IJCNN91 (Seattle)

NEURAL NETWORK TRAINING SCHEMES FOR NON-LINEAR ADAPTIVE FILTERING AND MODELLING

O. Nerrand, P. Roussel-Ragot, L. Personnaz, G. Dreyfus
Ecole Supérieure de Physique et de Chimie Industrielles de la Ville de Paris
Laboratoire d'Electronique
10, rue Vauquelin
F - 75005 PARIS - FRANCE

S. Marcos, O. Macchi, C. Vignat
Laboratoire des Signaux et Systèmes, CNRS - ESE
Plateau du Moulon
F - 91192 GIF SUR YVETTE Cedex - FRANCE

ABSTRACT

Neural networks are often advertised as being "adaptive". However, until now, neural networks have been used mainly in a non-adaptive way: the network is first trained, and subsequently used. This is in contrast with linear adaptive filters, which undergo continual adaptation; they have been widely used for many years, and investigated in great depth. It is only recently that neural network training schemes for truly adaptive operation have been proposed.
The purpose of the paper is to show that there is a wide variety (i) of cost functions, (ii) of techniques for estimating their gradient, and (iii) of adaptive algorithms for updating the coefficients of neural networks used as non-linear adaptive filters. We discuss the choices that have to be made, and we show that the training algorithms suggested recently for feedback networks are very closely related to (and, in some cases, identical to) the algorithms used classically for adapting recursive filters.

PRINCIPLES OF ADAPTIVE FILTERING AND MODELLING

A general filter is defined as a system with external time-dependent inputs, outputs, and an input-output, possibly recursive, relationship. The task that the system is supposed to perform is defined by a set of inputs and of corresponding desired output values. Generally, the input-output relationship features parameters which are estimated in order for the system to perform the desired task, i.e. for the actual outputs of the filter to be "as close as possible" (in a sense which will be defined mathematically later) to the desired outputs.
Two cases of interest arise:
(i) *non-adaptive filtering:* if the task is satisfactorily defined in advance by a finite set of input/desired output pairs, the estimation of the parameters can be carried out prior to using the filter (in the terminology of neural networks, the estimation is called "training");
(ii) *adaptive filtering:* otherwise, the parameters are estimated permanently while the filter is being used; thus, "training" never stops. This is a highly desirable feature in many cases; for instance, in a speech predictor, the parameters of the predictor must adapt to the characteristics of speech, which are known to be non stationary.

In the following, we consider a specific, although very widely used, type of discrete-time filter: we assume that the M external inputs consist of the values $\{u(n),u(n-1),....,u(n-M+1)\}$ of a signal u at successive instants of time; in addition, the filter has a single output y(n). Two kinds of filters are usually considered:
(i) non-recursive (also termed "transversal") filters, whose input-output relation is in the form
$y(n) = \Phi[u(n), u(n-1), \dots , u(n-M+1)]$, (1)
(ii) recursive filters, in which the output signal is delayed and fed back to the inputs; in this case, the input-output relation is in the form
$y(n) = \Phi[u(n), u(n-1), \dots , u(n-M+1), y(n-1), y(n-2), \dots , y(n-N)]$, (2)
where N is the order of the filter.
In this paper, we shall study adaptation algorithms for non linear recursive filters; in this context, transversal filters may be considered as a special kind of recursive filters.

In a more general approach, the inputs {u(n)} and output {y(n)} may be vectors. In addition to having a single output, the filters considered here have a further specificity: the output is the only variable which is fed back to the input. The present approach can be generalized to neural networks with *arbitrary* feedback structure.

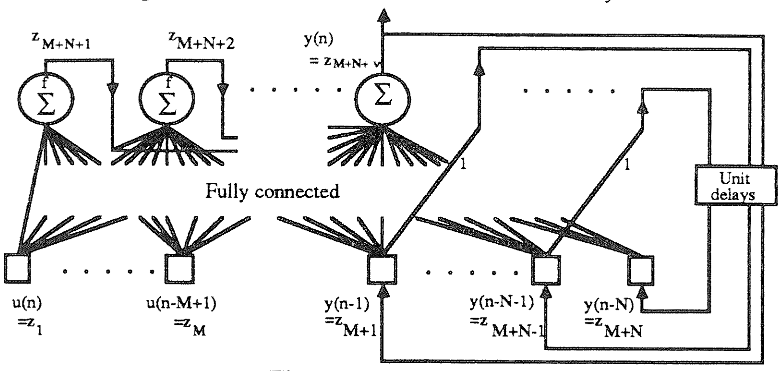

Figure 1.

An architecture of the type shown on Figure 1, featuring M external inputs, N feedback inputs and one output, can implement a fairly large class of functions Φ; the *non-recursive part* of the network (which implements function Φ) is a *fully-connected feedforward net.* This architecture is more general than the standard layered network; the latter makes sense in pattern recognition tasks (successive layers performing feature extraction and classification in a hierarchical way), but has no special significance for the filtering of time signals or for the modelling of physical processes. The basic building block of the filter is a "neuron", which performs a weighted sum of its inputs and computes an "activation function" f - usually non linear - of the weighted sum:

$$z_i = f_i[v_i] \quad \text{with } v_i = \sum_{j \in P_i} C_{ij}\, x_j \qquad (3)$$

where z_i denotes the output of neuron i, x_j denotes the j-th input of neuron i, and P_i is the set of indices of the inputs of neuron i. Note that the output neuron has a linear activation function.

As mentioned above, the task to be performed by the filter is defined by a (possibly infinite) set of inputs and corresponding desired outputs. At each sampling time n, an error e(n) is defined as the difference between the desired output d(n) and the actual output of the filter y(n): e(n) = d(n) - y(n). In general, d(n) is not known in advance: in a modelling process, the desired output of the network d(n) is the output of the process to be modelled at the same time n; in a predictor, it is desired that the output of the filter at time n be equal to the input signal at time n+1. The filter adaptation algorithms aim at finding the filter coefficients which minimize a given satisfaction criterion involving, usually, the squared error $e(n)^2$.
Thus, it is clear that adaptive filters and neural networks are formally equivalent, and that neural networks, which are potentially capable of realizing *non-linear* input-output relations, are simple generalizations of linear filters. In the next sections, we put into perspective the training algorithms developed for discrete-time recurrent neural networks and the algorithms used classically in adaptive filtering.

ADAPTATION CRITERION AND GRADIENT ESTIMATION TECHNIQUES

Criterion:

At time n, the most natural satisfaction criterion is: $J(n) = \frac{1}{n+1} \sum_{m=0}^{n} w(m)\, e(m)^2$.
In the stationary case, all weights w(m) can be taken equal to 1: J(n) is the best estimator of the expectation value of $e(m)^2$ (Mean Square Error criterion) [1, 2]. However, the amount of computation involved in this estimation grows unbounded. In addition, in many cases of practical interest, the system is not stationary, so that taking into account all the errors since the beginning of the optimization does not make sense; thus, one must weight the errors in order to implement a forgetting mechanism. In the present paper, we make all weights equal to 1 within a "sliding window" of length N_c; hence the following quantity:

$$J'(n) = \frac{1}{2} \sum_{m=n-N_c+1}^{n} e(m)^2 .$$

The choice of the length N_c of the window will be discussed below.

The present paper focusses on gradient estimation methods, whereby the updating of coefficients is given, at iteration k of the process, by

$$\Delta C_{ij}(k) = -\mu \left[\frac{\partial J'}{\partial C_{ij}}\right]_{C(k-1)} \quad (4)$$ where μ is the gradient step.

In the following, we give a presentation of the two techniques that can be used for gradient estimation.

Gradient estimation techniques:

As mentioned above, we have to compute, at time n, the gradient estimate

$$\left[\frac{\partial J'(n)}{\partial C_{ij}}\right]_{C(k-1)} = \left[\frac{\partial}{\partial C_{ij}}\left(\frac{1}{2}\sum_{m=n-N_c+1}^{n} e(m)^2\right)\right]_{C(k-1)} \quad (5) .$$

As a preliminary remark, one must notice that no assumption has been made as to the relationship between the iteration index k and time n. Several possibilities arise:

- *one can use a purely recursive algorithm,* whereby one iteration of the gradient (i.e. one modification of the coefficients) is performed per sampling interval n; in other words, k=n; this is the usual technique for the recursive adaptation of filters;
- *one can perform several modifications of C per sampling interval;* this may improve the tracking capabilities of the system in the non-stationary case, or speed up convergence to a minimum in the stationary case;
- *one can modify the coefficients after several sampling periods.*

For the gradient estimation to be meaningful, the coefficients C must be considered as being constant on the window of length N_c. Thus, the N_c errors {e(m)} appearing in relation (5) must be computed from a "training network", which is a static network made of N_c identical versions (hereinafter termed "copies") of the static part of the actual network. In the case of a purely recursive algorithm, the values of the coefficients used at time n are the coefficients C(n-1) which were computed at time n-1.

We introduce the following notation: all parameters computed from copy m of the training network will be denoted with superscript m. Hence, in the purely recursive case, the estimation (5) becomes

$$\frac{\partial}{\partial C_{ij}}\left[\frac{1}{2}\sum_{m=n-N_c+1}^{n} [e^m(n)]^2\right]_{C(n-1)} \quad \text{with } e^m(n) = d(m) - y^m(n) \quad (5') .$$

n denotes the actual time at which the computation is performed, so that $y^m(n)$ is the value of the state of the output neuron of copy m at time n (see figure 2, where M=2; N=3; ν=3; N_c=5).

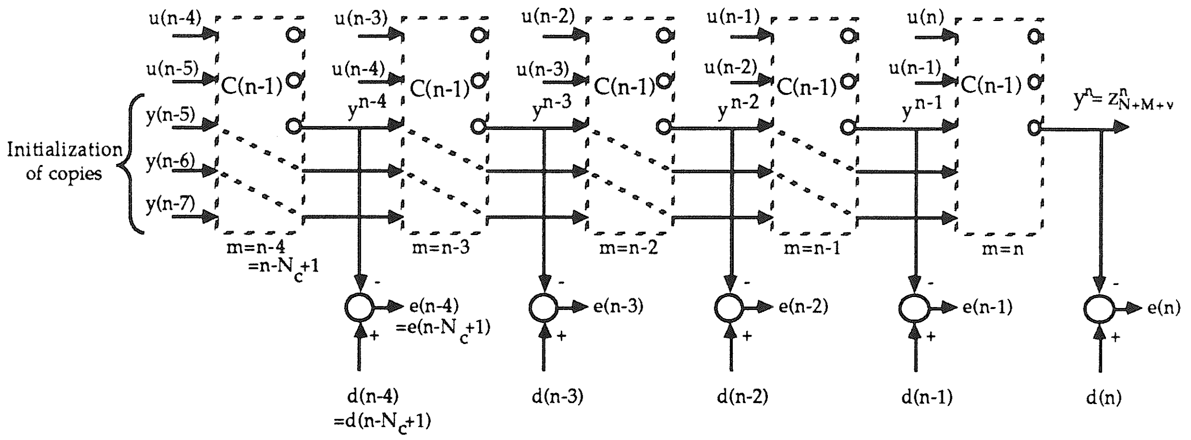

Figure 2.

Two techniques are available: a technique involving the forward computation of the gradient (commonly used in linear adaptive filtering) and a technique involving backpropagation (commonly used in neural network training). In both cases, we shall present the computations for the most general unconstrained architecture, in which the static part of the network is fully connected.

a) Forward computation of the gradient estimation at time n:

This technique was first suggested for the supervised training of neural networks in [3]; it is based on the following relation:

$$\left[\frac{\partial J'(n)}{\partial C_{ij}}\right]_{C(k-1)} = \left[\frac{\partial}{\partial C_{ij}}\left(\frac{1}{2}\sum_{m=n-N_c+1}^{n} e(m)^2\right)\right]_{C(k-1)} = -\sum_{m=n-N_c+1}^{n} e^m \frac{\partial y^m}{\partial C_{ij}} \,. \qquad (6)$$

Each term in the sum is related to one copy. Since all computations are performed *at time n*, we omit time, in the above relation as well as in the following ones.

For simplicity, we renumber the external inputs and the neurons of the network as indicated on Figure 1: we denote by z_1 to z_M the external inputs; we denote by z_{M+1} to z_{M+N} the feedback inputs; we denote by z_{M+N+1} to $z_{M+N+\nu}$ (where ν is the total number of neurons) the outputs of the neurons, ordered in the following way: neuron #p receives the outputs of neurons #(q<p) only.

With these notations, the outputs of the neurons of comy m are given by

$$z_p^m = f_p[v_p^m] \quad \text{with } v_p^m = \sum_{j\in P_p} C_{pj}\, z_j^m \quad (7),$$

and the derivative of the output of neuron #p of copy mwith respect to C_{ij} is given by:

$$\frac{\partial z_p^m}{\partial C_{ij}} = f_p'[v_p^m]\left[\delta_{pi}\, z_j^m + \sum_{h\in P_p} C_{ph} \frac{\partial z_h^m}{\partial C_{ij}}\right] \qquad (8) \qquad \text{with } p\in\{M+N+1, \ldots, M+N+\nu\} \text{ and } j<i,$$

where f_p denotes the activation function of neuron #p and δ_{pi} is the Kronecker symbol.

The recursion of relation (8) allows the forward computation of the derivatives of the outputs y^m of the N_c copies, the computations being carried out in the order of the numbering of the neurons, and from copy $n-N_c+1$ up to copy n, with:

(i) for p = 1 to M (external inputs):

$$z_p^m = u(m-p+1) \;;\; \frac{\partial z_p^m}{\partial C_{ij}} = 0 \;;$$

(ii) for p = M+1 to M+N (feedback inputs):

- for the first copy ($m=n-N_c+1$): *the values of z_p^m and of the partial derivatives must be chosen in order to initialize the computation;* for instance, one can use the desired output values at times $n-N_c$ to $n-N_c-N+1$:

  $$z_p^{n-N_c+1} = d(n-N_c-p+M+1)$$

  but other choices are possible. The values of the derivatives can be taken equal to zero, or alternatively one can use the values of the derivatives computed by the training network at the previous updating, with the coefficients C(k-1). We shall see that various choices lead to various algorithms;
- for the other copies ($m=n-N_c+2$ to n) one has

  for p=M+2 to M+N : $\quad z_p^m = z_{p-1}^{m-1}$

  for p=M+1 : $\quad z_{M+1}^m = z_{M+N+\nu}^{m-1} = y^{m-1}$

Once the N_c sets of partial derivatives of the output values y^m are obtained, relations (6) and (4) are used for computing the modifications of the coefficients.

b) Gradient estimation by backpropagation:

Backpropagation is a computational trick which can be used to obtain the same gradient estimate (5) as above, at time n [4-7] .

We consider the training network with N_c copies; first, $\{v_p^m\}$ and $\{z_p^m\}$ are computed from (7). We denote by C_{ij}^m the coefficient C_{ij} in copy m, so that we can make use of the following relation:

$$\frac{\partial J'(n)}{\partial C_{ij}^m} = \frac{\partial J'(n)}{\partial v_i^m}\frac{\partial v_i^m(n)}{\partial C_{ij}^m} = \frac{\partial J'(n)}{\partial v_i^m} z_j^m(n) \qquad \begin{array}{c}\text{for } i=M+N+1 \text{ to } M+N+\nu \\ \text{for } j=1 \text{ to } M+N+\nu-1 \\ \text{and } j<i\end{array}$$

Actually, the definition of the copies of the training network requires that all C_{ij}^m be equal. Thus, one has:

$$\Delta C_{ij}(n) = -\mu \frac{\partial J'(n)}{\partial C_{ij}} = -\mu \sum_{m=n-N_c+1}^{n}\left[\frac{\partial J'(n)}{\partial C_{ij}^m}\right] = -\mu \sum_{m=n-N_c+1}^{n}\left[\frac{\partial J'(n)}{\partial v_i^m}\right] z_j^m \,.$$

For a given copy m, we define the following set of variables q_i^m:

(i) for i=M+N+ν+N-1 down to M+N+ν+1:

$q_i^m = 0$ if m=n ; $q_i^m = q_{i-N-\nu+1}^{m+1}$ otherwise;

(ii) for i=M+N+ν (output neuron):

$q_i^m = e^n$ if m=n ; $q_i^m = e^m + q_{M+1}^{m+1}$ otherwise; (note that $q_i^m = -\frac{\partial J'}{\partial v_i^m}$) ;

(iii) for i = M+N+ν−1 down to M+N+1 (hidden neurons):

$q_i^m = f'_i(v_i^m) \sum_{h \in R_i} C_{hi}^m q_h^m$; $\left(q_i^m = -\frac{\partial J'}{\partial v_i^m} \right)$; where R_i is the set of indices of the neurons to which neuron i transmits its output;

(iv) for i = M+N (last feedback input):

$q_i^m = \sum_{h \in R_i} C_{hi}^m q_h^m$;

(v) for i=M+N-1 down to M+1 (other feedback inputs):

$q_i^m = \sum_{h \in R_i} C_{hi}^m q_h^m + q_{i+N+\nu}^m$.

Once the recursion has been carried out, the modifications of the coefficients at time n are computed as: $\Delta C_{ij}(n) = \mu \sum_{m=n-N_c+1}^{n} q_i^m z_j^m$ for i=M+N+1 to M+N+ν for j=1 to M+N+ν-1 and j<i

Note that computation by backpropagation implicitly assumes that the derivatives of the feedback inputs of the first copy with respect to the coefficients are equal to zero; this is in contrast with the forward computation of the gradient, where these values can be initialized arbitrarily.

c) Discussion

In the above section, we have shown that the gradient can be estimated either by forward computation or by backpropagation. In both cases, the total number of coefficients is equal to $\nu[(M+N)+(\nu-1)/2]$ if the static part of the network is fully connected.

For each copy, the forward computation involves $\nu[(\nu-1)/2+M+N][\nu(\nu+1)/2+\nu N+2]$ (i.e. $O(\nu^4)$) multiplications and the computation by backpropagation involves $(3/2)\nu^2+[2M+3N-(1/2)]\nu+N$ (i.e. $O(\nu^2)$). Thus, in general, the computational complexity of backpropagation is much smaller than the complexity of forward computation [8]. The consequences of this reduced complexity are discussed below.

ALGORITHMS

The present section discusses the various algorithms which result from various choices of criteria and of gradient estimation techniques. New algorithms are introduced, and the relations between the present work, the "Real Time Recurrent Learning" algorithm, and the "teacher forcing" technique are discussed.

Whichever gradient estimation technique is used, the following parameters must be chosen: (i) window length N_c, (ii) initial values z of the N feedback inputs of the first copy (state variable initialisation). In addition, as mentioned above, the initial values of the derivatives $\partial z/\partial C$ for the N feedback inputs of the first copy can be imposed to zero or non zero values in the case of forward computation, whereas these initial values are implicitly taken equal to zero if the computation is performed by backpropagation. It must be emphasized that both methods lead to the same result if the initial values of the derivatives are chosen equal to zero. If such a choice is not satisfactory, it is possible to take into account the dependency of the feedback inputs of copy #($n-N_c$) by using backpropagation with a number N_t of copies larger than N_c. This is reasonable because backpropagation is less computationally expensive than forward computation. It may also become possible to iterate the gradient descent between two consecutive sampling instants, which lead to a faster convergence rate. The choice of N_t and N_c results from a tradeoff between the accuracy of the estimation of the gradient and the computation time. In the stationary case, N_t and N_c are only limited by the computation time. In the non-stationary case, the optimal coefficients are time dependent and the typical time scale of the non-stationarity must be taken into account: N_t must be shorter than the time constant of the non-stationarity of the process (or than the mean duration of stationarity for a piecewise stationary process).

The *"Real-Time Recurrent Learning Algorithm"* [3] is a particular technique using the forward computation of the gradient estimation with $N_t=N_c=1$; the coefficient modifications are performed at each sampling instant n;

at time n, all the derivatives $\partial z/\partial C$ are computed using the N derivatives $\partial z/\partial C$ of the N state variables computed at time n-1 with coefficients C(n-1) (this algorithm is known as the Recursive Prediction Error algorithm, or IIR-LMS algorithm, in linear adaptive filtering). These drastic choices are justified if the coefficients change slowly, i.e. if the gradient step μ is small enough. It should be noticed that, with the same computation time, it might be possible to use backpropagation with several copies.
The above adaptation algorithms are complex non linear feedback processes which may lead to unstable behaviours. In order to generate stable adaptation algorithms, it is possible to modify the above algorithms in the following way: *the feedback inputs of copy m are no longer the outputs of copy m-1, but they are the desired values of these outputs.* This technique was introduced for the supervised training of neural networks under the name of *teacher forcing*.[9]. This family of algorithms is well known in Process Identification [10] (Series-Parallel algorithms) and in Adaptive Filtering (Equation-Error algorithms).

CONCLUSION

In this paper, we have shown that a large variety of techniques are available for training recurrent neural networks to perform adaptive filtering and modelling. The techniques used thus far are but a small fraction of the available possibilities. Neural networks, viewed as adaptive non-linear filters, have a considerable potential which needs be explored, and basic issues, such as the stability of the algorithms, are still open.

Acknowledgments
This work was supported in part by BRAIN contract ST2J O422 and by GRECO TdSI.

References

[1] Widrow B., Stearns S.D., Adaptive Signal Processing (Prentice-Hall, 1985).
[2] Mulgrew B, Cowan C.F.N., Adaptive Filters and Equalisers (Kluwer Academic Publishers, 1988).
[3] Williams R.J., Zipser D., A Learning Algorithm for Continually Running Fully Recurrent Neural Networks, Neural Computation 1, 270-280 (1989).
[4] Rumelhart, D. E., Hinton, G. E., and Williams R. J., in Parallel Distributed Processing 1 (M.I.T. Press, 1986).
[5] Robinson, A. J. and Fallside, F., in Anderson, D.Z. , ed., Neural Information Processing Systems (American Institute of Physics, 1988).
[6] Pineda, F., Generalization of Backpropagation to Recurrent Neural Networks, Phys. Rev. Lett. **59**, 2229 (1987).
[7] Werbos P.J., Backpropagation Through Time: What It Does And How To Do It, Proc. IEEE **78**, 1560 (1990).
[8] Pineda F., Recurrent Backpropagation and the Dynamical Approach to Adaptive Neural Computation, Neural Computation **1**, 161 (1989).
[9] Jordan M.I., Attractor Dynamics and Parallelism in a Connectionist Sequential Machine, Proc. Eighth Annual Conference of the Cognitive Science Society, 531 (1986).
[10] Narendra K.S., Parthasarathy K., Identification and Control of Dynamical Systems Using Neural Networks, IEEE Trans. on Neural Networks vol 1 n°1 p4-27 (1990).