RESEAUX DE NEURONES POUR LE FILTRAGE NON LINEAIRE ADAPTATIF

S. Marcos[*], P. Roussel-Ragot[**], L. Personnaz[**],
O. Nerrand[**], G. Dreyfus[**], C. Vignat[*]

[*]Laboratoire des Signaux et Systèmes
Ecole Supérieure d'Electricité
Plateau de Moulon
91192 GIF SUR YVETTE

[**]Ecole Supérieure de Physique et de Chimie Industrielles de la Ville de Paris
Laboratoire d'Electronique
10, rue Vauquelin
75005 PARIS

RESUME :

Nous introduisons une famille d'algorithmes adaptatifs permettant l'utilisation de réseaux de neurones comme filtres adaptatifs non linéaires, systèmes susceptibles de subir un apprentissage permanent à partir d'un nombre éventuellement infini d'exemples présentés dans un ordre déterminé. Ces algorithmes, fondés sur des techniques d'évaluation du gradient d'une fonction de coût, s'inscrivent dans un cadre différent de celui de l'apprentissage "classique" des réseaux de neurones, qui est habituellement non adaptatif.

I. INTRODUCTION

Le présent article, ainsi qu'un article publié précédemment [1], entrent dans le cadre d'un effort de clarification des relations conceptuelles qui existent entre l'apprentissage des réseaux de neurones et l'adaptation des filtres ; l'approche que nous décrivons ici nous permet d'introduire une famille de nouveaux algorithmes adaptatifs pour l'apprentissage des réseaux de neurones formels, qui sont susceptibles de trouver des applications originales en filtrage adaptatif non linéaire.

Un réseau de neurones formels est un ensemble de cellules non linéaires élémentaires interconnectées. Chaque connexion est caractérisée par un coefficient ou poids synaptique, et le comportement du réseau dépend essentiellement des valeurs de ces poids ; l'*apprentissage* est l'opération par laquelle les poids synaptiques sont calculés de telle manière que le système réalise bien la fonction qu'on attend de lui. Dans tout ce qui suit, nous nous plaçons dans le cadre d'un apprentissage au cours duquel on cherche à calculer les poids de manière à satisfaire à un critère qui fait intervenir la différence entre la réponse du réseau et une réponse désirée. Il a été prouvé que les réseaux de neurones formels sont des approximateurs universels [2], en ce sens que toute fonction multivariable non linéaire suffisamment régulière peut être approchée,

avec une précision fixée, par un réseau de neurones, pourvu que celui-ci possède une architecture et une taille appropriées, et qu'il soit soumis à un apprentissage efficace ; ces deux points - architecture et apprentissage - sont essentiels dans toute recherche sur les réseaux de neurones. Dans la mesure où l'approximation de fonctions et l'identification de signaux ou de systèmes sont au cœur des préoccupations du filtrage adaptatif et de celles de la commande adaptative, il est naturel de chercher à évaluer l'apport potentiel des réseaux de neurones dans ces domaines, notamment pour la réalisation d'opérations de filtrage non linéaire en traitement du signal.

Le cadre conceptuel dans lequel nous nous plaçons est très différent du cadre "classique" d'apprentissage et d'utilisation des réseaux de neurones ; dans le domaine de la classification, par exemple, l'apprentissage est effectué de manière *non adaptative* puisque l'ensemble d'apprentissage est connu à l'avance, de taille finie, et que l'ordre de présentation des exemples est arbitraire ; une fois l'apprentissage terminé, le réseau de neurones est muni des coefficients calculés durant la phase d'apprentissage, et il est utilisé sans modification de ces coefficients. Bien entendu, il est possible de traiter des signaux temporels de manière non adaptative : c'est ce qui a été fait dans la plupart des recherches concernant la reconnaissance de la parole par des réseaux de neurones, ainsi que la prédiction de séries temporelles par ces réseaux. C'est ainsi qu'ont été introduits les "Time Delay Neural Networks" (TDNN), qui ont une architecture de filtres transverses non linéaires mis en cascade, utilisés, par exemple, pour la reconnaissance de phonèmes [3] ; leur apprentissage n'est pas adaptatif.

Nous désignons par réseau de neurones *adaptatif* un réseau de neurones dont *l'apprentissage est effectué en permanence,* pendant son utilisation : ainsi, un prédicteur de parole reçoit en permanence un signal de parole échantillonné, et doit s'adapter en permanence aux caractéristiques du signal, lesquelles ne sont pas connues à l'avance et peuvent varier au cours du temps. L'apprentissage est donc effectué à partir d'un ensemble de données, éventuellement infini, dont l'ordre de présentation est imposé par la nature temporelle des informations traitées. Dans cette optique, nous montrons que les réseaux de neurones adaptatifs ne sont pas autre chose que des filtres adaptatifs non linéaires, qui peuvent être non bouclés (filtres transverses) ou bouclés (filtres récursifs). Il y a eu très peu d'études concernant des réseaux de neurones adaptatifs; nous montrons, dans le présent article, que les algorithmes proposés jusqu'à présent sont des cas particuliers des algorithmes généraux que nous introduisons ici .

Dans une première partie, nous rappelons quelques définitions concernant les filtres adaptatifs, et montrons qu'un filtre adaptatif linéaire, transverse ou récursif, peut être considéré comme un *neurone linéaire* unique. Bien entendu, les réseaux de neurones n'offrent aucun apport spécifique dans le cadre du filtrage linéaire, sauf dans un cas particulier étudié en détail dans [1]. Nous abordons ensuite les deux points essentiels mentionnés plus haut : architecture et apprentissage. Nous montrons tout d'abord qu'il est possible d'utiliser des réseaux de neurones comme des filtres non linéaires, transverses (réseaux non bouclés) ou récursifs (réseaux bouclés), et nous introduisons la notion de forme canonique d'un réseau. Nous proposons ensuite, dans un cadre très général, des algorithmes d'apprentissage adaptatifs originaux, et nous montrons les relations qui existent entre ces nouveaux algorithmes d'une part, et, d'autre part, les

algorithmes utilisés classiquement en filtrage, ainsi que les algorithmes adaptatifs d'apprentissage des réseaux de neurones qui ont été proposés par d'autres auteurs.

II. LES FILTRES ADAPTATIFS

A une suite de données d'entrée, ordonnées dans le temps, $\{u(0), u(1), u(2), \ldots, u(n), \ldots\}$, un filtre fait correspondre une suite de données de sortie $\{y(0), y(1), y(2), \ldots, y(n), \ldots.\}$. Selon le type de relation établie entre ces deux suites, on distingue les filtres transverses ou récursifs, linéaires ou non linéaires. Dans le cas d'un filtre linéaire transverse de mémoire M, la sortie $y(n)$ s'écrit à chaque instant n,

$$y(n) = C^{\mathrm{T}}U(n), \qquad \text{(II.1)}$$

avec

$$U(n) = [u(n), u(n-1), \ldots, u(n-M+1)]^{\mathrm{T}} \qquad \text{(II.2)}$$

et $C = [c_0, c_1, \ldots, c_{M-1}]^{\mathrm{T}}$.

Dans le cas d'un filtre linéaire récursif, la sortie $y(n)$ est déterminée par la relation

$$y(n) = C^{\mathrm{T}}X(n), \qquad \text{(II.3)}$$

avec

$$X(n) = [u(n), u(n-1), \ldots, u(n-M+1), y(n-1), \ldots, y(n-N)]^{\mathrm{T}} \qquad \text{(II.4)}$$

Le vecteur $C = [c_0, c_1, \ldots, c_{M-1}, c_M, \ldots, c_{M+N-1}]^{\mathrm{T}}$ des coefficients du filtre est donc de dimension $M + N$, M et N désignant respectivement les dimensions des parties transverse et récursive du filtre.

La tâche que doit réaliser un filtre, et donc la valeur des coefficients de ce filtre, sont définis par un critère d'optimisation. Si les coefficients sont remis à jour en permanence, pendant l'utilisation du filtre, en fonction de chaque nouvelle information disponible à son entrée, celui-ci est dit adaptatif.

La plupart des filtres adaptatifs actuels sont *linéaires*. Ils peuvent donc être mal adaptés à la modélisation de systèmes non linéaires, à la détection de signaux, ou à l'estimation de signaux non gaussiens. Plusieurs solutions ont été proposées pour tenter d'introduire des non linéarités dans le traitement du signal classique : généralisation des filtres linéaires à des filtres polynomiaux (Volterra) [4], fonctions de base radiales [5].

Dans cet article, nous montrons que les réseaux de neurones constituent une nouvelle famille de filtres non linéaires, transverses ou récursifs, et nous introduisons des algorithmes d'adaptation de ces filtres.

III. LES RESEAUX DE NEURONES UTILISES COMME FILTRES NON LINEAIRES

III.1 Modèle de neurone formel

Un neurone formel est une cellule élémentaire de calcul dont la structure est donnée par la Figure 1. On appelle sortie du neurone *i* la quantité

$$z_i = f_i(v_i) \ , \qquad \text{(III.1)}$$

et potentiel du neurone la quantité

$$v_i = \sum_{j \in P_i} c_{ij} \, z_j \qquad \text{(III.2)}$$

où $\{c_{ij}\}$ est l'ensemble des poids synaptiques (ou des connexions), $\{z_j\}$ est l'ensemble des entrées du neurone *i* et f_i est une fonction non linéaire dite "fonction d'activation". P_i est l'ensemble des indices des entrées du neurone *i.* La fonction f_i peut être quelconque ; cependant, les auteurs utilisent généralement des fonctions impaires, croissantes, telles que des fonctions signe ou sigmoïde. On supposera dans la suite que f_i est dérivable et bornée.

Il est clair qu'un filtre tranverse obéissant à la relation (II.1) peut être réalisé par un "neurone linéaire" unique (f_i=identité), possédant *M* entrées dont les valeurs à l'instant *n* sont $\{u(n), u(n-1), \ldots, u(n-M+1)\}$, et dont la sortie est $y(n)$. On remarque également qu'un filtre récursif obéissant à la relation (II.3) peut être considéré comme un "neurone linéaire" unique, comportant *M* entrées externes $\{u(n), u(n-1), \ldots, u(n-M+1)\}$, et *N* entrées de bouclage $\{y(n\text{-}1), \ldots, y(n-N)\}$.

Pour réaliser des fonctions plus complexes de filtrage non linéaire, il est naturel d'envisager l'utilisation d'un *réseau* de neurones formels, c'est-à-dire d'un ensemble de cellules élémentaires définies ci-dessus, connectées entre elles : un neurone *i* reçoit des informations $\{z_j\}$ en provenance d'autres neurones et/ou d'entrées externes du réseau. L'architecture du réseau définit une relation entre les entrées externes et la sortie : les réseaux non bouclés permettent de réaliser des filtres transverses non linéaires, et les réseaux bouclés permettent de réaliser des filtres récursifs non linéaires.

III.2 Réseaux de neurones non bouclés utilisés comme filtres transverses

Dans un réseau non bouclé, l'information circule uniquement des entrées du réseau vers sa sortie : il n'y a pas de boucle de retour. Un réseau non bouclé dont les entrées à l'instant *n* sont les valeurs successives $u(n), u(n-1), \ldots, u(n-M+1)$ peut alors être considéré comme un filtre transverse non linéaire dont la sortie à l'instant *n* obéit à la relation :

$$y(n) = \Phi(u(n), u(n-1), \ldots, u(n-M+1)) \qquad \text{(III.3)}$$

où Φ représente la fonction non linéaire globalement réalisée par le réseau. On voit apparaître clairement l'intérêt potentiel des réseaux de neurones formels pour réaliser des opérations de filtrage, dans la mesure où un réseau de neurones peut, au moins en principe, approcher par apprentissage n'importe quelle fonction non linéaire suffisamment régulière.

L'architecture la plus générale d'un réseau de neurones utilisé comme filtre transverse est celle d'un réseau complètement connecté (Figure 2). L'évolution d'un tel réseau, comprenant ν neurones, M entrées externes et une sortie, dont le résultat global est exprimé par (III.3), est régie à l'instant n par les équations suivantes :

$$z_i(n) = u(n-i+1)\ ; \qquad i = 1, \ldots, M,$$

$$z_i(n) = f_i(v_i(n))\ ,$$

$$v_i(n) = \sum_{j<i} c_{ij} z_j(n)\ \ ; \qquad i = M+1, \ldots, M+\nu\ ,$$

$$y(n) = z_{M+\nu}(n)\ . \qquad \text{(III.4)}$$

Les entrées externes ou les sorties des neurones ont été indifféremment appelées $z_i(n)$ et sont discernables par leurs indices. En l'absence de contrainte particulière sur l'amplitude du signal de sortie du système, on choisit pour $f_{M+\nu}$ la fonction identité.

L'architecture d'un réseau non bouclé, c'est à dire la topologie des connexions, peut être complètement ou partiellement imposée a priori par le problème à traiter : celui-ci impose en effet la suite des valeurs des signaux d'entrée et celle des sorties désirées, et, de surcroît, les connaissances a priori sur le problème peuvent donner des indications qui permettent de concevoir une architecture de réseau bien adaptée. Si cette architecture comporte des retards [6], il est toujours possible de représenter le réseau sous la forme canonique (III.3). Le passage d'une structure de réseau non bouclé quelconque à sa forme canonique, et en particulier la détermination de la taille M de la mémoire, sont détaillés dans [7]. Par exemple, il arrive que certaines applications en traitement du signal [8, 9] imposent des structures de plusieurs filtres adaptatifs mis en cascade ; l'optimisation conjointe de tous ces systèmes mis en cascade peut alors être envisagée. La représentation d'une cascade de filtres transverses linéaires ou non linéaires par un réseau non bouclé multicouche est étudiée dans [1] et ne sera pas reprise ici. Remarquons simplement qu'une cascade de filtres transverses n'a d'intérêt que si les filtres sont non linéaires ou bien si le problème impose la structure en cascade. Dans le cas contraire, des filtres linéaires mis en cascade (ou réseau linéaire multicouche) sans

que cela soit imposé par le problème, agissent simplement comme un seul filtre linéaire (ou réseau linéaire monocouche) adaptatif d'ordre plus élevé.

Si l'on ne dispose d'aucune connaissance a priori sur la structure à imposer au réseau, on peut utiliser une architecture de réseau non bouclé complètement connecté du type de celui de la Figure 2.

III.3 Réseaux de neurones bouclés utilisés comme filtres récursifs

Les filtres récursifs ont été introduits pour deux raisons essentielles : (i) la récursivité confère au filtre une mémoire infinie avec un nombre fini de coefficients ; (ii) pour modéliser des systèmes récursifs, il est naturel d'introduire des filtres récursifs. Ces deux mêmes raisons justifient l'utilisation de réseaux de neurones bouclés.

Les réseaux de neurones bouclés ont été largement étudiés en tant que mémoires associatives ; dans ce cadre, le réseau est considéré comme un système dynamique non linéaire, dont les attracteurs sont mis à profit pour retrouver des informations mémorisées pendant l'apprentissage. Dans cet article, nous nous plaçons dans un contexte complètement différent : le réseau n'a aucune raison, en général, d'atteindre un état d'équilibre ou un cycle limite.

Représentation canonique d'un réseau de neurones bouclé

Comme dans le cas des réseaux non bouclés, il est commode d'introduire une représentation canonique de n'importe quel réseau bouclé dont l'architecture a pu être imposée préalablement par le problème à traiter [6]. La dynamique d'un réseau bouclé peut être décrite par une équation aux différences d'ordre N, qui peut être exprimée sous la forme de N équations aux différences du premier ordre mettant en jeu, outre les M entrées externes, N variables, dites variables d'état. Par conséquent, tout réseau bouclé peut être mis sous une forme canonique [7], constituée

- d'un réseau non bouclé dont les sorties à l'instant n sont composées de la sortie du réseau bouclé et des sorties d'un nombre N de neurones dits neurones d'état,

- et de N bouclages, de retard unité, reliant les sorties des neurones d'état aux entrées d'état correspondantes.

L'évolution du réseau bouclé est déterminée par les équations d'état suivantes :

$$y(n) = \psi[X(n), U(n)],$$

$$X(n + 1) = \varphi[X(n), U(n)] \qquad \text{(III.5)}$$

où $U(n)$ est le vecteur des M dernières valeurs successives de l'entrée externe u et $X(n)$ est le vecteur d'état. La sortie du réseau peut être aussi une variable d'état. Dans la suite, nous nous limiterons au cas où l'état est constitué de la sortie à l'instant n et de ses $N - 1$ valeurs précédentes (Figure 3) ce qui correspond au modèle NARMAX [10]. Les valeurs des entrées externes du réseau et des sorties des neurones sont les suivantes :

- $i = 1, \ldots, M$; $z_i(n)$: valeurs des entrées externes,

- $i = M + 1, \ldots, M + N$; $z_i(n)$: valeurs des composantes de l'état $X(n)$ à l'entrée du réseau,

- $i = M + N + 1, \ldots, M + N + \nu - 1$; $z_i(n)$: valeurs des sorties des neurones cachés,

- $i = M + N + \nu, \ldots, M + 2N + \nu - 1$; $z_i(n)$: valeurs des composantes de l'état $X(n+1)$ en sortie du réseau (la sortie $z_{M+N+\nu}(n)$ du système global est la première variable d'état).

Avec ces notations, le système d'équations (III.5), qui décrit l'évolution du réseau représenté sur la figure 3, s'écrit

$$z_i(n) = u(n - i + 1) ; \qquad i = 1, \ldots, M,$$

$$z_{M+i}(n) = z_{M+N+\nu+i-1}(n - 1) \qquad i = 1, \ldots, N, \tag{III.6}$$

$$z_{M+N+\nu+i}(n) = z_{M+i}(n) \qquad i = 1, \ldots, N - 1, \tag{III.7}$$

$$z_i(n) = f_i(v_i(n)) ,$$

$$v_i(n)) = \sum_{j<i} c_{ij} z_j(n) \quad ; \qquad i = M + N + 1, \ldots, M + N + \nu. \tag{III.8}$$

Notons que la sortie du réseau à l'instant n est

$$y(n) = z_{M+N+\nu}(n) . \tag{III.9}$$

Cette écriture est conforme aux équations d'état (III.5), où le vecteur d'état $X(n)$ est composé des variables $z_{M+1}(n)=y(n-1)$ à $z_{M+N}(n)=y(n-N)$, où le vecteur d'état à l'instant suivant $X(n + 1)$ est composé des variables $z_{M+N+\nu}(n)$ à $z_{M+2N+\nu-1}(n)$, où le vecteur des entrées externes $U(n)$ est composé des variables $z_1(n)=u(n)$ à $z_M(n)=u(n-M+1)$, et enfin où φ et ψ sont des fonctions non linéaires qui dépendent à la fois des fonctions d'activation des neurones f_i et des coefficients c_{ij} . Remarquons que les neurones d'état, à l'exception du neurone de sortie, ne calculent pas leurs valeurs $z_{M+N+\nu+1}(n)$ à $z_{M+2N+\nu-1}(n)$: ils ne font

que transmettre ces valeurs suivant (III.7) ; grâce à ces notations, les algorithmes d'apprentissage introduits dans le paragraphe IV peuvent être présentés, et réalisés en logiciel, de manière complètement modulaire.

Si l'on ne dispose a priori d'aucune information sur la structure à adopter, il faut alors considérer directement le réseau le plus général possible. Le nombre ν de neurones, les valeurs de M et N sont à déterminer en fonction du problème. Différentes architectures ont été considérées dans la littérature sur les réseaux de neurones [11, 12, 13], mais elles restreignent souvent le type de fonction φ et ψ qui peuvent être réalisées.

IV. APPRENTISSAGE DES RESEAUX DE NEURONES ADAPTATIFS

A l'heure actuelle, l'apprentissage supervisé des réseaux de neurones est effectué, généralement, de manière non adaptative : l'ensemble d'apprentissage est de taille finie, connu à l'avance, et l'on distingue la phase d'apprentissage de la phase d'utilisation du réseau ; une fois l'apprentissage terminé, le réseau est utilisé avec les coefficients obtenus à l'issue de l'apprentissage. Si l'on veut modifier la tâche en cours d'utilisation du réseau, afin de prendre en considération de nouvelles informations, il faut arrêter la phase d'utilisation et reprendre la phase d'apprentissage. Or, pour qu'un réseau de neurones puisse réaliser des opérations de filtrage, il est essentiel qu'il puisse s'adapter en permanence à d'éventuelles évolutions des caractéristiques du signal, sans que son fonctionnement ne soit interrompu.

Dans ce paragraphe, nous établissons, dans un cadre général, des algorithmes d'apprentissage adaptatifs pour des réseaux de neurones utilisables comme filtres transverses ou récursifs ; nous montrons que les algorithmes adaptatifs proposés précédemment par d'autres auteurs sont des cas particuliers des algorithmes que nous présentons.

IV.1 Critère d'optimisation

Un réseau de neurones utilisé comme filtre fait correspondre à une séquence de données d'entrée $u(n)$, $u(n-1)$, …, une séquence de données de sortie $y(n)$, $y(n-1)$, …. La tâche qu'il doit réaliser est traduite par un critère d'optimisation. En principe, chaque application devrait posséder son propre critère : minimiser le taux d'erreur sur les bits transmis lors d'une communication numérique, maximiser le rapport signal sur bruit dans le cas d'un filtrage spatial, optimiser un critère d'écoute pour la prédiction de la parole, etc. Il se trouve que, dans beaucoup d'applications, le critère des moindres carrés est couramment utilisé parce qu'il conduit à des performances intéressantes et à des réalisations relativement simples. Ce critère est défini à partir d'une séquence d'entrée (éventuellement infinie) $u(n)$, $u(n-1)$, … et d'une séquence de réponses désirées correspondantes $d(n)$, $d(n-1)$, ….

Dans le cas où l'on dispose d'un nombre fini K de couples $\{u(m), d(m)\}$, le critère des moindres carrés (MC) se traduit par la minimisation de la fonction de coût

$$J_{\mathrm{MC}}(C) \triangleq \frac{1}{K} \sum_{m=1}^{K} e(m)^2 \qquad \text{(IV.1)}$$

$$e(m) = d(m) - y(m), \qquad \text{(IV.2)}$$

où C est le vecteur des coefficients à optimiser. C'est ce critère qui est couramment utilisé dans l'apprentissage supervisé des réseaux de neurones classifieurs : chaque couple correspond à un exemple $u(m)$ et à la classe $d(m)$ qui lui a été attribuée par le superviseur. Ce critère n'a de sens en *classification* que si la distribution des classes dans l'univers des formes est figée : les performances de généralisation du réseau (capacité à classer des formes qui n'ont pas été apprises durant la phase d'apprentissage) sont fonction de la représentativité de la distribution statistique des exemples présentés par rapport à l'ensemble de toutes les formes à classer. Ce critère n'a de sens en *filtrage* que si les séquences $u(m)$ et $d(m)$ sont stationnaires et si K est suffisamment grand pour représenter la statistique des séquences.

Comme nous l'avons mentionné plus haut, l'apprentissage d'un réseau ou d'un filtre reposant sur la minimisation de la fonction de coût (IV.1) est non adaptatif.

Lorsque l'on désire réaliser un *apprentissage permanent*, on cherche à trouver, à l'instant n, un ensemble de coefficients $C(n)$ tels que l'erreur à l'instant n+1 soit aussi petite que possible, en tenant compte de l'expérience passée, c'est-à-dire des propriétés statistiques des signaux d'entrée et des valeurs désirées correspondantes. Dans le cas où ces signaux sont stationnaires, les coefficients recherchés sont ceux qui minimisent la fonction

$$J^{n}_{\mathrm{MC}}(C) \triangleq \frac{1}{n} \sum_{m=1}^{n} e(m)^2 \qquad \text{(IV.3).}$$

Dans le cas où les signaux ne sont pas stationnaires, il n'est évidemment pas souhaitable de tenir compte de toute l'expérience passée pour modifier les coefficients du filtre ; pour calculer la modification des coefficients à l'instant n, on peut utiliser la fonction de coût

$$I(n) \triangleq \frac{1}{2} \sum_{m=n-N_{\mathrm{c}}+1}^{n} e(m)^2, \qquad \text{(IV.4)}$$

où N_{c} correspond à un intervalle de temps qui est petit devant l'échelle de temps typique de stationnarité des signaux.

IV.2 Algorithmes adaptatifs fondés sur l'estimation du gradient de la fonction de coût

La recherche d'algorithmes adaptatifs obéit essentiellement à trois motivations :

- la recherche analytique du minimum de la fonction de coût (IV.4) par rapport aux coefficients du filtre est inextricable pour des systèmes non linéaires et/ou bouclés ; il est donc intéressant de chercher un algorithme itératif de recherche de minimum de la fonction de coût ;

- pour limiter la complexité des calculs et les répartir dans le temps, il est intéressant de concevoir des algorithmes *récursifs* qui optimisent le système à l'instant n en fonction de ce qu'il était à un instant antérieur ;

- dans le cas non stationnaire, il est nécessaire d'adapter les coefficients du système, en fonction de l'évolution de l'environnement, au fur et à mesure que l'information arrive.

Les algorithmes utilisés habituellement pour l'apprentissage des réseaux de neurones tiennent compte essentiellement des deux premiers points ; le troisième point, en revanche, est rarement pris en considération. Ainsi, dans un contexte de traitement du signal par un réseau de neurones, un algorithme sera dit adaptatif s'il calcule, au fur et à mesure que le temps court, des modifications des coefficients du système à partir des informations passées ; il est évidemment souhaitable que l'algorithme soit capable de poursuivre d'éventuels changements des caractéristiques du signal.

Dans sa forme la plus générale, une modification, à l'instant n, du vecteur des coefficients du système, est calculée itérativement suivant

$$\Delta C(n) = C_{K_n}(n) - C_0(n) \qquad \text{(IV.5)}$$

avec

$$C_k(n) = C_{k-1}(n) + \mu_{k-1} D_{k-1}(n) \; ; \; k = 1, \ldots, K_n \, , \qquad \text{(IV.6)}$$

où K_n est le nombre d'itérations réalisées avant de modifier les coefficients; K_n peut dépendre de n. Dans la suite, on ne s'intéressera qu'à l'algorithme du gradient où

$$D_k = - \left. \frac{\partial I\,(n)}{\partial C} \right|_{C=C_k(n)} \; ; \qquad \text{(IV.7)}$$

et où μ_{k-1} est une suite de paramètres positifs, à déterminer, dont dépendent la vitesse de convergence, la stabilité, les capacités de poursuite de l'algorithme. Remarquons que d'autres algorithmes, par exemple les algorithmes de type "quasi-Newton", reposent également sur le calcul du gradient et peuvent donc entrer dans le cadre de ce qui suit.

Cet algorithme est récursif, car son initialisation, à l'instant n, tient compte des coefficients du système lors de la dernière modification de ceux-ci :

$$C_0(n) = C_{K_{n-T}}(n - T) \qquad \text{(IV.8)}$$

où T est l'intervalle de temps qui sépare deux modifications successives des coefficients.

Lors d'un apprentissage adaptatif, on veut pouvoir remettre à jour les coefficients même dans un cas non stationnaire : il faudrait donc, théoriquement, itérer l'algorithme du gradient (IV.6) K_n fois pour un même couple de données $\{u(n), d(n)\}$, afin de converger vers le minimum (éventuellement local) de la fonction de coût $I(n)$. Ceci correspond au trajet **1** sur la figure 4. Cependant, supposons qu'à l'instant $n + 1$, correspondant aux nouvelles données $\{u(n + 1), d(n + 1)\}$, la fonction de coût $I(n + 1)$ soit très proche de $I(n)$ (variations lentes de l'environnement) ; il n'est pas nécessaire d'itérer (IV.6) plusieurs fois (voir trajet **2** sur la figure 4). Par conséquent, dans un cas stationnaire, l'algorithme (IV.5)-(IV.8) peut se réécrire, en posant $K_n = 1$, $T = 1$, $D_0(n) = D(n - 1)$ et $C_1(n) = C(n)$

$$C(n) = C(n - 1) + \mu D(n - 1)\,. \qquad \text{(IV.9)}$$

Rappelons que, dans le cas où $N_c = 1$ dans la relation (IV.4), cet algorithme est connu sous le nom de gradient stochastique.

Si, au contraire, $I(n + 3)$, par exemple (courbe en pointillé sur la figure 4), est très différente de $I(n + 2)$, une seule itération de (IV.6) ne suffira pas à passer du minimum de $I(n + 2)$ au minimum de $I(n + 3)$ (voir trajet **3** sur la figure 4). Pour converger vers la solution optimale de $I(n + 3)$ appliqer la relation (IV.9) plusieurs fois.

Cependant, dans un contexte non stationnaire où les fonctions de coût $I(n + 2)$, $I(n + 3)$ ... seraient successivement très différentes, on n'a pas forcément intérêt à obtenir successivement le minimum de chacune d'elles : on doit donc s'efforcer de réaliser un compromis entre la minimisation de la fonction de coût à chaque instant et la capacité à suivre les non-stationnarités. C'est là un problème délicat, auquel on est souvent confronté en filtrage adaptatif.

Le choix des paramètres N_c, T et K_n introduits dans le critère (IV.4) et l'algorithme (IV.5)-(IV.8) est délicat et doit être fixé en fonction de connaissances a priori sur les caractéristiques des signaux à traiter (durée de stationnarité), de la tâche que doit réaliser le sytème, ainsi que du temps et des moyens dont on dispose pour effectuer les calculs. Si, par exemple, la modification des coefficients est faible, il n'est pas indispensable de prendre N_c plus grand que 1 dans la mesure où le moyennage sera alors réalisé au cours des itérations successives de l'algorithme [14] ; cependant, il faudra obligatoirement choisir $T = 1$ pour que le moyennage se fasse avec des valeurs de $C(n)$ assez voisines. Si au contraire, dans un cas stationnaire, on choisit $T > 1$, on aura intérêt à choisir $N_c > 1$ et éventuellement $K_n > 1$.

Dans un cas non stationnaire, N_c devra être choisi en fonction de la durée de stationnarité du signal. T sera choisi égal à 1 dans le cas où l'on a besoin d'adapter le système continuellement, mais on pourra envisager $T > 1$ dans le cas contraire. Dans le cas non stationnaire encore, on choisira K_n grand si l'on recherche à

chaque fois une grande précision et si l'on a le temps de faire les calculs (or, souvent, qui dit contexte non stationnaire dit temps de calcul réduit). Au contraire, on choisira $K_n = 1$ dans le cas où l'on cherche à poursuivre un comportement moyen.

Une fois ces différents paramètres fixés, le problème essentiel réside dans l'évaluation du gradient afin de pouvoir implanter l'algorithme. Le filtrage adaptatif utilise couramment une technique de calcul du gradient que nous désignerons sous le terme de "calcul direct" ; l'apprentissage des réseaux de neurones met couramment en œuvre une technique de calcul du gradient connue sous le nom de "rétropropagation" [6, 1].

IV.3 Evaluation du gradient dans un réseau non bouclé

IV.3.1 Calcul direct

Le calcul direct du gradient repose sur le développement le plus immédiat de (IV.4), (IV.2), soit

$$\frac{\partial I(n)}{\partial c_{pq}}\Bigg|_{C=C_k(n)} = -\sum_{m=n-N_c+1}^{n} e(m)\,\frac{\partial y(m)}{\partial c_{pq}}\Bigg|_{C=C_k(n)}. \qquad \text{(IV.10)}$$

Dans cette expression, les erreurs $e(m)$ et les dérivées partielles sont calculées avec les derniers coefficients connus à l'instant n, $C_k(n)$; elles correspondent aux valeurs que l'on aurait calculées aux instants $m = n - N_c + 1, \ldots, n$ si l'on avait disposé de ces coefficients. Les dérivées partielles des $y(m)$ par rapport aux coefficients c_{pq} (dans le réseau complètement connecté mais non bouclé de la figure 2 on a $p > q$) se calculent à partir des équations d'évolution (III.4) du réseau. Tous ces calculs sont faits à l'instant n avec les coefficients $C_k(n)$; pour alléger l'écriture, nous omettrons dans la suite la notation $\Big|_{C=C_k(n)}$.

Il vient pour chaque $m \in [n - N_c + 1, n]$

$$\frac{\partial z_i(m)}{\partial c_{pq}} = 0 \quad ; \; p > i \text{ ou } i = 1, \ldots, M,$$

$$\frac{\partial z_i(m)}{\partial c_{pq}} = f'_i(v_i(m))z_q(m) \; ; \; p = i \text{ et } i > M,$$

$$\frac{\partial z_i(m)}{\partial c_{pq}} = f'_i(v_i(m)) \sum_{p<j<i} c_{ij}\,\frac{\partial z_j(m)}{\partial c_{pq}} \quad ; \; p < i \text{ et } i > M. \qquad \text{(IV.11)}$$

Ainsi, le calcul des $\frac{\partial y(m)}{\partial c_{pq}}$ se fait de proche en proche par propagation, depuis les entrées vers les sorties du réseau non bouclé linéaire dont les poids sont $f'_i(v_i(m))c_{ij}$ (cf. (IV.11)), des dérivées partielles des sorties des neurones situés en amont. Dans le cas où $N_c \geq 1$, il faut calculer les N_c dérivées partielles et termes

d'erreurs intervenant dans (IV.10) avec les poids fixés à leur valeur $C = C_k(n)$. On dit alors que N_c copies du réseau à l'instant n sont utilisées pour calculer les valeurs nécessaires au calcul du gradient (IV.10) [7].

IV.3.2 Calcul par rétropropagation

Le calcul du gradient par rétropropagation [15] fait intervenir des dérivées partielles différentes de celles utilisées dans le calcul direct. A partir de (IV.4), on écrit

$$\frac{\partial I\,(n)}{\partial c_{pq}}\,\Big|_{C=C_k(n)} = \frac{1}{2}\sum_{m\,=\,n\,-\,N_c\,+\,1}^{n} \frac{\partial e^2(m)}{\partial v_p(m)}\,\frac{\partial v_p(m)}{\partial c_{pq}}\,\Big|_{C=C_k(n)}\,. \qquad \text{(IV.12)}$$

Comme précédemment, nous omettrons dans la suite la notation $\Big|_{C=C_k(n)}$.

D'après (III.4), on calcule aisément

$$\frac{\partial v_p(m)}{\partial c_{pq}} = z_q(m)\ ,\ p > M. \qquad \text{(IV.13)}$$

Les autres dérivées partielles intermédiaires intervenant dans (IV.12) peuvent se calculer également à partir de (III.4) :

$$\frac{\partial e^2(m)}{\partial v_p(m)} = \sum_{l\,>\,p} \frac{\partial e^2(m)}{\partial v_l(m)}\,\frac{\partial v_l(m)}{\partial v_p(m)} \qquad \text{(IV.14)}$$

soit encore

$$\frac{\partial e^2(m)}{\partial v_p(m)} = f'_p(v_p(m)) \sum_{l\,>\,p} c_{lp}\,\frac{\partial e^2(m)}{\partial v_l(m)}\quad . \qquad \text{(IV.15)}$$

La dernière relation exprime que les dérivées partielles $\frac{\partial e^2(m)}{\partial v_p(m)}$ peuvent être calculées à partir de celles déjà calculées en aval du réseau. C'est ce qu'on appelle le calcul du gradient par rétropropagation des dérivées partielles dans le réseau non bouclé linéarisé dont les coefficients sont $c_{lp}\,f'_p(v_p(m))$ (IV.15) [7].

On remarquera que la sommation se fait sur le premier indice l dans (IV.15) alors qu'elle se faisait sur le deuxième indice dans le calcul direct (IV.11) Dans un réseau non bouclé, les deux techniques de calcul du gradient présentées ci-dessus sont équivalentes du point de vue du résultat. Cependant, on a montré [1, 6]

que la rétropropagation et le calcul direct ont une complexité de calcul respectivement de l'ordre de ν^2 et de ν^4. En pratique, on utilisera donc la rétropropagation pour un réseau non bouclé.

IV.4 Sur la difficulté de calculer le gradient exact dans un réseau bouclé

L'évaluation du gradient nécessaire à l'adaptation des coefficients d'un réseau bouclé est beaucoup plus complexe que dans le cas d'un réseau non bouclé. En effet, on voit, sur les équations d'état (III.5) de la représentation canonique d'un réseau bouclé, que la sortie globale du système $y(n)$ est fonction non seulement des entrées externes au réseau $U(n)$, mais également de l'état $X(n)$ du réseau qui est lui même fonction des entrées du réseau et de son état calculé à l'instant précédent. Autrement dit, chacun des termes d'erreur $e(m)$ intervenant dans le critère $I(n)$ de (IV.4) ne dépend plus seulement des M dernières entrées externes $u(m)$, $u(m-1)$, …, $u(m-M+1)$, mais de toutes les valeurs des entrées depuis l'instant initial. De même, le calcul des dérivées partielles fait intervenir les dérivées partielles depuis l'instant initial. Par conséquent, pour calculer le gradient de $I(n)$ pour $C=C_k(n)$, que ce soit par le calcul direct (IV.10) ou par la rétropropagation (IV.12), il faudrait théoriquement calculer chaque $e(m)$ et chaque dérivée partielle avec toutes les données depuis l'instant initial, à C fixé.

Dans le cas du calcul direct des dérivées partielles permettant de calculer $\frac{\partial y(m)}{\partial c_{pq}}$ dans (IV.10), il vient d'après (III.6)-(III.9) exprimant l'évolution du réseau de la figure 3, et pour chaque $m \in [n - N_c + 1,\ n]$, où n désigne comme précédemment l'instant où sont faits les calculs,

$$\frac{\partial z_i(m)}{\partial c_{pq}} = 0 \quad ; \quad i = 1, \ldots, M,$$

$$\frac{\partial z_{M+l}(m)}{\partial c_{pq}} = \frac{\partial z_{M+N+\nu+l-1}(m-1)}{\partial c_{pq}} \quad ; l = 1, \ldots, N \ , \qquad \text{(IV.16)}$$

$$\frac{\partial z_{M+N+\nu+l}(m)}{\partial c_{pq}} = \frac{\partial z_{M+l}(m)}{\partial c_{pq}} \ ; \ l = 1, \ldots, N-1 \ , \qquad \text{(IV.17)}$$

$$\frac{\partial z_i(m)}{\partial c_{pq}} = f'_i(v_i(m)) \sum_{j < i} c_{ij,} \frac{\partial z_j(m)}{\partial c_{pq}} \quad ; \quad i = M+N+1, \ldots, M+N+\nu \ . \qquad \text{(IV.18)}$$

Cette écriture est donc différente de (IV.11) : elle fait apparaitre le caractère bouclé du réseau. En effet, d'après (IV.16), (IV.17), il vient

$$\frac{\partial z_{M+l}(m)}{\partial c_{pq}} = \frac{\partial z_{M+N+\nu}(m-l)}{\partial c_{pq}} = \frac{\partial y(m-l)}{\partial c_{pq}} \ ; \ l = 1, \ldots, N \ , \qquad \text{(IV.19)}$$

qui fait lui même intervenir d'après (IV.18), entre autres, les dérivées $\frac{\partial z_{M+j}(m-l)}{\partial c_{pq}}$, $j = 1, \ldots, N$. De plus, les N_c dérivées partielles $\frac{\partial y(m)}{\partial c_{pq}}$, nécessaires à l'adaptation de chaque coefficient de pondération c_{pq}, devraient donc être calculées à chaque itération de l'algorithme du gradient suivant les récurrences (IV.16)-(IV.18), en fixant tous les coefficients c_{pq} du réseau à leur valeur $C_k(n)$.

Dans le cas du calcul par rétropropagation, les récurrences permettant de calculer théoriquement les dérivées partielles intervenant dans (IV.12) sont les suivantes :

$$\frac{\partial e^2(m)}{\partial c_{pq}} = \sum_{k=0}^{m} \frac{\partial e^2(m)}{\partial v_p(m-k)} \frac{\partial v_p(m-k)}{\partial c_{pq}} \; ; \qquad \text{(IV.20)}$$

(i) $k = 0$,

$$\frac{\partial e^2(m)}{\partial v_p(m)} = -2e(m) \; ; \qquad p = M + N + \nu \, ,$$

$$\frac{\partial e^2(m)}{\partial v_{M+N+\nu+p}(m)} = 0 \; ; \qquad p = 1, \ldots, N-1, \qquad \text{(IV.21)}$$

$$\frac{\partial e^2(m)}{\partial v_p(m)} = f'_p(v_p(m)) \sum_{h>p} c_{hp} \frac{\partial e^2(m)}{\partial v_h(m)} \; ;$$

$$p = M + N + 1, \ldots, M + N + \nu - 1 \qquad \text{(IV.22)}$$

$$\frac{\partial e^2(m)}{\partial v_p(m)} = \sum_{h>p} c_{hp} \frac{\partial e^2(m)}{\partial v_h(m)} \; ; \; p = M+1, \ldots, M+N. \qquad \text{(IV.23)}$$

(ii) $k > 0$,

$$\frac{\partial e^2(m)}{\partial v_p(m-k)} = \frac{\partial e^2(m)}{\partial v_{p-N-\nu+1}(m-k+1)} \; ; p = M + N + \nu, \ldots, M + 2N + \nu - 1,$$

$$\text{(IV.24)}$$

$$\frac{\partial e^2(m)}{\partial v_p(m-k)} = f'_p(v_p(m-k)) \sum_{h>p} c_{hp} \frac{\partial e^2(m)}{\partial v_h(m-k)} \; ;$$

$$p = M + N + 1, \ldots, M + N + \nu - 1 \qquad \text{(IV.25)}$$

$$\frac{\partial e^2(m)}{\partial v_p(m-k)} = \sum_{h > p} c_{hp} \frac{\partial e^2(m)}{\partial v_h(m-k)} \quad ; \; p = 1, \ldots, M+N. \qquad \text{(IV.26)}$$

Par ailleurs, on a

$$\frac{\partial v_p(m-k)}{\partial c_{pq}}) = 0, \qquad p = 1, \ldots, M+N,$$

$$\frac{\partial v_p(m-k)}{\partial c_{pq}}) = z_q(m-k), \quad p = M+N+1, \ldots, M+2N+\nu. \qquad \text{(IV.27)}$$

Les relations (IV.22), (IV.23), (IV.25) et (IV.26) expriment bien le calcul par rétropropagation des dérivées partielles à un instant $m - k$ dans un réseau non bouclé, tandis que la relation (IV.24) exprime que cette rétropropagation des dérivées partielles se fait également dans le temps depuis l'instant m en remontant vers l'instant initial.

IV.5 Equivalence des notions de récurrence temporelle et de dépliement spatial

La représentation d'état d'un dispositif bouclé temporellement, qu'il soit linéaire ou non linéaire, comme celui de la figure 3, représentation issue du formalisme utilisé en automatique, permet une interprétation "spatiale" particulièrement simple des calculs de gradients présentés dans la section précédente. En effet, l'erreur $e(m)$ est calculée, à l'instant courant n, par le dispositif de la figure 5, obtenu en "dépliant" spatialement la récurrence temporelle du réseau de la figure 3. Ce dispositif est constitué d'une cascade de cellules élémentaires ou copies du réseau non bouclé de la forme canonique (§ III.3), contenant les mêmes valeurs de coefficients $C_k(n)$.

De cette façon, le calcul, à l'instant n, des erreurs $e(m)$ et des dérivées partielles de ces erreurs prend en considération un horizon temporel de longueur m ; ceci peut être interprété simplement comme le calcul de ces mêmes quantités sur une cascade non bouclée de m cellules identiques. C'est ce que nous appelons l'équivalence entre les notions de dépliement spatial et de récurrence temporelle.

IV.6 Algorithmes pour l'adaptation des réseaux bouclés

IV.6.1 Dépliement du réseau bouclé en un nombre fini de copies

Le calcul des erreurs et des dérivées partielles depuis l'instant initial n'est pas réalisable en pratique, car il nécessite un trop grand nombre de calculs, ainsi que le stockage en mémoire de tous les échantillons des signaux présentés au réseau ; de plus, dans le cas où les signaux sont non stationnaires, il n'est pas souhaitable de tenir compte de tout le passé.

On est donc amené à tronquer les récurrences (III.6)-(III.9), et (IV.16)-(IV.19) ou (IV.21)-(IV.26) sur une longueur N_t . Autrement dit, on remplace la représentation développée de la figure 5, qui utilise m copies pour calculer l'erreur $e(m)$, par celle de la figure 6, qui utilise un nombre fixe de copies N_t ($N_t < m$) pour estimer $e(m)$. Pour calculer les N_c termes du gradient, il y a donc N_c systèmes de ce type. Le système d'argument m est un réseau non bouclé dont les entrées sont les entrées externes $z_1^1(m)$, ..., $z_M^1(m)$, $z_1^2(m)$, ..., $z_M^2(m)$, ..., $z_1^{N_t}(m)$, ..., $z_M^{N_t}(m)$ ($z_i^l(m)$est la valeur de la i-ème entrée de la l-ème copie du m-ième système), les variables d'état de la première copie $z_{M+1}^1(m)$, ..., $z_{M+N}^1(m)$, et dont la sortie est $z_{M+N+\nu}^{N_t}(m)$, laquelle constitue une estimation de $y(m)$. La modification des coefficients à l'instant n est la somme des modifications calculées sur chacun des N_c systèmes décrits ci-dessus.

Afin de diminuer le nombre de calculs, on peut utiliser une seule récurrence, c'est à dire un seul système composé de N_t copies qui calcule les N_c erreurs et les dérivées partielles. Les entrées de ce système unique sont les entrées externes z_1^1, ..., z_M^1, z_1^2, ..., z_M^2, ..., $z_1^{N_t}$, ..., $z_M^{N_t}$, les variables d'état de la première copie z_{M+1}^1, ..., z_{M+N}^1 , et les sorties sont $z_{M+N+\nu}^{N_t-N_c+1}$, ...$z_{M+N+\nu}^{N_t}$. L'argument m, qui était le numéro de l'un des N_c systèmes, n'apparaît plus ici. Le résultat de ces calculs est évidemment différent de celui que l'on obtient avec les N_c systèmes indépendants [7].

IV.6.2 Une nouvelle famille d'algorithmes adaptatifs

Dans ce paragraphe, nous introduisons une nouvelle famille d'algorithmes adaptatifs qui s'appuient sur les systèmes décrits dans le paragraphe précédent et qui résultent des choix suivants :
- initialisation des variables d'état de la première copie et de leurs dérivées partielles;
- technique d'évaluation du gradient (calcul direct ou rétropropagation).

Les diverses possibilités sont résumées dans les tableaux 1 et 2.

Les tableaux présentés ci-dessus définissent autant d'algorithmes qu'il y a de lignes. Chaque algorithme est donc caractérisé (en plus des paramètres introduits précédemment comme le nombre de copies N_t, le nombre de termes N_c de la fonction de coût , le nombre d'itérations K_n du gradient, etc.) par la technique de calcul du gradient et les initialisations choisies. Bien que visant à minimiser une même fonction de coût et bien qu'utilisant un algorithme de gradient, les algorithmes donnés par les lignes des tableaux précédents sont différents.

Parmi les choix possibles d'initialisation, les lignes 1 et 4 du tableau 1 et la ligne 2 du tableau 2 sont naturelles : l'initialisation des dérivées partielles des états correspond à l'initialisation des états. Parmi ces choix naturels, la ligne 4 du tableau 1 et la ligne 2 du tableau 2 supposent que l'on connaît des réponses désirées pour toutes les variables d'état du réseau ; autrement dit, elles supposent que toutes les variables d'état sont les valeurs des sorties y précédentes du système. C'est le cas pour le modèle NARMAX, puisque les états sont des sorties à des instants précédents. Dans le cas plus général où l'on ne connaît pas

de réponses désirées pour tous les états, seul le choix de la ligne 1 du tableau 1 est à la fois naturel et réalisable.

Des algorithmes "hybrides" utilisant les initialisations des lignes 2 et 3 du tableau 1 et celles de la ligne 1 du tableau 2 sont envisageables.

Le choix du nombre de copies N_t n'est pas motivé par les mêmes considérations pour tous les algorithmes. Par exemple, pour l'algorithme de la ligne 1 du tableau 1, l'utilisation d'un grand nombre de copies n'a d'intérêt que lorsque les coefficients du système sont susceptibles d'être fortement modifiés d'une itération à l'autre (non stationnarité des signaux ou instabilité locale de l'algorithme) : à l'instant n, les coefficients $c_{pq}(n)$ sont tellement différents de ce qu'ils étaient à l'instant $n-1$, que, pour en tenir compte dans le calcul des erreurs $e(n)$, $e(n-1)$, … , $e(m)$, … qui vont servir à calculer $c_{pq}(n+1)$, on a intérêt à utiliser plusieurs copies. Dans un cas stable ou stationnaire, le nombre de copies n'est pas important à partir du moment où le caractère récursif du système est conservé par le type d'initialisation. En revanche, dans le cas des algorithmes des lignes 4 du tableau 1 et 2 du tableau 2, les copies permettent de prendre en considération le caractère récursif du filtre.

La rétropropagation suppose implicitement que les dérivées partielles des entrées d'état par rapport aux coefficients sont nulles pour la première copie. Les algorithmes 2 du tableau 1 et 1 du tableau 2 conduisent donc aux mêmes modifications des coefficients; il en est de même pour les algorithmes 4 du tableau 1 et 2 du tableau 2. Comme d'autre part la rétropropagation nécessite un nombre de calculs moins important, c'est l'algorithme correspondant qui doit être utilisé.

Nous allons montrer à présent que les algorithmes proposés dans la littérature pour l'adaptation des filtres récursifs et des réseaux de neurones bouclés sont des cas particuliers des algorithmes présentés ci-dessus.

IV.6.3 Classification des algorithmes existants en filtrage adaptatif et pour l'apprentissage des réseaux de neurones.

Les algorithmes du filtrage linéaire récursif adaptatif [16] entrent dans la classe plus générale des algorithmes définis ci-dessus. Ils correspondent tous à un calcul direct du gradient (tableau 1) et à $N_c=K_n=1$. L'algorithme "LMS étendu" correspond au cas $N_t=1$ et à la ligne 2. L'algorithme "RPE" (approche par erreur de sortie) coïncide avec $N_t=1$ et la ligne 1. Quant à l'algorithme avec erreur a posteriori, il est aussi associé à la ligne 1 mais avec $N_t=2$.

Les algorithmes qui sont apparus récemment pour l'apprentissage supervisé de réseaux bouclés (utilisés comme filtres) entrent également dans la classification précédente. Ils correspondent tous à $N_c=1$. On peut citer l'algorithme "Teacher Forcing" [17, 18] (correspondant à une approche de type Equation Error ou Série-Parallèle) qui correspond à la ligne 2 du tableau 2 pour $N_t=1$, le "Real Time Recurrent Learning Algorithm" [17, 18] (correspondant à une approche de type Output Error ou Parallèle) qui correspond à la

ligne 1 du tableau 1 avec N_t=1 et le "Truncated Backpropagation Through Time" [19] correspondant à la ligne 1 du tableau 2 avec N_t>1.

V CONCLUSION

Nous avons établi un cadre conceptuel général pour les algorithmes adaptatifs permettant l'apprentissage de réseaux de neurones formels non bouclés (utilisés comme filtres transverses non linéaires) ou bouclés (utilisés comme filtres récursifs non linéaires). Ces algorithmes sont fondés sur des techniques d'évaluation du gradient d'une fonction de coût. Les algorithmes classiques en filtrage adaptatif, ainsi que les algorithmes adaptatifs proposés par d'autres auteurs pour les réseaux de neurones, entrent dans ce cadre général ; de surcroît, cette approche nous permet de définir toute une famille d'algorithmes nouveaux, qui sont susceptibles d'applications dans le domaine du filtrage adaptatif non linéaire. Enfin, le caractère modulaire des algorithmes facilite les réalisations logicielles et suggère des implantations matérielles efficaces.

REMERCIEMENTS

Les auteurs tiennent à remercier O. Macchi, pour l'impulsion initiale à ce travail et l'intérêt qu'elle y porte.
Ce travail a été soutenu en partie par le GRECO Traitement du Signal et de l'Image.

REFERENCES

[1] S. MARCOS, O. MACCHI, C. VIGNAT, G. DREYFUS, L. PERSONNAZ, P. ROUSSEL-RAGOT, "A Unified Framework for Gradient Algorithms Used for Filter Adaptation and Neural Network Training", International Journal of Circuit Theory and Applications, à paraître.

[2] K. HORNIK, "Multilayer Feedforward Networks are Universal Approximators", Neural Networks, Vol.2, pp. 359-366, 1989.

[3] A. WAIBEL , T. HANAZAWA, G. HINTON, K. SHIKANO, and K. LANG, "Phoneme Recognition Using Time-Delay Neural Networks", IEEE Trans. on Acoustics, Speech, and Signal Processing 37, 328-339, 1989.

[4] P. CHEVALIER, P. DUVAUT, B. PICINBONO, "Le Filtrage de Volterra Transverse Réel et Complexe en Traitement du Signal", Traitement du Signal, Vol. 7, n° 5, pp. 451-476, 1990.

[5] M.J.D. POWELL, "Radial Basis Funstions for Multivariable Interpolation : A Review ", in Proceedings of IMA Conference on Algorithms for the Approximation of Functions and Data, RMCS Shrivenham, 1985.

T.J. SHEPHERD, D.S. BROOMHEAD, "Non Linear Signal Processing using Radial Basis Functions", SPIE, Vol.1348, Advanced Signal Processing Algorithms, Architectures, and Implementations, 1990.

[6] L. PERSONNAZ, O. NERRAND, G. DREYFUS, "Apprentissage et Mise en Oeuvre de Réseaux de Neurones Bouclés", Journées Internationales des Sciences Informatiques, Tunis, 1990.
O. NERRAND, P. ROUSSEL-RAGOT, L. PERSONNAZ, G. DREYFUS, S. MARCOS, "Training Discrete-Time Feedback Networks for Filtering and Control", Neural Network World, vol. 1, pp. 205-212, 1991.

[7] O. NERRAND, P. ROUSSEL-RAGOT, L. PERSONNAZ, G. DREYFUS, S. MARCOS, O. MACCHI, C. VIGNAT, "Neural Network Training Schemes for Non-linear Adaptive Filtering and Modelling", Proceedings of IJCNN, Seattle, 1991.
O. NERRAND, P. ROUSSEL-RAGOT, L. PERSONNAZ, G. DREYFUS, S. MARCOS, "Neural Networks and Non Linear Adaptive Filtering: Unifying Concepts and New Algorithms", soumis pour publication.

[8] S. MARCOS, O. MACCHI, "Joint adaptive echo cancellation and channel equalization for data transmission", Signal Processing, Vol.20, N°1, May 1990, pp. 43-65.

[9] J.M. TRAVASSOS ROMANO, "Localisation de Fréquences Bruitées par Filtrage Adaptatif et Implantation d'Algorithmes des Moindres Carrés Rapides", Thèse de doctorat, Orsay, 1987.

[10] I.J. LEONTARITIS, S.A. BILLINGS, "Input-output Parametric Models for Non-linear Systems", International Journal of Control, Vol. 41, n°2, pp. 303-328, 1985.

[11] J.L. ELMAN, "Finding Structure in Time", CRL Technical Report 8801, Center for Research in Language, University of California San Diego, 1988.

[12] M.I. JORDAN, "Serial Order: A Parallel Distributed Processing Approach", ICS Report 8604, Institute for Cognitive Science, University of California San Diego, 1986.

[13] P. PODDAR, K.P. UNNIKRISHNAN, "Efficient Real-Time Prediction and Recognition of Temporal Patterns", Neural Networks for Computing, Snowbird, 1991.

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

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

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

[17] R.J. WILLIAMS, D. ZIPSER, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks", Neural Computation 1, pp. 270-280, 1989.

[18] R.J. WILLIAMS, D. ZIPSER, "Experimental Analysis of the Real-Time Recurrent Learning Algorithm", Connection Science, Vol.**1**, pp.87-111, 1989.

[19] R.J. WILLIAMS, J. PENG, "An efficient gradient based algorithm for on-line training recurrent network trajectories", Neural Computation 2, p. 490-501, 1990.

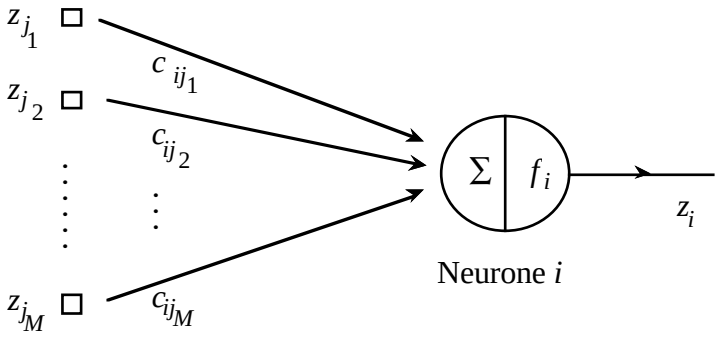

Figure 1. Neurone formel

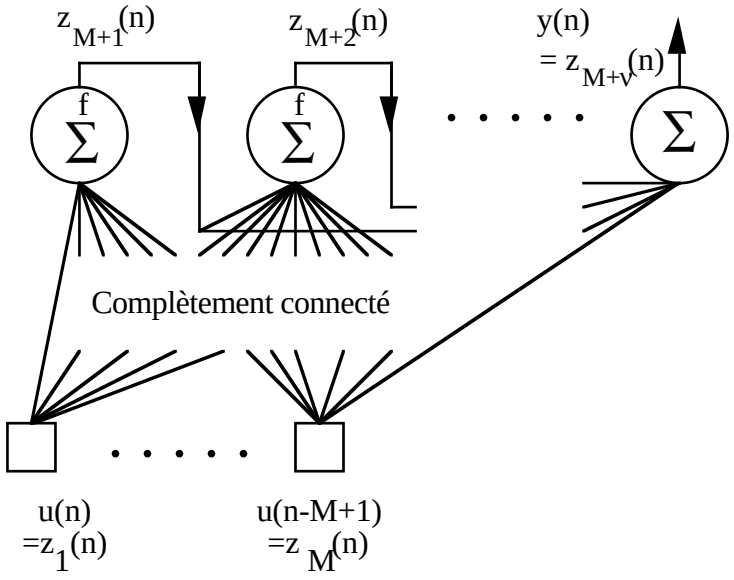

Figure 2. Réseau de neurones non bouclé complètement connecté utilisé comme filtre transverse non linéaire.

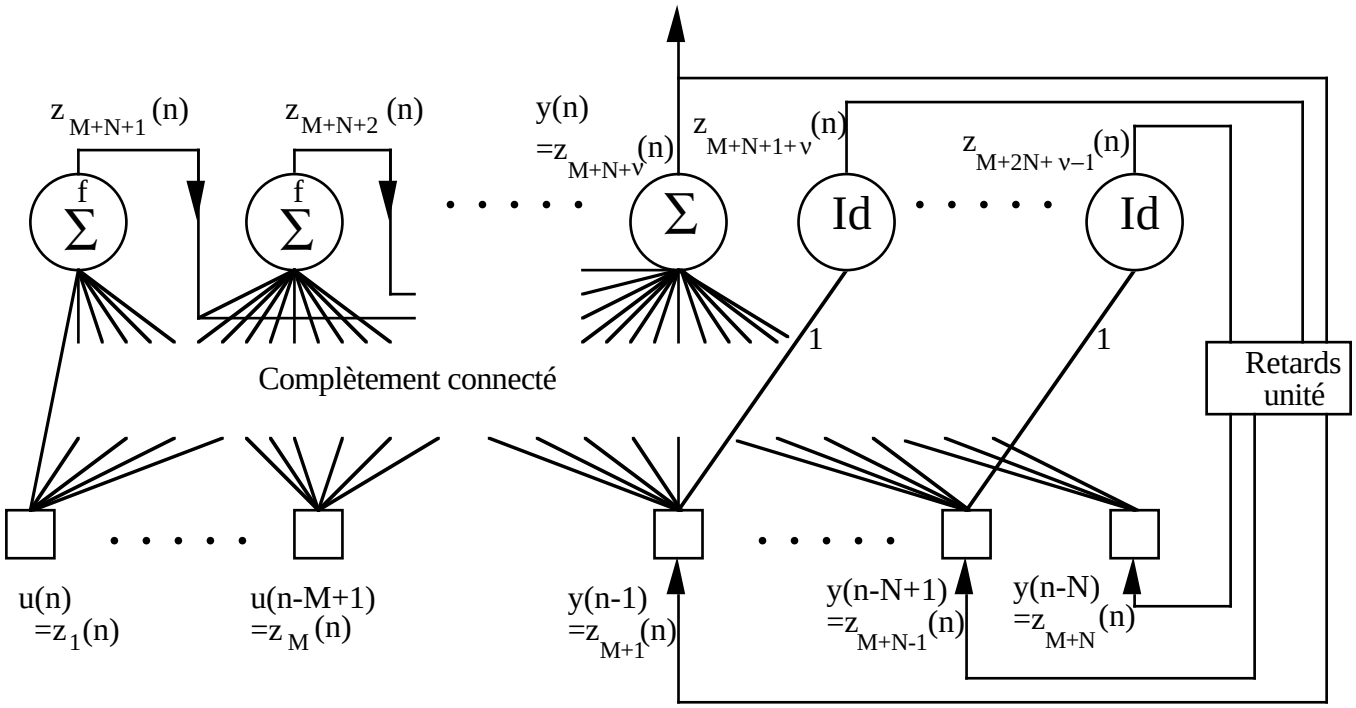

Figure 3. Réseau de neurones bouclé (modèle NARMAX) utilisé comme filtre récursif non linéaire.

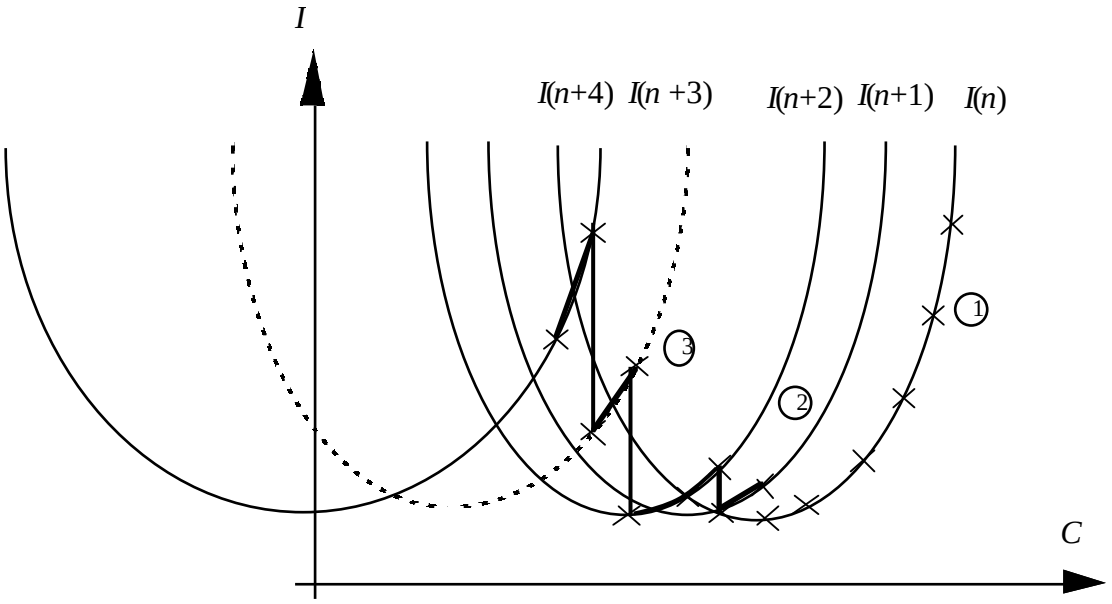

Figure 4. Surfaces d'erreur $I(n)$.

$z_{M+1}^{1}(m)\ ,\ \dots\ ,\ z_{M+N}^{1}(m)$	$\frac{\partial\, z_{M+1}^{1}(m)}{\partial c_{pq}}\ ,\ \dots\ ,\ \frac{\partial\, z_{M+N}^{1}(m)}{\partial c_{pq}}$
1) $z_{M+1}^{2}(m-1)\ ,\ \dots\ ,\ z_{M+N}^{2}(m-1)$	$\frac{\partial\, z_{M+1}^{2}(m-1)}{\partial c_{pq}}\ ,\ \dots\ ,\ \frac{\partial\, z_{M+N}^{2}(m-1)}{\partial c_{pq}}$
2) $z_{M+1}^{2}(m-1)\ ,\ \dots\ ,\ z_{M+N}^{2}(m-1)$	$0\ ,\ \dots\ ,\ 0$
3) réponses désirées	$\frac{\partial\, z_{M+1}^{2}(m-1)}{\partial c_{pq}}\ ,\ \dots\ ,\ \frac{\partial\, z_{M+N}^{2}(m-1)}{\partial c_{pq}}$
4) réponses désirées	$0\ ,\ \dots\ ,\ 0$

Tableau 1 : classification des algorithmes à partir des initialisations du système d'argument $m \in [n-N_c+2,\ n]$ pour la modification des coefficients à l'instant n : calcul direct. Pour les lignes 1, 2 et 3, on utilise les valeurs calculées à l'instant $n-1$ pour initialiser la première copie du système d'argument $m = n - N_c + 1$.

$z_{M+1}^{1}(m)\ ,\ \dots\ ,\ z_{M+N}^{1}(m)$	$\frac{\partial\, z_{M+1}^{1}(m)}{\partial c_{pq}}\ ,\ \dots\ ,\ \frac{\partial\, z_{M+N}^{1}(m)}{\partial c_{pq}}$
1) $z_{M+1}^{2}(m-1)\ ,\ \dots\ ,\ z_{M+N}^{2}(m-1)$	$0\ ,\ \dots\ ,\ 0$ (imposé par la rétropropagation)
2) réponses désirées	$0\ ,\ \dots\ ,\ 0$ (imposé par la rétropropagation)

Tableau 2 : classification des algorithmes à partir des initialisations du système d'argument $m \in [n-N_c+2,\ n]$ pour la modification des coefficients à l'instant n : rétropropagation. Pour la ligne 1, on utilise les valeurs calculées à l'instant $n-1$ pour initialiser la première copie du système d'argument $m = n - N_c + 1$.

Légendes.

Figure 1. Neurone formel

Figure 2. Réseau de neurones non bouclé complètement connecté utilisé comme filtre transverse non linéaire.

Figure 3. Réseau de neurones bouclé (modèle NARMAX) utilisé comme filtre récursif non linéaire.

Figure 4. Surfaces d'erreur $I(n)$.

Figure 5. Dépliement depuis l'instant initial du réseau bouclé pour le calcul de e(m) à l'instant n.

Figure 6. Système d'argument m pour le calcul de e(m) avec N_t copies à l'instant n.

Tableau 1 : classification des algorithmes à partir des initialisations du système d'argument $m \in [n-N_c+2,\ n]$ pour la modification des coefficients à l'instant n : calcul direct. Pour les lignes 1, 2 et 3, on utilise les valeurs calculées à l'instant $n-1$ pour initialiser la première copie du système d'argument $m = n - N_c + 1$.

Tableau 2 : classification des algorithmes à partir des initialisations du système d'argument $m \in [n-N_c+2,\ n]$ pour la modification des coefficients à l'instant n : rétropropagation. Pour la ligne 1, on utilise les valeurs calculées à l'instant $n-1$ pour initialiser la première copie du système d'argument $m = n - N_c + 1$.