorithm

There is a final level of simplification which consists in calculating the gradient $\nabla_H y(k)$ as if the filter $\mathbf{h}$ were non recursive. In the RHS of (4.33c), the vector $Y(k-1)$ is viewed as independent of the parameter values A and B. This approach is usual in Signal Processing [29] e.g. in the classical ADPCM system [25]. This approximation is valid at the end of convergence ($H(k) \approx \tilde{H}$) if the LMS optimum filter $\tilde{H}$ is good enough to restore the reference signal perfectly. Then

$$Y(k-1) = D(k-1),$$

which indeed does not depend on the parameters A and B of $\mathbf{h}$. This "extended LMS" (ELMS) algorithm reads

$$y(k) = \sum_{j=1}^{m} b_j(k-1)\ y(k-j) + \sum_{i=0}^{I-1} a_i(k-1)\ x(k-i)\ , \qquad (4.63)$$

$$A(k) = A(k-1) + \mu\ e(k)\ X(k)\ , \qquad (4.64)$$

$$B(k) = B(k-1) + \mu\ e(k)\ Y(k-1)\ . \qquad (4.65)$$

For obvious reasons, the word recursive does not appear in the name of this algorithm. Its computational complexity is

$$C(\text{ELMS}) = 2(N+1)\ . \qquad (4.66)$$

IV.3.6. Discussion/Comparison of algorithms

We have presented three major LMS algorithms for updating recursive filters

(M-RLMS) : the finite memory recursive LMS algorithm ;

(RLMS) : the standard recursive LMS algorithm ;

(ELMS) : the extended LMS algorithm.

The three of them are intended to approach the exact stochastic LMS gradient algorithm, whose objective is to asymptotically reach the vector $\tilde{H}$ minimizing the MS cost $J(H)$. All of them may reach a local minimum if the initial values $A(0)$, $B(0)$ are not suitable. There are several view points under which they can be compared to one another

(i) accurracy of convergence;

(ii) speed of convergence ;

(iii) computational complexity ;

(iv) stability. This last question is raised because the output $y(k)$ of a (fixed) recursive filter can grow unbounded depending on the value of vector B in (4.36). Evidently $y(k)$ can also diverge with an adaptive recursive filter in which $B(k)$ is changing.

When the algorithm is made simpler by switching from (M-RLMS) to (RLMS) and then to (ELMS), the complexity is reduced whereas the accuracy of convergence gets poorer. But it is not proved that stability gets poorer at the same time. For instance examples are known where (ELMS) is more stable than (RLMS) [30]. However, it is conjectured that introduction of the memory improves stability and that (M-RLMS) has better stability than (RLMS). For instance, in the context of Adaptive Control, Landau [23] has introduced a refinement of the (RLMS) algorithm which is more stable, under the name "hyperstable adaptive recursive filter" (HARF). It turns out that the HARF algorithm [31] is intermediate between (M-RLMS) with $M = 2$ and (ELMS) as shown below.

The HARF algorithm [32]. This algorithm uses an auxiliary signal $\bar{y}(k)$ according to

$$\bar{y}(k-1) = \sum_{j=1}^{m} b_j(k-1)\,\bar{y}(k-j-1) + \sum_{i=0}^{l-1} a_i(k-1)\,x(k-i-1), \qquad (4.67)$$

$$y(k) = \sum_{j=1}^{m} b_j(k-1)\,\bar{y}(k-j) + \sum_{i=0}^{l-1} a_i(k-1)\,x(k-i), \qquad (4.68)$$

$$a_i(k) = a_i(k-1) + \mu\,(d(k) - y(k))\,x(k-i), \qquad (4.69)$$

$$b_j(k) = b_j(k-1) + \mu\,(d(k) - y(k))\,\bar{y}(k-j). \qquad (4.70)$$

Now with $M = 2$, the formulae (4.49a') and (4.49b) of the (M-RLMS) algorithm read

$$y^k(k-1) = \sum_{j=1}^{m} b_j(k-1)\,y^{k-j}(k-j-1) + \sum_{i=0}^{l-1} a_i(k-1)\,x(k-i-1), \qquad (4.71)$$

(this is (4.49b) with $n = k-1$)

$$y^k(k) = \sum_{j=1}^{m} b_j(k-1)\,y^{k-j+1}(k-j) + \sum_{i=0}^{l-1} a_i(k-1)\,x(k-i) \qquad (4.72)$$

(this is (4.49b) with $n = k$).

Clearly (4.71) and (4.72) coincide with (4.67) and (4.68) respectively through the identification

$$y^k(k-1) = \bar{y}(k-1), \qquad (4.73)$$

$$y^k(k) = y(k). \qquad (4.74)$$

Concerning the update equations (4.69), (4.70) of the (HARF) algorithm, they are very similar to the corresponding equations (4.64), (4.65) of the (ELMS) algorithm.

In fact, (HARF) is a kind of (M-RLMS) algorithm with M=2. However, like (ELMS) it presents no recursive filtering in the updating equations. This (HARF) algorithm is thus intermediate between (M-RLMS) and (ELMS) and it presents a better stability than the standard (RLMS) algorithm. Its stability can be further improved by some suitable filtering of the error $e(k)$ (cf [32]).

Let us now illustrate the standard (STLMS) algorithm taking some important examples of (NN).

V. THE STOCHASTIC LMS ALGORITHM FOR THE TRAINING OF NEURAL NETWORKS

In this section, we consider the supervised training of (NN) with a finite collection of arbitrarily ordered examples. Throughout the section, the index k corresponds to the example presented to the (NN). As opposed to the problems of filtering considered in the previous Section, in the (NN) context the output $y(k)$ is nonlinear versus the inputs $x_1(k)$, ... , $x_N(k)$ of the net (except in Section V.2.3). Through the recursive (STLMS) algorithms, the network parameter H is updated each time a new pair $(X(k), d(k))$ is observed. In this way, the network training procedure can be called adaptive. Section V is devoted to certain structures for the network h.

V.1. The elementary neural cell

In the context of (NN), the simplest system h is the elementary neural cell depicted in fig. 8 [2],
where the output is

$$y_H(k) = f(X(k)^T H) \quad , \tag{5.1}$$

the input vector $X(k)$ associated to the k-th example being as in (1.1). In (5.1), f is a time independent nonlinear function, called the activation function of the neuron. It is often a monotonically increasing,

differentiable, bounded function. It may be a sigmoid function such as the example (1.3) given in the introduction. When

$$\rho = \infty \tag{5.2}$$

then

$$f(x) = A \ \mathrm{sgn}(x). \tag{5.3}$$

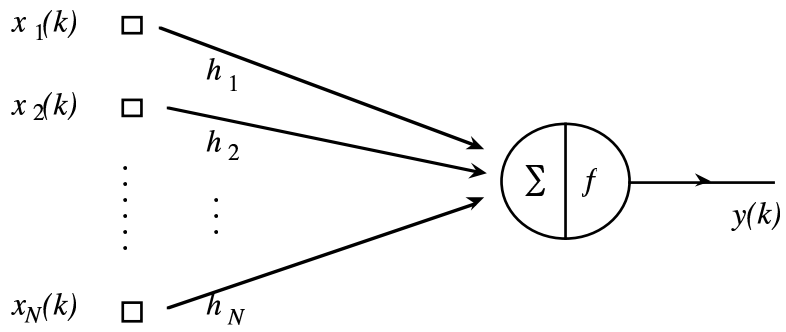

Fig. 8. The elementary neural cell

The corresponding cell is referred to as a linear threshold separator, or as a binary state neuron, in the (NN) literature [33], [34]. Note that in this case, the f function is neither continuous nor derivable.

Hereafter, derivability of f is of utmost importance to write explicitly the STLMS-based algorithm for the net. Boundedness of f is also a crucial point. Comparison of fig. 8 with fig. 2 shows that the elementary neural cell generalizes the notion of transversal filtering in two respects :

(i) the inputs x_i, $i = 1, \ldots, N$ are not necessarily related to one another by time delays ;

(ii) the output y involves a nonlinearity.

For the elementary cell, the standard stochastic LMS gradient algorithm (3.23) clearly reads

$$H(k) = H(k-1) + \mu \, e_H(k) f'(z_H(k)) X(k) \Big|_{H = H(k-1)} \tag{5.4}$$

where $f'(x)$ is the derivative of $f(x)$ and where

$$z_H(k) \triangleq X(k)^T H\square; \tag{5.5}$$

$z_H(k)$ is termed the potential of the neuron. Algorithm (5.4) is intended to find the minimum $\tilde{H}$ of the MS cost function

$$J(H) = E\,[(d(k) - f(X(k)^T H))^2]\ . \tag{5.6}$$

If the factor $f'(z(k))$ is omitted in algorithm (5.4), it results into the updating formula (4.7) which is the standard LMS algorithm used to adapt transversal (TF) (see § IV.1). In the (NN) context, algorithm (4.7) has been referred to [3] as the delta rule. An important difference between the (STLMS) algorithm (5.4) and the delta rule (4.7) is that the former is the gradient of the MS cost criterion, whereas the latter is not a gradient for a general function f. Yet for the specific function $f(x) = \mathrm{sgn}(x)$, algorithm (4.7) is shown in [35], [38] to be the stochastic gradient algorithm associated with the cost function $\mathcal{E}(H) = E[e\ X^T H]$. This algorithm has been widely used for training fully connected feedback (NN) designed to operate as associative memories.

Although the elementary neural cell is the simplest case of (NN), the nonlinearity of f induces the possibility of multiple minima in certain instances and makes it difficult to establish a general discussion of convergence for the corresponding (STLMS) algorithm. For gaussian inputs, this point is treated in [35], [37]. Then, as far as we are only concerned with the output error $e(k)$, the true (STLMS) algorithm (5.4) and its simplified version (4.7) works approximately in the same way. For non gaussian inputs, there is, to our knowledge, no general discussion available yet. Most papers rely on computer simulations.

In the following, we focus on the (NN) training, using the (STLMS) gradient algorithm (5.4) with **a derivable sigmoid function** f. If we do not take into account the computational complexity for calculating $f(x)$ and $f'(x)$, but retain only the multiplications, this algorithm has a low complexity (per step) namely

$$C(\mathrm{ELEMN}) = 2\,N + 2. \tag{5.7}$$

In a (NN), a number of such cells are interconnected. Because our objective is to emphasize similarities between (NN) and (TF), we restrict our presentation to layer feedforward networks and to feedback networks, which are respectively similar to cascaded filters (§ IV.2) and to recursive filters (§IV.3). However, it should be noticed that, for other applications, more general structures could be considered.

V.2. Multi-layer feedforward networks

V.2.1. Definition

The cascaded way to interconnect neural cells is to form layers as depicted in fig. 9 [39]-[42] with (L–1) layers of elementary cells. This structure has been refered to as a multilayer Perceptron [2]. Each cell works as the one of fig. 8. The l-th layer has N_l cells. The set $x_1, x_2, \ldots, x_I$ of inputs is viewed as a layer with label $l = 0$. According to the agreement made in Section I, for the sake of comparison with (TF), we consider the case of a single output, denoted y ; it is viewed as the L-th layer. Although the inputs and the output are not made of standard neural cells, their inclusion as layers makes notations more convenient because the whole network can be described in a systematic way by a set of L interconnexion matrices

$$\mathbf{h}^l = (h_{i,j}^l), \quad l = 1, 2, \ldots, L \quad , \tag{5.8}$$

where $h_{i,j}^l$ is the influence of the j-th cell of layer $\#\, l-1$ onto the i-th cell of layer $\#l$. Because the L-th layer has a single cell (which is the output y), the matrix $\mathbf{h}^L$ is a row matrix. We will use the notation $\mathbf{h}^{[1,L]}$ to designate the set of matrices $\mathbf{h}^1, \mathbf{h}^2, \ldots, \mathbf{h}^L$. The network is characterized by the set of equations

$$y_i^l = f(z_i^l) \quad , l = 1, 2, \ldots, L, \qquad i = 1, 2, \ldots, N_l \tag{5.9}$$

$$z_i^l = \sum_{j=1}^{N_{l-1}} h_{i,j}^l \; y_j^{l-1} \, , \tag{5.10}$$

$y = y_1^L$ (output) . (5.11)

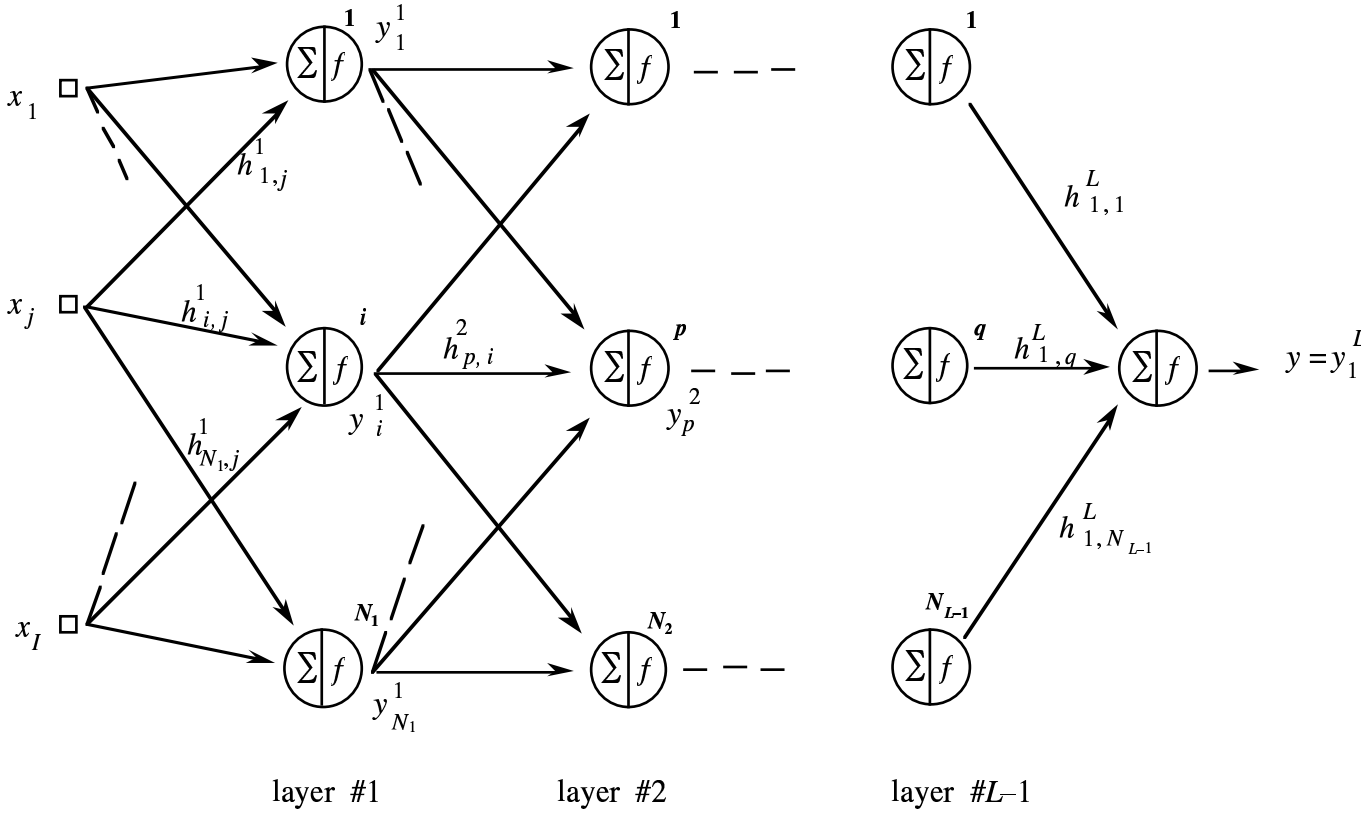

Fig. 9. A multilayer neural network

The quantities y_i^l are the outputs of the elementary cells. Because of the existence of a non linear function and because the neural cells are highly interconnected, in general, the overall cost function $\mathcal{E}(H)$ in (3.7) can have local minima (this makes a difference with the case of cascaded filters studied in Section III.2). However, it is expected that $\mathcal{E}(H)$ is "well behaved" in the vicinity of these minima. In addition, these minima can be found by gradient techniques. Hence the (STLMS) approach for multilayer neural networks remains meaningful.

V.2.2. The standard LMS algorithm

For the network (5.9) - (5.11), in order to implement the standard stochastic LMS gradient algorithm (3.23), it is necessary to evaluate the L matrices of derivatives $(l = 1, \dots, L)$

$$\mathbf{u}^l = (u_{i,j}^l) \quad j = 1, 2, \dots, N_{l-1}; \; i = 1, 2, \dots, N_l, \tag{5.12}$$

$$u_{i,j}^l \triangleq \frac{\partial y}{\partial h_{i,j}^l} .$$

There are two ways to conduct the calculation of the collection of partial derivatives (5.12). We call these two computational methods forward and backward propagation respectively. Both methods use the fact that, in a layer structure, the output y depends on $h_{i,j}^l$ only through the output y_i^l of the i-th cell of layer # l .

Forward propagation. In this method, the $u_{i,j}^l$ are calculated by direct differentiation of the net equations (5.9) - (5.11). It involves the partial derivatives $\frac{\partial y_p^m}{\partial h_{i,j}^l}$. Because of the layer structure, we successively have

$$\frac{\partial y_p^m}{\partial h_{i,j}^l} = 0 \qquad m < l \; ; \tag{5.13}$$

$$\frac{\partial y_p^l}{\partial h_{i,j}^l} = 0 \qquad p \neq i \; , \tag{5.14}$$

$$\frac{\partial y_i^l}{\partial h_{i,j}^l} = f'(z_i^l) \; y_j^{l-1} \; ; \tag{5.15}$$

and finally

$$\frac{\partial y_p^m}{\partial h_{i,j}^l} = f'(z_p^m) \sum_{q=1}^{N_{m-1}} h_{p,q}^m \; \frac{\partial y_q^{m-1}}{\partial h_{i,j}^l} , \qquad m > l \; . \tag{5.16a}$$

Iterating this last formula from layer $m = l + 1$ until layer $m = L$ ("forward propagation") will provide the desired quantity

$$u_{i,j}^{l} = \frac{\partial y_1^L}{\partial_i h_j^l} \quad . \tag{5.16b}$$

This method has a high computational complexity because it requires a very large number of intermediate quantities $\frac{\partial y_p^m}{\partial_i h_j^l}$. When the number of cells per layer is a constant N_c, the total complexity C(FPLN) for updating the layer network is bounded according to

$$(N_c + 2)\, N \leq C(\text{FPLN}) \leq \frac{1}{2}\, N\, (N + 4) \tag{5.17}$$

where N is the total number of parameters in the net. The lower bound corresponds to $L = 3$ (one hidden layer) and the upper bound corresponds to $N_c\ (L - 3) >> 4$. The value C(FPLN) includes all the mutliplications required to calculate the output and the updating and does not take into account the cost for calculating f and f'. This forward propagation way of computing the gradient is not usual in (NN). It was introduced in [12], [14].

Backward propagation [3], [5], [38]. The favorite method in (NN) is based on the fact that the network calculations are chained : all the dependency of y w.r.t. $h_{i,j}^{l}$ is summarized in the output y^l of the i-th cell of layer l. Hence

$$u_{i,j}^{l} = \frac{\partial y}{\partial y_i^l} \quad \frac{\partial y_i^l}{\partial_i h_j^l} \quad . \tag{5.18}$$

Using the quantities

$$g_i^l \triangleq \frac{\partial y}{\partial y_i^l} \tag{5.19}$$

it thus follows from (5.9) - (5.11) that

$$u_{i,j}^{l} = y_j^{l-1} f'(z_i^l)\, g_i^l \quad . \tag{5.20}$$

When the (NN) is implemented, the quantities z_i^l and y_j^{l-1} are necessarily calculated. Hence to implement the (STLMS) algorithm, it only remains to evaluate the partial gradients g_i^l in (5.19). This can be done with a backward recurrence over the layer label l according to

$$g_j^l = \sum_{i=1}^{N_{l+1}} g_i^{l+1} \frac{\partial y_i^{l+1}}{\partial y_j^l} \quad . \tag{5.21}$$

This last formula is an expression of the chained derivation rule [5]. It means that y depends on y_j^l only through the outputs y_i^{l+1} of the elementary cells to which y_j^l is connected, that are all the cells of the next layer $(l+1)$. It follows from (5.10) that

$$g_j^l = \sum_{i=1}^{N_{l+1}} g_i^{l+1} h_{i,j}^{l+1} \cdot f'(z_i^{l+1}) \quad . \tag{5.22}$$

Starting with the last partial gradient

$$g_1^L = \frac{\partial y}{\partial y_1^L} \equiv 1, \tag{5.23}$$

the rule (5.22) permits to calculate by backpropagation all the previous partial gradients g_j^l involved in (5.20). Hence the evaluation of the L matrices $\mathbf{u}^l$ in (5.12).

The "backward propagation layer network" algorithm (BPLN) Once the gradient matrices $\mathbf{u}^l$ have been evaluated using the backward propagation method (5.20) (5.22) (5.23), the network can be made adaptive according to the updating (5.24) which implements the (STLMS) algorithm (3.23). We call (BPLN) the corresponding procedure. Clearly all the quantities, $u_{i,j}^l$, y_j^{l-1}, z_i^l , g_i^l appearing in these equations also depend on the label k of the example presented to the network. The l-th layer will then be updated along

$$\mathbf{h}^l(k) = \mathbf{h}^l(k-1) + \mu\, e(k)\, \mathbf{u}^l(k) \quad \Big|_{\mathbf{h}^{[1,L]} = \mathbf{h}^{[1,L]}(k-1)} \,. \tag{5.24}$$

The "forward propagation layer network" algorithm (FPLN). The increment of the (STLMS) algorithm (5.24) can also be evaluated through the forward propagation method (5.16 a,b). The corresponding procedure is called (FPLN)

Computational complexity. The computation of the partial gradients g_j^l through (5.22) requires less arithmetical operations than the computation of the derivatives $\frac{\partial y_p^m}{\partial h_{ij}^l}$ necessary for the forward propagation method. The total number of multiplications, including the computation of the output and the updating of the net, is

$$C(\text{BPLN}) = 4\, N \tag{5.25}$$

where N is the total number of parameters. Comparing C(BPLN) to C(FPLN) shows why the (BPLN) algorithm is so attractive in (NN). The complexity (5.25) is very low, in the same kind of range as the complexity found for transversal filters and cascaded filters (cf. 4.10) and (4.31)).

Remark Although a cascade of filters is indeed a multilayer network (see below), formulae (4.31) and (5.25) are not exactly the same because all the corresponding layers do not have an equal number N_c of cells.

Formula (5.20) is interesting. It shows that, for the subgradients $\mathbf{u}^l$, the dependency on the various layer matrices $\mathbf{h}^m$ is distributed in three steps :

(i.N) the quantities y_j^{l-1} are the outputs of the $(l-1)$-th layer. They depend on the input X and on the $(l-1)$ first matrices $\mathbf{h}^1, \ldots, \mathbf{h}^{l-1}$, but they do not depend on $\mathbf{h}^l, \mathbf{h}^{l+1}, \ldots, \mathbf{h}^L$;

(ii.N) according to (5.22), the quantities g_i^l depend on the $(L-l)$ last matrices $\mathbf{h}^{l+1}, \ldots, \mathbf{h}^L$. They also depend on previous matrices but only through the derivative values $f'(z_i^{l+p}), p > 0$.

(iii.N) the quantities $f'(z_i^l)$ depend on the outputs of the l-th layer and on the l-th matrix $\mathbf{h}^l$.

It is interesting to notice that provided f has a derivative, the specific kind of nonlinearity has no influence on the structure of the (STLMS) algorithm for multilayer networks, even in the particular form of the (BPLN) algorithm. In particular, the algorithm structure would not be simpler in the case of a linear function f. Nevertheless, the presence of a non linear function is essential in the analysis of the algorithm performances (stationary points, speed and accuracy of convergence).

V.2.3. Comparison between cascaded filters and linear multilayer nets

To perform this comparison [43] [44], a linear layer network is assumed so that

$$f'(z) \equiv 1. \tag{5.26}$$

Then the above paragraphs disclose the similarities between cascaded filters and layer nets. In fact, for the cascade of filters in fig.5, according to (4.22) and (4.23), the subgradient used at time k in the updating of the coefficient h_p^l of the l-th filter is

$$u^l(k-p) = \mathbf{k}_{l+1}^{L}(y^{l-1}(k-p)) \tag{5.27.a}$$

where $y^{l-1}(k)$ is the output of the first $(l-1)$ filters when the input is $x(k)$ and where

$$\mathbf{k}_{l+1}^{L} = \mathbf{h}^L \circ \ldots \circ \mathbf{h}^{l+1} \tag{5.27.b}$$

represents the subcascade of the last $L-l$ filters. Thus the dependency of $u^l(k-p)$ on the various cascaded filters $\mathbf{h}^m$ is distributed in two steps :

(i.F) the quantity $y^{l-1}(k-p)$ is the output of the $(l-1)$-th filter. It depends on the input $x(k)$ and on the first $(l-1)$ filters $\mathbf{h}^1, \ldots, \mathbf{h}^{l-1}$; but it does not depend on $\mathbf{h}^l, \ldots \mathbf{h}^L$. This step is completely similar to the step (i.N) for a layer network ;

(ii.F) according to (5.27), the calculation of $\mathbf{k}_{l+1}^{L}$ in the second step requires the $(L-l)$ last filters $\mathbf{h}^{l+1}, \ldots, \mathbf{h}^L$ but not the other filters. Thanks to (5.26), in the network step (ii.N), f' is irrelevant. Again (ii.F) and (ii.N) are found completely similar.

Finally (5.26) makes step (iii.N) idle.

At first glance, one could think that the similarities end up here because passing $y^{l-1}(k-p)$ through the filter $\mathbf{k}_{l+1}^{L}$ (cf (5.27)) is not similar to the multiplication $y_j^{l-1} g_i^l$ involved in the network formula (5.20). In fact, the similarities can be pushed further as shown below

It is indeed possible to represent (see fig.10) any cascade of filter by a particular multilayer network, provided certain constraints are fulfilled, namely

$$h_{i,j}^{l} \text{ (network representation)} \equiv h_p^{l} \text{(filter representation)} \quad \forall (i, j) \ / \ j - i = p. \tag{5.28}$$

The other special constraints are that

$$x_i(k) = x(k - i + 1) \ , \tag{5.29}$$

and that the number of coefficients per layer is decreasing according to

$$N_{l+1} = N_l - M_l + 1 \tag{5.30}$$

Under these conditions the network formulae (5.20) (5.26) written at time k, namely

$$u_{i,i+p}^{l}(k) = y_{i+p}^{l-1}(k) \ g_i^l(k) \tag{5.31}$$

will provide the same value as the filtering formula (5.27a) for the gradient signal used to update the coefficient (5.28).

In the filtering representation (step (ii.F)) it can be recognized that the impulse response g_0^l, g_1^l, ... , g_j^l, ... of the filter $\mathbf{k}_l^L = \mathbf{k}_{l+1}^L \circ \mathbf{h}^{l+1}$ obeys

$$g_j^l = \sum_i g_i^{l+1} h_{j-i}^{l+1} \tag{5.32a}$$

where $h^{l+1}_0, h^{l+1}_1, \ldots h^{l+1}_{M_l-1}$ is the impulse response of $\mathbf{h}^{l+1}$. Formula (5.32a) is clearly a backward propagation one.

In the network representation (step (ii.N)) the partial gradients obey the backward propagation formula

$$g^l_j = \sum_i g^{l+1}_i h^{l+1}_{i,j}. \qquad (5.32b)$$

Clearly the constraint (5.28) ensures that the (TF) formula (5.32.a) coincides with the (NN) formula (5.32b).

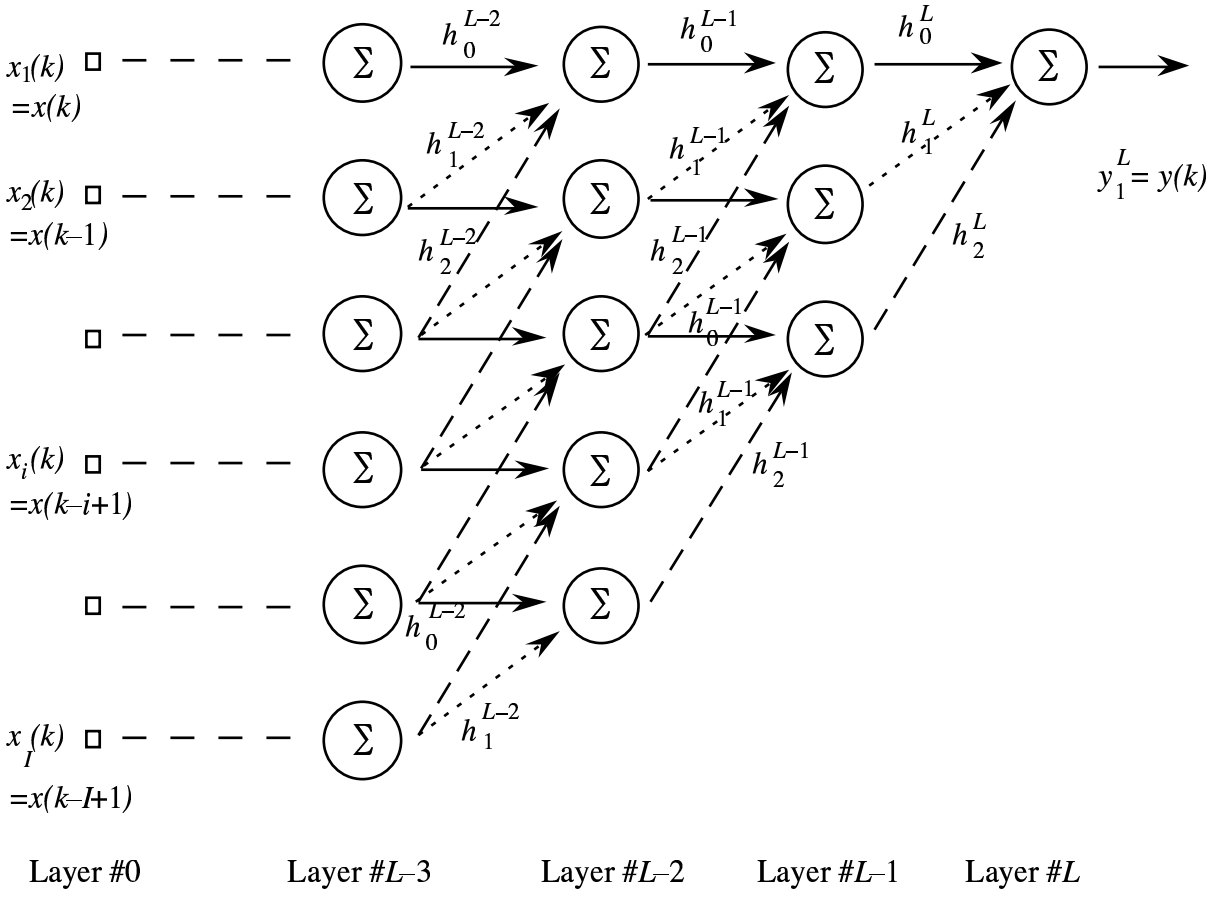

Fig. 10. The layer network representation of a cascade of filters (case $M_l = 3\ \forall l$)

Finally it is shown in Appendix that under the constraints (5.26), (5.28), (5.29), (5.30), the filtering formula (5.27.a) coincides with the network formula (5.20). This means that the very steps

implemented in a layer network by the (BPLN) algorithm are the same steps as those taken in a cascade of linear filters by the standard (LMS) algorithm. This is in spite of the fact that in the cascaded filters, the gradient was not evaluated through a backward approach but through a direct (forward) approach.

In other words the concept of backward propagation does not provide any new algorithm for a cascade of linear filters. On the contrary, it could bring new insights for the adaptation of a non-linear filter.

V.3. Feedback networks

V.3.1. The context

In a single layer or multilayer feedforward network, the (single) output $y(k)$ is computed from the input vector $X(k)$ after a (fixed) finite number of arithmetical operations (additions, multiplications, $f(.)$). The idea behind feedback networks is to implement some iterative mechanism rather than an expanded (explicit) formula by introducing a loop. Thus they are conceptually similar to recursive filters.

When introduced, feedback nets (also termed recurrent or dynamical nets), they were designed to operate as associative memories [45]. **In such a framework**, the network equations are of the kind

$$y_0(k), y_1(k), \ldots, y_{m-1}(k) \text{ fixed} , \tag{5.33a}$$

$$y_p(k) = F(H, X(k), y_{p-1}(k), \ldots, y_{p-m}(k)) , p \geq m , \tag{5.33b}$$

where H is the fixed vector of network parameters. When F is non linear w.r.t. y_{p-1}, y_{p-2}, ... , y_{p-m}, for a fixed example $X(k)$ presented to the net, the repeated iterations over p generally lead to limit point $y_\infty(k)$. Moreover, when $X(k)$ changes, the point $y_\infty(k)$ scans a finite collection of limit points. Therefore, there are a number of different (distorted) input patterns $X(k)$ which are associated to the

same limit point y_∞. Training consists in imposing the finite collection of attractor limit points. In this way, the network works as an associative memory.

In such a framework, for a single input $X(k)$, (5.33) is repeatedly implemented until convergence of the output $y_p(k)$. If P iterations ensure this convergence, the reference output $d(k)$ is compared to $y_P(k)$ in order to yield the error $e(k)$. To compute the gradient $\nabla_H y_P(k)$ necessary for updating the network, the net is unfolded in the way depicted in fig. 11 [3], [15] . In this figure, all the subnets are copies of the network **h**.

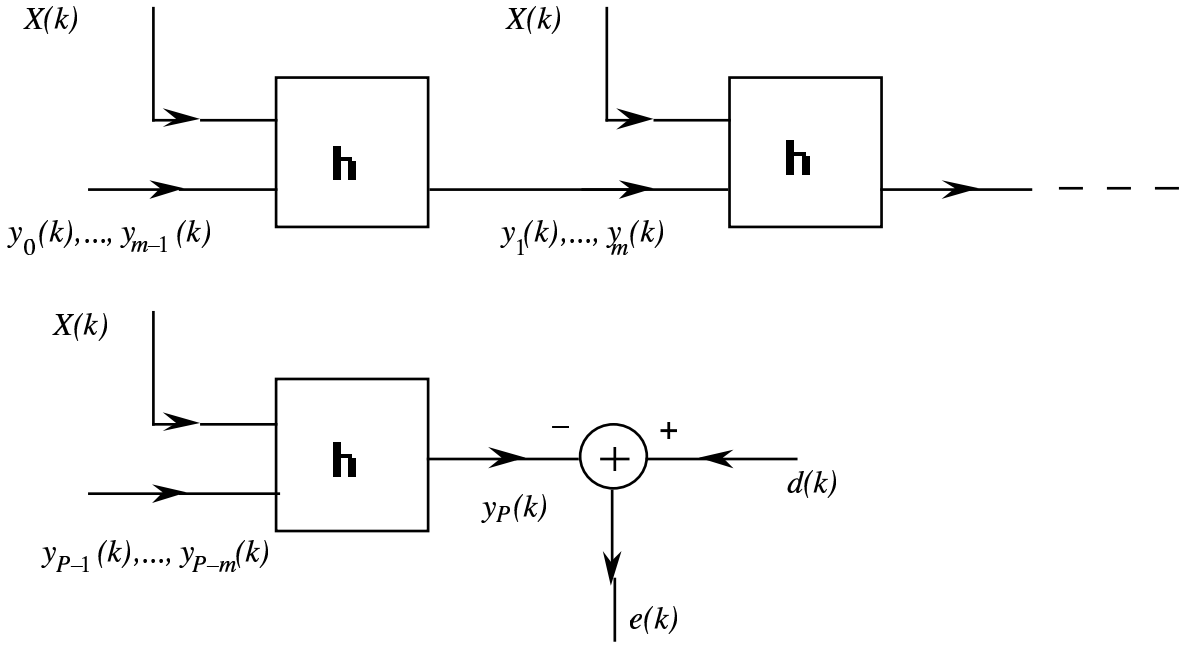

Fig. 11. The unfolding method for updating a feedback (NN)

This network is viewed as a non recursive (static) multilayer network, for which the intermediate gradients $\nabla_H y_p(k)$, $p = 1 , ... , P,$ are evaluated by the backward propagation method. Knowing that all the layers have the same parameter H, the final gradient is the sum of the P corresponding gradients $\nabla_H y_p(k)$. Other methods exist to evaluate the final gradient [45] - [48] in this framework.

In the present framework, as done in [11] [12], we consider a different context where a novel external information is presented to the net at each iteration (p) where the feedback loop is implemented. This means a one-to-one correspondence between the occurrence of a new example k and the operation of a new iteration p. In other words, $k = p$. Therefore, the network equations are, according to (5.33)

$$y(0), y(1), \ldots, y(m-1) \quad \text{fixed}, \tag{5.34a}$$

$$y(k) = F(H, X(k), y(k-1), \ldots, y(k-m)). \tag{5.34b}$$

But to allow the output $y(k)$ to retain some kind of convergence when k (= p) increases, $X(k)$ is not entirely renewed and there are some permanent features shared between $X(k-1)$ and $X(k)$. The easiest and most natural link between $X(k-1)$ and $X(k)$ is the time shifting property typical of the filtering context where

$$X(k) = (x(k), x(k-1), \ldots, x(k-I+1))^{T}. \tag{5.35}$$

Thus, the label k represents equivalently the time or the example. For notational convenience, let us group the delayed output samples into the sliding vector

$$Y(k-1) = (y(k-1), y(k-2), \ldots, y(k-m))^{T}. \tag{5.36}$$

Like $X(k)$, this vector has the shifting property. Then the network equations read

$$Y(m-1) \quad \text{fixed}, \tag{5.37a}$$

$$y(k) = F(H, X(k), Y(k-1)), \quad k \geq m. \tag{5.37b}$$

We are now faced with a system whose input ($x(k)$) and output ($y(k)$) are time signals. As a result of the recursive character of eq. (5.37), there is a strong resemblance with recursive (linear) filters, characterized by eq. (4.33). Thus, the problem of optimizing the net parameters H is similar to the optimization of recursive (linear) filters, detailed in subsection IV.3. The analysis below will thus closely follow the one of § IV.3.

For the sake of simplicity, we shall restrict our investigations to the case of **a "semi-linear" recursive network including a single neural cell;** the neuron potential $z(k)$ is linear w.r.t. the inputs and non linear in the delayed potentials $z(k-j)$' s according to

$$y(0)\ ,\ \ldots\ ,\ y(m-1) \text{ fixed} \tag{5.38a}$$

$$z(k) = \sum_{j=1}^{m} b_j\, y(k-j) + \sum_{i=0}^{I-1} a_i\, x(k-i)\ , \qquad k \geq m\ , \tag{5.38b}$$

$$y(k) = f(z(k))\ , \tag{5.38c}$$

the parameters of the network being

$$H^T = (A^T, B^T) \tag{5.39}$$

as in (4.34). The similarity between these nonlinear equations and the filtering equations (4.33) is obvious. The system can also be viewed as a semi-linear recursive filter. The corresponding system is depicted in fig. 12 where $\mathcal{A}$ and $\mathcal{B}$ are two transversal filters and z^{-1} denotes the unit delay. It closely resembles the recursive filter of fig.7.

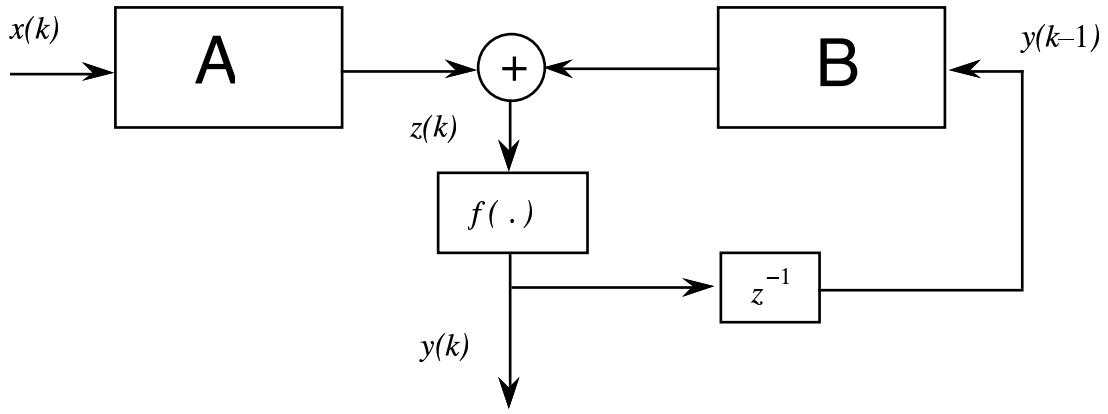

Fig. 12. A semi-linear recursive neural cell

In order to recursively optimize this system, the standard procedure is the (STLMS) algorithm, with equation (3.23). We are now going to derive it in two different ways.

V.3.2. The filtering approach

The steps taken to write the (STLMS) algorithm for a linear recursive filter (§ IV.3.2.) can be identically reproduced in the semi-linear case. The notations (4.37) are retained for partial derivatives. Differentiating (5.38) yields the two sets of equations

$$u_i(0) = u_i(1) = ... = u_i(m-1) = 0 \ , \ i = 0, ... , I-1 \ ; \qquad (5.40a)$$

$$v_j(0) = v_j(1) = ... = v_j(m-1) = 0, \ j = 1, ... , m \ ; \qquad (5.41a)$$

$$u_i(k) = f'(z(k)) \left[\sum_{n=1}^{m} b_n \ u_i(k-n) \ + x(k-i) \right] , \quad k \geq m \ ; \qquad (5.40b)$$

$$v_j(k) = f'(z(k)) \left[\sum_{n=1}^{m} b_n \ v_j(k-n) \ + y(k-j) \right] , \quad k \geq m \ . \qquad (5.41b)$$

The simplification found in the linear filtering case ($u_i(k) = u_0(k-i)$; $v_j(k) = v_0(k-j)$) is no longer valid in the present case, as a result of the inclusion of $f'(z(k))$ in the recurrence equations (5.40) and (5.41). Therefore, the sub-gradient vectors $U(k)$ and $V(k)$ defined in (4.44) and (4.45) are no longer sliding windows of the functions $u_0(k)$ and $v_0(k)$. This means that each one of the I (resp. m) coordinates of $U(k)$ (resp. $V(k)$) is calculated by a specific nonlinear recurrence (5.40) (resp. (5.41)). Nevertheless, the computational structure of the algorithm remains unchanged.

The (STLMS) algorithm follows, once $U(k)$ and $V(k)$ are known. It is written in (4.46), (4.47). In these equations, the signal $y(k)$ (in the error $e(k)$) and the derivative signals $u_0(k)$, ... , $u_{I-1}(k)$, $v_1(k)$, ... , $v_m(k)$ must be evaluated according to the full recurrences (5.38), (5.40), (5.41)

always taking the parameters (A, B) in the state $(A(k-1), B(k-1))$. For instance $y(k)$ must be computed along

$$y^k(0), \ldots, y^k(m-1) \text{ fixed} \quad ; \tag{5.42a}$$

$$z^k(n) = \sum_{j=1}^{m} b_j(k-1)\, y^k(n-j) + \sum_{i=0}^{I-1} a_i(k-1)\, x(n-i), \quad m \le n \le k \tag{5.42b}$$

$$y^k(n) = f(z^k(n)) \quad , \qquad m \le n \le k \quad ; \tag{5.42c}$$

$$y(k) = y^k(k) \quad ; \qquad e(k) = d(k) - y(k) \quad . \tag{5.42d}$$

The upper index k emphasizes that at times $n < k$ the output depends on the future system parameters $(A(k-1), B(k-1))$. Similarly with the calculations of $u_i(k)$, $v_j(k)$. Again we are faced with the requirement of a memory with indefinitely growing size. This problem is typical of gradient calculations for a system involving a loop.

It is clear that the MS cost $J(H)$ is non quadratic w.r.t. H and can have local minima. The (STLMS) algorithm will join a specific minimum depending on initial values for A, B and Y.

V.3.3. The network approach

The network approach uses two steps [5] : (i) the chained derivation rule ; (ii) the gradient backward propagation.

Step(i) : the chained derivation rule. This rule is characteristic of recurrences where the calculation of the successive values $y(n), n \le k$, is chained as in the system (5.38). The rule is written

$$u_i(k) \triangleq \frac{\partial y(k)}{\partial a_i} = f'(z(k)) \sum_{n=1}^{k} \frac{\partial z(k)}{\partial y(k-n)} \frac{\partial y(k-n)}{\partial a_i} + f'(z(k))\, x(k-i) \quad . \tag{5.43}$$

This can also be expressed by putting the last term of the RHS of (5.43) into the summation so that

$$u_i(k) = f'(z(k)) \sum_{n=0}^{k} \frac{\partial z(k)}{\partial y(k-n)} \frac{\partial y(k-n)}{\partial a_i} \quad (5.44)$$

The first comment is that the summation bound is $k - m$, since the m first values $y(0), \ldots, y(m-1)$ are fixed. The second comment is that each derivative $\frac{\partial y(k-n)}{\partial a_i}$ in (5.44), is evaluated considering the other variables $y(k-l)$, $l \neq n$, as quantities independent of a_i; the dependency on a_i being already taken into account via the summation (5.44). It thus follows that

$$\frac{\partial y(k-n)}{\partial a_i} = f'(z(k-n))\, x(k-n-i) \quad . \quad (5.45)$$

Therefore (5.44) reads

$$u_i(k) = f'(z(k)) \sum_{n=0}^{k} \frac{\partial z(k)}{\partial y(k-n)} f'(z(k-n))\, x(k-n-i) \quad . \quad (5.46)$$

The partial derivatives $v_j(k) \triangleq \frac{\partial y(k)}{\partial b_j}$ are evaluated similarly :

$$v_j(k) = f'(z(k)) \sum_{n=0}^{k} \frac{\partial z(k)}{\partial y(k-n)} f'(z(k-n))\, y(k-n-j) \quad . \quad (5.47)$$

Step (ii) : the gradient backward propagation. The derivative $\frac{\partial z(k)}{\partial y(k-n)}$ appearing in (5.46) and (5.47) can be evaluated on the basis of the quantities $\frac{\partial z(k)}{\partial y(k-n+1)}$, $\frac{\partial z(k)}{\partial y(k-n+2)}$, ... , and of the successive outputs of the neural cell. Indeed,

$$\forall\, n \geq 1 \;;\; \frac{\partial z(k)}{\partial y(k-n)} = \sum_{j=1}^{n} \frac{\partial z(k)}{\partial y(k-n+j)} \frac{\partial y(k-n+j)}{\partial y(k-n)} \quad . \quad (5.48)$$

This relation expresses that the only variables $y(k-l)$ depending on $y(k-n)$ are those such that $l < n$. Now from (5.38), it follows that the partial derivatives

$$\forall\, n \geq 1 \quad \frac{\partial y(k-n+j)}{\partial y(k-n)} = \begin{cases} f'(z(k-n+j))\, b_j & ;\ 1 \leq j \leq m \\ 0 & ;\ j > m \end{cases} \tag{5.49}$$

Clearly, (5.48) becomes

$$\forall\, n \geq 1\ , \quad \frac{\partial z(k)}{\partial y(k-n)} = \sum_{j=1}^{\mathrm{Inf}(n,m)} b_j f'(z(k-n+j)) \quad \frac{\partial z(k)}{\partial y(k-n+j)} \tag{5.50a}$$

$$\frac{\partial y(k)}{\partial z(k)} = \frac{1}{f'(z(k))}\ , \tag{5.50b}$$

Inf(n, m) denoting the smallest value of n and m. This formula computes the derivatives $\frac{\partial z(k)}{\partial y(l)}$ by successive decreases of the index l. Thus the recurrence (5.50) appears as a gradient backward propagation technique. Explicit formulae giving $u_i(k)$, $v_j(k)$ in terms of the network inputs x's, and network outputs y's , z's can be obtained by inserting the results of the backward recurrence (5.50) into the expressions (5.46), (547).

In a (NN) approach, the recursive cell can be unfolded in a way similar to that of fig. 11, where **h** is taken as the elementary semi-linear neural cell. The inputs of cell n° p are the values $y(k-p+1)$, ... , $y(k-p+m)$. Then the recurrence (5.50) computes the intermediate derivatives $\frac{\partial z(k)}{\partial y(k-p)}$ by a backward propagation technique, in the unfolded (static) multilayer network.

It can be checked that the resulting expressions for $u_i(k)$ and $v_j(k)$ coincide respectively with the recursive expressions (5.40), (5.41), which were obtained in the filtering approach of § V.3.2. This fact is not surprising.

For the sake of implementing the (STLMS) algorithm, the (NN) approach has the advantage over the filtering one to provide the explicit values (rather than recursive expressions) for the subgradients $U(k)$ and $V(k)$ used in the updating algorithm

$$A(k) = A(k-1) + \mu\, e(k)\, U(k) \quad \Big| \; H = H(k-1) \quad , \tag{5.51a}$$

$$B(k) = B(k-1) + \mu\, e(k)\, V(k) \quad \Big| \; H = H(k-1) \quad . \tag{5.51b}$$

V.3.4. The finite memory recursive LMS algorithm□

As already explained for recursive linear filters in subsection IV.3, the calculations of $y(k)$ and of its associated subgradient vectors $U(k)$, $V(k)$ appearing in (5.46) (5.47) requires that the memory has indefinitely growing size. This is a result of the recursive character of the system. To make the (STLMS) algorithm feasible, the memory will be truncated at a finite value M, in such a way that initial conditions will be given on the sliding time window $(k-M-m+1), \ldots, (k-m)$, rather than on the very ancient window $0, \ldots, m-1$. The ideas and procedures being essentially the same as in subsection IV.3, we keep for the update algorithm, the same denomination, i.e., "finite memory recursive LMS" abbreviated in (M-RLMS).

The (M-RLMS) equations turn out to be

$$y^k(k-M) = y^{k-1}(k-M) \quad , \tag{5.52a}$$

$$u_i^k(k-M) = u_i^{k-1}(k-M) \quad ; \; i = 0, \ldots, I-1 \quad , \tag{5.52b}$$

$$v_j^k(k-M) = v_j^{k-1}(k-M) \quad ; \; j = 1, \ldots, m \quad , \tag{5.52c}$$

$$y^k(n) = f(z^k(n)) \quad , \tag{5.53a}$$

$$z^k(n) = \sum_{j=1}^{m} b_j(k-1)\, y^k(n-j) + \sum_{i=0}^{I-1} a_i(k-1)\, x(n-i) \quad , \tag{5.53b}$$

$$u_i^k(n) = f'(z^k(n)) \left[\sum_{l=1}^{m} b_l(k-1)\, u_i^k(n-l) + x(n-i) \right] \quad , \tag{5.53c}$$

$$v_j^k(n) = f'(z^k(n)) \left[\sum_{l=1}^{m} b_l(k-1)\, v_j^k(n-l) + y^k(n-j) \right] \quad , \qquad (5.53d)$$

$$a_i(k) = a_i(k-1) + \mu\, (d(k) - y^k(k))\; u_i^k(k) \quad , \qquad (5.54a)$$

$$b_j(k) = b_j(k-1) + \mu\, (d(k) - y^k(k))\; v_j^k(k) \quad . \qquad (5.54b)$$

Note that equations (5.53) are written for $n \in [k-M+1, k]$ only.

Thus, it is clear that the (M-RLMS) algorithm for a recursive semi-linear neural cell is the same as for the recursive linear filter with the same structure, except for the presence of the derivative factor $f'(.)$ in the gradient coordinates. Consequently, the complexity is the same (if we don't take into account the computational load of the f and f' functions). It is given by eq. (4.55) and (4.56).

V.3.5. The standard recursive LMS algorithm.

The standard recursive LMS algorithm is denoted (R-LMS) as in subsection IV.3.4. It corresponds to the shortest memory $M = 1$ in the previous formulae. Then the writing gets simpler :

$$y(k) = f\left[\sum_{j=1}^{m} b_j(k-1)\, y(k-j) + \sum_{i=0}^{I-1} a_i(k-1)\, x(k-i) \right] \; ; \qquad (5.55a)$$

$$u_i(k) = f'(z(k)) \left[\sum_{l=1}^{m} b_l(k-1)\, u_i(k-l) + x(k-i) \right] \quad i = 0, \ldots, I-1 \; ; \qquad (5.55b)$$

$$v_j(k) = f'(z(k)) \left[\sum_{l=1}^{m} b_l(k-1)\, v_j(k-l) + y(k-j) \right] \quad j = 1, \ldots, m \; ; \qquad (5.55c)$$

$$a_i(k) = a_i(k-1) + \mu\, e(k)\, u_i(k) \; , \qquad i = 0, \ldots, I-1; \qquad (5.56a)$$

$$b_j(k) = b_j(k-1) + \mu\, e(k)\, v_j(k)\ , \qquad j = 1, \ldots, m\ . \quad (5.56b)$$

The associated complexity is given by eq. (4.62).

V.3.6. Discussion

In the present subsection, we have investigated the recursive updating of the semi-linear recursive neural cell which is the simplest example of neural network including a feedback loop. The input (example) presented to the network was a sliding window drawn from a time signal. In this context, the system would become similar to a feedback time filter, if it were not for the nonlinear function f. Therefore, the (STLMS) updating algorithm has the same structure and features as in the (TF) domain. In particular, feasibility requires to truncate the memory at a finite value M. With another kind of non-linearity in the net than semi-linearity, for instance with several semi-linear interconnected neural cells, the analysis would not be very different. However, a structure involving a full network has a broader range of applications than a single neuron. It can approximate any nonlinear function, with sufficient regularity. For instance, a (nonlinear) network with enough neurons can model any system with unknown nonlinearity. Conversely, the use of a single neuron requires that the nonlinearity in the modelled system be known in advance.

Because $y_H(k)$ is nonlinear w.r.t. the system parameters H, all the updating algorithms are subject to possible convergence towards a local (non global) minimum of the MS cost $J(H)$.

Furthermore, the feedback loops under investigation can generate unbounded signals, if the parameter vector H escapes its stability domain. Unfortunately, very little is known about this domain for two reasons :

(i) even for a linear recursive filter, the stability domain does not correspond to the classical criterion that the transfer function has all its poles inside the unit cercle. Indeed the updating algorithms $H(k-1) \rightarrow H(k)$ makes the filter non stationary and renders this criterion idle [30] ;

(ii) for a recursive semi-linear neural cell with the nonlinearity f in the loop, even with a fixed parameter H, the stability domain has not been investigated yet, except in a few specific cases [49].

In particular, an interesting open question is whether the memory M in the (M-RLMS) algorithm improves stability. This question is currently being investigated by the authors.

Finally, the approach of backward propagation used in the (NN) context yields new insights in the approach of adaptive recursive filtering, especially for non linear filtering. Similarly, the forward propagation approach typical of the (TF) context can bring new insights for the supervised training of (NN).

VI. CONCLUSION

The purpose of this paper was to set up a unified framework to deal with the gradient adaptation of certain important time linear filters and with the supervised training of neural networks using a finite number of arbitrarily ordered examples. In this framework, the similarities and dissimilarities are clearly apparent.

The major similarities lie in the optimization criteria and in the updating algorithms.

The major differences are in the nature and in the number of inputs $X(k)$ to the system.

The optimization criteria are indeed the same in both fields. The objective is to find the LS parameter H (corresponding to the least total squared output error at step k). In fact (even if it is implicit) there is always a probability distribution for the input $X(k)$. When the total number of inputs is large, this LS objective becomes equivalent to finding the LMS parameters H (corresponding to the least mean squared output error) in other words minimizing the MS cost $J(H)$.

In both contexts, for a large number of inputs, this minimization can be performed recursively using a variable filter $H= H(k)$ which obeys the stochastic LMS gradient algorithm given by eq. (3.23) in the text. Recursive updating algorithms are especially appreciated since they perform one updating each time a new input $X(k)$ occurs (with the associated reference output $d(k)$). Except in the specific case of a transversal linear time filter, the (STLMS) algorithm has essentially no competitor neither in (TF) nor in (NN). It is simple and yet ensures asymptotic optimality. It is known under the two popular denominations of "gradient" or "LMS" algorithm.

One major difference between the adaptation of (TF) and the training of (NN) lies in the nature of input samples. In the former case, $X(k)$ is a sliding window of a time process. This means that all

the coordinates of $X(k)$ play the same role, that the successive vectors $X(k)$ are naturally ordered, and that $X(k-1)$ and $X(k)$ have a great deal of similarities since they share all but one coordinates. In the (NN) training case, in general, none of these feature holds. In particular, the order of successive inputs $X(k)$ is arbitrary and this might influence the convergence process of the recursive algorithm $H(k)$. Another major difference is that the number of inputs is normally infinite (like the time) in (TF) but finite (like the collection of examples) in (NN) applications such as classification. This explains why certain (NN) updating algorithms are iterative but non recursive. For such algorithms, called "block" algorithms, implementation of all the parameter updatings (even the first one) requires the knowledge of the whole collection K of inputs. From the point of view of distributing the computing power, this can be viewed as inefficient. Therefore, in (NN) as well as in (TF), recursive algorithms are often preferred to block algorithms.

With the objective of comparing the approaches used in (TF) and in (NN) in order to implement the recursive (STLMS) algorithm, we have investigated in detail two categories of systems,

- systems with a modular repetitive structure,
- systems with a recursive (feedback) structure.

For the first structure, typical examples are cascaded (TF) and multilayer (NN). In the former case, the (STLMS) algorithm is easy to design and its computational complexity is low, namely $(3N + 1)$ multiplications per step where N is the total number of parameters. In the latter case, the (STLMS) algorithm can be written down following either a forward or a backward propagation approach in order to evaluate the gradient. It is the backward propagation approach which is the most similar to (TF) because it has a low complexity of approximately $4N$ multiplications per step. In fact, we have shown that cascaded filters constitute a specific case of linear multilayer networks, where the input is a time signal. In this case, there is a complete equivalence between the classical adaptive filtering approach, and the backward propagation approach used for training (NN).

For the structures which include a feedback loop, the similarities between (TF) and (NN) are still stronger, at least in the context where a novel input information is presented to the network each time the feedback loop of the net is implemented. In this context, the input is a time process and the

problem is to train the network for it to learn the time evolution of $X(k)$. For a single neural cell, if it were not for the nonlinear function included in the loop, the problem would merely be a time filtering problem. Again we find a complete equivalence between the classical adaptive filtering approach (for recursive filters) and the approach used in (NN) which is backward propagation. One major difficulty to exactly implement the (STLMS) is the indefinitely increasing size of the memory, the solution to this problem being to truncate the memory to a finite value.

Let us notice that the STLMS algorithms presented in this paper for updating the (TF), either transversal filters or recursive filters has been successfully generalized to certain non linear filters [24],[25]. As shown in this paper, these algorithms typical of (TF) and referred here to as the forward propagation approach can be applied for training (NN). Conversely, the backward propagation approach typical of (NN) could yield new insights in the context of non-linear filters.

In brief, we have shown that each time a linear (TF) problem can be embedded into the (NN) formalism, the gradient backward propagation approach is equivalent to the filtering methods. The use of backward propagation is justified for solving non linear time filtering problems, only.

In the specific case of transversal filters, many theoretical results are known about convergence of the (STLMS) algorithm. But in other cases, even with linear filters and a fortiori with non linear networks, the theoretical convergence results about the (STLMS) algorithm are very scarce in the literature. In particular, very little is known about the extra difficulty added by the non linearity. Another problem is the possible benefit for stability to increase the memory of the recursive LMS algorithm. This question and others constitute an open field of investigation.

ACKNOWLEDGMENT

This work was partially supported by the Direction des Constructions Navales under contract C9048603005 from GERDSM. The authors wish to thank Doctor Messaoud Benidir and Olivier Nerrand for helpful suggestions.

REFERENCES

[1] B. WIDROW, M.E. HOFF "Adaptive Switching Circuits" IRE WESCON Conven. Rec., p. 96-104, part 4, Sept. 1960.

[2] R.P. LIPPMAN "An Introduction to Computing with Neural Nets", IEEE ASSP Magazine, p. 4-22, 1987.

[3] D.E. RUMELHART, G.E. HINTON, R.J. WILLIAMS "Learning internal Representation by Error Propagation" in "Parallel Distributed Processing : Explorations in the Microstructure of Cognition. Vol. 1. Foundations, MIT Press, 1986, D. Rumelhart, J. McCleveland Editors.

[4] F. FOGELMAN-SOULIE, P. GALLINARI, Y. LE CUN, S. THIRIA "Automata Networks and Artificial Intelligence" In "Automata Networks in Computer Sciences" Manchester University Press, 1988.

[5] P.J. WERBOS "Backpropagation through time : What it does and How to do it", Proceedings of The IEEE, Vol.78, N°10, Oct. 1990.

[6] B. WIDROW, S.D. STEARNS "Adaptive Signal Processing", Prentice Hall, 1985.

[7] M.L. HONIG, D.G. MESSERSCHMIT "Adaptive Filters, Structures, Algorithms, and Applications", Kluwer Academic Publishers, 1984.

[8] S.T. ALEXANDER "Adaptive Signal Processing. Theory and Applications", Springer Verlag, 1986.

[9] M. BELLANGER "Adaptive Digital Filters and Signal Analysis" Marcel Dekker, New York, 1987.

[10] B. MULGREW, C.F.N. COWAN "Adaptive Filters and Equalisers", Kluwer Academic Publishers, 1988.

[11] A. LAPEDES, R. FARBER "Nonlinear Signal Processing Using Neural Networks : Prediction and System Modelling", Internal Report of the Los Alamos National Laboratory, July 1987.

[12] R.J. WILLIAMS, D. ZIPSER "A learning algorithm for continually running fully recurrent neural networks", Neural computation, vol.1, p. 270-280, 1989.

[13] D.D. NGUYEN, J.S.J. LEE "A New LMS-Based Algorithm for Rapid Adaptive Classification in Dynamic Environments", Neural Networks, Vol.2, p.215-228, 1989

[14] L. PERSONNAZ, O. NERRAND, G. DREYFUS "Apprentissage et Mise en Oeuvre de Réseaux de Neurones Bouclés", Proc. Journées Internat. des Sciences Inform. Tunis, Mars 1990.

[15] A.J. ROBINSON, F. FALLSIDE "The Utility Driven Dynamic Error Propagation Network", Internal Report, 1987

[16] J.A. CADZOW "Signal Processing via Least Squares Error Modeling", IEEE ASSP Magazine, p.12-31, Oct. 1990.

[17] O. MACCHI, E. EWEDA "Second Order Convergence Analysis of Stochastic Adaptive Linear Filtering", IEEE Trans. on Automatic Control, Vol.28, N°1, p. 76-85, 1983.

[18] F. FOGELMAN-SOULIE, P. GALLINARI, Y. LE CUN, S. THIRIA "Network Learning", Machine Learning, vol. 3, Kodratoff Michalski Editors, 1988.

[19] O. MACCHI "Advances in Adaptive Filtering", in "Digital Communications", E. Biglieri, G. Prati Ed., North-Holland, p. 41-57, 1986.

[20] S. MARCOS, O. MACCHI "Joint Adaptive Echo Cancellation and Channel Equalization for Data Transmission", Signal Processing 20, p.43-65, 1990.

[21] J.M. TRAVASSOS-ROMANO, M. BELLANGER, L.C. CORADINE "Least Squares Adaptive Filter in Cascade Form for Line Pair Spectrum Modelling", Proc. EUSIPCO 90, Barcelone, p. 249-252, 1990.

[22] U. FORSSEN "Adaptive Filters with Arbitrary Structures", Internal Report of the Royal Institute of Technology, Stockholm, Feb. 1990.

[23] I.D. LANDAU "Adaptive Control : the Model Reference Approach", Marcel Dekker, New York, 1979.

[24] J.R. TREICHLER, C.R. JOHNSON, M.G. LARIMORE "Theory and Design of Adaptive Filters", ch 5, Wiley Interscience, John Wiley and Sons, 1987.

[25] N.S. JAYANT, P. NOLL "Digital Coding of Wavefronts : Principles and Applications to Speech and Video", Prentice-Hall, Englewood Cliffs, N.J., 1984.

[26] O. MACCHI "A Common Formalism for Adaptive Identification in Signal Processing and in Control" paper accepted Proc. of IEEE, Nov. 1990.

[27] S.D. STEARNS "Error Surfaces of Recursive Adaptive Filters", IEEE Trans. on ASSP, Vol.ASSP-29, p.763-766, June 1981.

[28] J.J. SHYNK "Adaptive IIR filtering" IEEE ASSP Magazine, p. 4-21, 1989.

[29] P.L. FEINTUCH "An Adaptive Recursive LMS Filter" Proc. IEEE, p. 1622-1624, 1976.

[30] M. JAIDANE, O.MACCHI "Stability of Adaptive Recursive Filters", Proc. ICASSP, New York, p. 1503-1505, 1988.

[31] C.R. JOHNSON "A Convergence Proof for a Hyperstable Adaptive Recursive Filter", IEEE Trans. on IT, Vol.IT-25, N°6, pp.745-749, Nov. 1979.

[32] C.R. JOHNSON, I.D. LANDAU "On Adaptive IIR Filters and Parallel Adaptive Identifiers with Adaptive Error Filtering", Proc. ICASSP, p. 538-541, 1981.

[33] R. ROSENBLATT, Principles of Neurodynamics, New York, Spartan Books, 1959.

[34] P.C. TRELEAVEN "Neurocomputers", Internat. Jour. Neurocomputing, vol.1, n° 1, p. 4-31, 1989.

[35] J.J. SHYNK, "Performance Surfaces of a Single-Layer Perceptron", IEEE Trans. on Neural Nets, vol. 1, n° 3, p. 268-274, 1990.

[36] J.J. SHYNK, N.J. BERSHAD "Steady-State Analysis of a Single-Layered Perceptron Based on a System Model with Bias Terms", submitted IEEE Trans. ASSP, 1990.

[37] N.J. BERSHAD, J.J. SHYNK, P.L. FEINTUCH "Statistical Analysis of the Single-Layer Backpropagation Algorithm for Identification of a Nonlinear System with Gaussian Inputs", paper submitted, oct. 1990.

[38] Y. LE CUN "Modèles Connexionnistes de l'Apprentissage", Thèse de l'Université Paris VI, 1987.

[39] G.E. HINTON "Connectionist Learning Procedures" Machine Learning, vol. 3, Kodratof Michalski Editors, 1988.

[40] P. GALLINARI, S. THIRIA, F. FOGELMAN-SOULIE "Multilayer Perceptrons in Data Analysis", Proc. ICNN, San Diego, 1988.

[41] B. WIDROW, R.G. WINTER, R.A. BAXTER "Learning Phenomena in Layered Neural Networks", Proc. 1st Int. Conf. Neural Networks, p. 411-429, San Diego, 1987.

[42] A. PETROWSKI, L. PERSONNAZ, G. DREYFUS, C. GIRAULT "Implantation de Réseaux de Neurones Formels sur une Architecture Multiprocesseurs", 1er Colloque Européen sur les hypercubes et calculateurs distribués, Rennes, France, Oct.1989.

[43] C. VIGNAT, F. ROZYCKI "Réseaux de Neurones et Filtres Adaptatifs", Internal Report of the Laboratoire des Signaux et Systèmes, Gif sur Yvette, France, Sept. 1989.

[44] S. MARCOS, C. VIGNAT, O. MACCHI "L'Algorithme de Rétropropagation du Gradient en Réseaux de Neurones Comparé aux Algorithmes Adaptatifs du Traitement du Signal", Internal report of the Laboratoire des Signaux et Systèmes, Gif sur Yvette, France, July 1990.

[45] J.J. HOPFIELD "Artificial neural networks" IEEE Circuits and Devices Magazine, p. 3-10, sept 1988.

[46] R. ROHWER, S. RENALS "Training Recurrent Networks", Proc. nEURO88, L. Personnaz and G. Dreyfus, Editors, Paris, June 1988.

[47] F.J. PINEDA "Generalization of Back-Propagation to Recurrent Neural Networks", Physical Review Letters, Vol.59, N°19, p. 2229-2232, Nov. 1987.

[48] L.B. ALMEIDA "A Learning Rule for Asynchronous Perceptrons with Feedback in a Combinatorial Environment", Proc. First ICNN, San Diego, June 1987.

[49] C. UHL, O. MACCHI "When is DPCM a stable system ?", Proc. ICASSP, Albuquerque, p.2747-2750, April 1990.

APPENDIX

Equivalence between the filtering formula (5.27) and the network formula (5.31) in the case of cascaded linear filters.

This appendix proves that the simple product of y^{l-1}_{j} by g^{l}_{i} appearing in the network gradient $u^{l}_{i,j}$ of (5.31) is the convolution product of (5.27), when the constraints (5.26).(5.28) (5.29) hold.

In order to perform the cascaded filtering of fig. 5, the layered network must have the particular structure of fig. 10, for which the actual constraint (5.28) is in fact

$$h^{l}_{i,j} = \begin{cases} 0 & \text{if } j - i \geq M_l \\ h^{l}_{p} & \text{if } 0 \leq p = j - i \leq M_l - 1 \end{cases} \qquad \text{(A.1)}$$

With (A.1), the relation (5.32) is rewritten

$$g_i^l = \sum_{i_{l+1}=0}^{M_{l+1}-1} h_{i_{l+1}}^{l+1} \; g_{i-i_{l+1}}^{l+1} \quad , \qquad \text{(A.2)}$$

which, by an expansion until the last layer # L, gives

$$g_i^l = \sum_{i_{l+1}=0}^{M_{l+1}-1} \sum_{i_{l+2}=0}^{M_{l+2}-1} \cdots\cdots \sum_{i_L=0}^{M_L-1} h_{i_{l+1}}^{l+1} \, h_{i_{l+2}}^{l+2} \cdots\cdots$$

$$h_{i_L}^L \quad g_{i-i_{l+1}-i_{l+2}-\ldots-i_L}^L \; . \qquad \text{(A.3)}$$

In the same way, the output on cell j of layer $l-1$, y_j^{l-1} given by (5.9) (5.10) (5.11), is rewritten in the linear case and by using constraint (A.1) :

$$y_j^{l-1} = \sum_{i_{l-1}=0}^{M_{l-1}-1} h_{i_{l-1}}^{l-1} \; y_{j+i_{l-1}}^{l-2} \quad . \qquad \text{(A.4)}$$

Running recursion (A.4) from the input layer # 0 provides

$$y_j^{l-1} = \sum_{i_{l-1}=0}^{M_{l-1}-1} \sum_{i_{l-2}=0}^{M_{l-2}-1} \cdots \sum_{i_1=0}^{M_1-1} h_{i_{l-1}}^{l-1} \, h_{i_{l-2}}^{l-2} \cdots h_{i_1}^1$$

$$\cdot y_{j+i_{l-1}+i_{l-2}+\ldots+i_1}^0 . \qquad \text{(A.5)}$$

Now it follows form (5.29) that for the example #k at the network input

$$y_{j+i_{l-1}+i_{l-2}+\ldots i_1}^0 (k) = x(k - i_{l-1} - i_{l-2} - \ldots - i_1 - j + 1) \; . \qquad \text{(A.6)}$$

Thus it is recognized on (A.5) that

$$y_j^{l+1}(k) = h^{l-1} \circ h^{l-2} \circ \ldots \circ h^1(x(k-j)) \quad \text{(A.7)}$$

The product (5.31) of $g_i^l(k)$ by $y_j^{l+1}(k)$ whose expressions are given by (A.3) and (A.5) (A.6), respectively, yields

$$u_{i,i+p}^{l}(k) = \sum_{i_L=0}^{M_L-1} \ldots \sum_{i_{l+1}=0}^{M_{l+1}-1} \sum_{i_{l-1}=0}^{M_{l-1}-1} \ldots \sum_{i_1=0}^{M_1-1} h_{i_L}^{L} \ldots h_{i_{l+1}}^{l+1}$$

$$h_{i_{l-1}}^{l-1} \ldots h_{i_1}^{1} \cdot g_{i-i_{l+1}-i_{l+2}-\ldots-i_L}^{L} \cdot x(k-i_{l-1}-i_{l-2}-\ldots-i_1-i-p+1) \quad \text{(A.8)}$$

In addition, it should also be noticed, according to (5.23), the only term g_j^L which is non zero is $g_1^L = 1$. This implies that in the summation (A.8) the indeces i_l satisfy the constraint

$$i - i_{l+1} - i_{l+2} - \ldots - i_L = 1 \quad . \quad \text{(A.9)}$$

Finally, putting (5.23) and (A.9) into (A.8) gives

$$u_{i,i+p}^{l}(k) = \sum_{i_L=0}^{M_L-1} \ldots \sum_{i_{l+1}=0}^{M_{l+1}-1} \sum_{i_{l-1}=0}^{M_{l-1}-1} \ldots \sum_{i_1=0}^{M_1-1}$$

$$h_{i_L}^{L} \ldots h_{i_{l+1}}^{l+1} \; h_{i_{l-1}}^{l-1} \ldots h_{i_1}^{1} \; x(k-p-i_L-\ldots-i_{l+1}-\ldots-i_{l-1}-i_1) \quad \text{(A.10)}$$

which is nothing but (5.27).

APPENDIX B

Derivation of the backpropagation rule for recurrent nets (eq. 5.47)

The purpose, here, is to evaluate recursively the derivatives $\frac{\partial z(k)}{\partial y(k-n)}$ on the basis of the $\frac{\partial z(k)}{\partial y(k-n+j)}$, $j = 1, \ldots, m$. Indeed, according to (5.38),

$$\forall n \geq 1 \; ; \quad \frac{\partial z(k)}{\partial y(k-n)} = \sum_{j=1}^{\mathrm{Inf}(n,m)} \frac{\partial z(k)}{\partial y(k-j)} \frac{\partial y(k-j)}{\partial y(k-n)} . \tag{B.1}$$

The partial derivatives $\frac{\partial z(k)}{\partial y(k-j)}$, $j = 1, \ldots, \mathrm{Inf}(n, m)$, are direclty evaluated from (5.38) and yield

$$\frac{\partial z(k)}{\partial y(k-j)} = b_j \; ; j = 1, \ldots, \mathrm{Inf}(n, m) . \tag{B.2}$$

In according, according to relation (5.38)

$$y(k-j) = f(z(k-j)) , \tag{B.3a}$$

$$z(k-j) = \sum_{l=1}^{m} b_l \, y(k-j-l) + \sum_{i=0}^{I-1} a_i \, x(k-j-l) . \tag{B.3b}$$

$$\forall j \leq n \; ; \quad \frac{\partial y(k-j)}{\partial y(k-n)} = f'(z(k-j)) \frac{\partial z(k-j)}{\partial y(k-n)} . \tag{B.4}$$

Clearly, from (5.38) and (B.3), it appears that

$$\frac{\partial z(k-j)}{\partial y(k-n)} = \frac{\partial z(k)}{\partial y(k-n+j)} . \tag{B.5}$$

It follows that

$$\forall n \geq 1 ; \frac{\partial z(k)}{\partial y(k-n)} = \sum_{j=1}^{\mathrm{Inf}(n,m)} b_j f'(z(k-j)) \frac{\partial z(k)}{\partial y(k-n+j)} , \qquad \text{(B.6)}$$

which is exactly (5.47) in the text.

Now, we check that the result (5.45), (B.6) is the same as in the filtering approach. By replacing (B.6) into (5.43), we get

$$u_i(k) = f'(z(k)) \left[\sum_{n=1}^{k} \sum_{j=1}^{m} b_j f'(z(k-j)) \frac{\partial z(k)}{\partial y(k-n+j)} \right.$$

$$\left. \frac{\partial y(k-n)}{\partial a_i} + \frac{\partial y(k-1)}{\partial a_i} \right] \qquad \text{(B.7)}$$

and by taking (B.5) (B.4) into account, it can be written

$$u_i(k) = f'(z(k)) \sum_{j=1}^{\mathrm{Inf}(n,m)} b_j \sum_{n=0}^{k} \frac{\partial y(k-j)}{\partial y(k-n)} \frac{\partial y(k-n)}{\partial a_i} \frac{\partial y(k-1)}{\partial a_i}$$

$$+ f'(z(k))\, x(k-i) . \qquad \text{(B.8)}$$

Hence, from (5.43)

$$u_i(k) = f'(z(k)) \left[\sum_{j=1}^{\mathrm{Inf}(n,m)} b_j \, u_i(k-j) + f'(z(k))\, x(k-i) \right. , \qquad \text{(B.9)}$$

which is the relation (5.40)