UNIVERSITE PARIS 13 - INSTITUT GALILEE
Laboratoire d'Informatique de Paris Nord (L.I.P.N.)

N^o attribué par la bibliothèque

THESE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ PARIS 13

Discipline : **INFORMATIQUE**

présentée et soutenue publiquement

par

Méziane YACOUB

le **20 Janvier 1999**

Titre:

Sélection de Caractéristiques et Optimisation d'Architectures dans les Systèmes d'Apprentissage Connexionnistes.

Directeur de thèse : Younès BENNANI

JURY

M. Gérard PLATEAU	Président
M. Patrick GALLINARI	Rapporteur
M. Sarunas RAUDYS	Rapporteur
M. Younès BENNANI	Directeur
M. Bruce DENBY	Examinateur
M. Maurice NIVAT	Examinateur

Université Paris 13 - Institut Galilée - Avenue Jean-Baptiste Clément 93430 Villetaneuse (France)

Ajustement de la Capacité de Généralisation des Systèmes Connexionnistes : Sélection de Caractéristiques et Optimisation d'Architectures

---

Generalization Ability Adjustment in Connectionist Systems : Features Selection and Architectures Optimization

---

Sélection de Caractéristiques et Optimisation d'Architectures dans les Systèmes d'Apprentissage Connexionnistes.

Features Selection and Architectures Optimization in Connectionist Learning Systems.

Résumé

Cette thèse est consacrée au problème du choix d'une architecture dont la capacité est adaptée à la difficulté de la tâche. Nous proposons une approche structurelle du contrôle de la capacité de généralisation des systèmes d'apprentissage connexionnistes. Cette approche est basée sur une mesure de pertinence, baptisée HVS (Heuristic for Variable Selection), permettant d'évaluer l'importance de chacune des composantes du modèle. La mesure sera utilisée dans un premier temps pour le problème difficile de la sélection de variables puis étendue à l'optimisation d'architectures de type perceptron multicouche. Une autre utilisation de cette mesure nous a permis de développer une méthodologie d'aide au choix d'une architecture initiale pour traiter des séquences temporelles. Enfin on montre sur un problème réel d'identification de visages qu'une extension de la mesure de pertinence HVS permet de détecter et de sélectionner des zones discriminantes pour ce type d'application en reconnaissance des formes.

Abstract

This thesis deals with the problem of adapting the effective network complexity to the information contained in the training data set, and the task's difficulty. We propose a structural approach for generalization capacity control in connectionist learning systems. This approach is based on a heuristic measure, named HVS (Heuristic for Variable Selection), which allows one to evaluate the importance of each component of the model. First we use this measure for the difficult variable selection problem, then extend to architectures optimization in feedforward connectionist models. Another use of this method allows us to develop a methodology for initial architecture selection to treat temporal sequences. Finally we show on a real problem of face identification that an extension of the HVS measure allows to detect and select the discriminatory regions for this type of application in patterns recognition.

Résumé

Cette thèse est consacrée au problème du choix d'une architecture dont la capacité est adaptée à la difficulté de la tâche. Nous proposons une approche structurelle du contrôle de la capacité de généralisation des systèmes d'apprentissage connexionnistes. Cette approche est basée sur une mesure de pertinence, baptisée HVS (Heuristic for Variable Selection), permettant d'évaluer l'importance de chacune des composantes du modèle. La mesure sera utilisée dans un premier temps pour le problème difficile de la sélection de variables puis étendue à l'optimisation d'architectures de type perceptron multicouche. Une autre utilisation de cette mesure nous a permis de développer une méthodologie d'aide au choix d'une architecture initiale pour traiter des séquences temporelles. Enfin on montre sur un problème réel d'identification de visages qu'une extension de la mesure de pertinence HVS permet de détecter et de sélectionner des zones discriminantes pour ce type d'application en reconnaissance des formes.

Mots-Clés

Réseaux Connexionnistes, Apprentissage, Modélisation, Discrimination, Généralisation, Ajustement de la capacité, Régularisation Implicite, Sélection de Caractéristiques, Reconnaissance de Formes, Optimisation d'Architectures, selection d'architectures.

Abstract

This thesis deals with the problem of adapting the effective network complexity to the information contained in the training data set, and the task's difficulty. We propose a structural approach for generalization capacity control in connectionist learning systems. This approach is based on a heuristic measure, named HVS (Heuristic for Variable Selection), which allows one to evaluate the importance of each component of the model. First we use this measure for the difficult variable selection problem, then extend to architectures optimization in feedforward connectionist models. Another use of this method allows us to develop a methodology for initial architecture selection to treat temporal sequences. Finally we show on a real problem of face identification that an extension of the HVS measure allows to detect and select the discriminatory regions for this type of application in patterns recognition.

Key-Words

Connectionist Systems, Learning, Modelling, Discrimination, Generalization, Adjustement of the capacity, Implicite Regularization, Characteristics Selections, Pattern Recognition, architecture Optimization, architecture selection.

A mes parents

A mon frère

A mes soeurs

Remerciements

Je tiens tout d'abord à rendre hommage au Professeur Ahmed SAOUDI, mon premier directeur de thèse, décédé au mois d'Août 1993. C'est lui qui a initié mes recherches sur les réseaux connexionnistes. J'ai particulièrement apprécié ses compétences, ses qualités humaines et sa disponibilité. Sa disparition laisse un énorme vide.

Je tiens également à exprimer ma profonde reconnaissance à Monsieur Younès BENNANI, Maître de conférences habilité, pour avoir été l'instigateur de ce travail, et m'avoir permis de le mener à bien par ses précieux conseils et nombreuses suggestions. Merci pour sa disponibilité, sa bonne humeur et pour le soutien sans failles qu'il m'a accordé.

Je remercie Monsieur Gérard PLATEAU, professeur à l'Université Paris 13, qui m'a fait l'honneur de présider ce jury.

Monsieur Patrick GALLINARI, Professeur à l'Université Paris 6, et Monsieur Sarunas RAUDYS, Professeur à l'Institut de Mathématiques et Informatique de Vilnius, Lithuanie, ont accepté d'évaluer mon travail de thèse, et m'ont adressé des critiques instructives. Je tiens à les en remercier vivement.

Je remercie également Monsieur Bruce DENBY, Professeur à l'Université de Versailles et Monsieur Maurice NIVAT, Professeur à l'Université Paris 7, qui ont accepté d'être membres de ce jury.

Que tous les membres du Laboratoire d'Informatique de l'Université Paris 13 trouvent ici ma marque de sympathie.

Je remercie tous ceux qui m'ont aidé, de près ou de loin, à réaliser ce travail.

Je ne saurais comment remercier mes parents, ma famille et mon oncle pour m'avoir toujours soutenu et encouragé.

Table des matières

Table des figures

Liste des tableaux

Introduction

Généralement, on fait remonter le début des travaux sur les réseaux de neurones aux travaux de McCulloch et Pitts [66]. Ils proposent un modèle mathématique très simplifié d'un neurone biologique. Ce neurone de McCulloch et Pitts, ou neurone formel, est l'unité de base des modèles connexionnistes. Dans les années soixante, Rosenblatt propose un modèle de machine basé sur des neurones formels capable d'apprendre à partir d'exemples, le Perceptron [83]. Le Perceptron reste une étape importante pour l'étude des réseaux connexionnistes. Les limitations de cette machine ont été démontrées par Minsky et Papert [68] : elle ne peut résoudre que les problèmes linéairement séparables. Pour dépasser ces limitations, il fallait ajouter des couches intermédiaires de transformations "couches cachées" et construire un perceptron multicouche. Seulement là, on ne savait pas comment ajuster les poids par apprentissage. Comme on ne connaît pas lors d'une présentation d'exemple, l'état souhaité des unités cachées, on ne peut pas modifier les poids des connexions se terminant sur ces unités suivant l'algorithme du perceptron, c'est ce qu'on appelle le "credit assignment problem". Non seulement Minsky et Papert montrent les limites théoriques du perceptron, mais ils conclurent qu'il n'existait probablement pas de procédure d'apprentissage pour le perceptron multicouche. L'effet fut presque immédiat : la majorité des chercheurs se désintéressèrent des réseaux de neurones pour se tourner vers l'approche symbolique de l'intelligence artificielle, qui semblait beaucoup plus prometteuse.

On considère en général que c'est l'article de Hopfield [46] qui a motivé les chercheurs à s'intéresser de nouveau aux réseaux connexionnistes. Ce redémarrage de l'intérêt fût allimenté, entre autres, par les travaux de Kohonen [51] sur les mémoires associatives et de Ackley, Hinton et Sejnowski [1] avec des modèles dynamiques stochastiques. Le véritable évènement arrive en 1985-1986 : un algo-

rithme d'apprentissage permettant d'utiliser des perceptrons multicouche, appelé algorithme de rétro-propagation du gradient, est mis au point [86, 57]. Il constitue une solution algorithmique aux problèmes rencontrés par le modèle du perceptron. En fait, on peut considérer cet évènement comme une redécouverte de l'algorithme de la rétro-propagation du gradient. En effet, ce type d'algorithme a été largement utilisé depuis les années soixante dans d'autres domaines comme le contrôle [18].

Le Pereceptron multicouche est le modèle le plus utilisé dans le domaine des réseaux connexionnistes. Cette popularité est dûe essentiellement à sa propriété d'approximateur universel [23, 33, 44, 48, 97]. Cependant, pour un problème donné, on ne connait pas le nombre nécessaire de couches et de neurones par couche. Ceci est important puisqu'un nombre de neurones trop petit induira une modélisation insuffisante, mais un trop grand nombre de neurones entraîne une surparamétrisation du modèle, qui nuira aux performances en généralisation, c'est-à-dire les performances du réseau sur les exemples qu'il n'a pas vus lors de l'apprentissage. Dans ce dernier cas, on parle de sur-apprentissage : le modèle apprend par coeur les exemples d'apprentissage.

Ces dix dernières années, les recherches très actives dans le domaine connexionniste ont permis l'apparition de nouveaux modèles, de nouveaux algorithmes et de nouveaux résultats théoriques. Les réseaux connexionnistes offrent ainsi des techniques pour divers domaines d'application : la discrimination, la classification, la modélisation, la prévision. Ils ont prouvé leur efficacité sur de nombreux problèmes dans chacun de ces domaines.

Dans cette thèse, nous nous intéressons au problème du contrôle de la capacité de généralisation des systèmes d'apprentissage connexionnistes. Nous nous intéressons essentiellement aux réseaux multicouche qui constituent le modèle le plus largement utilisé pour les applications pratiques. Nous proposons une nouvelle mesure heuristique, nommée HVS (Heuristic for Variable Selection), permettant d'estimer l'importance de chaque unité dans un perceptron multicouche. Nous l'utiliserons pour la sélection de variables, puis pour l'optimisation d'architecture. Une autre direction de recherche que nous avons commencé à explorer consiste à optimiser des architectures incluant des connaissances a priori sur le problème, comme dans le cas du domaine de traitement d'images par exemple.

Nos résultats préliminaires montrent qu'il est intéressant de généraliser la notion de sélection de variables aux notions de sélection de traits et de zones discriminantes. Nous nous intéressons aussi aux réseaux temporels, notamment à ceux utilisés pour l'apprentissage de machines d'état fini. Nous montrons comment on peut extraire de la base d'apprentissage une information utile pour le choix de l'architecture initiale.

Cette thèse est divisée en cinq chapitres :

- Dans le chapitre 1, nous présentons quelques notions liées aux systèmes d'apprentissage supervisé à partir d'exemples, les problématiques de généralisation de ces systèmes, quelques résultats de l'approche développée par Vapnik et quelques éléments de base sur les réseaux de neurones. Nous présentons à la fin de ce chapitre une brève description des heuristiques développées pour améliorer les performances en généralisation et nous situons nos contributions.

- Dans le chapitre 2, nous nous intéressons au problème difficile de la sélection de variables. Nous présentons une nouvelle mesure heuristique, nommée HVS (Heuristic for Variable Selection), permettant d'évaluer l'importance de chaque variable pour l'estimation d'un processus. Ensuite, nous présentons une procédure neuronale de sélection de variables, basée sur la mesure HVS. Nous testons son efficacité sur un problème de discrimination et un problème de régression et nous comparons nos résultats à ceux obtenus, par d'autres chercheurs, en utilisant d'autres méthodes.

- Le problème auquel on s'intéresse dans le chapite 3 est l'ajustement de la taille du réseau afin que sa complexité soit adaptée à la difficulté du problème à résoudre. Nous proposons d'utiliser HVS pour l'optimisation d'architecture en étendant son application aux unités des couches cachées.

- L'objectif du chapitre 4 est d'utiliser notre méthode de sélection de variables pour détecter et sélectionner des zones discriminantes dans une image. Nous proposons une autre extension de l'application de HVS permettant au système d'apprentissage de se concentrer sur l'information utile pour extraire des traits et effectuer une bonne discrimination. Dans ce chapitre nous avons utilisé des architectures incorporant des connaissances a

priori sur le problème. L'utilisation de HVS dans ce cas peut être vu comme une sélection de connaissances a priori.

– Dans le chapitre 5, nous nous intéressons aux réseaux temporels, particulièrement à ceux utilisés pour l'apprentissage de machines d'état fini. Nous proposons une méthode permettant d'extraire de la base d'apprentissage une connaissance utile pour le choix d'une architecture initiale. Cette architecture permettra ensuite la modèlisation de la machine étudiée.

Chapitre 1

Réseaux de Neurones : Apprentissage et Généralisation

Dans ce chapitre nous allons principalement voir que les résultats de Vapnik et Chervonenskis sur la théorie de l'apprentissage sont difficilement applicables dans le domaine des réseaux connexionnistes. Cette difficulté a motivé le développement de plusieurs approches heuristiques, parallèlement aux études théoriques, pour améliorer les performances en généralisation de ces systèmes d'apprentissage. Nous situons nos contributions par rapport à ces approches.

1.1 Introduction

Dans notre étude nous nous baserons essentiellement sur un modèle générique qui est le perceptron multicouche (PMC) [86].

Le perceptron multicouche est le modèle connexionniste le plus étudié et le plus utilsé du fait de sa simplicité et de son efficacité. Ce modèle a plusieurs domaines d'application, tels que la discrimination, l'identification et la modélisation. Néanmoins, il faut signaler qu'une utilisation aveugle de ce modèle peut donner des résultats aberrants.

D'un point de vue théorique, on sait que les perceptrons multicouche à une seule couche cachée sont des approximateurs universels [23, 33, 44, 48, 97]. Il suffit de choisir un réseau suffisamment complexe pour pouvoir obtenir une erreur sur l'ensemble d'apprentissage aussi petite que l'on désire. En raison de cette propriété d'approximateur universel, le PMC représente une alternative très prometteuse à la modélisation paramétrique. Cependant, le choix d'un modèle trop complexe peut mener à minimiser l'erreur en apprentissage, sans toutefois diminuer l'erreur en généralisation, c'est-à-dire l'erreur sur l'ensemble des exemples non appris. Le problème de la généralisation a été abordé par différents chercheurs. Les résultats les plus importants sont ceux obtenus dans le cadre de l'approche développée par Vapnik [91]. Malgré l'importance de ces résultats, ils restent cependant difficilement applicables dans le domaine des systèmes d'apprentissage connexionnistes. Parallèlement aux études théoriques, plusieurs approches heuristiques ont été proposées et largement utilisées. Pour une excellente synthèse de ces approches heuristiques, le lecteur pourra consulter [34].

Dans ce chapitre, nous présentons quelques notions liées aux systèmes d'apprentissage supervisé à partir d'exemples, les problématiques de généralisation de ces systèmes, quelques résultats de l'approche développée par Vapnik et quelques notions de bases sur les réseaux de neurones. A la fin de ce chapitre, nous présentons une brève description des heuristiques les plus populaires développées pour améliorer les performances en généralisation en situant nos contributions parmi elles.

1.2 Apprentissage supervisé à partir d'exemples

1.2.1 Les principales composantes du modèle d'apprentissage

Un modèle général d'apprentissage à partir d'exemples peut être décrit à travers trois principales composantes [91]: un générateur d'observations, un superviseur et un système d'apprentissage (figure 1.1).

- Le générateur d'observations génère des observations $x \in \mathbb{R}^n$ avec une probabilité $p(x)$ fixe mais inconnue.

- Le superviseur associe une sortie y à chaque entrée x selon une distribution de probabilité conditionnelle $p(x|y)$ également fixe mais inconnue.

- Le système d'apprentissage est capable d'implémenter un ensemble de fonctions :

$$\mathcal{F} = \{f(x, w),\ w \in \Lambda\}, \quad \text{où } \Lambda \text{ est un ensemble de paramètres.}$$

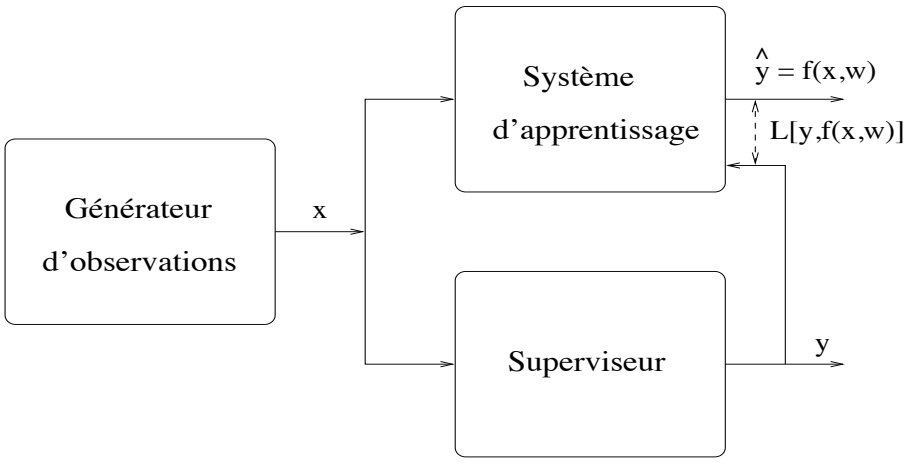


FIG. 1.1 – *Schéma général d'un modèle d'apprentissage.*

1.2.2 Objectif de l'apprentissage

L'objectif de l'apprentissage est d'extraire une relation pertinente liant les couples (x_i, y_i) et de sélectionner parmi les fonctions $f(x, w)$, $w \in \Lambda$ celle qui approxime au mieux la réponse y du superviseur. Cette sélection sera basée sur un échantillon $D_l = \{(x_i, y_i)\}_{i=1}^{l}$ de l observations indépendantes et identiquement distribuées (i.i.d.) de distribution de probabilité conjointe $p(x, y) = p(x)p(y|x)$ inconnue.

Pour sélectionner dans $\mathcal{F}$ la fonction qui approxime au mieux la réponse du superviseur, on utilise une fonction de perte $L[y, f(x, w)]$ qui mesure l'écart entre la réponse y du superviseur associée à l'observation x et la réponse $f(x, w)$ produite par le système d'apprentissage (figure 1.1).

Le choix de la fonction de perte dépend du problème d'apprentissage :

- en régression : le but est d'estimer au mieux une fonction à valeurs réelles

$$L[y, f(x, w)] = (y - f(x, w))^2$$

- en discrimination : le but est de séparer des données en classes distinctes

$$L[y, f(x, w)] = \begin{cases} 0 & \text{si } \; y = f(x, w) \\ \\ 1 & \text{si } \; y \neq f(x, w) \end{cases}$$

- en estimation de densité : on cherche à estimer la distribution de probabilité des données, ce qui correspond à maximiser leur vraissemblance

$$L[y, f(x, w)] = -log f(x, w)$$

L'espérance mathématique de $L[y, f(x, w)]$ définit alors la fonction de risque $R(w)$:

$$R(w) = \int L[y, f(x, w)] p(x, y) dx dy \quad w \in \Lambda \tag{1.1}$$

Ce risque R(w) représente l'erreur de généralisation.

L'apprentissage revient donc à trouver la fonction $f(.,w^*)$ dans $\mathcal{F}$ qui minimise le risque $R(w)$:

$$f(.,w^*) = arg \min_{f(.,w)} R(w) \tag{1.2}$$

1.2.3 La Minimisation du Risque Empirique

1.2.3.1 Principe

La minimisation de $R(w)$ n'est pas possible du fait que la densité de probabilité $p(x,y)$ nécessaire pour le calcul de $R(w)$ est inconnue. Une solution consiste à utiliser un principe d'induction et à remplacer $R(w)$ par son estimation empirique $R_{emp}(w)$ calculée sur un ensemble d'apprentissage D_l de taille l:

$$R_{emp}(w) = \frac{1}{l} \sum_{i=1}^{l} L[y_i, f(x_i, w)] \tag{1.3}$$

Ce principe est connu comme la minimisation du risque empirique (MRE). Ce nouveau risque $R_{emp}(w)$ représente l'erreur d'apprentissage.

L'apprentissage à partir de D_l consiste donc à approcher la fonction optimale $f(.,w^*)$ par une autre fonction $f(.,w^0)$ où w^0 minimise $R_{emp}(w)$:

$$f(.,w^0) = arg \min_{f(.,w)} R_{emp}(w) \tag{1.4}$$

Il est alors important de savoir si $f(.,w^0)$ est proche de $f(.,w^*)$. En d'autres termes: est-ce que le principe MRE est consistant?

1.2.3.2 Consistance

Etablir une condition nécessaire et suffisante pour que le principe de MRE soit consistant permettrait de savoir sous quelles conditions un problème d'apprentis-

sage admet une solution. Vapnik et Chervonenskis montrent le résultat important (théorème clé) suivant :

Théorème 1.2.1 *Si $A \leq R(w) \leq B$ alors le principe MRE est consistant si et seulement si $R_{emp}(w)$ converge uniformément vers $R(w)$ sur $\{f(.,w)/w \in \Lambda\}$:*

$$\lim_{l\to\infty} P(\sup_{w\in\Lambda} |R(w) - R_{emp}(w)| > \varepsilon) = 0, \quad \forall\varepsilon > 0 \tag{1.5}$$

Ce résultat (théorique) n'est pas applicable puisque $R(w)$ n'est pas calculable. Vapnik et Chervonenkis montrent d'autres conditions suffisantes pour la consistance du principe MRE . Parmi leurs résultats, le principal est celui qui se base sur la dimension de Vapnik-Chervonenkis (VC-dimension). D'une part, il ne dépend d'aucune distribution de probabilité et d'autre part, on sait calculer la VC-dimension de certains modèles d'apprentissage. Ils montrent que la consistance de MRE dépend de la VC-dimension (ou capacité) de la famille de fonctions où l'on cherche la solution. Ils fournissent un intervalle de confiance reliant $R(w)$ (erreur de généralisation) et $R_{emp}(w)$ (erreur d'apprentissage) pour toute valeur de w.

Avec une probabilité $1 - \eta$, l'erreur de généralisation est telle que :

$$\forall w \in \Lambda,\ R_{emp}(w) - \Phi(l,h,\eta) \leq R(w) \leq R_{emp}(w) + \Phi(l,h,\eta) \tag{1.6}$$

Par exemple, dans le cas où $f(x,w) \in \{0,1\}$ la largeur Φ est définie par

$$\Phi(l,h,\eta) = \sqrt{\frac{h}{l}(1 + log\frac{2l}{h}) - \frac{1}{l}log\frac{\eta}{4}}$$

La largeur $\Phi(l,h,\eta)$ de cet intervalle de confiance dépend de la taille l de l'échantillon, de la VC-dimension h de la famille de fonctions paramétriques $\{f(.,w), w \in \Lambda\}$ et du niveau de confiance $1 - \eta$.

Le principe de la minimisation du risque empirique est alors efficace si la largeur $\Phi(l,h,\eta)$ de l'intervalle de confiance est faible. Vapnik et Chervonenkis montrent que la largeur $\Phi(l,h,\eta)$ de l'intervalle de confiance est faible lorsque le rapport $\frac{l}{h}$ est suffisamment grand. En revanche, lorsque le ratio $\frac{l}{h}$ est faible, la largeur de l'intervalle de confiance devient significative. Dans ce dernier cas, lorsque le

nombre l d'exemples est fixé, Vapnik a proposé un autre principe d'induction: la minimisation du risque structurel (MRS) [90]. Ce principe consiste à minimiser le risque garanti, c'est-à-dire la somme du risque empirique $R_{emp}(w)$ et de l'intervalle de confiance $\Phi(l, h, \eta)$

$$R_{garanti}(w) = R_{emp}(w) + \Phi(l, h, \eta) \tag{1.7}$$

1.2.4 La Minimisation du Risque Structurel

Ce principe d'induction proposé par Vapnik consiste à définir une suite de sous-ensembles imbriqués $S_i = \{f(., w), w \in \Lambda_i\}$ de l'ensemble des fonctions $S = \{f(., w), w \in \Lambda\}$ implémentées par le système d'apprentissage:

$$S_1 \subset S_2 \subset S_3 \cdots \subset S_n$$

et dont les capacités h_i forment une suite croissante:

$$h_1 < h_2 < \cdots < h_n.$$

Le principe MRS consiste alors à sélectionner la fonction qui minimise le risque empirique dans le sous-ensemble qui minimise le risque garanti. On réalise alors une optimisation en deux étapes (figure 1.2)

Etape 1: calcul du minimum du risque empirique dans chaque sous-ensemble,

$$R_{emp}(w_i^0) = Min_{w \in \Lambda_i} R_{emp}(w)$$

Etape 2: sélection du sous-ensemble présentant le meilleur risque garanti

$$Min_i R_{emp}(w_i^0) + \Phi(l, h_i, \eta)$$

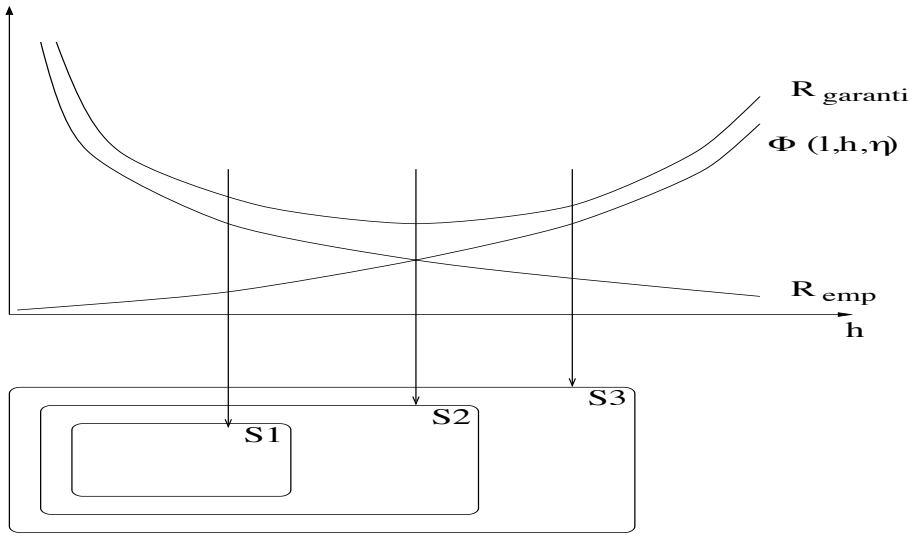


FIG. 1.2 – *Minimisation du risque structurel*

Pour certains modèles d'apprentissage, on ne sait pas calculer la VC-dimension, mais on sait parfois la borner en fonction du nombre de paramètres du modèle. Dans le cas des réseaux de neurones, la dimension de Vapnik est difficile à évaluer. Ainsi, bien que les travaux de Vapnik et Chervonenkis sont d'une très grande importance, ils restent difficilement applicables dans le domaine des réseaux connexionnistes.

La détermination de la VC-dimension des réseaux de neurones est encore très largement ouverte, il n'y a que quelques résultats sur ses bornes [64, 4, 5, 7].

Un autre principe d'induction, le principe de longueur de description minimale, a été proposé par Rissanen [81]. Ce principe est basé sur la complexité de Kolmogorov [52].

Dans la communauté réseaux connexionnistes, parallèlement aux études théoriques, plusieurs heuristiques ont été proposées pour l'ajustement de la capacité de généralisation. Avant de présenter les idées principales sur lesquelles se basent ces heuristiques, nous allons d'abord rappeler quelques notions de base sur les réseaux de neurones et plus particulièrement le perceptron multicouche. Nous reviendrons dans les chapitres 4 et 5 sur des variantes de ces modèles : les réseaux à masques et les réseaux récurrents.

1.3 Les réseaux de neurones

Dans cette section, nous commencerons par présenter l'élément de base, le neurone formel, puis les PMC et leur capacité d'approximation universelle. Nous rappelons ensuite le principe de l'apprentissage dans ces modules.

1.3.1 le neurone

On appelle neurone (figure 1.3) une unité de calcul qui :

- reçoit des données en entrée x_i, sous la forme d'un vecteur de $\mathbf{R}^p$,
- calcule son activation A, en fonction de ses entrées et ses poids w_i, en utilisant une fonction d'activation h,
- et produit une sortie y, à partir de son activation, en utilisant une fonction de transition g.

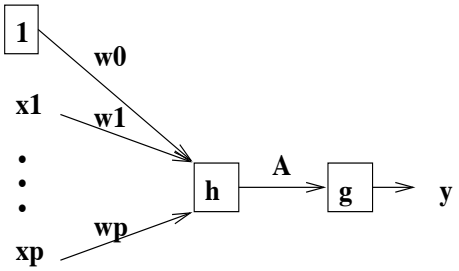


FIG. 1.3 – *Un neurone formel.*

Il existe plusieurs types de neurones, en particulier :

- le neurone produit scalaire : la fonction h est une opération du type produit scalaire ($A = w_0 + \sum_i w_i x_i$) et la fonction g peut être l'identité, une fonction à seuil ou une fonction sigmoïde (figure 1.4).

– le neurone distance: la fonction h est une distance entre les entrées et les poids ($A = ||w - x||$) et la fonction g peut être l'identité ou une fonction noyau (figure 1.4).

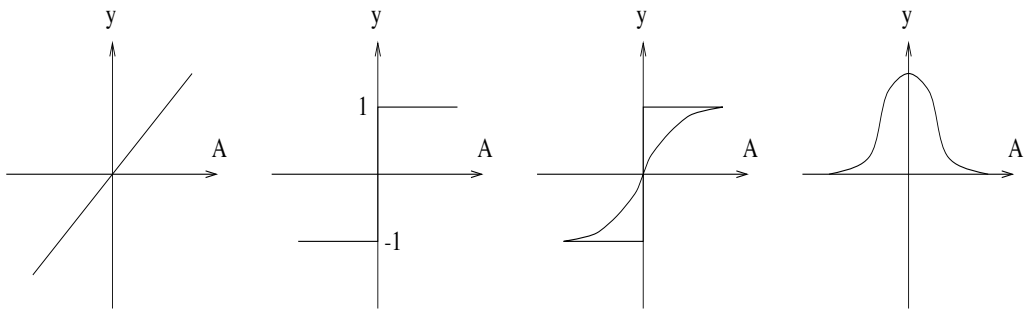


FIG. 1.4 – *De gauche à droite: fonction identité, fonction à seuil, fonction sigmoïde et fonction noyau.*

Les paramètres $w = (w_1, w_2, \ldots, w_p)$ sont les poids de la cellule, w_0 est le biais, $x = (x_1, x_2, \ldots, x_p)$ est l'entrée de la cellule , et y la sortie calculée. Pour simplifier cette notation, on considère généralement que la cellule a pour entrée $x = (x_0, x_1, x_2, \ldots, x_p)$ avec $x_0 = 1$, et w_0 est alors un poids comme les autres.

Un réseau de neurones est un ensemble de neurones interconnectés. Il est entièrement caractérisé par

– son architecture: le nombre de neurones et leur schéma d'interconnexion,

– les fonctions h et g et les poids de ses neurones.

1.3.2 Le perceptron multicouche (PMC)

Un perceptron multicouche [86] est un modèle connexionniste dont les neurones sont organisés en couches successives, et les connexions ne sont autorisées que dans un seul sens, de l'entrée vers la sortie. La première couche est dite couche

d'entrée, et la dernière couche de sortie. Les autres sont des couches cachées. La figure 1.5 montre un exemple de PMC avec une couche d'entrée, deux couches cachées et une couche de sortie.

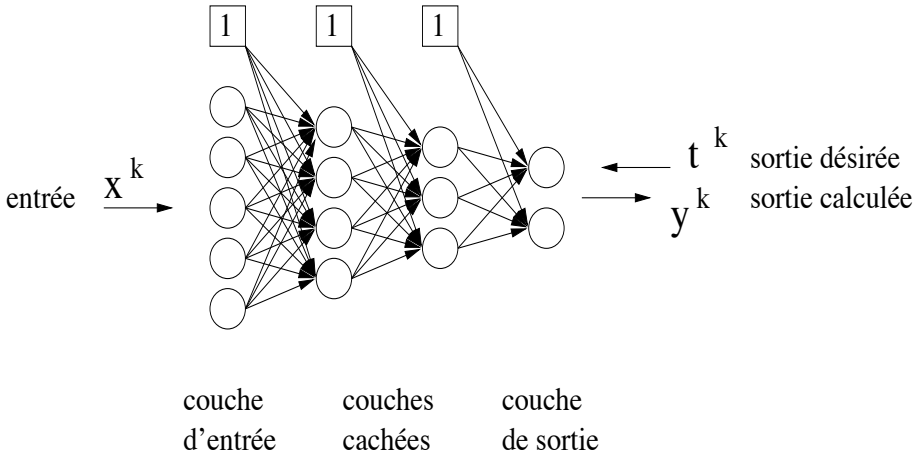


FIG. 1.5 – *Un perceptron multicouche.*

1.3.3 L'apprentissage des PMC

Le choix d'une architecture détermine l'ensemble $\mathcal{F}$ des fonctions dans lequel on va rechercher la solution au problème traité. A chaque ensemble de poids correspond une fonction de transfert $f \in \mathcal{F}$. L'apprentissage s'inscrit alors dans le cadre de de la minimisation de risque empirique. Le PMC est entraîné à partir d'une base d'apprentissage (l'échantillon D_l) de façon à déterminer les meilleurs poids w^0 qui minimisent une fonction de coût (une mesure empirique du risque) qui représente l'écart entre sorties calculées ($y^k = f(x^k, w)$) et sorties désirées (t^k)

$$w^0 = arg\min_w \frac{1}{l} \sum_{k=1}^{l} [t^k - f(x^k, w)]^2 \tag{1.8}$$

Pour résoudre l'équation (1.8), on utilise généralement des procédures de gradient (voir par exemple [6] pour un ensemble de méthodes d'apprentissage).

1.3.4 Propriétés d'approximation universelle des PMC

Un des résultats les plus importants provient des preuves apportées sur les capacités d'approximation universelles des PMC. En effet, plusieurs études théoriques[23, 33, 44, 48, 97] montrent qu'il existe toujours un réseau à une couche cachée capable d'approcher toute fonction continue avec une précision arbitraire.

D'un point de vue théorique, on sait donc que les PMC, et notamment ceux à une couche cachée, sont des approximateurs universels. Par conséquent, ces modèles sont des candidats possibles pour de nombreux problèmes. Cependant, toutes les approches développées ne fournissent que des théorèmes d'existence d'un PMC à une couche cachée. Ces théorèmes ne démontrent pas qu'un PMC à une seule couche cachée est optimal, il peut exister un PMC à deux ou plusieurs couches cachées ayant un nombre de paramètres plus faible et approchant la fonction en question avec une aussi bonne précision. On ne sait pas non plus combien de neurones sont nécessaires dans la couche cachée.

Cette indétermination du nombre de couches et du nombre de neurones par couche n'est pas sans conséquence puisqu'un nombre de neurones trop petit induira une modélisation insuffisante, et un nombre de neurones trop grand entraîne une surparamétrisation du modèle, qui nuira aux performances en généralisation. Afin que la complexité du réseau soit adaptée au problème à résoudre, plusieurs approches heuristiques ont été développées afin d'ajuster la taille du réseau pendant l'apprentissage. Nous décrivons quelques unes de ces approches dans la section suivante.

1.4 Heuristiques pour l'ajustement de la capacité de généralisation

Pour améliorer les performances en généralisation, plusieurs méthodes heuristiques ont été développées, qui peuvent être regroupées en deux grandes classes :

1. les méthodes de régularisation formelle
2. les méthodes de régularisation structurelle

1.4.1 Les méthodes de régularisation formelle

Cette classe consiste à introduire une information a priori dans le processus d'apprentissage. Les principales méthodes sont :

- l'introduction des termes de pénalisation dans la fonction de coût, permettant de limiter le domaine des solutions admissibles du système. La fonction de coût à optimiser lors de l'apprentissage est alors composée de deux termes :

$$R(w) = R_1(w) + \lambda R_2(w)$$

où $R_1(w)$ est un coût sur les données, $R_2(w)$ un coût sur la complexité et λ le paramètre de régularisation permettant de contrôler l'importance relative de $R_2(w)$.

  - par exemple le "weight decay" [76, 54] contraint les poids à tendre vers zéro, le terme associé est :

$$R_2(w) = \sum_i w_i^2.$$

  - dans la méthode de "weight elimination" [95], le terme utilisé est :

$$R_2(w) = \sum_i \frac{\frac{w_i^2}{w_0^2}}{1 + \frac{w_i^2}{w_0^2}}.$$

    Contrairement au "weight decay" qui a tendance à favoriser plusieurs poids faibles (puisque tous les poids sont attirés vers zéro), avec ce terme seuls les poids faibles seront attirés vers zéro. En quelque sorte, ce terme favorise quelques poids forts plutôt que plusieurs poids faibles.

  - en 1995, Raudys [79] montre que dans certains cas de grands poids peuvent aussi améliorer les performances en généralisation. Il a proposé un terme qualifié d'anti-régularisateur :

$$R_2(w) = -\sum_i w_i^2$$

En effet ce terme est sensé favoriser de fortes valeurs pour certains poids.

- d'autres termes ont été proposés, comme le "soft weight sharing" [74] ou le terme proposé par Chauvin [19] qui pénalise poids et activités des neurones cachés.

- "early stopping" qui consiste à arrêter l'apprentissage avant l'optimum pour éviter le phénomème de sur-apprentissage. La figure 1.6 montre l'évolution typique de l'erreur sur l'ensemble d'apprentissage et sur l'ensemble de test, en fonction des itérations d'apprentissage. L'erreur sur l'ensemble d'apprentissage diminue toujours, alors que sur l'ensemble de test l'erreur diminue, atteint son minimum puis augmente. Il faut donc arrêter l'apprentissage à l'instant où les performances en généralisation sont meilleures. On utilise alors un ensemble de validation distinct de l'ensemble de test et on arrête l'apprentissage au minimum de l'erreur sur cet ensemble.

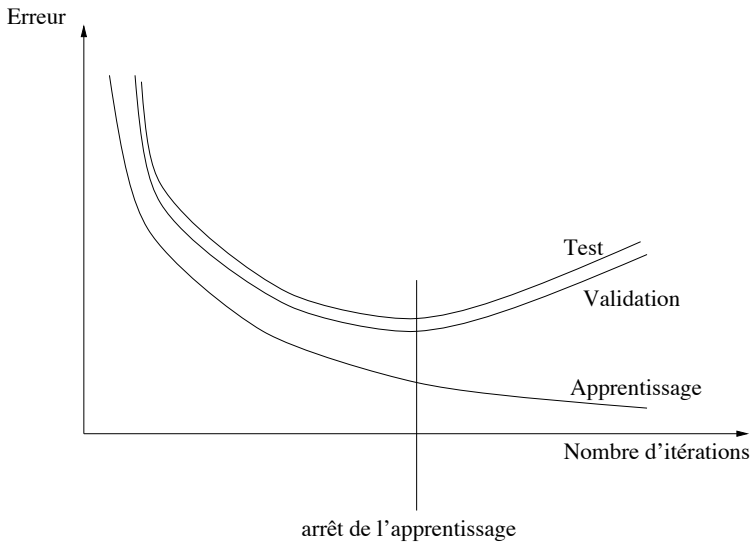


FIG. 1.6 – *Evolution typique de l'erreur sur les bases d'apprentissage, de validation et de test au cours de l'apprentissage.*

- le bruitage des données d'apprentissage: le principe de cette heuristique consiste à ajouter, pendant la phase d'apprentissage, un bruit aux entrées,

les sorties désirées étant inchangées. Cette heuristique accroît donc artificiellement le nombre d'exemples dans la base d'apprentissage et pénalise les solutions sensibles aux petites variations des entrées. Pour un certain type de problèmes, cette heuristique permet d'améliorer les résultats en généralisation. Une analyse de l'injection de bruit peut être trouvée dans [65, 13] et une étude plus détaillée dans [41].

1.4.2 Les méthodes de régularisation structurelle

Dans cette classe on trouve notamment les techniques constructives qui font croître l'architecture du système, les techniques d'élagage des poids ou des unités superflus du système et les modèles qui incorporent des connaissances a priori sur le problème à traiter

- les méthodes constructives proposent de rendre l'architecture incrémentale. En partant d'un réseau de faible dimension, on ajoute des neurones ou des couches selon un certain critère. On peut citer, par exemple, "Upstart algorithm"[32], "Tiling algorithm"[67] et l'algorithme "cascade correlation" [28].

  Les méthodes basées sur cette approche sont généralement "gourmandes" en nombre de neurones ajoutés, elles favorisent un apprentissage par coeur et ne donnent donc pas de bons résults en généralisation.

- à l'opposé des méthodes constructives, les méthodes d'élagage proposent de partir d'un réseau surdimensionné et de le simplifier au cours de l'apprentissage. Les techniques d'élagage ont donc pour but d'éliminer les paramètres inutiles du système.

  La plupart de ces méthodes sont fondées sur un calcul de sensibilité. Pour sélectionner le poids ou le neurone à supprimer, on calcule la sensibilité de l'erreur à la suppression des paramètres. La technique d'élagage la plus populaire est Optimal Brain Damage (OBD) proposée par Le Cun et al [60]. Dans OBD, la sensibilité d'un poids w_j est définie par $s_j = \frac{1}{2}\frac{\partial^2 MSE}{\partial w_j^2} w_j^2$ où MSE est l'erreur quadratique moyenne. Pour d'autres techniques d'élagage, voir par exemple [80].

- l'architecture d'un réseau détermine sa complexité potentielle, c'est-à-dire la classe des fonctions calculables par celui-ci. D'un côté, nous avons vu que les réseaux MLP à une couche cachée sont des approximateurs universels. D'un autre côté, les travaux de Vapnik montrent que le choix d'un modèle trop complexe peut mener à minimiser le coût empirique, sans toutefois diminuer le coût réel. La qualité de l'apprentissage dépend donc de l'architecture du modèle utilisé. L'utilisation des connaissances a priori sur le problème joue un très grand rôle dans le choix de l'architecture. Les TDNN (Time Delay Neural Network) par exemple sont très utilsés dans le domaine de la parole [55] et de l'image [58]. Ils permettent, par exemple, de mieux prendre en compte l'aspet temporel de la parole et introduisent une certaine invariance par translation des images. Les systèmes modulaires sont aussi très utilisés [9, 12]. La notion de modularité permet de concevoir des systèmes correspondant à la décomposition d'une tâche complexe en sous-tâches beaucoup plus simples à traiter. Elle introduit ainsi dans l'architecture les connaissances de base que l'on possède sur le problème.

1.5 Conclusion du chapitre

Dans ce chapitre, nous avons surtout voulu montrer l'intérêt des approches heuristiques développées pour améliorer les performances en généralisation des systèmes connexionnistes.

Nous avons commencé par donner quelques notions liées à l'apprentissage à partir d'exemples, les problématiques de généralisation de ces systèmes et quelques résultats des études développées par Vapnik. Nous avons vu que ces résultats, bien que très importants, restent difficilement applicables dans le domaine des réseaux connexionnistes. Des approches heuristiques ont été développées et utilisées. Ces approches peuvent être regroupées en deux grandes familles : les méthodes de régularisation formelle et les méthodes de régularisation structurelle.

Les techniques d'ajustement de la capacité de généralisation que nous avons développées appartiennent principalement à la classe des techniques de régularisation

structurelle. Dans la suite de cette thèse, nous proposons :

- pour les PMC

  une nouvelle mesure heuristique, nommée HVS, permettant d'évaluer l'importance de chaque unité d'un PMC. Nous l'utiliserons pour la sélection de variables puis pour l'optimisation d'architectures.

- pour les réseaux de type TDNN utilisés en traitement d'images

  nous étendons l'application de HVS pour ce type d'architectures. Nous proposons une méthode permettant d'une part de réduire davantage la taille des architectures de ce type, et d'autre part de sélectionner les zones discriminantes les plus pertinentes.

- pour les réseau temporels

  une méthode d'aide au choix d'une architecture initiale pour l'apprentissage de machines d'état fini est proposée.

Chapitre 2

Sélection de Variables en Discrimination et en Modélisation

Ce chapitre est consacré essentiellement à notre mesure heuristique, nommée HVS (Heuristique for Variable Selection)[98], que nous utiliserons pour la sélection de variables. HVS ne demande que peu de calculs simples, faciles à implémenter. Nous testerons son efficacité sur un problème de discrimination et un problème de régression, après avoir montré sa capacité de détection et de quantification de pertinence.[1]

1. Résultats publiés dans ANNIE'97 [98], ICANN'98 [99] et soumis à la revue IJNS [102] .

2.1 Introduction

Dans ce chapitre, nous nous intéressons au problème difficile de la sélection de variables. La sélection de variables[2] consiste à choisir un sous-ensemble de variables de l'ensemble qu'on a choisi au départ. Elle représente une des difficultés majeures de l'estimation d'un modèle. Il faut utiliser suffisamment de variables afin d'expliquer convenablement le processus à modéliser, mais d'un autre côté, inclure trop de variables aura pour conséquence d'augmenter artificiellement la complexité du modèle. Généralement, il n'y a pas un sous-ensemble meilleur que les autres ; il en existe la plupart du temps plusieurs. Ce problème est, la plupart du temps, non monotone, c'est-à-dire que le meilleur sous-ensemble à p variables ne contient pas forcément le meilleur sous-ensemble à q variables ($q < p$). Nous écartons l'examen de toutes les combinaisons possibles, pour p variables il faudrait en examiner $2^p - 1$, ce qui n'est réalisable que si p est faible. Notez qu'avec 30 variables nous avons plus d'un milliard de combinaisons! S'il ne faut en moyenne qu'un dixième de seconde pour calculer les éléments d'une combinaison, il faudrait plus de trois ans pour les faire toutes. La sélection de variables nécessite donc des mesures (ou des critères) pour évaluer l'importance de chaque variable ou de chaque sous-ensemble de variables, une stratégie de sélection et un critère d'arrêt de la sélection. Nous proposons une nouvelle mesure heuristique, que nous avons appelée HVS, permettant d'évaluer l'importance de chaque variable pour l'estimation d'un processus. Ensuite, nous présentons une procédure neuronale de sélection de variables, basée sur la mesure HVS.

Premièrement, nous rappelons quelques notions de base liées au problème de la sélection de variables. Ensuite nous présentons brièvement les méthodes statistiques puis les approches neuronales de sélection de variables les plus utilisées. Enfin nous présentons notre méthode et testons son efficacité sur quelques problèmes que nous avons choisis.

2. Nous nommerons variables ou caractéristiques les descripteurs qui caractérisent une forme.

2.2 Principe de la sélection de variables

Les méthodes de sélection de variables permettent de retenir un sous-ensemble de variables, de l'ensemble qu'on a choisi au départ, pertinent pour l'objectif qu'on s'est fixé. Il faut distinguer la sélection de variables de l'extraction de variables (figure 2.1)

- l'extraction consiste en la définition de nouvelles variables à partir de l'ensemble qu'on a choisi au départ : partant de d variables définies a priori, on en recherche $d'(d' < d)$ fonction des d primitives. Une technique simple consiste à faire une ACP (Analyse en Composantes Principales) sur les échantillons et prendre comme nouvelles variables les p composantes principales. Ces nouvelles caractéristiques sont calculées sous forme de combinaisons linéaires des variables originelles.

- les méthodes de sélection de variables ont pour but de chercher d' variables parmi les d originelles.

Dans cette étude, nous nous intéresserons qu'à la sélection de variables.

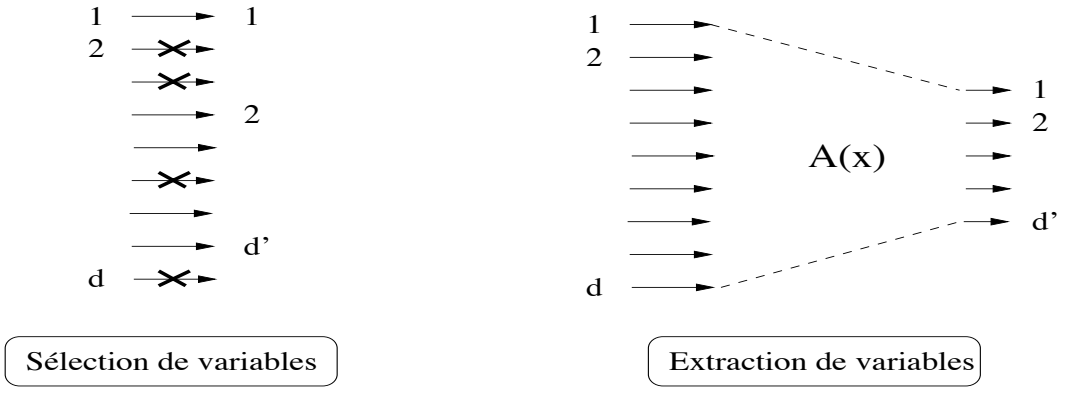


FIG. 2.1 – *La sélection de variables consiste à choisir d' variables parmi les d originelles alors que l'extraction de variables consiste en la définition de d' variables fonction des d originelles.*

D'une façon générale, une méthode de sélection de variables est composée des trois éléments suivants:

- un critère J d'évaluation de l'importance d'une variable ou d'un sous-ensemble de variables,
- une procédure de recherche d'un sous ensemble de variables et
- un critère d'arrêt de la procédure de recherche.

• Critères d'évaluation

D'une façon générale, le but poursuivi lors de la sélection de variables est d'augmenter la puissance de discrimination ou la qualité de prédiction d'un système. De nombreux critères ont été proposés, basés sur des considérations statistiques ou heuristiques. Pour un problème de discrimination, les méthodes classiques de sélection de variables utilisent souvent les critères basés sur les matrices de dispersion, car ils sont directement liés à la géométrie des classes en présence. Les vecteurs moyennes traduisent la position des classes dans l'espace, la matrice de variance-covariance intraclasse caractérise la dispersion des points dans les classes et la matrice de variance-covariance interclasse représente la dispersion des classes entre elles. Des critères liés à des considérations de distances probabilistes ou des mesures d'entropie ont été aussi proposés. Pour un problème de régression, les critères sont plutôt basés sur des estimations de l'erreur de prédiction.

• Techniques de recherche

Une solution possible est l'étude de toutes les combinaisons de variables possibles : c'est une procédure exhaustive très lourde puisque pour p variables il faudrait en examiner $2^p - 1$. Dans beaucoup d'applications, une telle recheche exhaustive n'est pas envisageable, elle n'est possible que si le nombre de variables est faible. On peut alors, soit faire appel à une méthode d'optimisation de type Branch and Bound [72] soit avoir recours à des méthodes sous-optimales. Comme la plupart des critères de sélection ne sont pas monotones, seules des méthodes sous-optimales sont utilisées. Les procédure sous-optimales de sélection

de variables les plus simples sont des procédures séquentielles comme la sélection ascendante, l'élimination descendante et la «stepwise selection»

- la sélection ascendante est effectuée par agrégations successives. A chaque étape, la variable sélectionnée est celle qui, parmi les variables restantes, réalise une optimisation du critère obtenu exprimé avec les variables déjà retenues auxquelles on ajoute une variable.

  Soit E l'ensemble des variables, E_k l'ensemble des variables sélectionnées à la $k$ième étape, et $v_1, v_2, \cdots, v_{d-k}$ les variables encore disponibles. On retient la variable v_i telle que :

  $$J(E_k) = \max_{v_i \in E - E_{k-1}} J(E_{k-1} \cup \{v_i\}) \tag{2.1}$$

  Au départ, l'ensemble des variables est initialisé à l'ensemble vide.

- la procédure élimination descendante est une procédure inverse de la précédente. On part de l'ensemble complet des variables. Au niveau k, on ôte la variable v_i telle que :

  $$J(E_k) = \max_{v_i \in E_{k+1}} J(E_{k+1} - \{v_i\}) \tag{2.2}$$

- la procédure «stepwise selection» est un mélange de la sélection ascendante et de l'élimination descendante. On sélectionne les variables une à une, mais à chaque étape on teste à nouveau si une d'elles ne peut être éliminée.

• Critère d'arrêt

Si on ne connait pas le nombre de variables à sélectionner, il faut introduire un critère pour arrêter la procédure de recherche. La plupart des méthodes de sélection de variables utilisent soit des tests statistiques, soit des critères d'arrêt heuristiques.

Dans le cas de données gaussiennes, Raudys [78] montre, dans le sens de l'erreur de classement, que le nombre optimal de variables dépend de la taille de la base d'apprentissage et de la règle de décision utilisée.

D'autres auteurs comme Jain et Waller [49] montrent que pour des données gaussiennes et un classifieur linéaire le nombre optimal de variables est $d^0 \simeq N - 1$, N étant la taille de l'ensemble d'apprentissage.

2.2.1 Les méthodes statistiques de sélection de variables

Les méthodes statistiques réalisent la sélection de variables indépendamment du modèle utilisé. Dans cette section nous allons présenter quelques mesures d'évaluation proposées dans le cadre de la régression linéaire et du classement. Nous conseillons au lecteur intéressé le livre de Duda et Hart [26] où on trouve deux chapitres sur ce problème.

• En classement, de façon intuitive, plus les classes sont séparées, meilleure sera la discrimination. Par conséquent, il faut sélectionner les variables qui permettent de séparer les classes le mieux possible.

Soit $Y = \{y_j, j = 1, \ldots, D\}$ l'ensemble des variables, la sélection de variables en classement consiste alors a chercher le meilleur sous-ensemble X, $X = \{x_i, i = 1, \ldots, d, x_i \in Y(d < D)\}$, dans Y. L'optimalité de X est définie au sens d'un critère fonctionnel J. Idéalement, $J(X)$ est la probabilité de classement correcte en utilisant X, relativement a n'importe quelle autre combinaison $\Xi = \{\xi_i | i = 1, \ldots, d\}$ prise dans Y. En d'autres termes, X satisfait

$$J(X) = \max_{\Xi} J(\Xi) \tag{2.3}$$

On notera les ensembles X et Ξ par les vecteurs $\underline{x}$ et $\underline{\xi}$ respectivement.

$$\underline{x} = (x_1, x_2, \ldots, x_d)^t$$
$$\underline{\xi} = (\xi_1, \xi_2, \ldots, \xi_d)^t$$

Avec cette notation, le critère s'écrit alors

$$J(\underline{x}) = \max_{\underline{\xi}} J(\underline{\xi}) \tag{2.4}$$

La probabilité d'erreur de classement est une mesure idéale de la séparabilité des classes, elle est définie comme suit

$$e = \int (1 - \max_i P(w_i|\underline{\xi}))P(\underline{\xi})d\underline{\xi} \tag{2.5}$$

où w_i désigne la classe i.

Cette mesure est peu commode à calculer puisque les différentes probabilités intervenant dans son expression sont généralement inconnues. On la remplace par des mesures fondées sur d'autres concepts de séparabilité.

La distance interclasse est la notion la plus simple de séparabilité. Soient w_1 et w_2 deux classes d'effectifs respectifs M et N, résultant d'une phase d'apprentissage. Elles sont d'autant mieux séparables que :

$$d(w_1, w_2) = \frac{1}{MN} \sum_{k,l=1}^{M,N} d(\underline{\xi}_{1k}, \underline{\xi}_{2l}) \text{ est grand (moyenne des distances par paires)} \tag{2.6}$$

Ainsi, $d(w_1, w_2)$ peut être calculée sans détermination préliminaire de la structure probabiliste des classes. Son défaut majeur est qu'elle ne peut donner d'indication sur le recouvrement de w_1 et w_2. Il manque en fait des informations de type probabiliste.

Il faut donc introduire un concept de séparabilité des classes utilisant l'information complète sur la structure probabiliste des classes. Cette information est fournie, par exemple, par les densités de probabilités conditionnelles des classes (ddpc), $p(\underline{\xi}|w_i)$ et les probabilités a priori des classes $p(w_i)$. Intuitivement, le recouvrement de deux densités peut être estimé par la distance entre $p(\underline{\xi}|w_1)$ et $p(\underline{\xi}|w_2)$. En générale, toute fonction J de type

$$J = \int g[p(\underline{\xi}|w_1), p(\underline{\xi}|w_2), P(w_1), P(w_2)] d\underline{\xi} \tag{2.7}$$

satisfaisant :

1. $J \geq 0$
2. Si les deux classes sont totalement séparables, alors J est maximum, i.e.
   J est maximum si $p(\underline{\xi}|w_1) = 0$ quand $p(\underline{\xi}|w_2) \neq 0$, $\forall \underline{\xi}$
3. J vaut 0, quand les densités de probabilités conditionnelles des classes sont identiques, i.e
   $J = 0$ si $p(\underline{\xi}|w_1) = p(\underline{\xi}|w_2)$

peut être utilisée comme distance probabiliste de la séparabilité de deux classes.

Alternativement, la mesure du recouvrement peut être formulée avec la dépendance probabiliste entre les vecteurs $\underline{\xi}$ et les classes w_i. Si les variables aléatoires

$\underline{\xi}$ et w_i sont statistiquement indépendantes alors la probabilité conjointe $P(\underline{\xi}, w_i)$ est donnée par :

$$p(\underline{\xi}, w_i) = P(\underline{\xi})P(w_i) \tag{2.8}$$

alors qu'en général : $p(\underline{\xi}, w_i) = p(\underline{\xi}|w_i)P(w_i)$

Lorsque $\underline{\xi}$ et w_i sont statistiquement indépendantes, l'observation de $\underline{\xi}$ n'apporte aucune information sur w_i : la ddpc $p(\underline{\xi}|w_i)$ est égale à la ddp mixte $p(\underline{\xi})$.

D'un autre coté, lorsque $\underline{\xi}$ est statistiquement dépendant de w_i, la ième ddp $p(\underline{\xi}|w_i)$ est différente de la ddp mixte $p(\underline{\xi})$

Ainsi, la détermination de la dépendance probabiliste entre formes observées et classes d'appartenance relève de la différence (au sens d'une distance fonctionnelle) entre ddp conditionnelles et mixtes. La dépendance probabiliste constitue donc un concept naturel de séparabilité des classes.

Le problème de la sélection de variable peut être abordé en termes de la théorie de l'information. Soit w une variable aléatoire prenant les valeurs w_i, $i = 1, \ldots, c$ avec les probabilités a priori $P(w_i)$. Etant donné $\underline{\xi}$, quelle quantité d'information apporte la connaissance de la valeur de w, ou alternativement, quelle est notre incertitude sur w?
Si pour un certain $\underline{\xi}$ il existe i tel que $P(w_i|\underline{\xi}) = 1$ et $P(w_j|\underline{\xi}) = 0$, $\forall j \neq i$, alors le résultat $w = w_i$ est certain et on apporte aucune information en observant la réalisation de w. Par contre si $P(w_i|\underline{\xi}) = cte$, $\forall i$, l'incertitude est maximale et l'observation de la réalisation de w apporte un gain non nul d'information. Dans ce contexte, plus l'incertitude est petite plus le vecteur $\underline{\xi}$ est pertinent.

Dans la théorie de l'information, l'incertitude est quantifiée par ce qu'on appelle l'entropie $J[P(w_1|\underline{\xi}), \ldots, P(w_c|\underline{\xi})]$, qui est la mesure d'information gagnée d'une experience. Il existe plusieurs mesures d'information, comme par exemple l'entropie généralisée de degré α définie par :
$J^\alpha[P(w_1|\underline{\xi}), P(w_2|\underline{\xi}), \ldots, P(w_c|\underline{\xi})] = (2^{1-\alpha} - 1)^{-1}[\sum_{i=1}^{c} P^\alpha(w_i|\underline{\xi}) - 1]$ où α est un réel positif, $\alpha \neq 1$.

• dans le cas de la régression linéaire, plusieurs mesures d'évaluation ont été proposées. Nous présentons ici deux de ces mesures : S_p et C_p (voir par exemple [88]).

Soit le modèle de régression linéaire standard

$$y = \sum_{i=1}^{k} b_i x_i + e$$

avec une erreur résiduelle e qui est un bruit indépendant suivant une loi $N(0, \sigma^2)$

Avec la notation suivante :

$\hat{y}_j^i$: $j^{\text{ième}}$ estimation de y avec un modèle utilisant i variables

N : nombre d'exemples

$SSR_p = \sum_{j=1}^{N} (\hat{y}_j^p)^2$, somme des carrés des valeurs estimées pour p ($p < k$) variables, après l'élimination de $(k - p)$ variables.

$SST = \sum_{i=1}^{N} y_j^2$, somme des carrés de y.

$MSE_p = \frac{(SST - SSR_p)}{N-k}$

Les mesures S_p et C_p sont définies comme suit :

$$S_p = \frac{(SST - SSR_p)}{(N-p)(N-p-2)} = \frac{MSE_p}{(N-p-2)} \tag{2.9}$$

$$C_p = \frac{(SST - SSR_p)}{\sigma^2} - N + 2p \tag{2.10}$$

2.2.2 Les méthodes neuronales de sélection de variables

Contrairement aux méthodes précédentes, les méthodes neuronales de sélection de variables réalisent la sélection parallèlement au processus d'apprentissage. On peut ainsi surveiller directement leur influence sur les performances du système.

Pour estimer la pertinence d'une variable, la plupart des mesures proposées se basent sur l'une des idées suivantes :

- calcul des dérivées de la sortie par rapport aux entrées [24, 84, 85],
- analyse de la perturbation de la fonction coût lorsqu'on supprime la variable du modèle [69],
- utilisation des informations fournies par les techniques d'élagage [20, 61],
- utilisation des paramètres du modèle pour estimer la pertinence [10, 98].

• La dérivée de la fonction F que représente le réseau par rapport à chacune de ses variables x_i est très utilisée comme mesure de pertinence de la variable x_i. Si une dérivée est proche de zéro pour tous les exemples, alors la variable correspondante n'est pas utilisée par le réseau, et peut donc être éliminée. Dans le cas des PMC, cette dérivée peut se calculer comme une extension de l'algorithme d'apprentissage [42]. Comme ces dérivées peuvent prendre aussi bien des valeurs positives que négatives, produisant une moyenne proche de zéro, c'est la moyenne des valeurs absolues qui est généralement utilisée (ce sont les grandeurs des dérivées qui nous intéressent). Plusieurs mesures de ce type ont été proposées. Par exemple Ruck et al., dans [85], proposent la mesure de pertinence suivante :

$$S_i = \sum_{p=1}^{N} \sum_{s=1}^{m} \left| \frac{\partial F_s}{\partial x_i}(x^p) \right| \tag{2.11}$$

où N est la taille de la base d'apprentissage, m est le nombre de variables, x^p est le $p^{ème}$ exemple et x_i est la $i^{ème}$ variable.

Pour un problème de discrimination, Rossi [84] propose de ne considérer que les exemples qui sont près des frontières interclasses

$$S_i = \frac{1}{m} \sum_{s=1}^{m} \sum_{x \in D, \left\| \frac{\partial F_s}{\partial x}(x) \right\| > \epsilon} \frac{\left| \frac{\partial F_s}{\partial x_i}(x) \right|}{\left\| \frac{\partial F_s}{\partial x}(x) \right\|} \tag{2.12}$$

où ϵ est un seuil choisi.

Czernichow [24] utilise le quantille à 95% de la distribution des valeurs absolues des dérivées de chaque variable

$$S_i = q95 \left| \frac{\partial F}{\partial x_i}(x) \right| \tag{2.13}$$

• La sensibilité de l'erreur à la suppression de chaque variable est utilisée par Moody et Utans dans [69]. Une mesure de sensibilité S_i est calculée pour chaque variable x_i pour évaluer la variation de l'erreur en apprentissage si cette variable est supprimée du réseau. La définition de S_i est :

$$S_i = \frac{1}{N} \sum_{j=1}^{N} (SE(\bar{x}_i) - SE(x_{ij})) \qquad avec \qquad \bar{x}_i = \frac{1}{N} \sum_{j=1}^{N} x_{ij} \tag{2.14}$$

où N est la taille de la base d'apprentissage et x_{ij} la valeur de la $i^{ème}$ variable de l'exemple j. S_i mesure l'effet sur l'erreur quadratique en apprentissage (SE) en remplaçant la variable x_i par sa valeur moyenne sur l'ensemble des exemples. Le remplacement d'une variable par sa moyenne supprime son influence sur la sortie du réseau. Si le nombre de variables est grand, le coût de calcul de S_i est élevé. Ils proposent de l'approximer par le terme linéaire :

$$S_i = \frac{1}{N} \sum_{j=1}^{N} \frac{\partial SE}{\partial x_{ij}} dx_{ij} \qquad avec \qquad dx_{ij} = \bar{x}_i - x_{ij} \tag{2.15}$$

• Certaines des méthodes neuronales de sélection de variables sont des extensions des techniques d'élagage des poids. L'importance de chaque variable s'obtient en sommant les importances des connexions qui partent de celle-ci

$$S_i = \sum_{j \in fan-out(i)} s_{ji} \tag{2.16}$$

où s_{ji} est la sensibilité du poids w_{ji} et $fan - out(i)$ est l'ensemble des neurones qui utilisent comme entrée la sortie du neurone i, par définition on a : $\forall i \in$ couche de sortie, $fan - out(i) = \emptyset$.

En utilisant les informations fournies par l'algorithme OBD [60], Cibas et al., dans [20], proposent la mesure de pertinence suivante :

$$S_i = \sum_{j \in fan-out(i)} \frac{1}{2} \frac{\partial^2 E}{\partial w_{ji}^2} w_{ji}^2 \tag{2.17}$$

De la même façon, en utilisant les informations de EBD (Early Brain Damage) [89], qui est une extension de OBD, Leray et Gallinari proposent la mesure de pertinence suivante :

$$S_i = \sum_{j \in fan-out(i)} \frac{1}{2} \frac{\partial^2 E}{\partial w_{ji}^2} w_{ji}^2 - \frac{\partial E}{\partial w_{ji}} w_{ji} + \frac{1}{2} \frac{(\frac{\partial E}{\partial w_{ji}})^2}{\frac{\partial^2 E}{\partial w_{ji}^2}} \tag{2.18}$$

• La mesure que nous proposons dans la section suivante repose sur les paramètres et la structure du réseau. Dans le cas d'un PMC à une seule couche cachée, elle est définie par :

$$S_i = \sum_{j \in H} \left(\frac{|w_{ji}|}{\sum_{i' \in I} |w_{ji'}|} \sum_{k \in O} \frac{|w_{kj}|}{\sum_{j' \in H} |w_{kj'}|} \right)$$

où I est la couche d'entrée, H la couche cachée et O la couche de sortie. Elle peut se généraliser dans le cas d'un PMC à plusieurs couches cachées. Nous verrons en détail cette mesure dans la section suivante.

Cette mesure se simplifie sous l'hypothèse où chaque vecteur de poids arrivant à un neurone de H et O est de norme L_1 unitaire, sous la forme :

$$S_i = \sum_{j \in H} \sum_{k \in O} |w_{ji} w_{kj}|$$

Dans ce cas, la pertinence partielle de la variable i pour la sortie k est donc la somme des valeurs absolues des produits des poids allant de l'unité i à l'unité j et des poids allant de l'unité j à la sortie k, sur tous les chemins possibles allant de i à k. La pertinence totale de la variable i est estimée alors par la somme des pertinences partielles sur toutes les sorties.

2.3 HVS: une nouvelle mesure heuristique pour la sélection de variables

Dans cette section, nous présentons une nouvelle méthode de sélection de variables, basée sur notre mesure HVS. Dans un premier temps, nous présentons la mesure HVS et nous testons ses capacités à estimer l'importance des variables sur des exemples dont l'importance théorique de chaque variable est connue. Ensuite nous décrivons notre méthode de sélection de variables et nous testons son efficacité sur un problème de discrimination et un problème de régression. Nous comparons nos résultats à ceux obtenus par d'autres chercheurs, en utilisant d'autres méthodes.

2.3.1 La mesure HVS

Soit un réseau multicouche défini par une architecture A(I,H,O), avec une couche d'entrée I, une couche cachée H et une couche de sortie O, et une matrice de poids W. La valeur de la connexion w_{ij} entre deux neurones j et i, reflète l'importance de leur lien. Cette valeur peut être positive ou négative selon que la connexion est excitatrice ou inhibitrice. Dans notre étude nous nous sommes intéressé à la force de ces connexions. Cette force nous la quantifions par $|w_{ij}|$. Dans ce cas, on définit la contribution partielle de chaque unité j connectée à une unité i par un poids w_{ij} (figure 2.2) comme :

$$\pi_{ij} = \frac{|w_{ij}|}{\sum_{k \in fan-in(i)} |w_{ik}|} \tag{2.19}$$

où $fan - in(i)$ est l'ensemble des neurones dont les sorties servent d'entrées au neurone i; par définition on a : $\forall i \in I, fan - in(i) = \emptyset$.

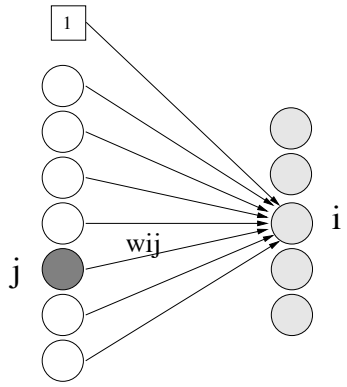


FIG. 2.2 – *La contribution partielle de l'unité j connectée à une unité i par un poids w_{ij} est définie comme le rapport entre la valeur absolue de w_{ij} et la somme des valeurs absolues des poids des connexions arrivant au neurone i.*

Le numérateur de l'équation 2.19 mesure la force de la relation entre l'unité j et l'unité i. Le dénominateur n'est qu'un terme de normalisation. Le terme π_{ij} définit la contribution partielle de l'unité j connectée à l'unité i comme une proportion de toutes les autres forces des connexions arrivant à l'unité i.

Maintenant ce qui nous intéresse est d'estimer la contribution de l'unité j à la décision finale du système. L'unité j envoie des connexions à un ensemble d'unités $fan-out(j)$ avec des contributions partielles π_{ij}, $i \in fan-out(j)$ (figure 2.3).

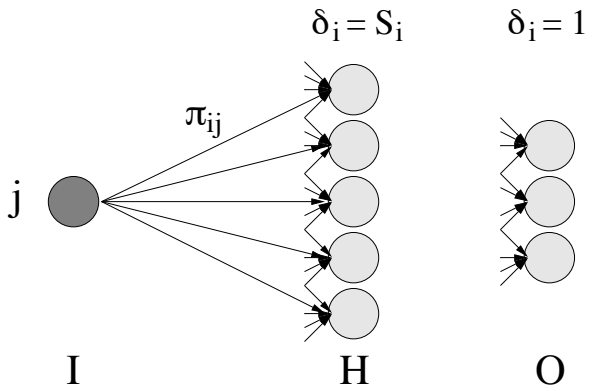


FIG. 2.3 – *La contribution relative de l'unité j à la décision finale du réseau dépend de l'importance des unités qui utilisent sa sortie comme entrée et de sa contribution partielle à chacune de ses unités.*

Chaque élément de $fan-out(j)$ possède une contribution relative S_i (égale à 1 s'il s'agit d'une unité de sortie). A ce stade nous définissons la contribution relative de l'unité j à la décision finale du système comme une somme pondérée de ses contributions partielles par les contributions relatives des unités auxquelles elle envoie une connexion ($\in fan - out(j)$). On obtient alors l'expression suivante :

$$S_j = \sum_{i \in fan-out(j)} \pi_{ij} \delta_i \tag{2.20}$$

avec

$$\delta_i = \begin{cases} 1 & \text{si l'unité i} \in \text{O} \\ \\ S_i & \text{si l'unité i} \in \text{H} \end{cases} \tag{2.21}$$

Interprétation intuitive de la mesure HVS :

Une interprétation intuitive de la définition de HVS peut être donnée comme suit :

La contribution partielle π_{ij}, définie comme le rapport entre la valeur absolue de w_{ij} et la somme des valeurs absolues de tous les poids appartenant à $fan-in(i)$, estime la proportion de la contribution du neurone j au neurone i. Cette proportion est calculée pour chaque neurone i appartenant à $fan-out(j)$, et le terme δ_i est utilisé comme un coefficient de pondération des contributions partielles pour évaluer la contribution relative S_j.

En d'autres termes, on peut résumer le fondement de la mesure HVS par le principe suivant : "il vaut mieux participer moyennement à une décision importante que de participer fortement à une décision inutile".

HVS s'applique à un PMC ayant un nombre quelconque de couches cachées :

La mesure HVS s'applique à un PMC ayant un nombre quelconque de couches cachées. On peut calculer les contributions partielles de tous les neurones du réseau en utilisant l'équation 2.19. Ensuite, connaissant les importances des neurones de la couche de sortie (d'après l'équation 2.21), on peut calculer les contributions relatives de tous les neurones de la dernière couche cachée en utilisant l'équation 2.20. Connaissant les contributions relatives des neurones de la dernière couche cachée, on peut calculer celles des neurones de l'avant dernière couche cachée, et ainsi de suite ...

2.3.2 Capacités d'HVS à estimer l'importance des variables

Dans cette section, nous voulons vérifier si HVS est capable d'ordonner les variables selon un ordre d'importance significatif. Nous allons donc verifier si HVS est capable d'une part de détecter les variables inutiles, et d'autre part d'estimer le rapport entre les importances des variables utiles. Pour se faire, nous avons choisi deux problèmes relativement simples dont l'importance théorique de chaque variable est connue. Ils sont très utilisés en apprentissage symbolique ; ils révèlent la faiblesse de certains algorithmes d'apprentissage.

2.3.2.1 Détection de Pertinence

Afin de tester les capacités de HVS à identifier les variables significatives à partir du signal d'entrée, nous avons choisi le problème "Even-Parity 3". Dans ce problème (voir figure 2.4), les objets sont représentés par des vecteurs à valeurs binaires P_i, i variant de 1 à n ($n \geq 3$, dans notre cas n=10). Un objet est considéré comme une instance positive si et seulement si ses trois premiers bits forment un ensemble dans lequel le nombre de "1" est pair. Dans le cas contraire, l'objet sera considéré comme une instance négative. Il s'agit alors d'un problème de discrimination entre deux classes. Il est donc clair que, dans ce problème, seules les trois premières variables sont importantes et ont une même importance.

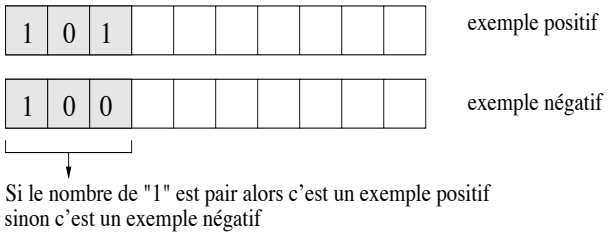


FIG. 2.4 – *Le problème "Even parity 3" : seules les trois premiers bits sont importants.*

Nous avons généré une base de 2956 échantillons : deux tiers de la base pour l'apprentissage et l'autre tier pour la validation. Nous avons utilisé une architecture $< 10|2|2 >$, c.à.d une couche d'entrée I composée de 10 neurones, une couche cachée H de 2 neurones et une couche de sortie de 2 neurones.

A la fin de l'apprentissage, nous avons utilisé HVS pour estimer l'importance de chaque variable. Comme le montre la figure 2.5, seules les trois premières variables sont considérées comme importantes. De plus, il est à remarquer que HVS leur attribue des importances approximativement égales. Nous avons calculé aussi les saliences des variables en utilisant la méthode OCD. Comme on le voit sur figure 2.6, OCD donne des résultats similaires à ceux de HVS.

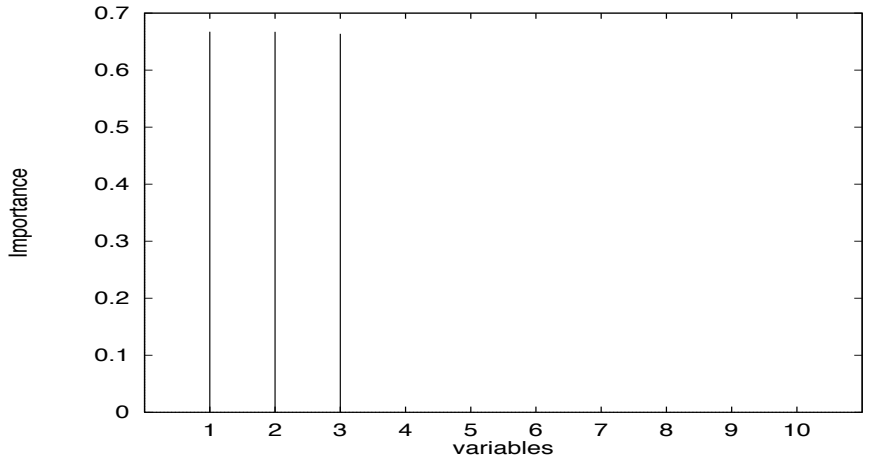


FIG. 2.5 – *Le problème "Even parity 3" : importance des variables selon HVS.*

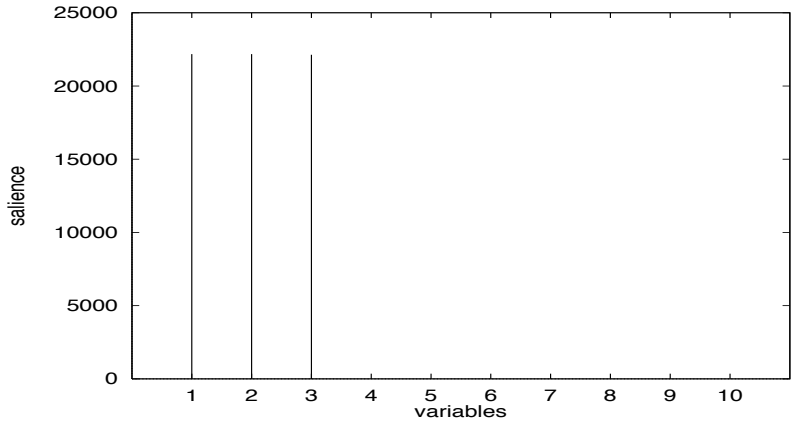


FIG. 2.6 – *Le problème "Even parity 3" : "salience" des variables selon OCD.*

2.3.2.2 Quantification de Pertinence

Afin de vérifier si HVS est capable d'estimer le rapport entre les importances significatives des variables, nous avons choisi le problème du "Multiplexeur 11".

Dans ce problème (voir figure 2.7), chaque instance est un vecteur booléen divisé en trois bits adresse et 2^3 bits de données. Les bits adresse permettent d'adresser les bits restants. Si le bit adressé est égal à 1, l'instance est positive, sinon elle est négative. Par conséquent, tous les bits sont importants. Les trois premiers bits ont une même importance I_1 et les huit bits restants ont aussi une même importance I_2. Le rapport théorique entre I_2 et I_1 est : $\frac{I_2}{I_1} = \frac{1}{8}$.

FIG. 2.7 – *Le problème du "Multiplexer11" : les 3 premiers bits définissent le bit adressé. Dans cet exemple il s'agit du* $6^{ème}$ *bit* $(1\times2^2+1\times2^1+0\times2^0)$. *L'instance est positive dans le cas où le* $6^{ème}$ *bit vaut 1 et elle est négative dans le cas où le* $6^{ème}$ *bit vaut 0*

Nous avons utilisé une base de 2048 exemples : deux tiers de la base pour l'apprentissage et l'autre tier pour la validation. A la fin de l'apprentissage, nous avons utilisé HVS pour estimer l'importance de chaque variable. Le résultat est illustré dans la figure 2.8.

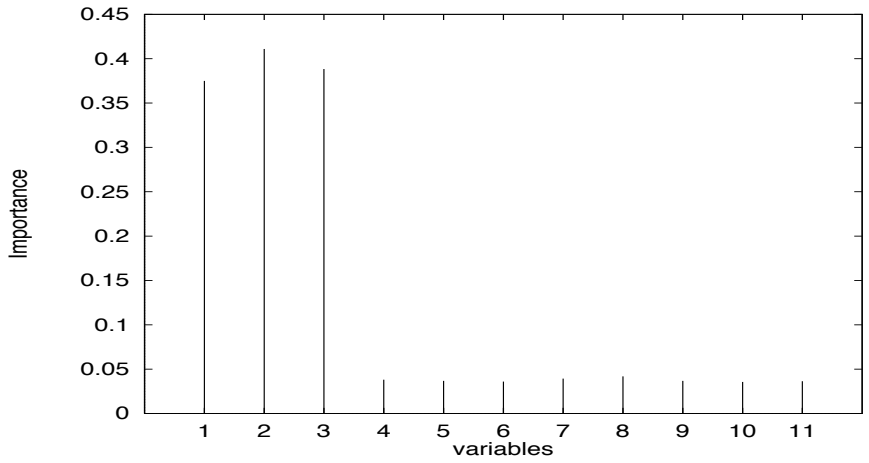


FIG. 2.8 – *Le problème du "Multiplexer11" : importance des variables selon HVS.*

La figure 2.8 montre que la mesure HVS estime que :

- toutes les variables sont importantes,
- les trois premières variables ont des importances approximativement égales,
- les huit variables restantes ont des importances approximativement égales.

Mais le plus important dans ce calcul d'importance est que le rapport entre l'importance d'une des trois premières variables et l'une des huit variables restantes n'est pas loin du rapport théorique.

Nous avons aussi calculé les saliences des variables en utilisant OCD. De même que HVS, OCD a séparé les variables en deux classes selon leur importance. Cependant, le rapport entre la première classe et la seconde est tellement élevé qu'on a tendance a supposer que les variables 4 à 11 sont non pertinentes (figure 2.9).

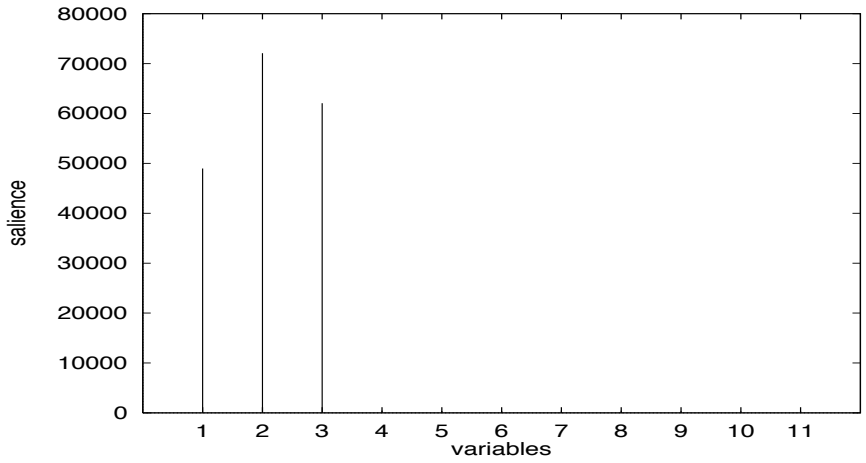


FIG. 2.9 – *Le problème du "Multiplexer11" : "salience" des variables selon OCD.*

2.3.2.3 Conclusion partielle

Les résultats des expériences précédentes montrent que :

- HVS est capable de détecter les variables pertinentes.
- HVS est capable de quantifier les importances des variables pertinentes.

Nous allons maintenant utiliser HVS pour la sélection de variables.

2.3.3 HVS pour la Sélection de Variables

2.3.3.1 Principe de la méthode

La méthode de sélection de variables que nous proposons est de type "backward selection" basée sur la mesure HVS. Nous supprimons les variables une à une jusqu'à la dernière variable. A chaque fois qu'on supprime une variable, un réapprentissage est nécessaire.

L'algorithme de sélection que nous proposons est le suivant :

1) Atteindre un minimum local

2) Calculer la pertinence de chaque variable selon 2.20

3) Trier les variables selon un ordre croissant de pertinence

4) Supprimer la variable de plus faible importance

5) reprendre à 1) jusqu'à la dernière variable.

Sélection d'un sous-ensemble de variables L'algorithme précédent génère un ensemble de k réseaux (k étant le nombre de variables) ayant de moins en moins de variables.
Notons $PMC(p)$ $p = 1, \cdots, k$ ces réseaux et $E(p)$ l'erreur estimée sur une base de validation.

A ce niveau deux approches sont possibles pour la sélection d'un sous-ensemble de variables :

- Choisir le réseau $PMC(p^*)$ qui obtient la plus petite erreur

$$PMC(p^*) = \arg \min_{PMC(P)} E(p)$$

  et choisir le sous-ensemble des p^* variables associées à $PMC(p^*)$.

- la deuxième consiste à utiliser un test statistique (test de Fisher) et chercher parmi tous les réseaux $PMC(p)$ ceux qui sont statistiquement proches de $PMC(p^*)$. Ce principe permet d'obtenir un ensemble de réseaux tels que $E(p^i) \approx E(p^*)$.

  Le sous-ensemble de variables sélectionnées est choisi comme le plus petit sous-ensemble de p^0 variables statistiquement proches de $PMC(p^*)$

$$p^0 = \min_i \{p^i\}$$

Pour des raisons de simplicité, nous avons choisi de développer la première approche.

2.3.3.2 HVS pour un problème de discrimination

Pour valider notre méthode de sélection de variables sur un problème de discrimination, nous avons choisi le problème des vagues de Breiman, proposé par Breiman et al. [16]. C'est un problème à trois classes. Les exemples sont des vecteurs à valeurs réelles de dimension 21. La représentation de ce problème dans l'espace des deux premières composantes principales est illustrée par la figure 2.10. Les polygones montrent les enveloppes convexes de chaque classe. On peut remarquer le grand chevauchement entre les trois classes.

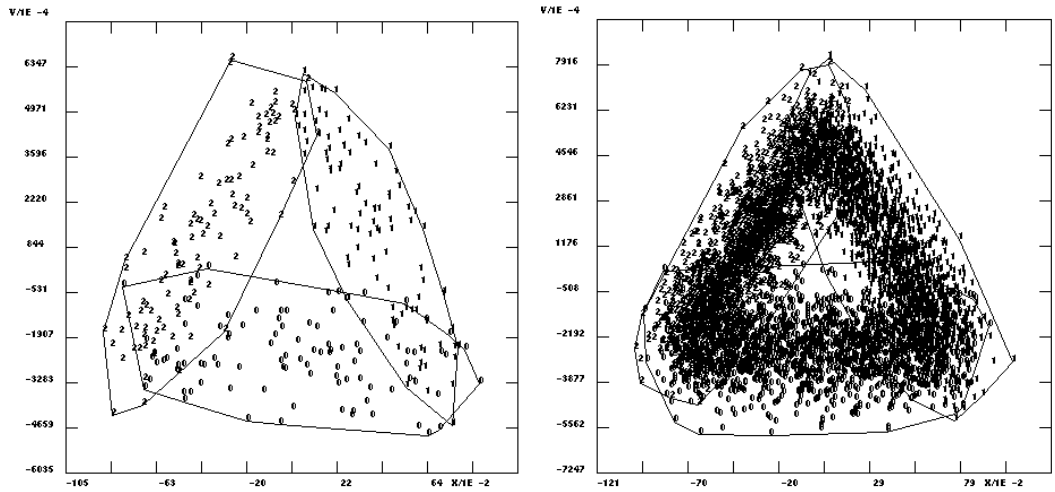


FIG. 2.10 – *Représentation des vagues de Breiman dans l'espace des deux premières composantes principales pour l'ensemble d'apprentissage (à gauche) et pour l'ensemble de test (à droite). Les polygones montrent les enveloppes convexes de chaque classe.*

Pour ce problème, on utilise un ensemble d'apprentissage D^l de 300 exemples et un ensemble D^v de 700 exemples pour la validation. Pour le test on utilise un ensemble D^t de 4300 exemples. Nous avons utilisé une architecture intiale définie par $< 21|10|3 >$.

Comme le montre la Figure 2.11, la procédure HVS a sélectionné le sous-ensemble de variables $S^v(\text{HVS}) = \{4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19\}$ et a éliminé les variables $\{1, 2, 3, 20, 21\}$.

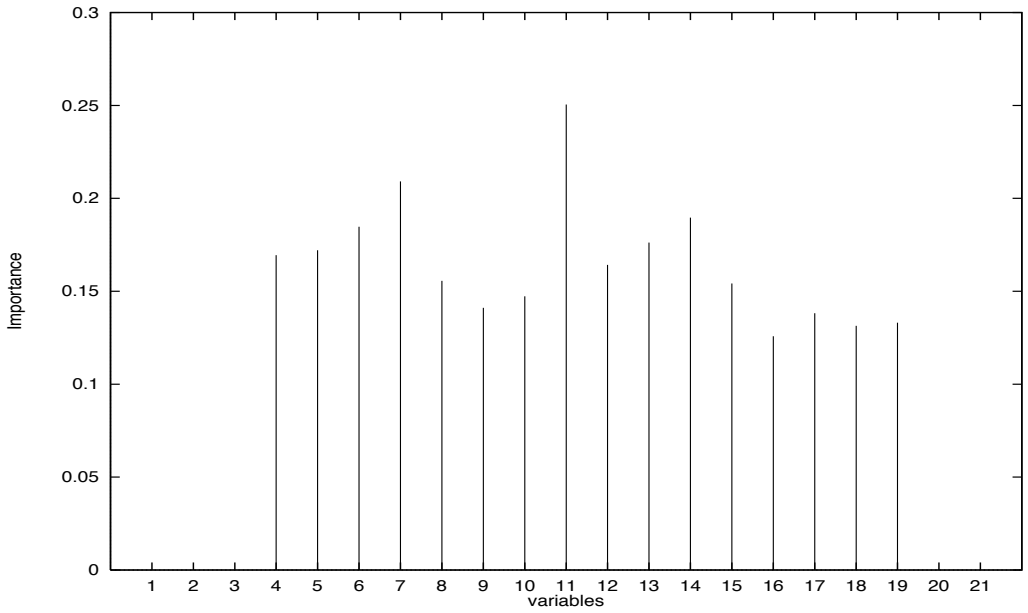


FIG. 2.11 – *HVS a selectionné les variables 4 à 19 comme étant les plus importantes.*

La table 2.1 donne les performances avant et après sélection des variables. Il faut noter que les performances sur D^t après avoir sélectionné les variables les plus importantes, selon la mesure HVS, sont passées de 85.23% à 85.79%.

Il faut noter également que ces performances ne sont pas loin des performances du classifieur Bayesien (limite théorique) qui ont été estimées par Breiman à 86% de bonne décision.

	performances sur		
	D^l	D^v	D^t
Avant Sélection	87.66 [83.46 , 90.92]	85.57 [82.77 , 87.98]	85.23 [84.14 , 86.26]
Après Sélection	88 [83.83 , 91.20]	86.14 [83.39 , 88.51]	85.79 [84.72 , 86.80]

TAB. 2.1 – *Performances et intervalles de confiance à* 95% *avant et après sélection de variables en utilisant HVS.*

Conclusion partielle : Pour ce problème, nous avons obtenu de très bonnes performances en sélectionnant 16 variables.

Nous allons maintenant tester notre méthode de sélection de variables sur une version bruitée de ce problème. Cette version comporte un nombre important de variables inutiles.

2.3.3.3 Sélection de variables en présence de bruit

Pour tester l'efficacité de HVS en présence de bruit, nous avons choisi la version bruitée du problème des vagues de Breiman, qui a été utilisée par De Bollivier et al. [15]. Les exemples sont des vecteurs à valeurs réelles de dimension 40 incluant 19 composantes bruitées.

De même que pour le problème précédent, on utilise un ensemble d'apprentissage D^l de 300 exemples et un ensemble D^v de 700 exemples pour la validation. Pour le test on utilise un ensemble D^t de 4300 exemples. Nous avons utilisé une architecture intiale $< 40|10|3 >$.

Comme le montre la Figure 2.12, la procédure HVS a sélectionné le sous-ensemble de variables $S^v(\text{HVS}) = \{4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19\}$ et a éliminé les variables 1 à 3 et 20 à 40. Il est à remarquer que HVS a sélectionné les variables les plus importantes dans la première partie du signal et a rejeté toutes les variables dans la partie bruitée.

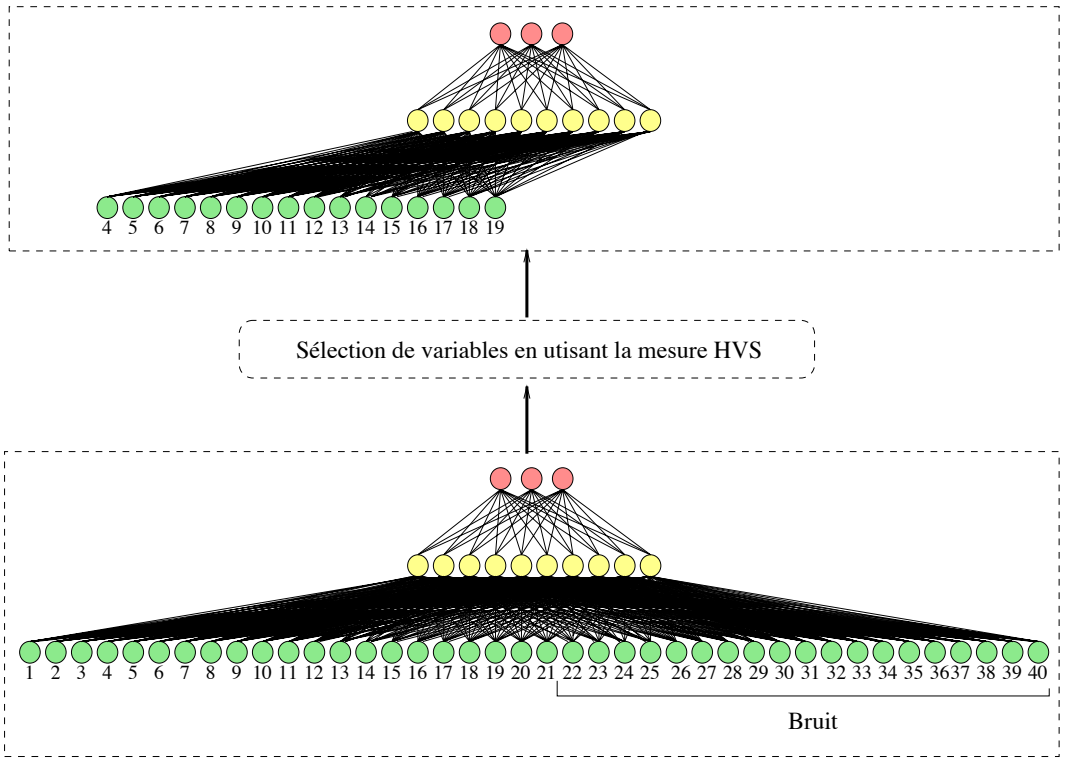


FIG. 2.12 – *HVS a sélectionné les variables 4 à 19 comme étant les plus importantes.*

Comme le montre la table 2.2, les performances sur D^t après avoir sélectionné les variables les plus importantes selon la mesure HVS sont passées de 82.83 à 85.37.

	performances sur		
	D^l	D^v	D^t
Avant Sélection	92.00 [88.37 , 94.57]	83.85 [80.95 , 86.40]	82.83 [81.68 , 83.93]
Après Sélection	88.66 [84.58 , 91.78]	85.57 [82.77 , 87.98]	85.37 [84.28 , 86.40]

TAB. 2.2 – *Performances avant et après sélection de variables en utilisant HVS.*

La table 2.3 présente les résultats obtenus sur ce même problème en utilisant la méthode OCD. Contrairement à la méthode HVS, OCD a sélectionné dans la partie bruitée.

Méthode	# v.s.	variables sélectionnées	performances sur D^t
OCD	21	00011111111111111000 0010000010010101100	83.14 [82.00 , 84.23]
HVS	16	00011111111111111100 0000000000000000000	85.37 [84.28 , 86.40]

TAB. 2.3 – *Résultats obtenus en utilisant la méthode HVS et la méthode OCD. Dans cette table, les variables sélectionnées sont représentées par 1 et celles supprimées sont représentées par 0.*

Conclusion partielle Pour ce problème bruité, HVS a éliminé toutes les variables inutiles et a sélectionné 16 variables dans la partie non bruitée. Les performances sur D^t après avoir sélectionné les variables les plus importantes selon la mesure HVS sont passées de 82.83 à 85.37. Nous verrons dans le chapitre suivant que ces performances seront encore améliorées en appliquant notre méthode d'ajustement d'architecture.

Nous allons maintenant tester notre méthode de sélection de variables sur un problème de régression dont la base d'exemples est de petite taille. Il s'agit dans ce cas d'un problème de détermination de la dimension intrinsèque d'une série chronologique.

2.3.3.4 HVS pour un Problème de Régression

2.3.3.4.1 Le problème "sunspots" Comme problème de régression, nous avons choisi la série des taches solaires : "sunspot". Il s'agit d'une série très courte et célèbre dont l'équation sous jacente est inconnue. Il s'agit du nombre moyen annuel de taches noires observées sur le soleil entre 1700 et 1979 (280 points). La Figure 2.13 montre l'évolution annuelle de cette série.

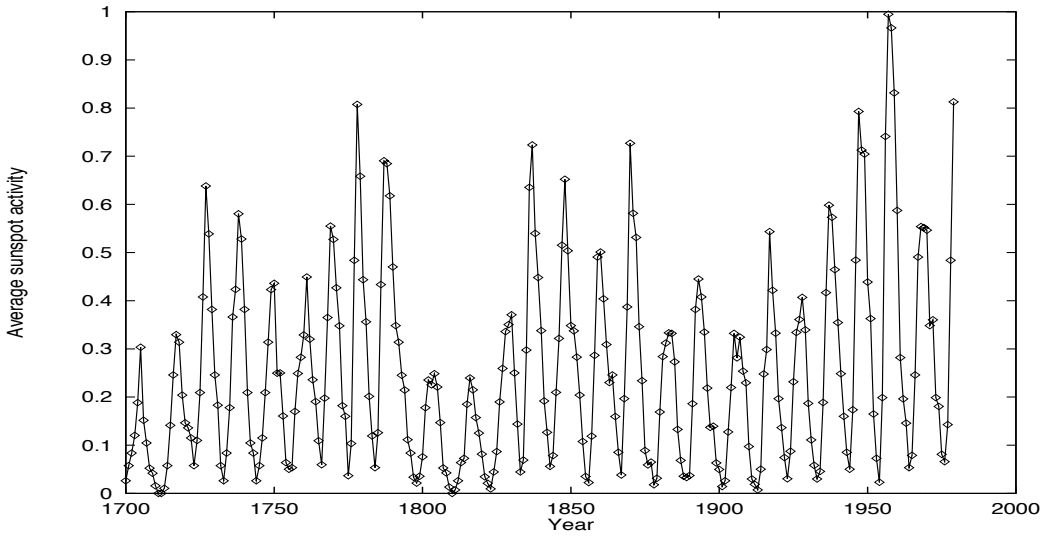


FIG. 2.13 – *série sunspot normalisée de 1700 à 1979.*

Pour ce problème, on essaie de prédire une valeur en utilisant les 12 valeurs précédentes (figure 2.14), ce qui conduit à utiliser un réseau avec une couche d'entrée de 12 neurones et une couche de sortie de 1 neurone.

On utilise les données de 1712 à 1920 pour l'apprentissage (un ensemble D^l de 209 exemples) et les données de 1921 à 1955 pour la validation (un ensemble D^v de 35 exemples). Pour le test, on utilise les données de 1956 à 1979 (un ensemble D^t de 24 exemples). Nous avons utilisé une architecture initiale $< 12|10|1 >$ avec une tangente hyperbolique et identité comme fonctions de transition de la couche cachée et de la couche de sortie respectivement. Nous avons commencé avec 12 neurones en couche d'entrée car Weigend et al [96] ont fait remarquer que l'augmentation du nombre de neurones de la couche d'entrée de 12 à 25 ne

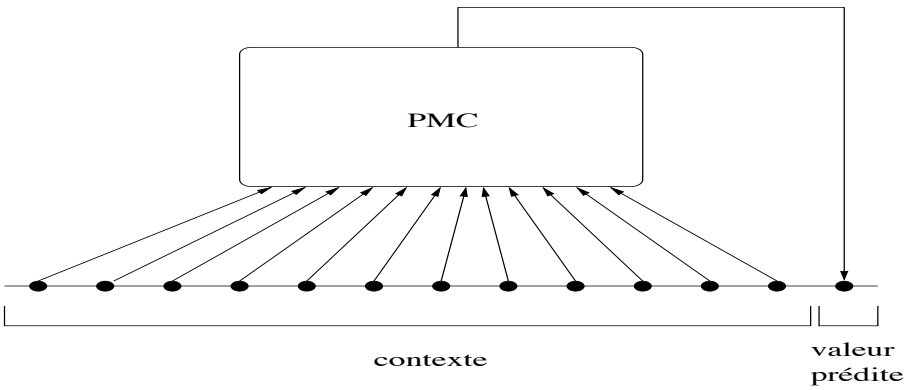


FIG. 2.14 – *Un PMC pour la régression.*

conduit pas à des améliorations significatives.

2.3.3.4.2 Mesure de qualité Les performances du modèle sont calculées en utilisant le critère ARV (Average Relative Variance), qui est le rapport entre l'erreur quadratique moyenne du modèle et la variance des données. Il est à Noter que la définition de l'ARV établit un rapport entre l'erreur du modèle et la variance des données calculées sur le même ensemble D (pris de l'ensemble entier S).

$$arv(D) = \frac{\sum_{i \in D} (y^{(i)} - f(x^{(i)}))^2}{\sum_{i \in D} (y^{(i)} - \mu_D)^2} \tag{2.22}$$

où $y^{(i)}$ est la valeur de la série à l'instant i, $f(x^{(i)})$ est la sortie du réseau à l'instant i, et μ_D la moyenne de la valeur désirée dans D.

Cependant, la définition de l'ARV la plus utilisée utilise la variance totale des données (de la série)

$$arv(D) = \frac{\frac{1}{|D|} \sum_{i \in D} (y^{(i)} - f(x^{(i)}))^2}{\frac{1}{|S|} \sum_{i \in S} (y^{(i)} - \mu_S)^2} \tag{2.23}$$

Afin de comparer notre arv avec celles obtenues en utilisant d'autres méthodes, on maintient la définition la plus utilisée malgré son biais.

2.3.3.4.3 Dimension intrinsèque de la série chronologique Le problème de la sélection de variable peut se formuler en régression comme un problème d'estimation de la dimension intrinsèque d'une série chronologique: détermination des délais nécessaires pour mieux prédire. Pour la série "sunspot", nous avons fait 30 expériences. Pour toutes ces expériences, nous avons utilisé une architecture initiale $< 12|10|1 >$. Les delais: $x_{t-12}, x_{t-11}, x_{t-8}, x_{t-3}, x_{t-2}$, et x_{t-1} ont été selectionnés dans la majorité de nos experiences.

Nous allons décrire un peu plus en détail notre méthode à travers une de ces experiences.

phase d'apprentissage Nous avons utilisé le "Early Stopping" comme critère d'arrêt de la procédure d'apprentissage. La figure 2.15 montre l'évolution de l'arv sur D^v durant la phase d'apprentissage (la meilleure arv sur D^v obtenue pendant cette phase est 0.0906).

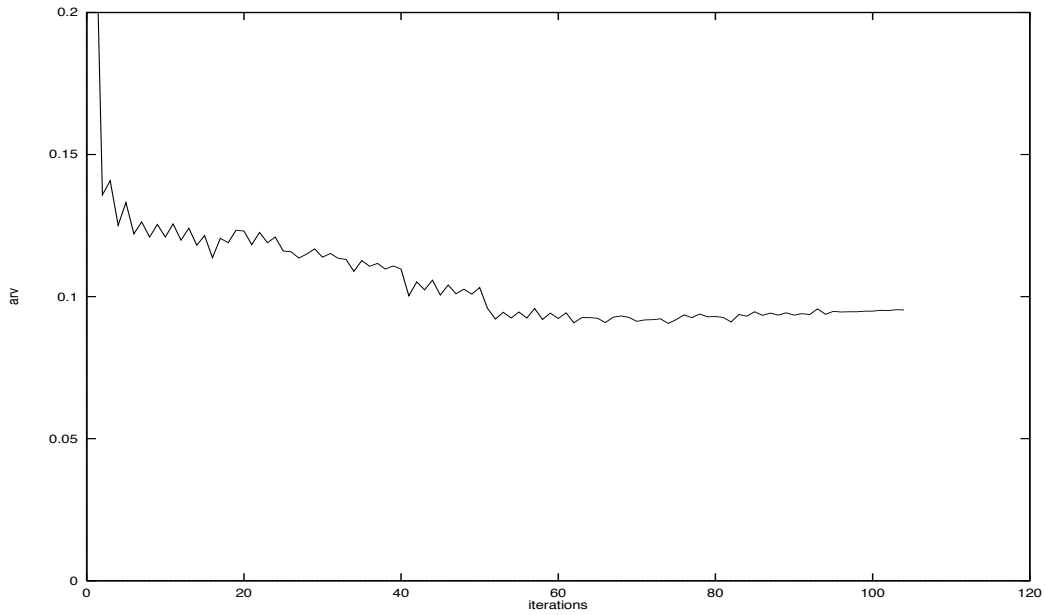


FIG. 2.15 – *Evolution de l'Arv sur D^v durant la phase d'apprentissage.*

phase de sélection de variables : détermination des délais A la fin de la phase d'apprentissage, nous avons lancé la procédure de sélection de variables. La figure 2.16 donne l'évolution de l'arv sur D^v durant cette phase de sélection de variables (pour une expérience). Les intervalles délimités par les lignes verticales correspondent aux différentes phases d'élagage. Chaque phase d'élagage consiste à

- utilser les poids correspondant à la meilleure arv obtenue durant la phase d'élagage précédente,
- éliminer, parmi les variables restantes, la variable la moins pertinente selon la mesure HVS,
- réapprendre en utilisant le "Early Stopping" comme critère d'arrêt.

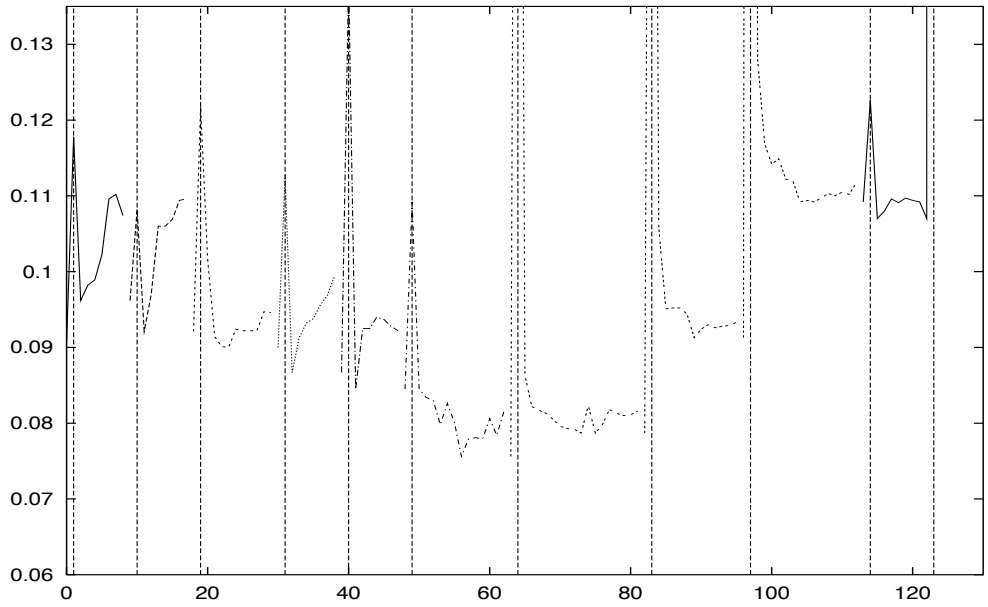


FIG. 2.16 – *Evolution de l'Arv sur D^v durant la sélection de variables pour un réseau. Les intervalles délimités par les lignes verticales correspondent aux différentes phases d'élagage.*

Nous allons expliquer en détail la figure 2.16.

- le premier point de la courbe (itération 0) correspond à la meilleure arv

sur D^v obtenue durant la phase d'apprentissage (0.0906).

- chaque ligne verticale correspond au moment où nous avons supprimé une variable.
- chaque fois qu'on supprime une variable, un réapprentissage est nécessaire. L'évolution de l'arv sur D^v pendant cette phase de réapprentissage est illustrée par une courbe qui commence à la ligne verticale où la variable est supprimée et se termine juste avant la ligne verticale suivante. Pour arrêter cette phase de réapprentissage nous avons utilisé le "Early Stopping".
- le point qui est juste avant chaque ligne verticale représente la meilleure arv sur D^v obtenue pendant la phase d'élage précédente.

Pour sélectionner le meilleur système (le meilleur sous-ensemble de variables), nous avons utilisé un "Early Stopping" global. En d'autres termes, nous avons choisi le système ayant obtenu la meilleure arv sur D^v. La figure 2.17 donne la meilleure arv de chaque système, chaque ligne verticale représente un sytème. Le système i correspond au sytème obtenu après avoir supprimé les i variables les moins importantes selon HVS. Comme le montre cette figure, le meilleur système est obtenu après avoir supprimé 6 variables. La figure 2.18 montre l'ensemble des variables sélectionnées.

Il est à noter que dans ce cas on peut aussi utiliser des tests statistiques et choisir le modèle possédant le plus petit nombre de variables et qui reste statistiquement proche des performances du meilleur modèle.

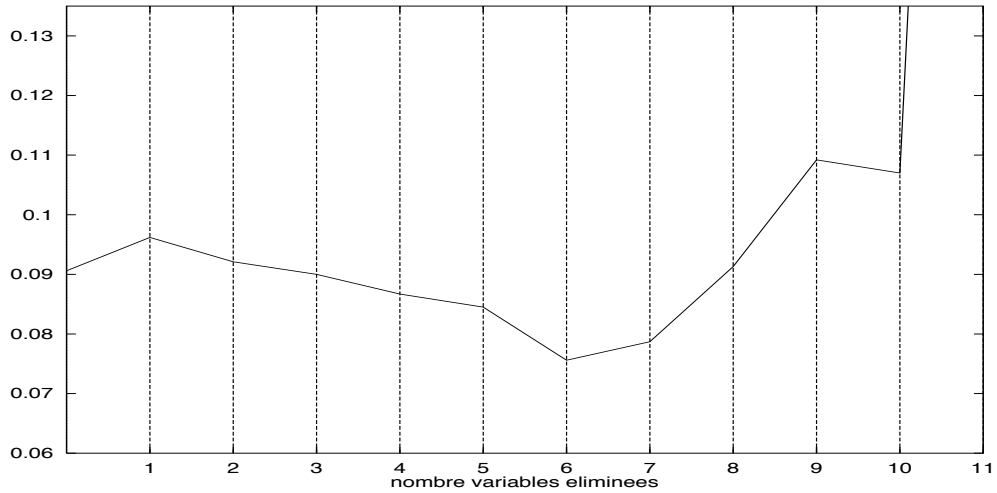


FIG. 2.17 – *Evolution globale de l'Arv sur D^v durant la sélection de variables pour un réseau. L'arv après élimination de i variables correspond à l'Arv optimale de la phase d'élagage i dans la figure 2.16.*

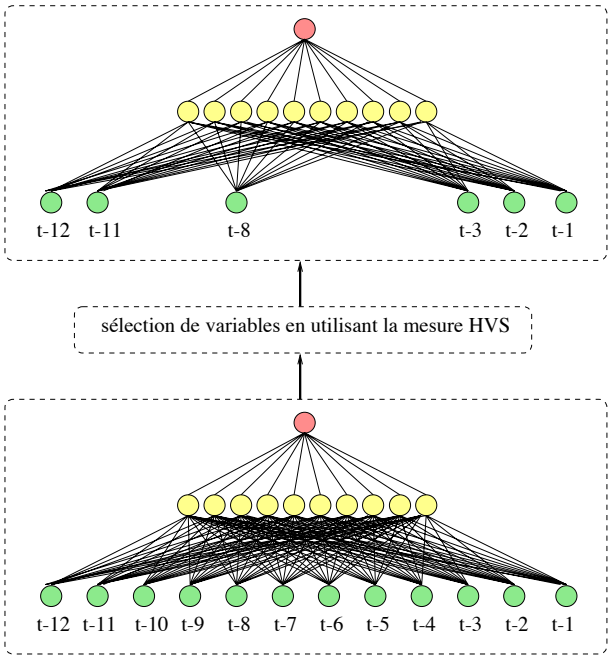


FIG. 2.18 – *Nous avons fait 30 expériences. Les delais $x_{t-12}, x_{t-11}, x_{t-8}, x_{t-3}, x_{t-2}$, et x_{t-1} ont été selectionnés dans la majorité de nos expériences.*

Modèle/arv	D^l(1712-1920)	D^v(1921-1955)	D^t(1956-1979)
Yacoub & Bennani	0.089	0.076	0.331
Goutte[40]	0.082	0.082	0.357
Svar & al.[87]	0.090	0.082	0.35
Weigend & al.[96]	0.082	0.086	0.35
Linéaire	0.131	0.128	0.36

TAB. 2.4 – *Comparaison des résultats obtenus en utilisant différentes méthodes. Notre résultat est la moyenne de 30 expériences.*

La table 2.4 présente nos résultats, moyenne des 30 expériences, après sélection de variables, ainsi que quelques résultats obtenus en utilisant d'autres méthodes. Il est à remarquer que notre méthode permet d'obtenir des prédictions plus précises que celles obtenues par d'autres approches.

Conclusion partielle Pour ce problème de régression dont la base est de petite taille, nous avons obtenu des prédictions plus précises que celles obtenues par d'autres approches.

2.4 Conclusion du chapitre

Dans ce chapitre, nous avons présenté une mesure heuristique, nommé HVS, permettant d'estimer l'importance de chaque variable. Dans un premier temps, nous avons testé ses capacités sur des problèmes dont l'importance théorique de chaque variable est connue. Ensuite, nous l'avons utilisé pour la sélection de variables. Nous avons testé son efficacité sur un problème de discrimination et sur un problème de régression. HVS est une mesure efficace et ne demande que peu de calculs simples, faciles à implémenter.

Chapitre 3

Ajustement de la Complexité des Architectures Connexionnistes

Le problème auquel on s'intéresse dans ce chapite est l'ajustement de la taille du réseau afin que sa complexité soit adaptée à la difficulté du problème à résoudre. Nous proposons une extension de la méthode d'élagage HVS permettant d'éliminer les neurones internes non indispensables à l'estimation d'un processus.[1]

1. Résultats publiés dans ICANN'98 [99] et soumis à la revue IJNS [102].

3.1 Introduction

Comme nous l'avons déjà mentionné, les PMC sont des approximateurs universels. Cependant, pour un problème donné, on ne connait pas le nombre nécessaire de couches et de neurones par couche. Ceci est très important puisqu'un nombre de neurones trop petit induira une modélisation insuffisante, et un nombre de neurones trop grand entraîre une surparamétrisation du modèle, qui nuira aux performances en généralisation. Une étude sur l'évolution des performances en fonction du nombre de cellules cachées [21] montre d'une façon expérimentale le comportement des PMC sur un problème de discrimination. Cette étude montre que le phénomène de sur-apprentissage est plus complexe qu'on le pense. L'idée d'ajuster, pendant l'apprentissage, la taille du réseau afin que sa compléxité soit adaptée au problème à résoudre a conduit au développement de deux grandes familles de méthodes : les méthodes constructives et les méthodes destructives. Nous les avons déjà introduites dans le chapitre 1, lorsque nous avons présenté les méthodes heuristiques de régularisation structurelle.

Dans ce chapitre, nous proposons d'étendre notre méthode de sélection de variables aux neurones de la couche cachée. On diminuera ainsi la complexité du modèle en éliminant les neurones internes inutiles. Notre méthode appartient à la famille des méthodes destructives.

Nous donnons d'abord un bref état de l'art sur ce thème afin de situer notre contribution, puis nous présentons le principe de la méthode d'ajustement d'architectures que nous proposons et ensuite nous la validons sur les problèmes évoqués dans le chapitre précédent.

3.2 Techniques d'optimisation d'architectures connexionnistes

L'idée d'ajuster, pendant l'apprentissage, la taille du réseau afin que sa compléxité soit adaptée au problème à résoudre a conduit au développement de deux grandes familles de méthodes : les méthodes constructives et les méthodes destructives.

• Dans les méthodes constructives, l'algorithme d'apprentissage augmente progressivement le nombre de paramètres libres en ajoutant des neurones cachés au réseau. L'algorithme arrête d'ajouter des neurones cachés lorsqu'un critère de validation indique que les performances du réseau sont suffisamment bonnes. Dans cette famille on trouve, par exemple, "Upstrart algorithm"[32], "Tiling algorithm"[67] et l'algorithme "cascade correlation"[28].

Par exemple, l'algorithme cascade correlation combine deux idées : la première est l'architecture cascade, dans laquelle les unités cachées sont ajoutées une seule à la fois et ne changent pas après avoir été ajoutées. La figure 3.1 montre l'architecture du réseau après l'ajout de deux unités cachées. La deuxième est l'algorithme d'apprentissage, qui crée et connecte des nouvelles unités cachées. Pour chaque nouvelle unité cachée ajoutée, l'algorithme essaie de maximiser la grandeur de la corrélation entre la sortie de cette nouvelle unité et l'erreur résiduelle des sorties du réseau avant l'ajout de cette unité.

Le réseau est initialisé sans unité cachée et chaque entrée est connectée à chaque neurone de sortie par des connexions ajustables (carrés noirs dans la figure 3.1). Les poids des neurones de sortie sont déterminés par minimisation de l'erreur quadratique entre les sorties désirées et celles obtenues. Tant que les performances du réseau sont jugées insuffisantes, une unité cachée est ajoutée suivie d'un réapprentissage du réseau. L'ajout d'une unité cachée se fait comme suit :

- premièrement, elle est connectée (en amont, reçoit des connexions) aux entrées et aux sorties des unités cachées déjà ajoutées. On maximise alors la corrélation entre la sortie de cette nouvelle unité et l'erreur résiduelle des sorties du réseau en ajustant les poids arrivant à cette unité ajoutée. Cette corrélation est définie par

$$R(z^p, \epsilon^p) = \sum_k \left| \sum_p (z^p - \bar{z})(\epsilon_k^p - \bar{\epsilon}_k) \right| \tag{3.1}$$

  où ϵ_k^p est l'erreur sur l'unité de sortie k pour l'exemple p et z^p la sortie de l'unité ajoutée pour l'exemple p. $\bar{z}$ et $\bar{\epsilon}_k$ sont les moyennes sur tous les exemples de la base d'apprentissage.

- ensuite on la connecte (en aval) aux unités de sortie et on fige les poids

des connexions arrivant à elle (ceux qu'on vient de déterminer). C'est à ce moment que la phase de réapprentissage commence (en ne réapprenant que les poids arrivant aux unités de sortie).

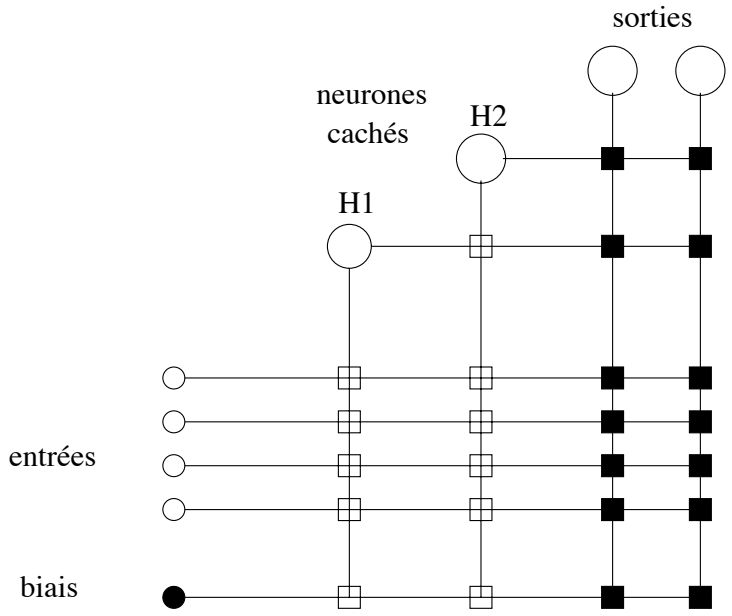


FIG. 3.1 – *Architecture de l'algorithme "cascade correlation". Les carrés blancs correspondent aux poids qui sont appris puis figés, les carrés noirs aux poids qui sont réappris après chaque ajout d'une unité cachée. L'unité H1 est ajoutée avant H2.*

La plupart des méthodes constructives apparaissent comme très "gourmandes" en nombre de neurones. Elles favorisent un apprentissage par coeur et par conséquent généralisent mal [45].

• Dans les méthodes destructives, on propose au contraire de partir d'un réseau surdimentionné et de le simplifier au cours de l'apprentissage. Cette famille contient les méthodes d'élagage des poids [60, 43] ou des unités cachées [19, 99] et les méthodes qui introduisent des termes de pénalisation dans la fonction de coût [95, 19, 74].

Par exemple, la méthode proposée par Chauvin dans [19] permet de réduire le

nombre de neurones cachés. La fonction de coût utilisée est :

$$R = \alpha R_1 + \beta \sum_p \sum_h e((a_h^p)^2) \tag{3.2}$$

où a_h^p est l'activation de la cellule cachée h lorsqu'on présente l'exemple p, R_1 le critère standard (erreur quadratique moyenne), α et β sont des termes de pondération et e une fonction d'énergie, comme par exemple $e(x^2) = \frac{x^2}{1+x^2}$. On peut aussi ajouter un terme de pénalisation des poids.

Le deuxième terme de l'équation 3.2 mesure la variation de l'activité de chaque unité sur l'ensemble d'apprentissage. Une cellule dont la valeur varie peu sera considérée comme inutile.

3.3 HVS pour l'Ajustement d'Architecture

Dans cette section, nous proposons d'utiliser HVS pour l'optimisation d'architecture en étendant son application aux unités de la couche cachée. Nous testerons son efficacité sur les problèmes déjà décrits dans le chapitre précédent.

3.3.1 Principe de la méthode

Le principe de la méthode que nous proposons pour l'ajustement d'architectures est très simple. Il consiste à considérer chaque couche cachée comme la couche d'entrée d'un sous-réseau composé de cette couche, de la couche de sortie, et de toutes les couches cachées qui sont entre ces deux couches. Le problème d'optimisation d'architecture devient alors un problème de sélection de variables, dans le sous-réseau en question. Par exemple dans la figure 3.2, éliminer les neurones internes inutiles du réseau R revient à sélectionner les variables dans le sous-réseau R1.

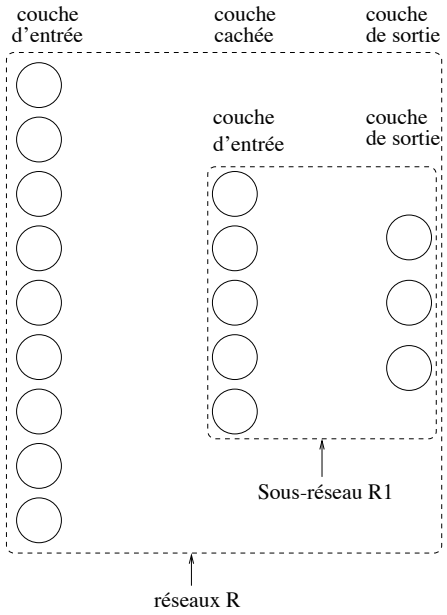


FIG. 3.2 – *La couche cachée du réseau R est considérée comme la couche d'entrée du sous-réseau R1.*

3.3.2 Validation de la méthode

Pour illustrer l'effet de l'application de HVS sur les unités de la couche cachée, nous avons choisi de poursuivre les expériences sur le problème bruité des vagues de Breiman et sur le problème sunspot décrit dans le chapitre précédent.

3.3.2.1 Ajustement d'architecture en classement

Pour valider notre méthode sur un problème de classement, nous l'appliquons au problème des vagues de Breiman avec 40 variables (version bruitée). Pour ce problème, comme nous l'avons décrit au chapitre précédent, nous avons utilisé une architecture initiale $< 40|10|3 >$. Notre méthode de sélection de variables nous a permi de supprimer 24 variables, l'architecture initiale de la procédure d'ajustement d'architecture est donc $< 16|10|3 >$.

Après avoir sélectionné les meilleures variables selon HVS, nous avons appliqué notre méthode d'optimisation d'architecture. La figure 3.3 montre l'évolution des

performances sur D^v durant l'élagage des neurones cachés (voir figure 3.4 pour l'agrandissement de la partie intéressante de cette figure).

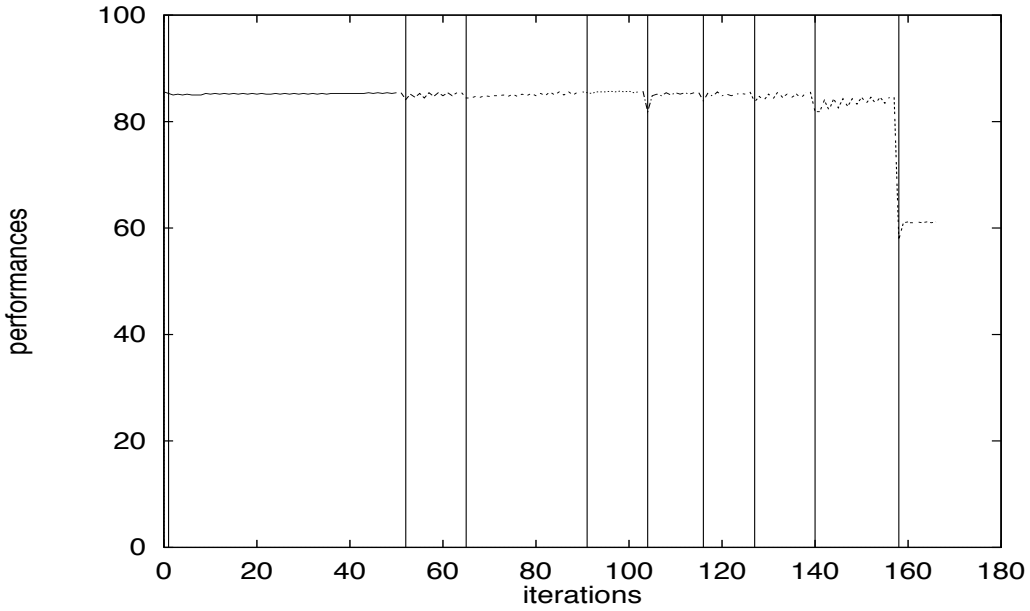


FIG. 3.3 – *Evolution des performances sur D^v durant l'élagage des neurones cachés.*

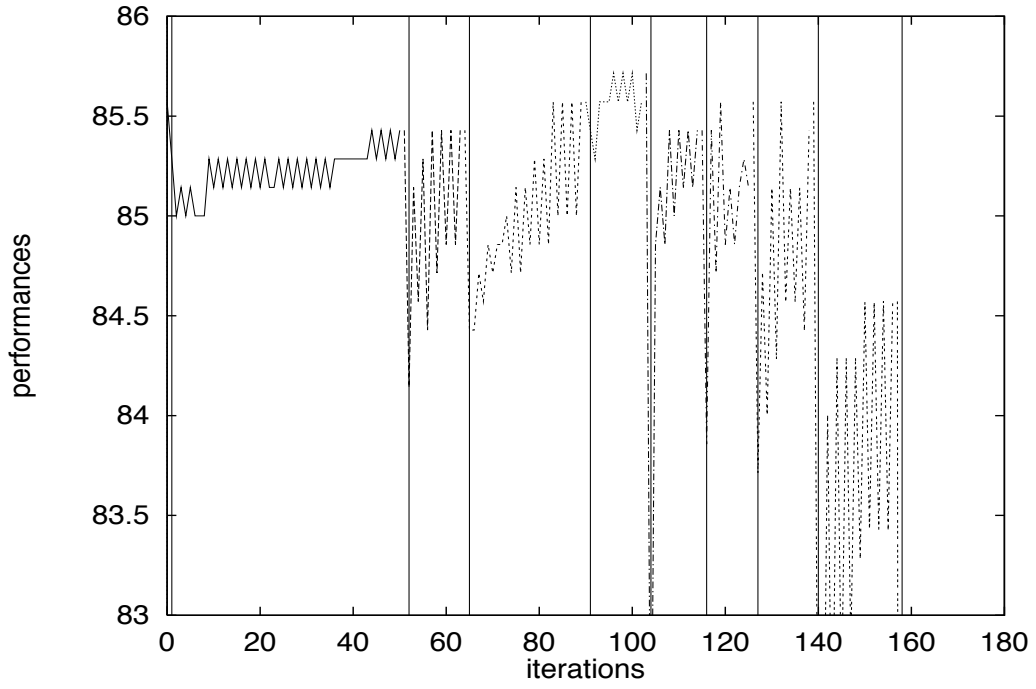


FIG. 3.4 – *Evolution des performances sur D^v durant l'élagage des neurones cachés avec un agrandissement de l'échelle.*

Au départ (à l'itération 0), les performances sur l'ensemble D^v sont égales à 85.57, elles correspondent aux performances obtenues après sélection de variables. A l'itération 1 (la première ligne verticale) nous avons supprimé le neurone caché le moins important selon la mesure HVS. Les performances sur D^v juste après élimination de ce neurone sont égales à 85.28. On peut noter la baisse attendue de performance dûe à l'élagage. L'intervalle délimité par les deux premières lignes verticales correspond à la phase d'apprentissage appliquée après avoir supprimé le premier neurone caché. Les meilleures performances obtenues pendant cette phase sont égales à 85.42. Juste avant la deuxième ligne verticale, on peut voir une courbe qui se termine (correspondant à la fin de la phase d'apprentissage après élagage du premier neurone caché) et une deuxième courbe qui commence (correspondant au début de la phase d'élagage suivante). Chaque phase d'élagage commence au point correspondant aux meilleures performances obtenues durant la phase d'élagage précédente, ensuite on supprime le neurone caché le moins important (ligne verticale) puis on réapprend en utilisant le "Early Stopping" comme critère d'arrêt.

Les meilleures performances sur D^v obtenues en fonction du nombre de neurones cachés supprimés sont montrées sur la figure 3.5. Comme le montre cette figure, la meilleure performance sur D^v est obtenue après élagage de 4 neurones cachés.

Un autre choix consiste à utiliser le réseau possédant 3 neurones cachés (élagage de 7 neurones cachés). En effet ce réseau arrive à une performance de 85.57%, ce qui est statistiquement très proche du réseau possédant 6 neurones cachés.

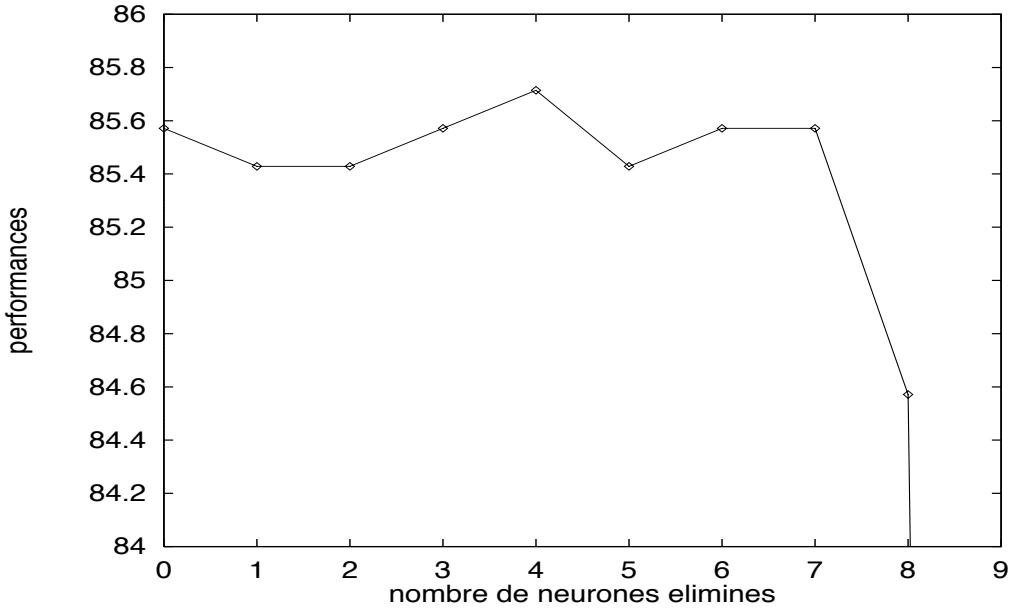


FIG. 3.5 – *l'évolution globale des performances sur D^v durant le processus d'élagage des neurones cachés.*

Comme le montre la table 3.1, notre méthode d'ajustement nous a permis d'améliorer les performances sur D^t. Elles sont passées de 85.37 à 85.46. Notez que nous sommes approximativement à 0.54 de l'erreur théorique Bayesienne qui est égale à 14% sur D^t.

	Architecture	performances sur		
		D^l	D^v	D^t
Avant élagage	$< 40\|10\|3 >$ (443 paramètres)	92.00 [88.37 , 94.57]	83.85 [80.95 , 86.40]	82.83 [81.68 , 83.93]
Après Sélection de variables	$< 16\|10\|3 >$ (203 paramètres)	88.66 [84.58 , 91.78]	85.57 [82.77 , 87.98]	85.37 [84.28 , 86.40]
Après élimination de neurones cachés	$< 16\|6\|3 >$ (123 paramètres)	89 [84.95 , 92.06]	85.71 [82.93 , 88.11]	85.46 [84.38 , 86.49]

TAB. 3.1 – *Nombre de paramètres et performances avant élagage, après sélection de variables, et après sélection de variables et ajustement d'architecture.*

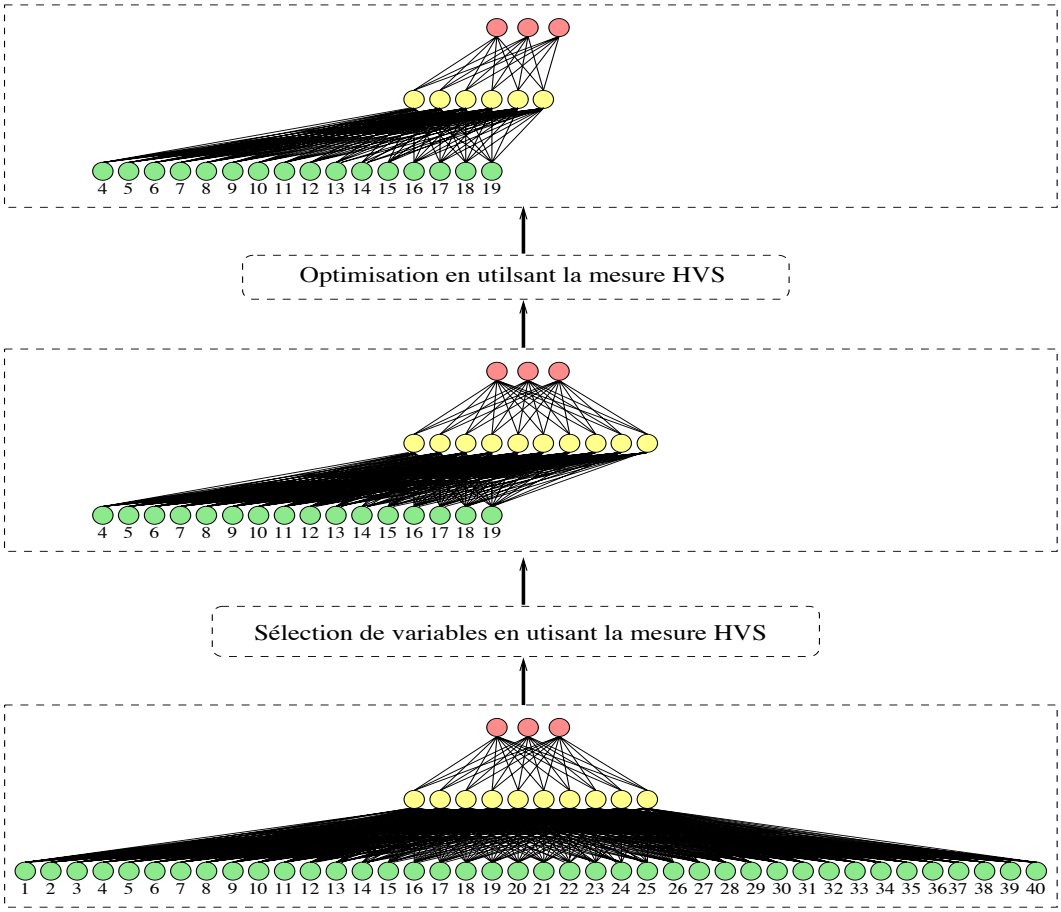


FIG. 3.6 – *l'application de HVS aux variables et aux neurones cachés nous a permi de supprimer 320 paramètres, correspondant à un pourcentage d'élagage de 72.23%*

Comme le montre la figure 3.6, l'application de HVS aux variables et aux neurones cachés nous a permis de supprimer 320 paramètres, correspondant à un pourcentage d'élagage de 72.23%.

3.3.2.2 Ajustement d'architecture en régression

Pour valider notre méthode d'ajustement d'architecture sur un problème de régression, nous avons choisi de continuer nos expériences sur le problème sunspots. Nous avons repris les 30 expériences faites au chapitre précédent, lors de la sélection de variables. Pour chaque expérience, nous avons appliqué notre technique d'élagage des neurones internes, à la fin de la procédure de sélection de variables.

La figure 3.7 montre l'évolution de l'arv sur D^v durant le processus d'élagage des neurones cachés pour un réseau (celui qu'on a montré lors de la sélection de variables).

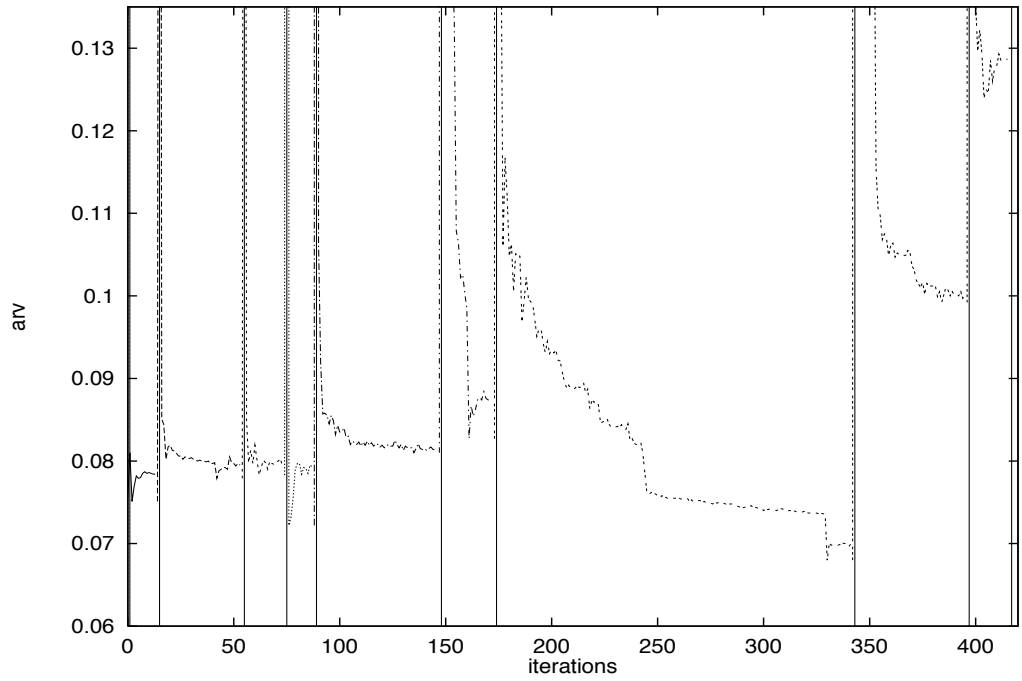


FIG. 3.7 – *l'évolution de l'arv durant le processus d'élagage des neurones cachés pour un réseau.*

De même que pour la sélection de variables, les intervalles délimités par les lignes verticales correspondent aux différentes phases d'élagage. A chaque suppression d'un neurone caché, un réapprentissage est nécessaire. Le "Early Stopping" est le critère utilisé pour arrêter la phase de réapprentissage.

Les meilleures arv sur D^v obtenues en fonction du nombre de neurones cachés supprimés sont montrées sur la figure 3.8. Cette figure illustre en quelque sorte l'évolution globale de l'arv sur D^v durant la phase d'optimisation. Pour choisir le meilleur système, nous avons appliqué un "Early Stopping" global. Nous avons choisi donc le système ayant donné la meilleure arv sur D^v. Comme on peut le voir sur la figure 3.8, la meilleure arv sur D^v (pour la même expérience) est obtenue après élagage de 7 neurones cachés.

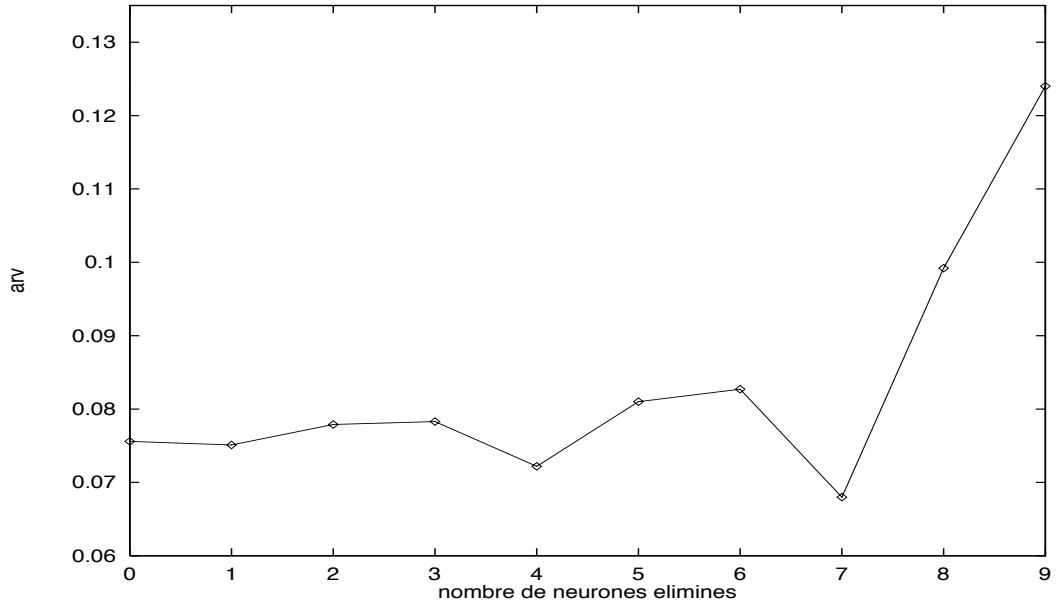


FIG. 3.8 – *l'évolution globale de l'arv durant le processus d'élagage des neurones cachés pour un réseau.*

La table 3.2 donne, pour cette expérience, les arv obtenues à la fin de la phase d'apprentissage, après sélection de variables et après sélection de variables et ajustement d'architecture.

	D^l	D^v	D^t
A la fin de la phase d'apprentissage	0.0832	0.0906	0.4032
Après sélection de variables	0.0802	0.0756	0.3437
Après sélection de variables et ajustement d'architecture	0.0869	0.068	0.2968

TAB. 3.2 – *Résultats d'une expérience.*

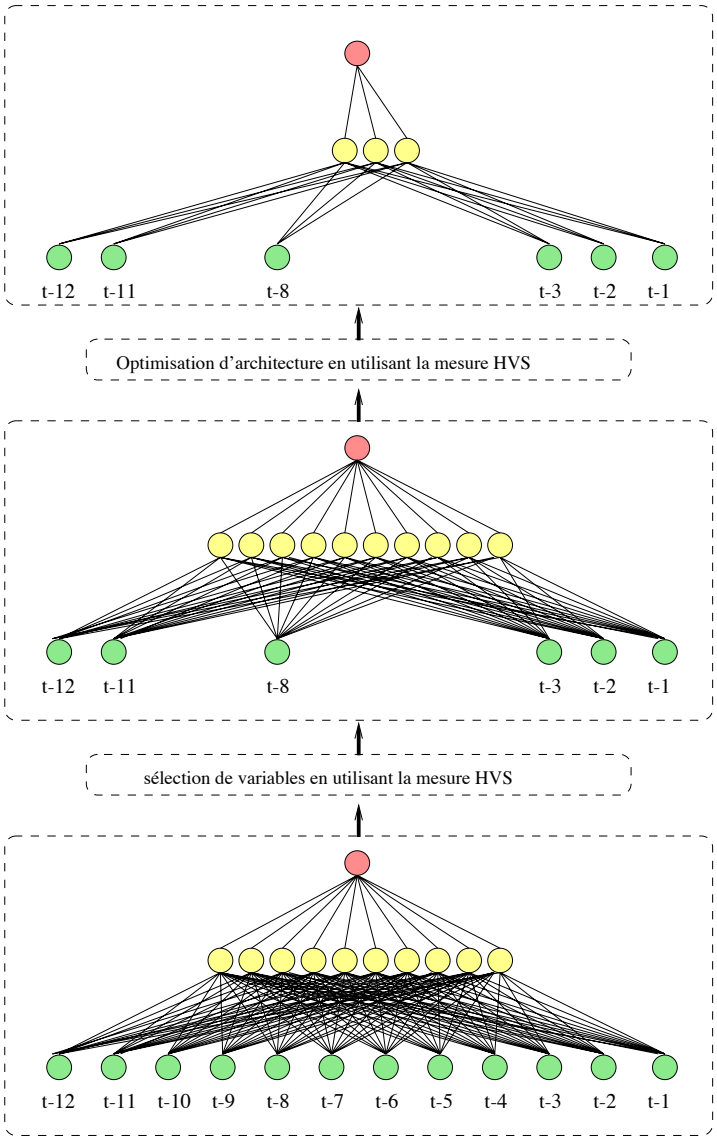


FIG. 3.9 – *l'application de HVS aux variables et aux neurones cachés.*

La figure 3.9 illustre, pour ce réseau, l'effet de l'utilisation de la mesure HVS pour la sélection de variables et l'élagage de neurones cachés. Cette utilisation nous a permis de supprimer 116 paramètres, correspondant à un pourcentage d'élagage de 82.27%.

La table 3.3 présente nos résultats, comme une moyenne des 30 expériences, après sélection de variables et ajustement d'architecture, ainsi que quelques résultats obtenus en utilisant d'autres méthodes. Remarquez que notre méthode permet non seulement de réduire la complexité effective du modèle mais aussi d'obtenir des prédictions plus précises que celles obtenues par d'autres approches.

Modèle/arv	D^l(1712-1920)	D^v(1921-1955)	D^t(1956-1979)
Yacoub & Bennani	0.084	0.072	0.310
Goutte[40]	0.082	0.082	0.357
Svar & al.[87]	0.090	0.082	0.35
Weigend & al.[96]	0.082	0.086	0.35
Linèaire	0.131	0.128	0.36

TAB. 3.3 – *Comparaison des résultats obetenus sur la série des taches solaires en utilisant différentes méthodes. Notre résultat est la moyenne de 30 expériences.*

3.4 Conclusion du chapitre

Nous avons montré, à travers deux problèmes types d'utilisation des réseaux connexionnistes: discrimination et modélisation, que notre méthode de sélection de variable peut être étendue aux neurones internes. Nous disposons donc d'une méthode qui permet de sélectionner les variables pertinentes et d'optimiser l'architecture, en éliminant les neurones cachés inutiles. Les exemples que nous avons présentés montrent que cette façon de faire permet non seulement d'obtenir de très bons résultats de prédiction ou de discrimination en comparaison avec d'autres résultats obtenus par d'autres chercheurs, mais aussi de réduire considérablement la compléxité effective du modèle (plus de 50% des paramètres du système sont supprimés).

Chapitre 4

Détection et sélection de zones discriminantes

Dans ce chapitre, nous proposons d'étendre l'application de la mesure HVS à un autre type d'architectures utilisé dans le domaine du traitement d'images. Notre objectif est d'une part d'optimiser la taille de ce type d'architectures et d'autre part de détecter et sélectionner des zones discriminantes les plus pertinentes. L'utilisation de HVS dans ce cas peut être vu comme une optimisation de la connaissance a priori utilisée lors de la conception de ce type de système. [1]

1. Accepté à IJCNN'99 [101]

4.1 Introduction

Jusqu'à présent, nous nous sommes intéressé aux réseaux multicouche dans lesquels les liaisons entre couches étaient totales. Ces réseaux ont montré leur efficacité sur de nombreux problèmes. Néanmoins, ils sont en général inadaptés aux problèmes dans lesquels les formes à traiter sont de très grande taille, comme dans le cas du domaine de traitement d'images. Pour ce type de problème, l'utilisation d'un réseau multicouche entièrement connecté induit trop de paramètres à estimer alors qu'on dispose, en générale, que de peu d'exemples pour l'apprentissage. Les architectures souvent utilisées pour ce type de problème sont celles incluant des connaissances a priori sur le problème. C'est à ce type d'architectures que nous nous intéressons dans ce chapitre.

L'utilisation de ce type d'architectures (incluant des connaissances a priori sur le problème) permet de réduire considérablement le nombre de paramètres libres du système d'apprentissage. Ces architectures sont composées d'un module d'extraction de traits et d'un module de classement. Le classifieur est le module qui utilise le plus de paramètres libres (connexions totales avec le module extracteur de traits). Le nombre de paramètres libres utilisés par ce dernier est directement lié au nombre de traits extraits par le module d'extraction. Ces traits ne sont pas tous informatifs et la plupart sont redondants voir non pertinents. Nous proposons de sélectionner les traits pertinents, en utilisant la mesure HVS, et de rétropropager l'effet de cette sélection sur la rétine pour détecter et sélectionner les zones discriminantes les plus pertinentes. Nous verrons que cette façon de faire permet de réduire considérablement le nombre de paramètres dans ce type d'architectures.

Dans ce chapitre, nous présentons tout d'abord ce type d'architectures en illustrant leur utilisation dans le domaine de la reconnaissance de visages. Nous décrivons ensuite notre méthode de sélection de zones discriminantes et d'optimisation de ce type d'architectures. Pour finir, nous testons son éfficacité sur un problème de reconnaissance de visages et nous donnons nos perspectives dans cette direction de recherche.

4.2 Systèmes de discrimination et connaissances a priori

Dans cette section, nous introduisons les architectures multicouche incluant des connaissances a priori du problème à traiter. Tout d'abord nous rappelons les différents types de connexions classiques. Ensuite nous abordons le problème de reconnaissance de visages en illustrant comment ce type d'architectures a été utilisé dans ce domaine.

4.2.1 Connexions classiques

Nous présentons les trois types de connexions classiquement utilisés dans les architectures organisées en couches où chaque unité n'est connectée qu'à des unités de la couche précédentes : connexions globales, connexions locales et connexions à masques.

4.2.1.1 Connexions globales

Dans ce schéma de connexions, tous les neurones de la couche $k+1$ sont connectés aux neurones de la couches k (figure 4.1). En d'autres termes, l'état de chaque neurone de la couche $k+1$ est calculé en fonction des états de tous les neurones de la couche k.

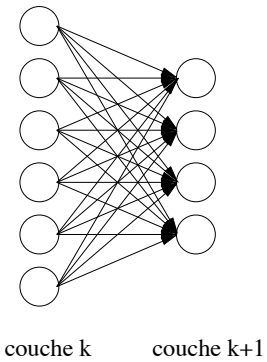


FIG. 4.1 – *connexions globales. Les neurones de la couche $k+1$ sont connectés à tous les neurones de la couche k.*

Ce type de connexions est simple à réaliser car aucune connaissance ni choix n'est nécessaire concernant les unités à connecter. Avec des connexions globales on atteint rapidement un très grand nombre de paramètres à ajuster (les poids des connexions). Plus il y a de paramètres libres à ajuster, plus on a besoin d'informations (exemples) pour estimer la fonction globale du système. Pour limiter le nombre de paramètres de ces modèles, d'autres types de connexions ont été conçus. Nous présentons quelques exemples dans la suite.

4.2.1.2 Connexions locales

Dans ce schéma, chaque neurone de la couche $k+1$ n'est connecté qu'à un sous ensemble de neurones de la couche k. En d'autres termes, l'état de chaque neurone de la couche $k+1$ est fonction seulement des états d'un sous ensemble de neurones de la couche k. La figure 4.2 illustre les connexions locales entre deux couches de neurones. Outre le concept de connexion locale, on voit aussi la notion de recouvrement des poids. En effet, parmi les trois cellules dans la couche k connectées à une cellule i dans la couche $k+1$ deux sont connectés aussi à la cellule $i+1$. On parle alors de recouvrement de 2 unités, ou de façon complémentaire, d'un déplacement de 1.

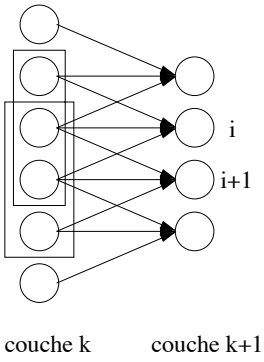


FIG. 4.2 – *Connexions locales entre deux couches de neurones. Chaque neurone de la couche $k+1$ est connecté seulement à 3 neurones de la couche k.*

Les connexions locales permettent de réduire le nombre de paramètres d'un réseau, mais surtout de spécifier, dans l'architecture du réseau, des caractéristiques du problème à résoudre.

4.2.1.3 Connexions à poids partagés

Dans ce schéma, les neurones d'une couche partagent tous les mêmes poids. La figure 4.3 illustre un exemple de connexions à masques, ou à poids partagés. Les neurones $(j, j+1, j+2)$ de la couche k sont connectés au neurone i de la couche $k+1$, avec les poids de connexions (w_1, w_2, w_3). De même, les neurones $(j+1, j+2, j+3)$ de la couche k sont connectés au neurone $i+1$ de la couche $k+1$ avec exactement les mêmes poids (w_1, w_2, w_3), et ainsi de suite. L'information arrivant à chaque neurone de la couche $k+1$ est traitée de la même façon. L'utilisation de masques permet ainsi d'imposer un traitement invariant par translation. De plus, le partage des poids permet de réduire significativement le nombre de paramètres libres dans le réseau.

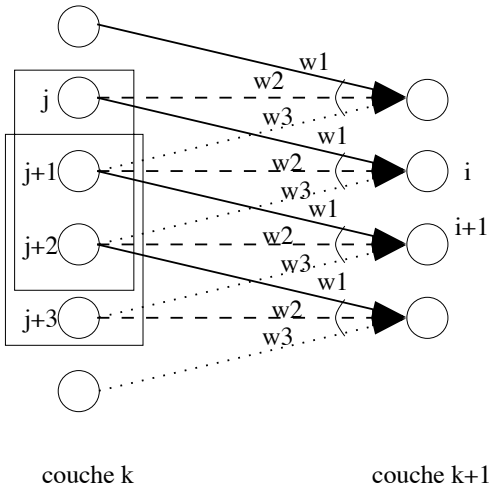


FIG. 4.3 – *connexions locales à masque ou à poids partagés. Dans cette figure, un même type de pointillé dans les flèches représente une même valeur du poids de connexion.*

Dans la figure 4.3, l'arrangement de la couche k est unidimensionnel et le masque est déplacé, en longueur, sur cette couche. C'est un masque unidimensionel. La figure 4.4 illustre un masque bidimensionel : l'arrangement de la couche k est bidimensionel et le masque est déplacé dans les deux sens, en largeur et en longueur.

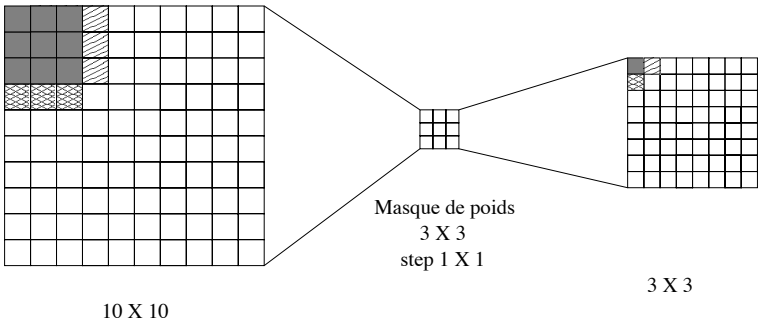


FIG. 4.4 – *Un masque de taille* 3×3 *se décalant par pas de 1 dans chaque direction.*

0	-1	0
-1	+4	-1
0	-1	0

Laplacien

-1	0	+1
-2	0	+2
-1	0	+1

Sobel

-1	0	+1
-1	0	+1
-1	0	+1

Gradient

FIG. 4.5 – *Différents masques utilisés couramment pour des tâches d'extraction des contours dans une application de traitement des images par de méthodes classiques.*

La figure 4.5 montre trois masques typiques utilisés par des méthodes classiques de traitement d'images pour effectuer des transformations de bas niveau sur l'information. Ces masques peuvent être utilisés dans un réseaux multicouche avec poids partagés en donnant de bonnes valeurs aux poids de connexions. Cependant, la capacité d'apprentissage des méthodes connexionnistes est telle que le réseau lui même peut trouver le masque nécessaire afin de réaliser l'opération désirée.

La figure 4.6 montre un exemple de réseau très populaire en reconnaissance de chiffres, Le réseau LeNet [59], qui utilise des masques.

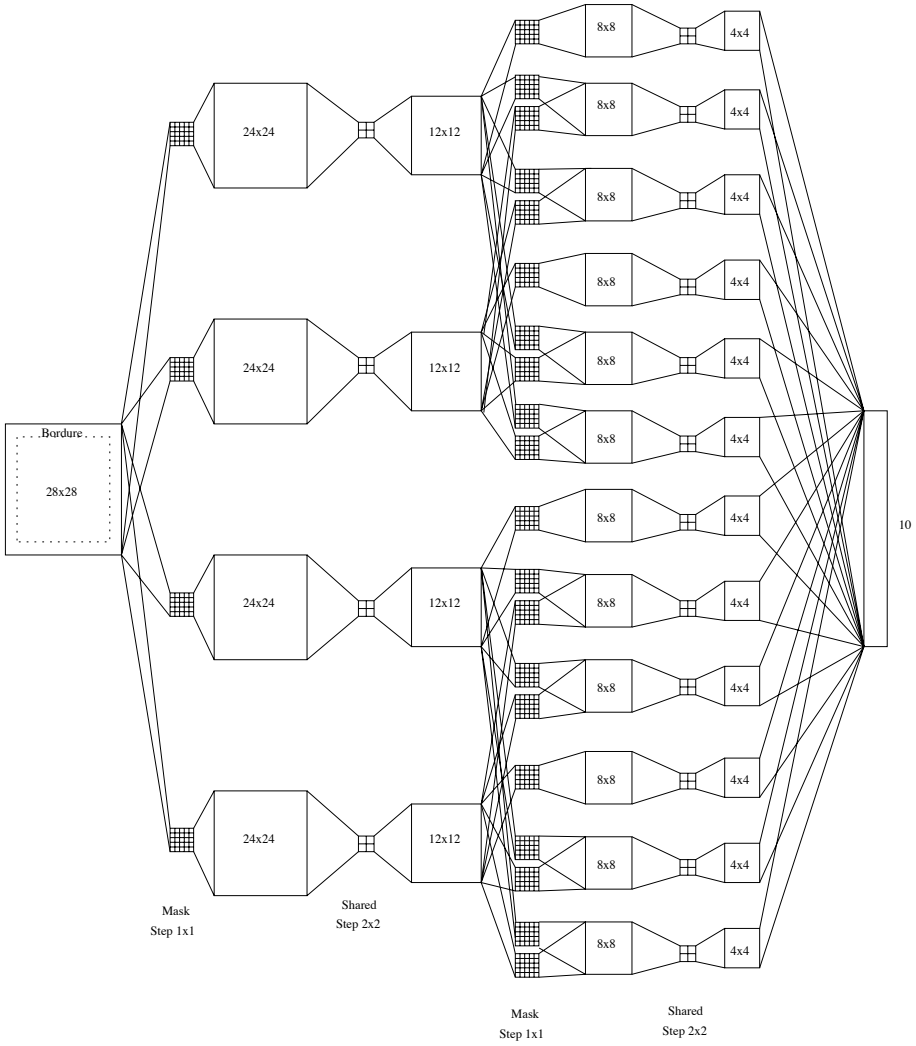


FIG. 4.6 – *Architecture du réseau LeNet.*

4.2.2 Application à l'identification de visages

Comme nous avons choisi de valider notre méthode dans le domaine de la reconnaissance de visages, nous présentons quelques systèmes connexionnistes proposés dans ce domaine. Nous nous intéressons plus particulièrement à la tâche d'identification de visages. Iditentifier un visage revient à retrouver l'identité d'une personne à partir d'une vue du visage.

Les deux principales approches utilisées dans le domaine de la reconnaissance

de visages sont : les approches basées sur les caractéristiques du visage et les approches basées sur le visage. Dans la première approche, les visages sont représentés en termes de distances, angles et surfaces dérivés de caractéristiques élémentaires comme le nez, la bouche, le menton ou les yeux [17]. Dans la deuxième approche les images de visages sont numérisées et représentées par l'intensité lumineuse de chaque pixel [25, 29, 39, 92]. Nous nous intéréssons qu'aux méthodes de la deuxième approche.

Nous allons présenter maintenant quelques systèmes connexionnistes d'identification de visages. D'abord nous présentons deux systèmes parmi ceux qui traitent l'image comme un vecteur mono-dimensionnel [25, 29] ensuite nous présentons un système de type TDNN qui permet de prendre en compte la structure bi-dimensionnelle de l'image [92].

Le système proposé par Fleming et Cottrell [29] est composé d'un réseau auto-associatif (diabolo) et d'un réseau de classement (figure 4.7).

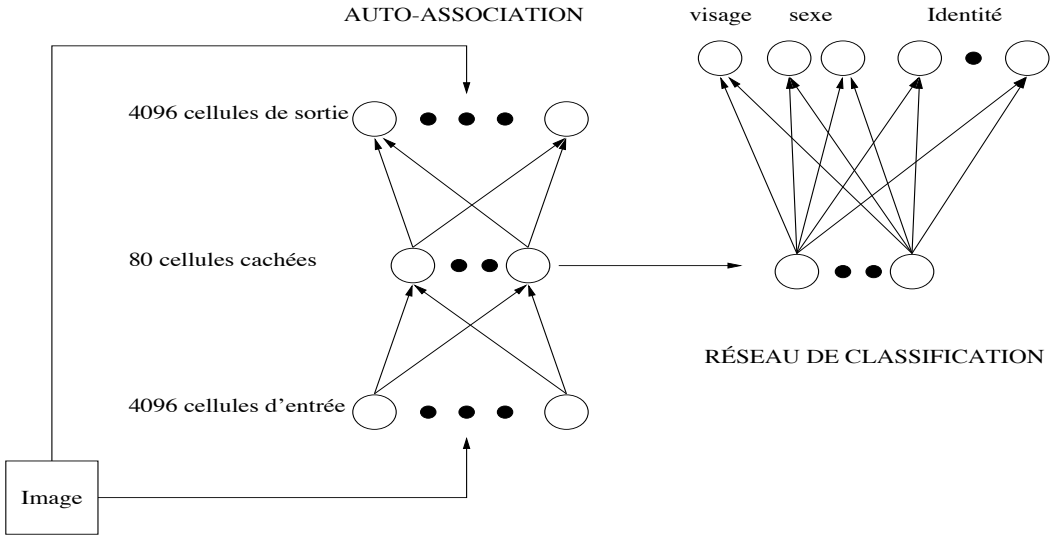


FIG. 4.7 – *Architecture du système de Fleming et Cottrel [29].*

Dans un premier temps le réseau auto-associatif est entraîné à reproduire les images qui lui sont présentées en entrée. Une fois cet apprentissage terminé, le réseau de classification est entraîné à détecter le sexe, l'identité et si c'est un visage. Les entrées de ce réseau sont les sorties de la couche cachée du réseau auto-

associatif. Ce type d'approche a été largement utilisé dans différents domaines. Par exemple en reconnaissance de la parole, pour une tâche d'identification du locuteur où les chaines de Markov cachées ont été entrainées sur le codage des TDNN [2, 9]

Le réseau est entraîné avec 64 images de 11 visages et 13 images de "non-visages". De bons résultats ont été rapportés sur cette base de petite taille. Notez que le réseau auto-associatif utilise plus de 655000 connexions.

Le réseau précédent n'utilise qu'une couche cachée qui fonctionne comme une ACP, et ne semble sensible qu'aux liaisons linéaires. DeMers et Cottrel [25] proposent un réseau qui devrait détecter des relations non-linéaires (figure 4.8).

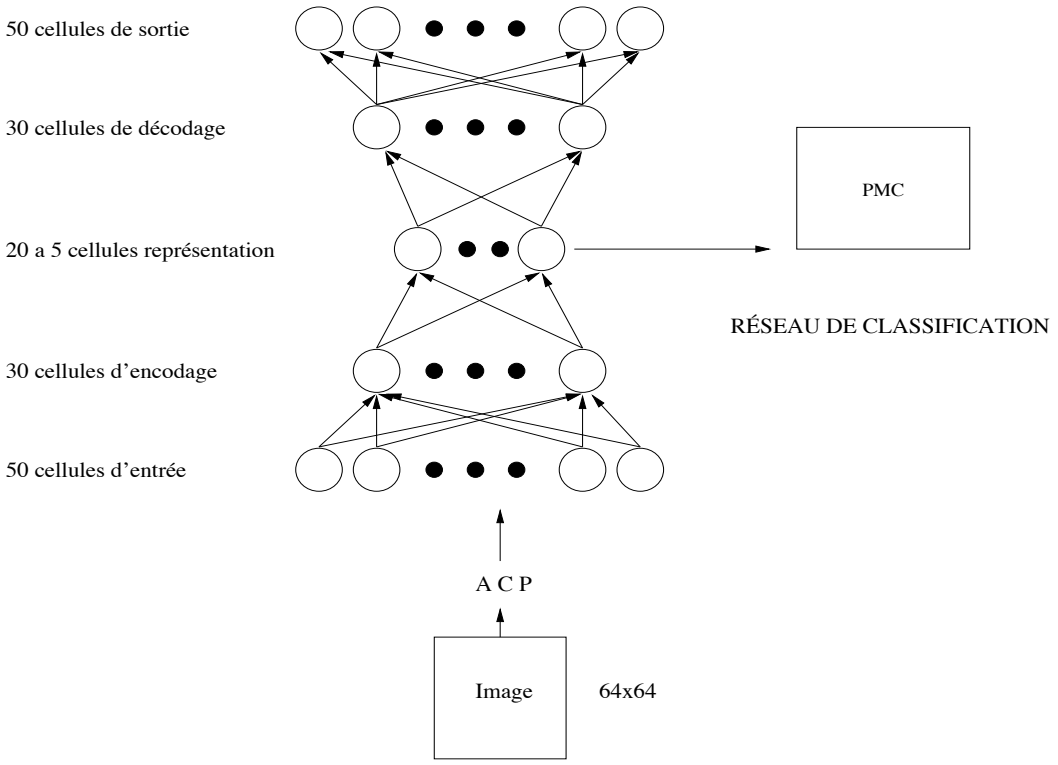


FIG. 4.8 – *Architecture du système de DeMers et Cottrel [25].*

Les images de taille 64×64 sont tout d'abord représentées par leur 50 premières composantes principales qui servent d'entrée à un réseau auto-associatif. Ensuite le réseau auto-associatif est entraîné. Puis le nombre de cellules de la couche de

représentation est réduit en éliminant les cellules avec la variance d'activation la plus faible, puis le réseau est ré-entraîné. Le réseau arrive à reconstituer de manière satisfaisante des visages avec une couche de représentation de 5 cellules. Une fois terminé avec ce réseau auto-associatif, les sorties des cellules de représentations sont utilisées comme entrée d'un PMC qui est ensuite entraîné pour détecter le sexe et l'identité. La base utilisée est composée de 20 personnes photographiées avec 8 expressions différentes. De bons résultats sont rapportés. Le réseau auto-associatif utilise 3300 connexions.

Contrairement aux sytèmes précédents qui traitent l'image comme un vecteur mono-dimensionnel et donc n'utilisent aucune connaissance du problème, les systèmes utilisant des réseaux de type TDNN permettent d'exprimer par exemple une certaine invariance de la reconnaissance au translation de l'image. L'architecture du réseau TDNN utilisé par Viennet dans [92] pour la reconnaissance de visages est montrée dans la figure 4.9. Cette architecture comprend 30431 connexions mais du fait de l'utilisation des masques de poids partagés fait que l'architecture ne comprend que 1844 poids (paramètres libres). De bons résultats ont été rapportés sur deux bases différentes.

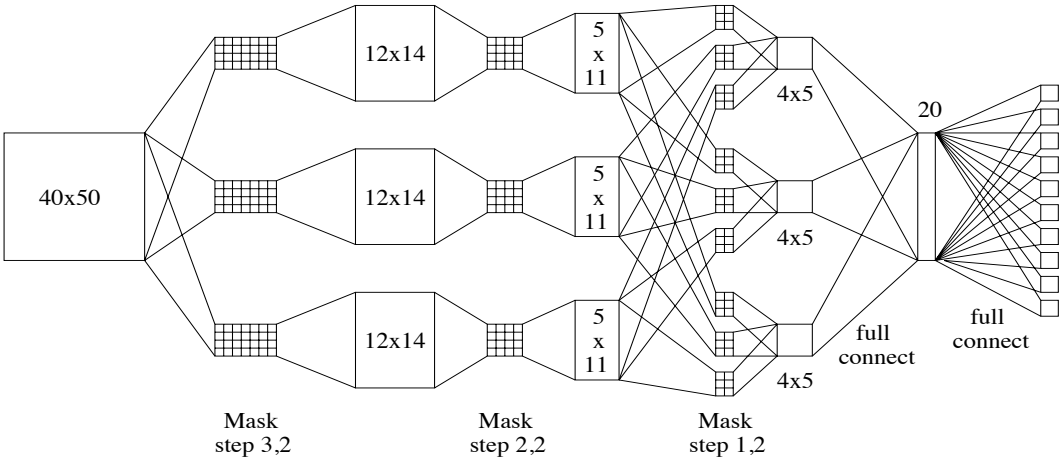


FIG. 4.9 – *Architecture du réseau TDNN «faci40×50».*

Le système d'identification de visages que nous utilisons est un réseau de type TDNN. Dans la section suivante nous proposons une méthode permettant de simplifier ce type d'architectures.

4.3 Détection et sélection de zones discriminantes

Dans cette section, nous présentons notre méthode permettant de réduire encore plus le nombre de paramètres dans ce type d'architectures incluant des connaissances a priori sur le problème. Nous proposons de réduire le nombre de paramètres du classifieur en considérant ce problème comme un problème classique de sélection de variables. Dans ce cas les variables sont représentées par les traits extraits par les couches cachées (le résultat du codage). En d'autres termes, nous procédons à une sélection de traits. Une fois le classifieur simplifié, l'effet de l'élagage des traits non pertinents sera rétropropagé sur les couches inférieures jusqu'à la rétine. Cette deuxième phase nous permet d'une part de simplifier le module d'extraction de caractéristiques et d'autre part de détecter les zones discriminantes de la rétine.

Nous avons choisi de valider notre approche dans le domaine de reconnaissance de visages. Tout d'abord nous présentons le système sur lequel nous allons travailler ainsi que la base de données utilisée. Nous décrivons ensuite le principe de notre méthode et nous testons son efficacité.

4.3.1 Présentation du système

Le système que nous allons utiliser est un réseau TDNN. Il est composé de deux modules (figure 4.10):

1. le premier module réalise une extraction de traits à partir de l'image présentée au réseau;
2. le deuxième module effectue une discrimination à partir des traits extraits par le premier module.

Cette architecture composée de deux modules peut être basée sur la coopération de deux systèmes indépendants. En effet de Bollivier et al. [14] proposent de remplacer le module de discrimination dans un PMC (dernière couche) par un autre classifieur plus puissant de type LVQ. Cette idée a été adaptée aux modèles TDNN en reconnaissance de la parole par Bennani et al. [11]. En image le même

principe a été utilisé par Fogelman Soulié et al. pour l'identification de visages, où ils proposent l'utilisation de l'algorithme RBF pour la discrimination [30]. Dans notre cas le deuxième module est un PMC.

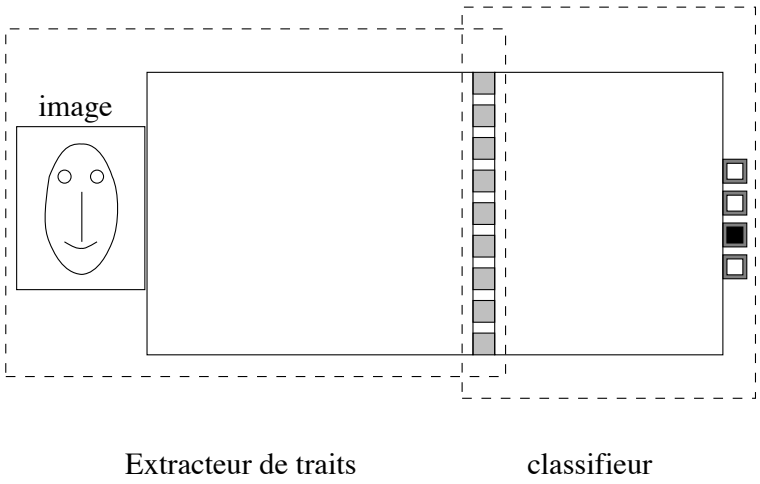


FIG. 4.10 – *Le réseau est composé d'un extracteur de traits et d'un classifieur*

4.3.2 Base de données

Il s'agit d'une base d'images représentant les visages de 4 personnes dans différentes positions et contexte. Elle contient 92 photos prises sous différents angles. Les photos sont digitalisées avec une résolution de 40 x 50 pixels, avec une palette de 256 niveaux de gris. Aucun pré-traitement n'est effectué sur les données : elles ont été simplement introduites dans la couche d'entrée de 2000 unités.

4.3.3 Principe de la détection et de la sélection

La méthode que nous proposons pour sélectionner des zones discriminantes procède en trois étapes : extraction de traits, sélection de de traits et sélection de zones discriminantes.

Etape 1 : Extraction de traits

Cette étape consiste à faire l'apprentissage en utilisant les deux modules, elle permet au premier module d'extraire des traits.

Etape 2 : Sélection de traits

Pour supprimer les traits non pertinents proposés par le premier module, nous appliquons notre méthode de sélection de variables au deuxième module.

Il est très important de noter que pour évaluer l'importances des traits extraits par le premier module, la mesure HVS n'utilise que les poids du second module (le classifieur). Par conséquent, la sélection de traits en utilisant la mesure HVS ne demande pas beaucoup de calculs relativement à ce type d'architectures.

Etape 3 : Sélection de zones discriminantes (rétine)

Pour sélectionner les zones discriminantes de la rétine, il suffit de rétropropager l'effet de la méthode de sélection de variable sur le premier module. Le principe est très simple : si l'ensemble des neurones qui utilisent comme entrée la sortie d'une cellule est vide alors éliminer toutes les connexions arrivant à cette cellule, c.à.d. si $fan{-}out(i) = \emptyset$ alors $\forall j \in fan{-}in(i), w_{ij} = 0$. On applique ce principe en commençant par la dernière couche de l'extracteur de traits. Remarquez que cette étape ne demande aucun calcul, on ne fait que des tests sur les $fan - out(i)$ où i est une cellule. L'ensemble des cellules sélectionnées de la rétine représente les zones discriminantes.

4.3.4 Expérimentation

Nous avons choisi un système dont l'architecture est très redondante. L'architecture que nous avons utilisée est représentée sur la figure 4.11. Le visage à reconnaître est présenté sur la rétine de taille 40 x 50. La première couche cachée est constituée de deux groupes de cellules, de taille 20 x 25, connectés à la rétine

par des masques de poids partagés de taille 3 x 3 se décalant par pas de 2 dans chaque direction. La deuxième couche cachée est constituée de quatre groupes de cellules de taille 6 x 7. Chaque groupe de la première couche cachée est connecté à deux groupes de la deuxième couche cachée par des masques de poids partagés de taille 5 x 7 se décalant par pas de 3 dans chaque direction. Enfin, la deuxième couche cachée est totalement connectée aux quatre cellules de sortie (une par personne).

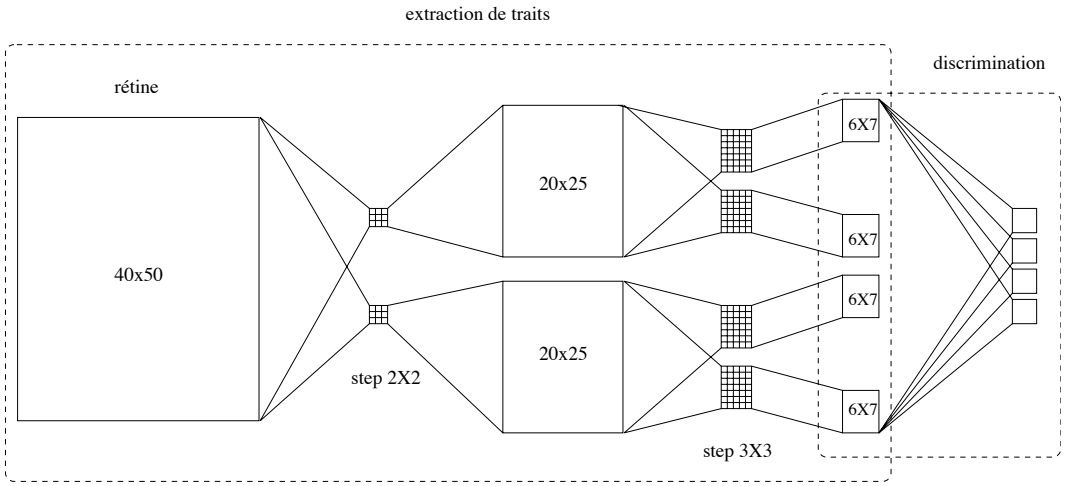


FIG. 4.11 – *Architecture du système d'identification de visages.*

Le réseau est ainsi composé d'un extracteur de traits, fournissant un codage sur 168 valeurs, et d'un classifieur qui effectue une discrimination basée sur les traits extraits. Au total, l'architecture contient 3172 cellules : 2000 sur la rétine, 1000 sur la première couche cachée, 168 sur la deuxième couche cachée et 4 sur la couche de sortie. La table 4.1 donne le nombre de poids et de connexions que comporte chaque module.

Etape 1

Nous avons utilisé un ensemble d'apprentissage D^l de 76 exemples et un ensemble de validation D^v de 16 exemples. A la fin de la phase d'apprentissage, le taux de

	Extracteur	Classifieur
nombre de connexions	14880	672
nombre de poids (libres)	158	672

TAB. 4.1 – *Nombre de connexions et de poids que comporte chaque module du système d'identification.*

reconnaissance est de 100% sur D^l et de 100% sur D^v.

Etape 2

A la fin de l'apprentissage, nous avons appliqué au classifieur notre méthode de sélection de variables. La table 4.2 donne les performances et le nombre de traits avant et après élagage en utilisant la mesure HVS. Remarquez que nous avons supprimé 94% de traits tout en gardant les mêmes performances sur D^l et D^v.

Cette suppression a permis aussi de réduire le nombre de paramètres du classifieur en passant de 672 paramètres à 40 paramètres.

	nombre de traits	performances sur D^l	performances sur D^v	nombre de paramètres du classifieur
Avant Sélection	168	100% [95.19 , 100]	100% [80.64 , 100]	672
Après Sélection	10	100% [95.19 , 100]	100% [80.64 , 100]	40

TAB. 4.2 – *Parmi les 168 traits extraits par le premier module, seuls 10 ont été sélectionnés par HVS, tout en gardant les performances à* 100% *sur* D^l *et sur* D^v.

Etape 3

Dans un premier temps, nous avons rétropropagé l'effet de HVS sur la première couche cachée. Nous avons supprimé 518 cellules sur cette couche, correspondant à un pourcentage d'élagage de 51.8%. Ensuite, nous avons rétropropagé l'effet de HVS sur la rétine. Nous avons supprimé 923 pixels sur la rétine, correspondant

à un pourcentage d'élagage de 46%. La figure 4.12 illustre l'effet de HVS sur la rétine et les cellules cachées.

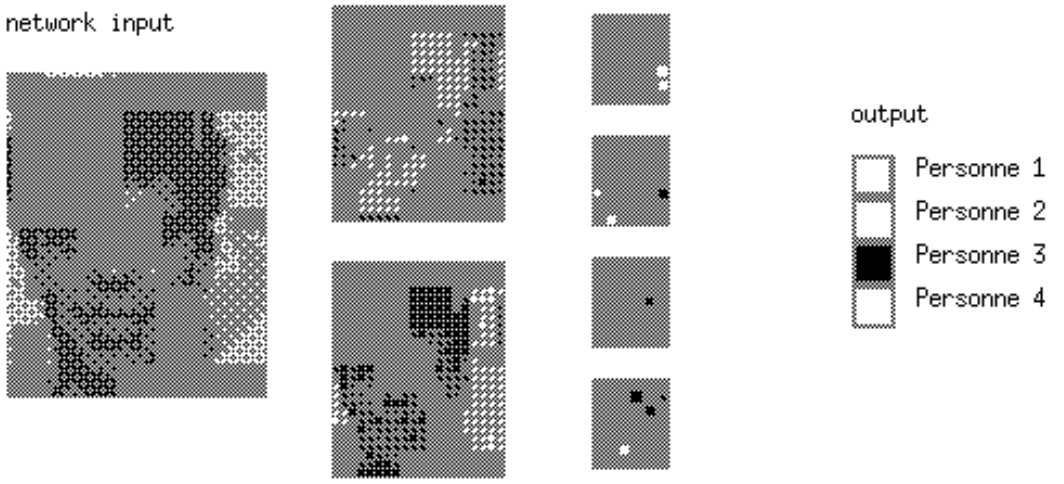


FIG. 4.12 – *Effet de HVS sur la rétine et les unités des couches cachées.*

	nombre de cellules			performances	
	rétine	1ère couche cachée	2ème couche cachée	D^l	D^v
Avant Sélection	2000	1000	168	100% [95.19 , 100]	100% [80.64 , 100]
Après Sélection	1077	482	10	100% [95.19 , 100]	100% [80.64 , 100]
pourcentage d'élagage	46 %	51.8 %	94 %		

TAB. 4.3 – *Effets de HVS sur les cellules et les performances du système.*

La table 4.3 illustre l'effet de l'application de la mesure HVS pour la sélection de zones discriminantes. Nous avons élagué presque la moitié des unités de la rétine tout en gardant les performances du système à 100% sur l'ensemble D^l et sur l'ensemble D^v.

La figure 4.13 montre quelques images après sélection de zones discriminantes les plus pertinentes. Malgré la petite taille de la base de donnée, on peut néanmoins

apporter quelques interprétations aux résultats obtenus. Les images de la figure 4.13 montrent que les zones discriminantes sélectionnées comportent le plus souvent des régions du visage supposées discriminantes : les yeux, le nez, la bouche, le menton et les cheveux. Ce résultat confirme les constatations rapportées par d'autres chercheurs, comme Lawrence et al. [56] qui montrent que pour le problème d'identification de visages, les yeux, le nez, la bouche, le menton et les cheveux sont les régions les plus importantes pour cette tâche. La validation de ces interprétations est en cours d'étude sur une grande base de données (ORL [2]).

Evidemment, il n'y a aucune raison pour que les zones discriminantes sélectionnées par la mesure HVS correspondent forcément aux caractéristiques conseillées dans le domaine. Mais on peut envisager une coopération bidirectionnelle entre les informations fournies par les experts et les résultats de notre approche. Les caractéristiques proposées par les experts peuvent être utilisées comme un point de départ de notre méthode. Une autre coopération consiste à utiliser nos résultats pour compléter voir améliorer les résultats des experts.

2. Cette base contient 400 images prises entre Avril 1992 et Avril 1994 au "Olivetti Research Laboratory" à Cambrdge, UK. Elle est disponible à l'adresse suivante :
http://www.cam-orl.co.uk/facedatabase.html

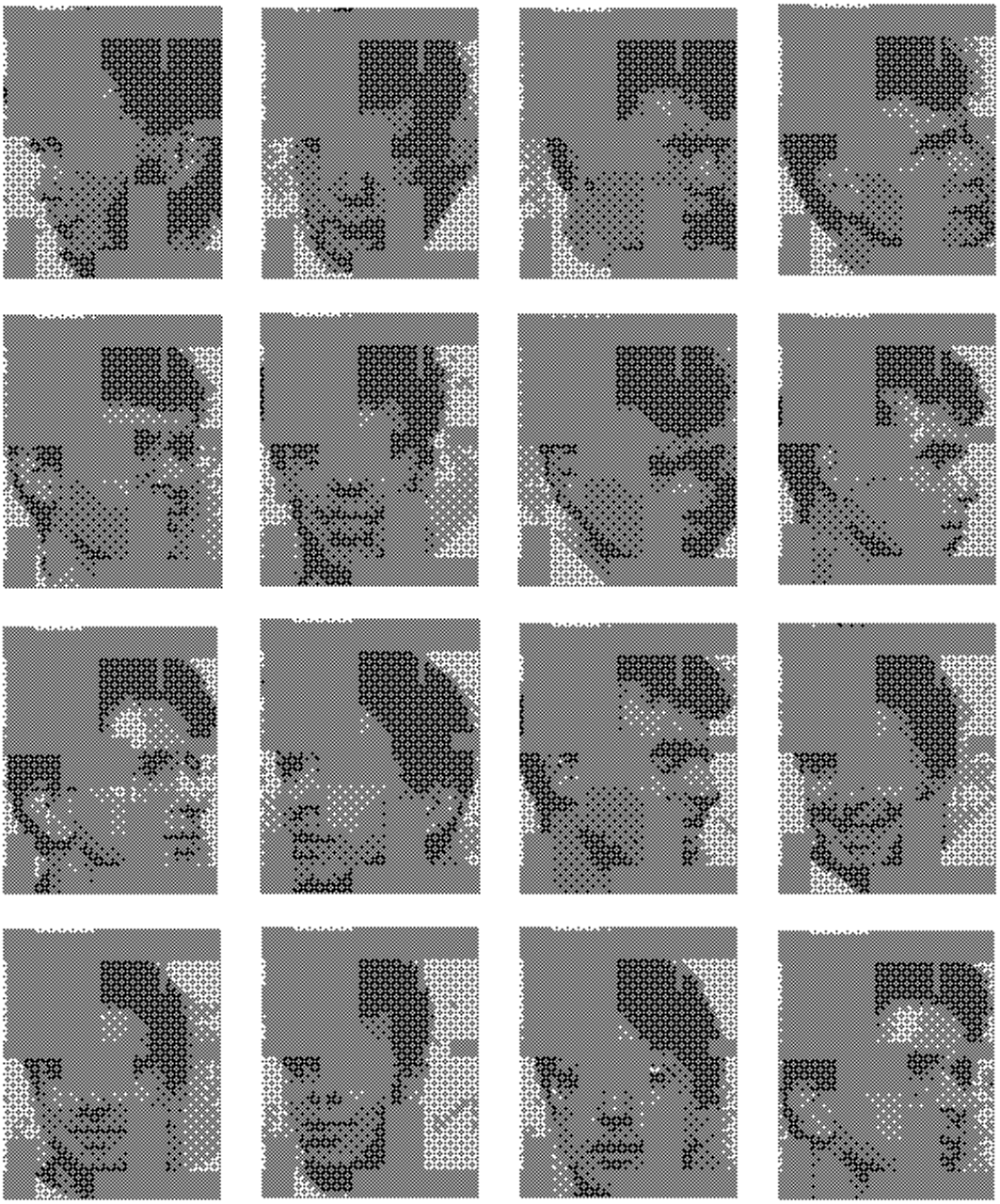


FIG. 4.13 – *Quelques images après élagage de* 46% *de pixels sur la rétine en utilisant la mesure HVS.*

4.4 Conclusion

Dans ce chapitre, nous nous sommes intéressés au type d'architectures très utilisé dans le domaine du traitement d'images, celles incluant des connaissances a priori sur le problème. L'utilisation de ce type d'architectures permet déjà de réduire considérablement le nombre de paramètres libres des systèmes. Nous avons proposé une méthode permettant d'une part de réduire davantage la taille des architectures de ce type, et d'autre part de sélectionner les zones discriminantes les plus pertinentes. De plus, la méthode que nous avons proposé ne demande que peu de calculs simples.

Nous avons testé l'éfficacité de cette méthode sur un problème de reconnaissance de visages. Les résultats rapportés sur une petite base de 4 personnes contenant 24 images par personnes sont significatifs : notre méthode nous a permi de réduire la rétine de 46% sans que les performances du réseau baissent. L'analyse approfondie de notre méthode sur une grande base est une de nos perspectives immédiates.

Chapitre 5

Extraction de connaissances pour la sélection d'architecture dédiée à l'apprentissage de machines d'état fini

Pour traiter des séquences temporelles, nous avons besoin d'une forme de mémoire à court terme pour rendre le réseau dynamique. Plusieurs architectures dotées de cette mémoire ont été proposées. Dans ce chapitre, nous proposons une méthode d'extraction de connaissances à partir d'exemples permettant de mieux choisir parmi ces architectures la plus adaptée à la machine d'état fini que l'on cherche à modéliser.[1]

1. Résultats publiés dans IJCNN'98 [100]

5.1 Introduction

Dans ce chapitre, nous nous intéressons aux réseaux connexionnistes dédiés à la modélisation de machines d'état fini. Certains de ces réseaux ne sont pas réccurents, par conséquent ils ne peuvent capturer que la dépendance entre la valeur désirée et un nombre limité des entrées passées. Les réseaux récurrents permettent de traiter un ensemble de séquences temporelles beaucoup plus riche. Cependant, certaines études démontrent l'incapacité des algorithmes d'apprentissage fondés sur la descente du gradient à capturer les contingences temporelles présentes dans la séquence d'entrée et qui s'étendent sur de longs intervalles de temps [8, 71]. On trouvera une collection de papiers sur les réseaux récurrents dans [37] et une excellente comparaison expérimentale de différents réseaux récurrents dans [47]. Il a été rapporté [47, 63] que la descente du gradient peut être plus efficace dans les réseaux NARX (Nonlinear AutoRegressive models with eXogenous inputs) que dans les autres architectures récurrentes avec des "états cachés". La classe des machines que ces réseaux sont capables de simuler est appelée "finite memory machine", une sous-classe des machines d'état fini. Un cas particulier de ces réseaux est le TDNN (Time Delay Neural Network). La classe des machines que les TDNN sont capables de simuler est appelée "definite memory machine" [22], une sous-classe des "finite memory machine" [2]. Une étude expérimentale récente montre que les capacités d'apprentissage et les performances en généralisation d'un réseau NARX sont fortement liées à l'ordre de la mémoire à courte terme de ce réseau [62]. Pour déterminer l'ordre optimal de la mémoire d'un réseau NARX, Giles et al. [36] proposent un algorithme, appelé "Delay Damage Algorithm", qui consiste à commencer avec un ordre suffisamment grand et de le simplifier à la fin de l'apprentissage en élagant les ordres de mémoire non pertinents. Cette méthode est basée sur le calcul de la dérivée seconde de la fonction coût par rapport à chaque ordre de mémoire [75]. On peut procéder de la même façon en utilisant HVS comme mesure de pertinence des ordres de mémoire.

Dans la suite de ce chapitre, nous nous intéressons essentiellement à l'apprentissage de machines d'état fini. La connaissance de la classe de la machine que

2. Dans la suite de ce chapitre, on utilisera machines à mémoire fini à la place de "finite memory machine" et machines à mémoire fini sur les entrées à la place de "definite memory machine"

l'on désire apprendre est importante pour le choix de l'architecture initiale. Nous proposons une méthode permettant d'extraire cette information de la base d'apprentissage. Nous donnons tout d'abord quelques notions sur les machines d'état fini. Ensuite nous présentons quelques types de réseaux de neurones dédiés au traitement de séquences temporelles et les liens qui existent entre certains types de ces réseaux et certaines classes de machines d'état fini. Pour finir nous décrivons notre méthode d'extraction de connaissances [100] et nous donnons quelques résultats experimentaux.

5.2 Notions sur les machines d'état fini

Dans cette section, nous donnons quelques notions de base sur les machines d'état fini. Pour plus de détails, voir par exemple [50]. Dans cette étude, nous nous intéressons aux automates d'état fini munis d'une sortie, en plus de l'entrée, définissant des fonctions dites de transduction ou de codage (souvent nommé transducteur). Dans la suite, toutes les machines sont supposées être déterministes.

Les machines d'état fini

Une machine d'état fini est un système abstrait que l'on peut décrire par un graphe cyclique orienté étiqueté, composé d'entrées, des états et de sorties. La machine peut se trouver dans l'un des états qui sont en nombre fini. L'état de la machine résume l'information concernant le passé des entrées nécessaire pour déterminer le comportement de la machine sur les entrées ultérieures. Pour ces machines, il y a égalité entre la longueur de la séquence présentée en entrée et la longueur de la séquence de sortie qui en résulte. A chaque instant, la machine se trouve exactement dans un état et elle ne produit une sortie que lorsqu'elle reçoit une entrée.

Les machines séquentielles

Une machine séquentielle désigne la réalisation matérielle d'une machine d'état fini, constituée de portes logiques et d'éléments de mémoire tels que les délais. Nous verrons qu'il y a des similitudes topologiques entre diverses machines séquentielles et réseaux de neurones temporels. Un exemple de machine séquentielle

est montré sur la Figure 5.1.

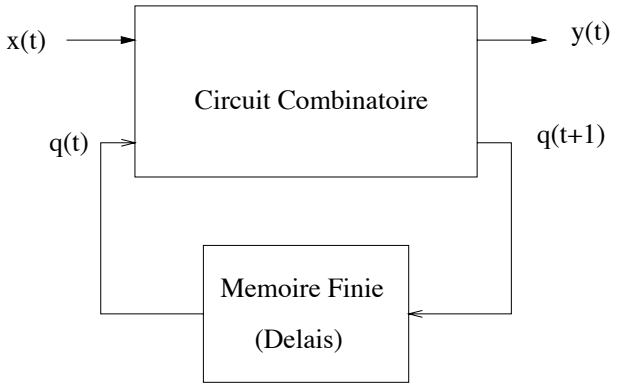


FIG. 5.1 – *Une machine séquentielle*

Les machines à mémoire finie

Les machines à mémoire finie forment une sous-classe des machines d'état fini. Elles sont définies comme suit:

Définition 5.2.1 *Une machine d'état fini M est dite à mémoire finie d'ordre n sur les entrées et d'ordre m sur les sorties si n et m sont les plus petits entiers tels que l'état actuel de M peut toujours être déterminé d'une manière unique à partir des n dernières entrées et m dernières sorties de toute séquence de longueur max(n,m).*

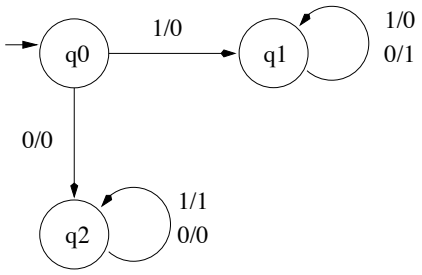


FIG. 5.2 – *Une machine à mémoire finie d'ordre 1 sur les entrées et 1 sur les sorties.*

Remarquez que la définition exclut toute connaissance sur l'état initial de la machine. Par exemple, la machine d'état fini montrée sur la figure 5.2 à un

degré 1 sur les entrées et un degré 1 sur les sorties puisqu'il suffit de connaître la dernière entrée et la dernière sortie de la machine pour déterminer son état actuel comme l'indique la table 5.1

entrée	sortie	
e(t-1)	y(t-1)	état courant
0	0	q_2
0	1	q_1
1	0	q_1
1	1	q_2

TAB. 5.1 – *Etat courant de la machine de la figure 5.2 en fonction de la dernière entrée et de la dernière sortie*

Les machines d'état fini n'ont pas toutes une mémoire finie, certaines ont un ordre infini. Par exemple, la machine d'état fini montrée sur la figure 5.3 à un ordre infini puisqu'on peut observer un nombre infini de 1 en entrée et un nombre infini de 1 en sortie sans être capable de déterminer si la machine est dans l'état q_1 ou q_2 sauf si on connait l'historique complet du processus d'entrée.

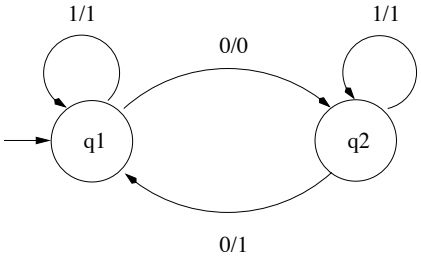


FIG. 5.3 – *Une machine d'état fini ayant un ordre infini*

Comme l'état d'une machine à mémoire finie dépend uniquement d'un nombre fini des entrées précédentes et d'un nombre fini des sorties précédentes, le circuit de la machine séquentielle d'une machine à mémoire finie peut toujours être réalisé par des délais sur les entrées et les sorties et d'un circuit combinatoire comme le montre la figure 5.4.

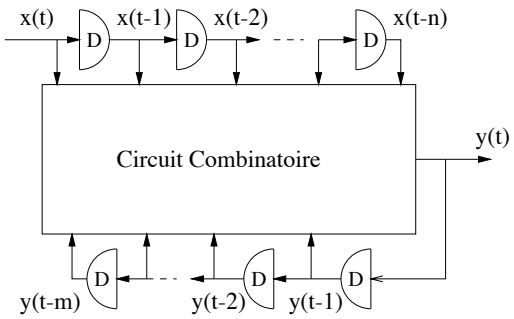


FIG. 5.4 – *Circuit de la machine séquentielle d'une machine à mémoire finie.*

Une machine à mémoire finie d'odre n sur les entrées et d'ordre 0 sur les sorties est dite machine à mémoire fini sur les entrées. Par exemple, la machine montrée sur la figure 5.5 est une machine d'ordre 2 sur les entrées puisqu'il suffit de connaître les deux dernières entrées pour déterminer son état actuel comme le montre la table 5.2.

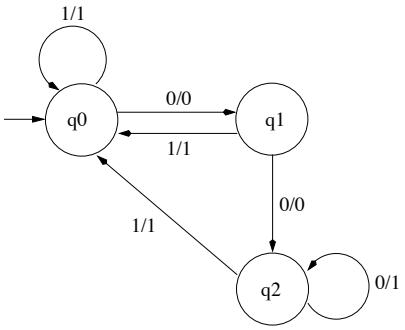


FIG. 5.5 – *Une machine d'ordre 2 sur les entrées*

e(t-2)	e(t-1)	état courant
0	0	q_2
0	1	q_0
1	0	q_1
1	1	q_0

TAB. 5.2 – *Etat courant de la machine de la figure 5.5 en fonction des deux dernières entrées.*

La réalisation de telle machine ne demande pas de récursion du circuit combinatoire comme le montre la figure 5.6.

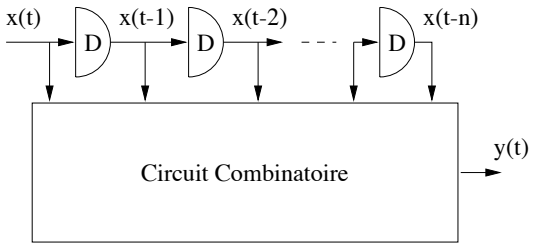


FIG. 5.6 – *Circuit de la machine séquentielle d'une machine à mémoire fini sur les entrées.*

5.3 Les réseaux temporels

Dans cette section, nous présentons quelques réseaux connexionnistes dédiés au traitement de séquences temporelles. Ces réseaux sont dotés d'une mémoire à long terme et d'une mémoire à court terme. La mémoire à long terme est construite dans le réseau durant la phase d'apprentissage, sous forme de poids de connexions. Une fois que le réseau est suffisamment entraîné, cette mémoire reste fixe durant l'utilisation du réseau. La mémoire à court terme est représentée par les valeurs de sortie de certains neurones. Cette mémoire permet au réseau de prendre en compte l'aspect dynamique du problème à traiter. Contrairement à la mémoire à long terme, qui reste fixe une fois l'apprentissage terminé, cette mémoire est recalculée pour chaque nouvelle entrée, que ce soit pendant la phase d'apprentissage ou pendant l'utilisation du réseau. Cette mémoire à court terme permet en quelques sortes de calculer l'état du réseau. On peut trouver une taxonomie développée par Mozer pour ces réseaux dans [70] et une autre développée par Horne et Giles dans [47]. Par exemple la taxonomie développée par Horne et Giles [47] consiste à regrouper, au premier niveau, ces réseaux en deux classes selon que les états du réseau sont observables[3] [55, 73, 93] ou non[4] [3, 27, 31, 38, 77, 82, 94].

– Dans la première classe on trouve les réseaux NARX [73], les TDNN [55] et les réseaux Gamma [93]. Dans le modèle NARX, la sortie du réseau

3. l'état du réseau est donné uniquement par les activations des unités d'entrée et de sortie
4. l'état du réseau est donné par les activations des unités cachées

est calculée par un PMC ayant comme entrées une fenêtre sur les entrées et sorties passées (figure 5.7). En d'autres termes, un réseau NARX est composé d'un PMC et d'une mémoire à court terme qui est formée par des lignes de délai sur les entrées et des lignes de délai sur les sorties.

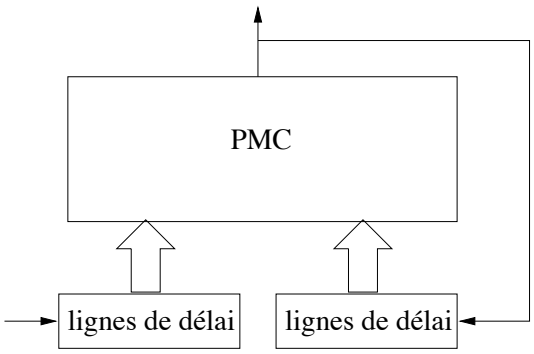


FIG. 5.7 – *Les réseaux NARX*

Remarquez que si on remplace le ciruit combinatoire par un PMC, le circuit de la machine séquentielle d'une machine à mémoire finie, illustrée sur la figure 5.4 devient similaire à réseau NARX, comme rapporté dans [35] par Giles et al..

Un cas particulier de ce réseau est le réseau où la mémoire à court terme n'est représentée que par des lignes de délai sur les entrées (Figure 5.8). Ce type d'architecture est connu sous le nom de TDNN[5]. La figure 5.9 illustre un peu plus en détail la mémoire à court terme d'un réseau TDNN. C'est une mémoire d'ordre d, de résolution 1 et de profondeur[6] d.

5. Dans [22], les auteurs montrent que la classe des machines calculables par les TDNN (retards sur les entrées et couches cachées) et les IDNN, pour Input Delay Neural Networks (retards uniquement sur les entrées) est la même

6. depth en anglais

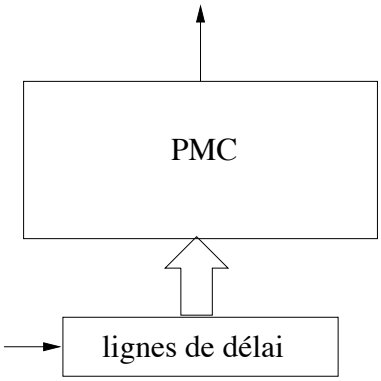


FIG. 5.8 – *Les réseaux à délai.*

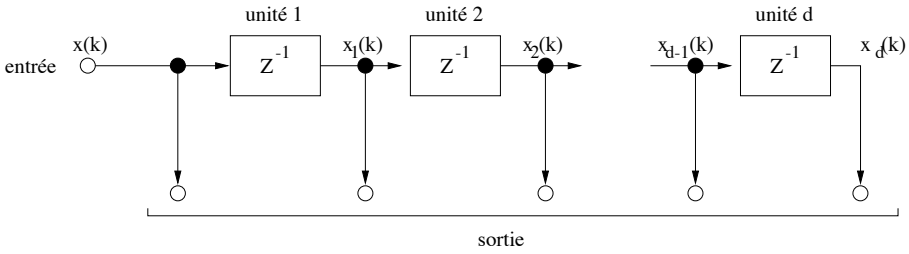


FIG. 5.9 – *mémoire ordinaire d'ordre d, utilisée dans les TDNN.* Z^{-1} *est un opérateur qui lorsqu'il reçoit un signal x(k) donne sa version retardée d'une unité de temps.*

De nouveau, si on remplace le ciruit combinatoire par un PMC, le circuit de la machine séquentielle d'une machine à mémoire finie sur les entrées, illustrée sur la figure 5.6 devient similaire à réseau à délai (figure 5.8).

La mémoire à court terme d'un réseau Gamma est montrée dans la figure 5.10. Cette mémoire a un odre d, une résolution μ et une profondeur $\frac{d}{\mu}$. La profondeur et la résolution de la mémoire à court terme sont donc contrôlées par le paramètre μ ($0 < \mu < 2$), qui est ajustable. Quand $\mu = 1$, on retrouve la mémoire à court terme des TDNN.

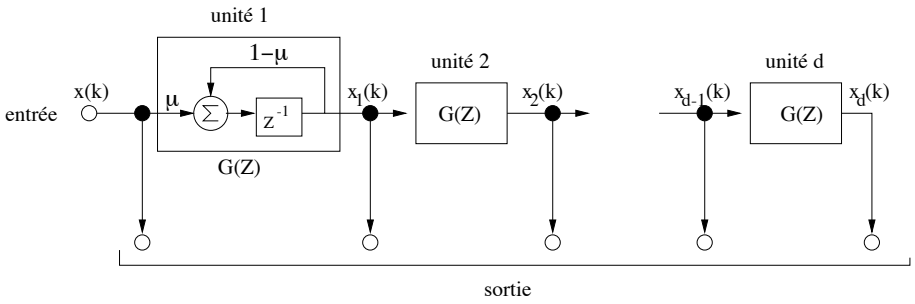


FIG. 5.10 – *Mémoire à court terme d'un réseau Gamma. L'opérateur G(z) est montré plus en détail dans l'unité 1. La sortie de l'unité 1, par exemple, s'obtient comme suit :* $x_1(k+1) = \mu x(k) + (1-\mu)x_1(k)$

– Dans la deuxième classe, le réseau le plus populaire est le réseau d'Elman [27]. Dans ce réseau la mémoire à court terme du réseau est représentée par la couche de contexte, qui contient les valeurs récentes des sorties des neurones de la couche cachée (figure 5.11).

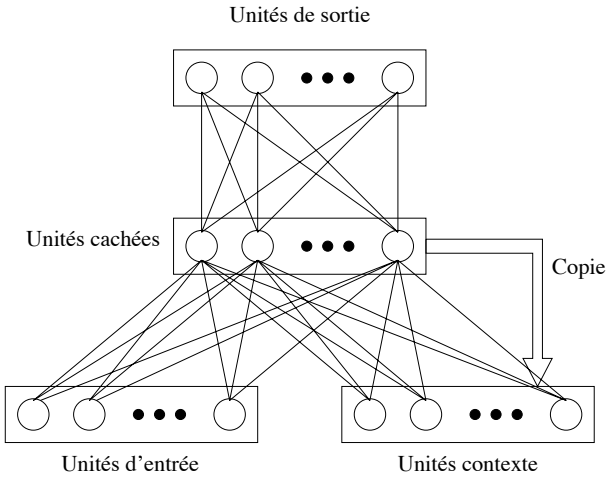


FIG. 5.11 – *Réseau d'Elman.*

Dans [53], Kremer montre que ce réseau peut simuler n'importe quelle machine d'état fini.

5.4 Identification de la classe d'une machine d'état fini

En pratique, il a été rapporté que la descente du gradient peut être plus efficace dans les réseaux NARX que dans les autres architectures récurrentes avec des "états cachés" [47]. Une étude expérimentale récente [62] montre que les capacités d'apprentissage et les performances en généralisation d'un réseau NARX sont fortement liés à l'ordre de la mémoire de ce réseau. Dans cette étude les auteurs ont montré que les capacités d'apprentissage d'un réseau NARX ne possédant pas assez de mémoire peuvent être sérieusement inssufisantes. Par conséquent, la taille de mémoire est essentielle pour un réseau NARX afin d'être entrainé convenablement. Cependant, les performances en généralisation peuvent être faibles si cette mémoire est inutilement large. Une approche possible consiste alors à commencer avec un ordre suffisamment grand pour faciliter l'apprentissage puis élaguer les ordres non pertinents pour avoir de bonnes performances en généralisation. Ce principe est utilisé par Giles et al. dans [36]. Leur méthode est basée sur le calcul de la dérivée seconde de la fonction coût par rapport à chaque ordre de mémoire. On peut procéder de la même façon en utilisant HVS comme mesure de pertinence des ordres de mémoires. Dans cette section, nous allons aborder le problème d'une façon un peu différente. Nous avons vu les liens qui existent entre certains types de réseaux et certaines classes de machines d'état fini. La connaissance de la classe de la machine que l'on désire apprendre est par conséquent utile pour le choix de l'architecture à utiliser pour son apprentissage. Nous proposons une méthode permettant d'extraire cette information de la base d'apprentissage. En d'autres termes, nous nous intéressons à estimer, pour une machine d'état fini donnée, le minimum de connaissances nécessaires sur ces entrées-sorties antérieures afin de rendre son comportement futur complétement prévisible.

Dans la suite, nous dirons qu'une machine est à mémoire finie si son ordre sur les entrées et son ordre sur les sorties sont tous deux inférieurs ou égaux à une certaine valeur d fixée à l'avance. De même, nous dirons qu'elle est à mémoire finie sur les entrées si son ordre sur les entrées est inférieur ou égal à d et son ordre sur les sorties est nul.

Dans les expériences que nous présentons dans la section suivante, nous avons fixé d à 3.

La méthode que nous proposons est basée sur deux modules (figure 5.12). Le module M1 est utilisé principalement pour identifier si cette machine est à mémoire finie sur les entrées et d'estimer son ordre dans le cas d'une réponse positive. Il permet aussi d'identifier si les entrées "futures" interviennent dans le calcul de la sortie actuelle. Le module M2 est utilisé pour identifier si la machine est à mémoire finie ou a un ordre infini.

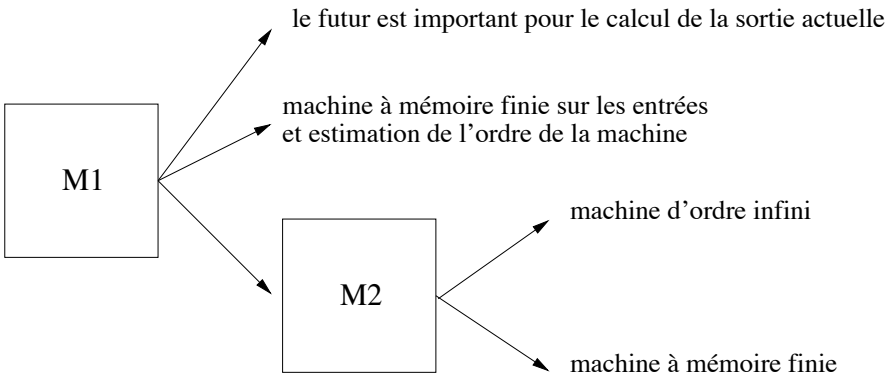


FIG. 5.12 – *Schéma général de la méthode*

L'idée générale de la méthode que nous proposons consiste à étendre l'application de notre mesure de pertinence pour estimer les délais.

Le module M1 utilise un PMC pour calculer la fonction suivante :

$$y(t) = f(x(t+d_2), \cdots, x(t), \cdots, x(t-d), \cdots x(t-d_1)) \tag{5.1}$$

A la fin de la phase d'apprentissage nous appliquons notre mesure de pertinence pour estimer les délais. Nous dirons que la machine est à mémoire finie sur les entrées si les délais sélectionnés sont : $\{x(t-t')/0 \leq t' \leq d\}$. De plus si les délais sélectionnés sont : $x(t-k) \cdots x(t)$ avec $0 \leq k \leq d$ nous approximons l'ordre de la machine par k.

Dans le cas où le module M1 estime que la machine n'est pas à mémoire finie sur les entrées et que le futur n'intervient pas dans le calcul de la sortie actuelle,

nous utilisons le module M2 pour identifier si la machine est à mémoire finie. Le module M2 utilise un PMC pour calculer la fonction suivante :

$$y(t) = f(x(t), \cdots, x(t-d), \cdots x(t-d_1), y(t-1), \cdots, y(t-d), \cdots y(t-d_1)) \quad (5.2)$$

De même que précédement, à la fin de la phase d'apprentissage nous utilisons notre mesure de pertinence pour sélectionner les délais les plus importants. Nous dirons que la machine est à mémoire finie si les délais sélectionnés sont : $\{x(t-t')$ et $y(t-t")/0 \leq t' \leq d, 1 \leq t" \leq d\}$. De plus si le module M2 a sélectionné les délais $x(t-k_1) \cdots x(t)$ et $y(t-k_2) \cdots y(t-1)$ avec $0 \leq k_1 \leq d$ et $1 \leq k_2 \leq d$ nous dirons que la machine est à mémoire finie d'ordre k_1 sur les entrées et d'ordre k_2 sur les sorties.

Comme nous utilisons un PMC (la taille de l'entrée est fixe), nous avons besoin de générer des exemples de même taille. Deux cas se présentent :

1. **la machine est une "boite noire"** : nous supposons que la machine est disponible sous forme d'une boite noire, c'est à dire que nous avons accès à ses terminaux d'entrée et de sortie, sans pouvoir examiner ses composants internes et leurs interconnexions. Nous pouvons ainsi présenter une séquence à son entrée et la machine nous donne la sortie correspondante.

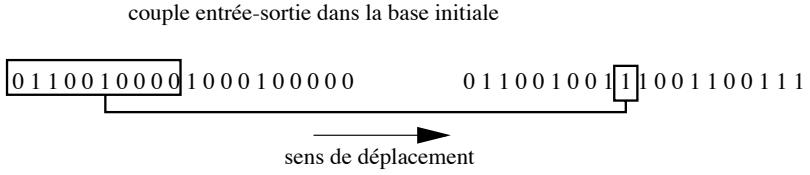
2. **un ensemble d'exemples généré par la machine** : nous supposons que la machine n'est pas disponible, mais nous disposons d'un ensemble d'exemples entrée-sortie générés par cette machine.

Dans le premier cas, pour que la base d'apprentissage contienne des séquences de même taille, il suffit de présenter à cette machine des séquences de même taille. Dans le deuxième cas, la base d'apprentissage peut contenir des séquences de taille différentes, comme le montre la figure 5.13.

base d'apprentissage	
entrée	sortie
01100100001000100000	01100100111001100111
10101010101010101	10101010101010101
00000000000000000000	00111111111111111111
00000000010000000000	00111111110011111111
11111111111111	11111111111111
00100100100100000	00100100100100011
000	001
...	...

FIG. 5.13 – *Base initiale*

Dans ce cas, nous allons générer une autre base contenant des séquences de même taille en faisant glisser une fenêtre sur les séquences de la première base, comme le montre la figure 5.14.



0110010000	1
1100100001	1
1001000010	0
0010000100	0
0100001000	1
1000010001	1
0000100010	0
0001000100	0
0010001000	1
0100010000	1
1000100000	1

FIG. 5.14 – *Séquences engendrées en faisant glisser une fenêtre de taille 10 sur les entrées et 1 sur les sorties*

5.5 Résultats Expérimentaux

Dans cette section, nous rapportons un ensemble de résultats expérimentaux.

Resultats du module M1

Bases utilisées

Nous avons généré quatre bases d'apprentissages selon le principe du premier cas. Nous avons utilisé quatre machines différentes et pour chaque machine nous avons généré une base d'apprentissage composée de séquences ayant la forme suivante :

$$x(1) \cdots x(10) \rightarrow y(6)$$

où $y(1) \cdots y(6) \cdots y(10)$ est la séquence produite en sortie par la machine d'état fini lorsqu'elle lit la séquence $x(1) \cdots x(10)$.

Remarquez que pour calculer $y(t)$ nous utilisons même les valeurs $x(t)$, $t > 6$. Cette façon de faire nous permet aussi d'avoir une idée sur le comportement de notre méthode en présence de valeurs futures, relativement à un instant t donné, selon que la machine utilisée est déterministe ou pas. En d'autres termes, nous voulons voir si notre méthode éliminera les $x(t)$ avec $7 \leq t \leq 10$ selon que la machine est déterministe ou non.

Base 1 cette base est générée en utilisant la machine de la figure 5.5. La séquence de sortie est obtenue à partir de la séquence d'entrée comme suit : la machine produit une sortie 1 chaque fois qu'elle reçoit une entrée 1. Quand la machine reçoit une entrée 0, sa sortie dépend des deux entrées précédentes : si les deux sont des 0 alors la machine produit une sortie 1, sinon elle produit une sortie 0.

Base 2 Cette base de donnée est générée en utilisant la machine de la figure 5.2. La séquence de sortie est obtenue à partir de la séquence d'entrée comme suit : si le premier élément de la séquence en entrée est un 0 alors la machine

produit une sortie 1 chaque fois qu'elle reçoit une entrée 1 et une sortie 0 chaque fois qu'elle reçoit une entrée 0. Autrement, elle produit une sortie 0 chaque fois qu'elle reçoit une entrée 1 et une sortie 1 chaque fois qu'elle reçoit une entrée 0.

Base 3 Cette base de donnée est générée en utilisant la machine de la figure 5.3. La séquence de sortie est obtenue à partir de la séquence d'entrée comme suit : la machine produit une sortie 1 chaque fois qu'elle reçoit une entrée 1. Quand la machine reçoit une entrée 0, sa sortie dépend du nombre de 0 qui se sont présentés avant l'entrée courante : s'il est pair alors elle produit un 1, sinon elle produit un 0.

Base 4 Cette base d'apprentissage est générée en utilisant la machine de la Figure 5.15.

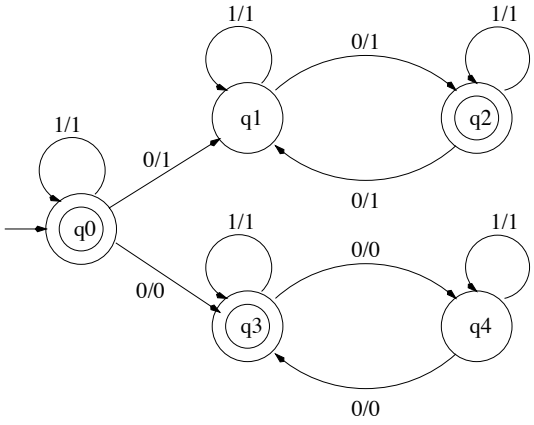


FIG. 5.15 – *Une machine d'état fini non déterministe.*

En utilisant cette machine, la séquence de sortie est obtenue à partir de la séquence d'entrée comme suit : la machine produit une sortie 1 chaque fois qu'elle reçoit une entrée 1. Lorsque la machine reçoit une entrée 0, sa sortie dépend de la parité du nombre de 0 présents dans la séquence d'entrée. S'il est pair alors la machine produit un 1, sinon elle produit un 0. La fonction codée par cette machine n'est pas séquentielle: quand une entrée 0 se produit, la sortie ne peut être déterminée que si on connait la totalité de la séquence d'entrée.

Résultats

Pour chaque base, nous avons procédé à l'apprentissage de la sortie $y(6)$ en utilisant les entrées $x(1) \cdots x(10)$. Nous avons utilisé des architectures $< 10|2|1 >$ pour les bases 1 et 2 et $< 10|4|1 >$ pour les bases 3 et 4. A la fin de la phase d'apprentissage nous avons utilisé la mesure HVS pour calculer l'importance des neurones d'entrée, les résultats sont les suivants :

- **Base 1 (Cas d'une machine à mémoire finie sur les entrées)**

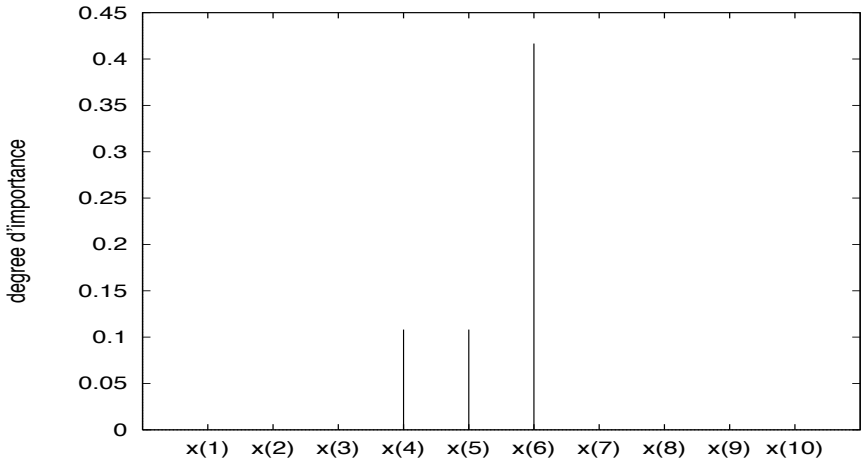


FIG. 5.16 – *Importance des entrées $x(1), \cdots x(10)$ pour déterminer la sortie y(6), selon la mesure HVS. Les exemples de la base d'apprentissage sont générés en utilisant la machine à mémoire finie sur les entrées de la figure 5.5*

Comme le montre la figure 5.16, on peut constater que seules les entrées x(4), x(5) et x(6) sont importantes pour le calcul de y(6). Ce résultat confirme la propriété de la machine étudiée : machine à mémoire finie d'ordre 2 sur les entrées. Pour cette machine, l'architecture initiale que nous choisirons pour son apprentissage est de type TDNN avec une mémoire à court terme d'ordre 2.

• **Base 2 (Cas d'une machine à mémoire finie)**

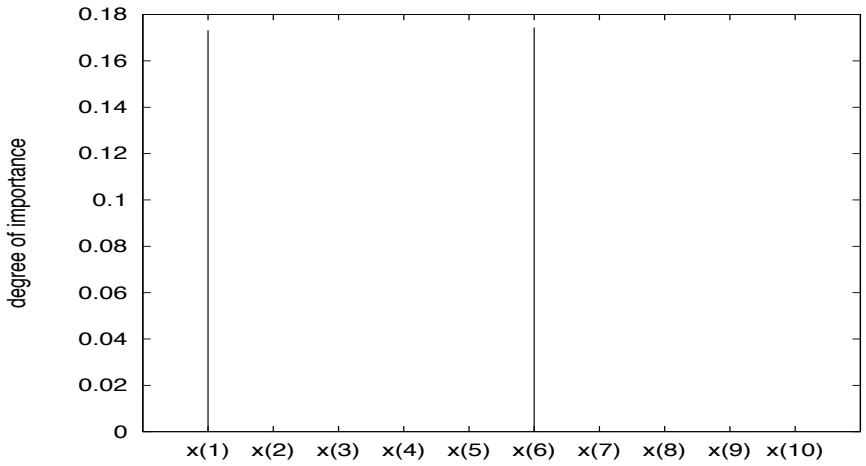


FIG. 5.17 – *Importance des entrées* $x(1), \cdots x(10)$ *pour déterminer la sortie y(6), selon la mesure HVS. Les exemples de la base d'apprentissage sont générés en utilisant la machine à mémoire finie (figure 5.2)*

Comme le montre la figure 5.17, on peut constater que les entrées importantes sont x(1) et x(6). Cette machine n'est donc pas une machine à mémoire finie sur les entrées, selon la définition que nous avons adoptée. Il reste à identifier si elle est une machine à mémoire finie en utilisant le module M2.

• **Base 3 (Cas d'une machine d'ordre infini)**

Comme le montre la figure 5.18, on peut constater que les entrées importantes sont $x(1), \cdots, x(6)$. De même que précédemment, cette machine n'est donc pas une machine à mémoire finie sur les entrées, selon la définition que nous avons adoptée. Il reste à identifier si elle est une machine à mémoire finie en utilisant le module M2.

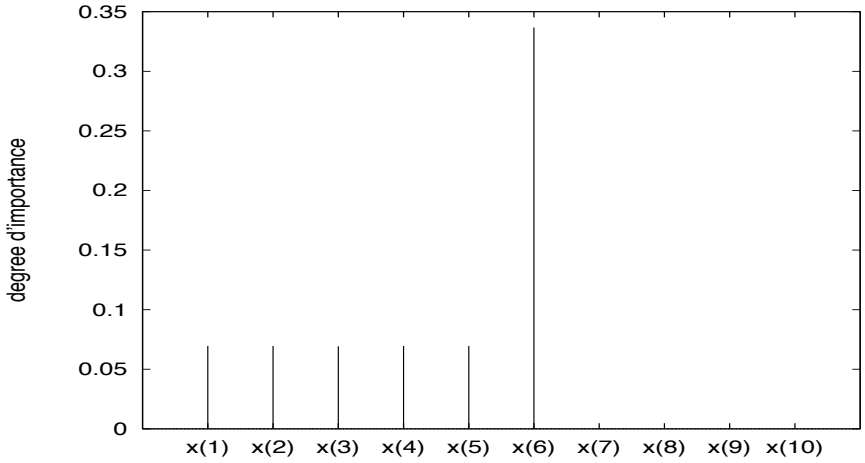


FIG. 5.18 – *Importance des entrées $x(1), \cdots x(10)$ pour déterminer la sortie y(6), selon la mesure HVS. Les exemples de la base d'apprentissage sont générés en utilisant la machine d'ordre infini (figure 5.3)*

• **Base 4 (Cas d'une machine non déterministe)**

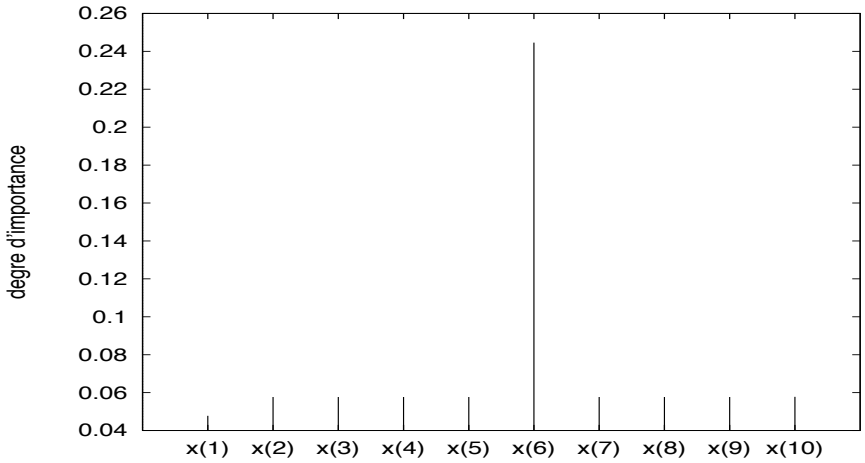


FIG. 5.19 – *Importance des entrées $x(1) \cdots x(10)$ pour déterminer la sortie y(6), selon la mesure HVS. Les exemples de la base d'apprentissage sont générés en utilisant la machine non déterministe de la figure 5.15*

Comme le montre la figure 5.19, on peut constater que toutes les entrées sont importantes, y compris les entrées "futures" relativement à x(6). Ce résultat

confirme la propriété de la machine étudiée : machine non déterministe.

Conclusion partielle

HVS a identifié que

- la machine 1 est à mémoire finie d'ordre 2 sur les entrées
- pour les machines 2 et 3, seules les entrées présentées dans le passé sont nécessaires pour produire correctement la sortie actuelle
- pour la machine 4, les entrées "futures" interviennent dans le calcul de la sortie actuelle (machine non déterministe).

D'après ces résultats, le module M2 est concerné par les 2 machines utilisées pour générer les bases 2 et 3.

Resultats du module M2

Nous avons généré deux bases d'apprentissage en utilisant les machines 2 et 3. Chaque base d'apprentissage est composée de séquences qui sont de la forme suivante :

$$x(1)\cdots x(6)y(1)\cdots y(5) \rightarrow y(6)$$

où $y(1)\cdots y(6)$ est la séquence produite en sortie par la machine lorsqu'elle lit la séquence $x(1)\cdots x(6)$.

Pour chaque base d'apprentissage, nous avons procédé à l'apprentissage de la sortie $y(6)$ en utilisant les entrées $x(2)\cdots x(6)$ et $y(2)\cdots y(5)$. A la fin de la phase d'apprentissage nous avons calculé les importances des neurones d'entrée selon la mesure HVS.

Résultats

• **cas d'une machine à mémoire finie (utilisée pour générer la base 2)**
Comme le montre la figure 5.20, HVS estime que les variables importantes sont

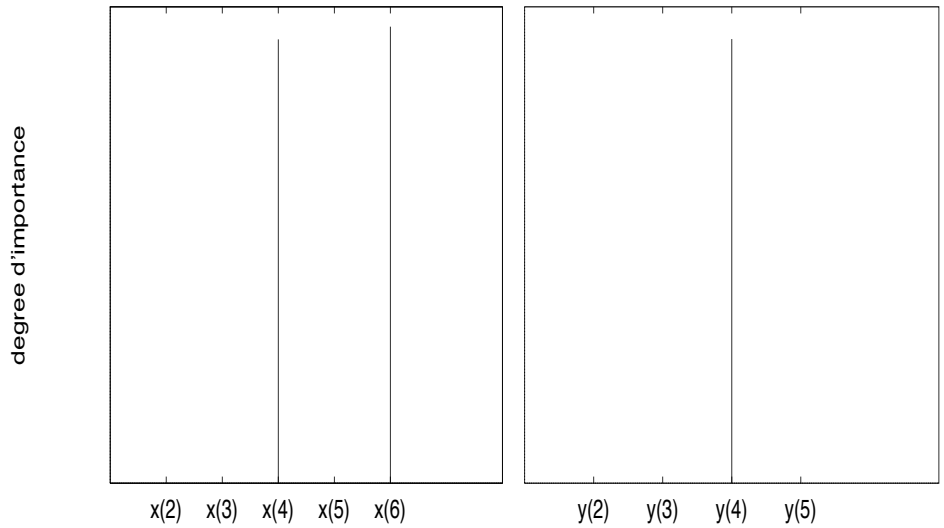


FIG. 5.20 – *Importance des entrées* $x(1), \cdots x(6)$ *et* $y(1), \cdots y(5)$ *pour déterminer la sortie y(6), selon la mesure HVS. Les exemples de la base d'apprentissage sont générés en utilisant la machine à mémoire finie (figure 5.2)*

$x(4)$, $x(6)$ et $y(4)$.

Dans un premier temps, ce résultat est un peu surprenant car on s'attendait plutôt à ce que HVS sélectionne les variables $x(5)$, $x(6)$ et $y(5)$ puisqu'il s'agit d'une machine à mémoire fini d'ordre 1 sur les entrées et 1 sur les sorties. Pour comprendre ce qui s'est passé reprenant de nouveau cette machine (figure 5.21).

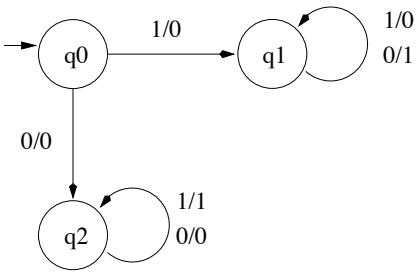


FIG. 5.21 – *Une machine à mémoire finie d'ordre 1 sur les entrées et 1 sur les sorties.*

Remarquez que parmi les trois états de la machine, q_0 est un état "transient", c'est à dire quand la machine est dans cet état elle le quitte imédiatement et n'y

revient jamais. Les état q_1 et q_2 sont des états absorbants, c'est à dire quand la machine entre dans l'un de ces états elle ne le quitte jamais. En d'autres termes, une fois que la machine a commencé à fonctionner, elle ne peut se trouver que dans l'état q_1 lequel elle ne quittera jamais ou dans l'état q_2 lequel elle ne quittera jamais. De plus, remarquez que lorsque la machine est dans l'état q_1 les transitions sont toutes de la forme $x(t)/y(t)$ avec $x(t) \neq y(t)$ et lorsqu'elle est dans l'état q_2, les transitions sont toutes de la forme $x(t)/y(t)$ avec $x(t) = y(t)$. Il suffit donc de connaître une transition quelconque qui s'est produite dans le passé pour déterminer avec exactitude si la machine est dans l'état q_1 ou dans l'état q_2. Par conséquent la mesure HVS a sélectionné la transition qui s'est produite au temps $t-2$, relativement à $t = 6$ et l'entrée actuelle $x(6)$. En fait la transition $x(4)/y(4)$ lui permet d'identifier l'état courant, et une fois cet état identifié l'entrée actuelle lui permet d'identifier la sortie actuelle.

Nous avons fait d'autres expériences parmi lesquelles HVS a sélectionné les variables $x(5)$, $x(6)$ et $y(5)$. Cependant, cette remarque nous fait penser à la méthode proposée par Giles et al. [36], qui consiste à commencer avec un ordre suffisamment grand. Il serait intéressant de tester cette méthode sur cette machine afin de voir quelle transition elle va sélectionner. Une alternative pour régler ce problème serait de contraindre la méthode de sélection de délais à avantager d'abord les délais récents. Une autre alternative serait de commencer plutôt par un ordre minimal et de le faire croitre si nécessaire. Cela nous fait penser à la méthode de weigend et al.[7] [96], où ils sélectionnent les délais en se basant sur les performances du réseau en faisant croitre le nombre de délais utilisés.

• **cas d'une machine d'ordre infini (utilisée pour générer la base 3)** Pour cette machine, nous n'avons réussi à éliminer aucun délai en entrée. Nous avons donc conclu qu'elle n'est pas à mémoire finie selon la définition que nous avions adoptée.

7. A la page 201 de [96], on peut lire :"To estimate the embedding in the presence of noise, we compared networks with 6, 12 and 25 inputs units. The performance on the training set improved by increasing the number of input units from 6 to 12. Further increase from 12 to 25 input units did not lead to significant further improvement".

5.6 Conclusion

Dans ce chapitre, nous nous sommes intéressés à l'apprentissage de machines d'état fini. Nous avons étendu l'application de la mesure HVS pour identifier les délais adéquats pour apprendre une machine donnée. Plus précisemment, nous avons proposé une méthode permettant d'identifier la classe de cette machine, qui est une information très utile pour le choix de l'architecture à utiliser pour son apprentissage. Nous l'avons tester sur différentes classes de machines. Les résultats obtenus sont très encourageants.

Conclusion et perspectives

Nous nous sommes intéressés dans cette thèse au problème du contrôle de la capacité de généralisation des systèmes d'apprentissage connexionnistes. Nous avons étudié principalement des modèles de type perceptron multicouche.

Nous avons d'abord abordé le problème délicat de la sélection de variables. Nous avons proposé une nouvelle mesure heuristique, que nous avons appelée HVS, permettant d'estimer l'importance de chaque variable. Afin de tester l'efficacité de la mesure HVS, nous avons vérifié ses estimations sur des problèmes dont l'importance théorique de chaque variable est connue. Ensuite nous l'avons utilisé pour la sélection de variables. Nous avons testé son efficacité sur un problème de discrimination relativement difficile et sur un problème réel dont la base de données est très courte, il s'agit de la célèbre série chronologique "sunspot". Pour chacun de ces problèmes, nous avons comparé nos résultats à ceux obtenus par d'autres chercheurs en utilisant d'autres méthodes. Les résultats de ces comparaisons nous ont permis de conclure que HVS est une mesure très efficace. Une autre caractéristique de HVS est qu'elle ne demande que peu de calculs simples.

Nous avons ensuite abordé le problème d'ajustement d'architecture. Nous avons proposé une méthode, basée sur la mesure HVS, permettant de simplifier le réseau en élagant les neurones cachés inutiles. En effet, la mesure HVS n'estime pas uniquement l'importance des variables, mais également l'importance de toutes les unités du réseau. Nous avons donc étendu son application aux neurones cachés. De plus, par définition de la mesure HVS, pour estimer l'importance des unités d'une couche du réseau, on n'utilise que les paramètres du sous-réseau composé de cette couche, de la couche de sortie, et des couches qui sont entre les deux. Par conséquent notre méthode d'optimisation d'architecture demande

moins de calculs que notre méthode de sélection de variables. Nous avons testé son efficacité aussi bien sur un problème de discrimination que sur un problème de régression. En fait, nous avons continué nos tests sur les mêmes problèmes que nous avons évoqués lors de la sélection de variables. Les résultats de nos expériences montrent que cette façon de faire nous a permis non seulement d'obtenir de très bons résultats de prédiction ou de discrimination en comparaison avec d'autres résultats obtenus par d'autres chercheurs, mais aussi de réduire considérablement la complexité effective du système.

Ainsi, pour les perceptrons multicouche, nous avons mis au point une nouvelle mesure heuristique permettant d'estimer l'importance de chaque unité du modèle et nous avons montré son efficacité aussi bien pour la sélection de variables que pour l'optimisation d'architecture. Nous avons voulu voir si l'application de cette mesure pouvait être étendue à d'autres types de réseaux.

Une direction de recherche que nous avons commencée à explorer concerne les architectures incluant des connaissances a priori sur le problème, très utilisées dans le domaine du traitement des images. Notre objectif est d'une part de simplifier ces architectures et d'autre part de sélectionner les zones discriminantes les plus pertinentes. L'idée que nous avons développée ne demande pas beaucoup de calculs relativement à la taille de ce type d'architectures. En effet, d'une part, pour sélectionner les traits pertinents, la mesure HVS ne fait intervenir que les paramètres du module de classement. D'autre part, l'étape qui consiste à sélectionner les zones discriminantes ne demande aucun calcul, on ne fait que rétropropager l'effet de l'élagage des traits jugés non pertinents selon la mesure HVS. Dans un premier temps, nous avons voulu tester cette méthode sur des exemples dont on dispose d'une certaine connaissance sur les zones importantes. Nous avons choisi de travailler sur la reconnaissace de visages parce que les résultats rapportés par certains chercheurs montrent que dans ce domaine, les régions les plus importantes sont les yeux, le nez, la bouche, le menton et les cheveux. Nous avons commencé à travailler sur une petite base de quatre personnes. Les résultats rapportés montrent qu'il est intéressant de poursuivre cette direction de recherche. Nous avons supprimé presque la moitié des unités de la rétine sans

baisser les performances du système.

Nous n'avons pas approfondi notre étude sur cette base, nous avons voulu passer à une base plus grande et utilisée par d'autres chercheurs. L'intêret est de pouvoir mieux analyser notre méthode en comparant nos résultats avec ceux obtenus par d'autres chercheurs. Dans l'immédiat, nous pensons travailler sur la base ORL. Ensuite, l'objectif suivant consiste à aborder un problème dont on ne dipose que de peu d'informations sur les zones discriminantes. Nous pensons particulièrement travailler sur le problème de discrimination de signaux sonars, qui consiste à détecter et identifier un événement causé par un bruiteur sous-marin ou de surface.

Nous nous sommes aussi intéressés aux réseaux temporels. Nous avons limité notre étude à l'apprentissage de machines d'état fini. Nous avons proposé une méthode permettant d'identifier la classe de la machine à apprendre. Cette information est utile pour le choix de l'architecture initiale. Nous l'avons testée sur différentes classes de machine. Les résultats préliminaires que nous avons rapportés sont encourageants.

Publications

1. **M. Yacoub & Y. Bennani** Discriminative Feature Extraction and Selection applied to Face Recognition. Accepté à International Joint Conference on Neural Networks (IJCNN'99).
2. **M. Yacoub & Y. Bennani** Architecture optimization in feedforward connectionist models. Proc. IEEE International Conference on Artificial Neural Networks (ICANN'98), Skövde, Suède, pp. 881-886, 1998.
3. **M. Yacoub & Y. Bennani** A Neural Network Methodology for Machine's Class Identification. Proc. IEEE International Joint Conference on Neural Networks (IJCNN'98), Anchorage, Alaska, USA, pp. 322-325, 1998.
4. **M. Yacoub & Y. Bennani** HVS: A Heuristic for Variable Selection in Multilayer Artificial Neural Network Classifier. In Intelligent Engineering Systems Through Artificial Neural Networks, Vol 7: C. Dagli, M. Akay, O. Ersoy, B. Fernandez and A. Smith (Editors), St Louis, Missouri, USA, pp. 527-532, 1997.
5. **M. Yacoub & Y. Bennani** Features Selection and Architectures Optimization in Connectionist Systems. Soumis à International Journal on Neural Systems.
6. **M. Yacoub** Normalization of Representations in the Fibonacci Numeration System using Neural Networks. Inter. Conf. on Neural Information Processing (ICONIP'94), Seoul, Corée, Vol. 2, pp. 1347-1350, 1994.
7. **M. Yacoub** Recurent Neural Networks and Fibonacci Numeration System of order s ($s \geq 2$). Proc. of IEEE Inter. Conf. on Neural Networks (ICNN'94), Orlando, Floride, USA, Vol. 3, pp. 1503-1506, 1994.
8. **M. Yacoub & A. Saoudi** Recurrent Neural Networks and Fibonacci Numeration System. Proc. of the Inter. Joint Conf. on Neural Networks (IJCNN'93), Nagoya, Japon, Vol. 3, pp 2331-2334, 1993.

Bibliographie

[1] D.H. Ackley, G.E. Hinton, and T.J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

[2] T. Artieres, Y. Bennani, P. Gallinari, and Montacie C. Connectionist and conventional models for free-text talker identification tasks. *Neuro-Nimes*, 1991.

[3] A.D. Back and A.C. Tsoi. FIR and IIR synapses, a new neural network architecture for time series modelling. *Neural Computation*, 3(3):375–385, 1991.

[4] P.L. Bartlett. Vapnik-chervonenskis dimension bounds for two and three layer networks. *Neural Computation*, 5(3):371–373, 1993.

[5] P.L. Bartlett and R.C. Williamson. The vc dimension and pseudodimension of two-layer neural networks with discrete inputs. *Neural Computation*, 8(3):625–628, 1996.

[6] R. Battiti. First and second order methods for learning : between steepest descent and newton's method. *Neural Computation*, 4:141–146, 1992.

[7] E.B. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation*, 1(1):151–160, 1989.

[8] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[9] Y. Bennani. Approches connexionnistes pour la reconnaissance du locuteur : modélisation et identification. *Thèse de doctorat, Université Paris-Sud*, 1992.

[10] Y. Bennani and F. Bossaert. A neural network based variable selector. *ANNIE*, 5:425–430, 1995.

[11] Y. Bennani, N. Chaourar, P. Gallinari, and A. Mellouk. Validation of neural net architectures on speech recognition tasks. *Proc. of ICASSP*, pages 97–100, 1991.

[12] Y. Bennani and P. Gallinari. Task decomposition through a modular connectionist architecture : a talker identification system. *IEEE International Conference on Artificial Neural Networks*, 1992.

[13] C. Bishop. Training with noise is equivalent to tikhonov regularisation. *Neural Computation*, 7(1):108–116, 1995.

[14] M. de Bollivier, P. Gallinari, and S. Thiria. Cooperation of neural nets for robust classification. *IJCNN'90*, 1:113–120, 1990.

[15] M. de Bollivier, P. Gallinari, and S. Thiria. Cooperation of neural nets and task decomposition. *IJCNN'91*, 2:573–576, 1991.

[16] L. Breiman, J. Freidman, R. Olshen, and C. Stone. Classification and regression trees. *Wadsworth Int. Group.*, 1984.

[17] R. Brunelli and T. Poggio. Hyperbf networks for gender classification. *Proceedings DARPA Image Understanding Workshop*, pages 311–314, 1992.

[18] A. Bryson, W. Denham, and S. Dreyfuss. Optimal programming problem with inequality constraints. *I: Necessary conditions for extermal solutions. AIAA Journal.*, 1:25–44, 1963.

[19] Y. Chauvin. Dynamic behavior of constrained back-propagation networks. *Advances in Neural Information Processing*, 2:642–649, 1990.

[20] T. Cibas, F. Fogelman-Soulie, P. Gallinari, and S. Raudys. Variable selection with optimal cell damage. *ICANN'94*, 1:727–730, 1994.

[21] T. Cibas, P. Gallinari, and O. Gascuel. Experimental investigation on the complexity-performance relations in multilayer perceptrons. *ICANN'95*, 1:569–574, 1995.

[22] D.S. Clouse, C.L. Giles, B.G. Horne, and G.W. Cottrell. Time-delay neural networks: Representation and induction of finite state machines. *IEEE Transactions on Neural Networks*, 8(5):1065, 1997.

[23] G. Cybenko. Approximation by superpositions of sigmoidal functions. *Mathematics of Control, Signal and Systems*, 2(4):303–314, 1989.

[24] T. Czernichow and A. Munoz. Variable selection through statistical sensivity analysis: Application to feedforward and recurrent neural networks. *Technical Report*, 1995.

[25] D. DeMers and G. W. Cottrel. Non-linear dimensionality reduction. In S.J. Hanson, J.D. Cowan, and C.L. Giles, editors, *Advances in Neural Information Processing Systems 4*, volume 5, pages 580–587, San Mateo, CA, 1992. Morgan Kaufmann Publishers.

[26] R.O. Duda and P.E. Hart. *Pattern classification and scene analysis*. John Wiley et Sons, Cambridge, UK, 1973.

[27] J.L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[28] S.E. Fahlman and C. Lebiere. The cascade-correlation learning algorithm. *Advances in Neural Information Processing Systems*, 2:525–532, 1990.

[29] M.K. Fleming and G.W. Cottrel. Categorization of faces using unsupervised feature extraction. *International Joint Conference on Neural Networks*, 2, 1990.

[30] F. Fogelman Soulié, B. Lamy, and E. Viennet. Multi-modular neural networks architectures for pattern recognition: Applications in Optical Characters Recognition and Human Face Recognition. *Int. J. Pattern Recognition and Artificial Intelligence*, 7(4), 1993. Extended version as Tech. Report 827, LRI 1993.

[31] P. Frasconi, M. Gori, and G. Soda. Local feedback multilayered networks. *Neural Computation*, 4:120–130, 1992.

[32] M. Frean. The upstart algorithm : A method for constructing and training feedforward neural networks. *Neural Computation*, pages 198–209, 1991.

[33] K.I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.

[34] P. Gallinari. Heuristiques pour la généralisation. *Chapitre 14, Statistique et Méthodes Neuronales, Thiria S., Lechevallier Y., Gascuel O. Cannu S., Eds. Dunod, Paris.*, 1997.

[35] C.L. Giles, B.G. Horne, and T. Lin. Learning a class of large finite state machines with recurrent neural networks. *Tech. Rep. UMIACS-TR-94-94 and CS-TR-3328, Institute for Advanced ComputerStudies, University of Maryland, College Park, Maryland*, 1994.

[36] C.L. Giles, T. Lin B.G. Horne, , and S.Y. Kung. The past is important : a method for determining memory structure in narx neural networks. *IEEE International Joint Conference on Neural Networks*, pages 1834–1839, 1998.

[37] C.L. Giles, G.M. Kuhn, and R.J. Williams. Dynamic recurrent neural networks: Theory and applications. *IEEE Transactions on Neural Networks*, 5(2), 1994. Special Issue.

[38] C.L. Giles, C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun, and Y.C. Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, 1992.

[39] B.A. Golomb, D.T. Lawrence, and T.J. Sejnowski. Sexnet: a neural network identifies sex from human face. *Advances in Neural Information Processing*, 3:572–577, 1991.

[40] C. Goutte. On the use of pruning prior for neural networks. *Neural Network for Signal Processing*, VI:52–61, 1996.

[41] Y. Grandvallet. Injection de bruit dans les perceptrons multicouches. *Thèse de Doctorat de l'Université de Technologie de Compiègne*, 1995.

[42] S. Hashem. Sensitivity analysis for feedforward artificial neural networks with differentiable activation functions. *International Joint Conference on Neural Networks*, I:419–424, 1992.

[43] B. Hassibi and D.G. Stork. Second order derivatives for networks pruning: Optimal brain surgeon. *Advances in Neural Information Processing Systems*, 5:164–171, 1993.

[44] R. Hecht-Nielsen. Kolmogorov's mapping neural network existence theorem. *Proc. of the IEEE 1st. Conf. on Neural Networks*, 3:11–14, 1987.

[45] J. Hérault and C. Jutten. *Réseaux neuronaux et traitement du signal.* Editions Hermes, 1994.

[46] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *P.N.A.S. USA*, 79:2554–2558, 1982.

[47] B.G. Horne and C.L. Giles. An experimental comparison of recurrent neural networks. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 697–704. MIT Press, 1995.

[48] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedformard networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[49] A. Jain and W. Waller. On the optimal number of features in the classification of multivariate gaussian data. *Pattern Recognition*, 10:365–374, 1978.

[50] Z. Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill, Inc, NY,, 1978.

[51] T. Kohonen. *Self organization and associative memory*, volume 8 of *Springer series in information sciences.* Springer Verlag, 1984.

[52] A. N. Kolmogorov. Three approaches to the quantitative definitions of information. *Problems of Information Transmission*, 1(1):1–7, 1965.

[53] S.C. Kremer. On the computational power of elman-style recurrent networks. *IEEE Transactions on Neural Networks*, 6(4):1000–1004, 1995.

[54] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. *Advances in Neural Information Processing Systems*, 4:950–957, 1992.

[55] K. J. Lang, A. H. Waibel, and G. E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3(1):23–44, 1990.

[56] S. Lawrence, C. Lee Giles, A.C. Tsoi, and A.D. Back. Face recognition: A convolutional neural network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, 1997.

[57] Y. Le Cun. A learning scheme for asymmetric threshold networks. *Proc. Cognitiva 85, CESTA, Paris*, pages 599–604, 1985.

[58] Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R. Howard, W. Hubbard, and L.D. Jackel. Back-propagation applied to handwritten zipcode recognition. *Neural Computation*, 1:541–551, 1989.

[59] Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R. Howard, W. Hubbard, and L.D. Jackel. Handwritten digit recognition with a back-propagation network. *Advances in Neural Information Processing Systems*, 2:396–404, 1990.

[60] Y. Le Cun, J.S. Denker, and S.A. Solla. Optimal brain damage. *Advances in Neural Information Processing Systems*, 2:598–605, 1990.

[61] P. Leray and P. Gallinari. Feature selection with neural networks. *to appear in behaviormetrika, special issue on Analysis of Knowledge representations in Neural Networks Models*, 1998.

[62] T. Lin, B.G. Horne, C.L. Giles, and S.Y. Kung. What to rember: how memory order affects the performance of narx neural networks. *IEEE International Joint Conference on Neural Networks*, pages 1051–1056, 1998.

[63] T. Lin, B.G. Horne, P. Tino, and C.L. Giles. Learning long-term dependencies in narx recurrent neural networks. *IEEE Transactions on Neural Networks*, 7(6):1329–1338, 1996.

[64] W. Maas. Neural nets with superlinear vc-dimension. *Neural Computation*, 6(5):877–884, 1994.

[65] K. Matsuoka. Noise injection into inputs in back-propagation learning. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(3):436–440, 1992.

[66] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.*, 5:115–133, 1943.

[67] M. Mézard and J.P. Nadal. Learning in feedforward layered networks : the tiling algorithm. *J. Phys. A : Math. Gen.*, 22:2191–2203, 1989.

[68] M. Minsky and S. Papert. Perceptrons. *MIT Press, Cambridge*, 1969.

[69] J. Moody. *Prediction Risk and Architecture Selection for Neural Networks - Theory and Pattern Recognition Application.* Spring-Verglas, 1994.

[70] M. C. Mozer. Neural net architectures for temporal sequence processing. In A.S. Weigend and N.A. Gershenfeld, editors, *Time Series Prediction*, pages 243–264. Addison–Wesley, 1994.

[71] M.C. Mozer. Induction of multiscale temporal structure. In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 275–282, San Mateo, CA, 1992. Morgan Kaufmann Publishers.

[72] K. S. Narenda and K. Parthasarathy. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, 26(9):917–922, 1977.

[73] K. S. Narenda and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Trans. on Neural Networks*, 1(1):4–27, 1990.

[74] S. J. Nowlan and G. E. Hinton. Simplifying neural networks by soft weights sharing. *Neural Computation*, 4:473–493, 1992.

[75] M.W. Pedersen and L.K. Hansen. Recurrent networks: Second order properties and pruning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7*. MIT Press, 1995.

[76] D.C. Plaut, S.J. Nowlan, and G.E. Hinton. Experiments on learning by backpropagation. *Tech. Rep. CMU-CS-86-126, Carnegie Mellon University, Pittsburg,USA*, 1986.

[77] P. Poddar and K.P. Unnikrishnan. Nonlinear prediction of speech signals using memory neuron networks. In B.H. Juang, S.Y. Kung, and C A. Camm, editors, *Neural Networks for Signal Processing: Proceedings of the 1991 IEEE Workshop*, pages 395–404, Piscataway, NJ, 1991. IEEE Press.

[78] S. Raudys. On dimensionality, learning sample size and complexity of classification algorithms. *Int. Conf. on Pattern Recognition*, pages 166–169, 1976.

[79] S. Raudys. A negative weight decay or antiregularization. *ICANN'95*, pages 449–454, 1995.

[80] R. Reed. Pruning algorithms a survey. *IEEE Trans. Neural Networks*, 4(5):740–747, 1993.

[81] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[82] A.J. Robinson and F. Fallside. Static and dynamic error propagation networks with application to speech coding. In D.Z. Anderson, editor, *Neural Information Processing Systems*, pages 632–641, New York, NY, 1988. American Institute of Physics.

[83] F. Rosenblatt. Prindiples of neurodynamics: perceptrons and the theory of brain mechanisms. *Spartan Books, Washington*, 1961.

[84] F. Rossi. Attribute suppression with multi-layer perceptron. *Proceedings of IEEE IMACS'96*, 1996.

[85] D.W. Ruck, S.K. Rogers, and M. Kabrisky. Feature selection using a multilayer perceptron. *Neural Network Comput.*, 2(2):40–48, 1990.

[86] D. Rumelhart, G. Hinton, and R. William. Learning internal representation by error propagation. *Parallel Distributed Processing: Explorations in the Micro-structure of Cognition*, 1, 1986.

[87] C. Svarer, L.K. Hansen, and J. Larsen. On design and evaluation of tapped-delay neural networks architectures. *IEEE International Conference on Neural Networks*, pages 46–51, 1993.

[88] M. L. Thompson. Selection of variables in multiple regression. *Part I: A review and evaluation Part II: Computation and examples. International Statistical Review*, 46:1–19 and 129–146, 1978.

[89] V. Tresp, R. Neuneier, and G. Zimmermann. Early brain damage. *Neural Information Processing Systems*, 9:669–675, 1997.

[90] V.N. Vapnik. Principles of risk minimisation for learning theory. *in Advances in neural Information Processing Systems*, 4:831–840, 1992.

[91] V.N. Vapnik. *The nature of statistical learning theory.* Spring-Verglas, 1995.

[92] E. Viennet. Architectures connexionnistes multi-modulaires application à l'analyse de scène. *Thèse de doctorat, Université Paris-Sud*, 1993.

[93] B. de Vries and J. Principe. The gamma model - a new neural network for temporal processing. *Neural Networks*, 5(4):565–576, 1992.

[94] R.L. Watrous and G.M. Kuhn. Induction of finite state automata using second-order recurrent networks. In J.E. Moody, S.J. Hanson, and R.P Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 309–316, San Mateo, CA, 1992. Morgan Kaufmann Publishers.

[95] A. Weigend, D. Rumelhart, and B. Huberman. Generalization by weight-elimination with application to forecasting. *Advances in Neural Information Processing Systems*, 3:875–882, 1991.

[96] A.S. Weigend, B.A. Huberman, and D.E. Rumelhart. Predicting the furure: a connectionist approach. *Int. Journal of Neural Systems*, 1(3):193–209, 1990.

[97] H. White. Connectionist nonparametric regression : multilayer feedforward networks can learn arbitrary mappings. *Neural Networks*, 3:535–549, 1990.

[98] M. Yacoub and Y. Bennani. HVS : A heuristic for variable selection in multilayer artificial neural networkclassifier. *Intelligent Engineering Systems Through Artificial Neural Networks*, 7:527–532, 1997.

[99] M. Yacoub and Y. Bennani. Architecture optimization in feedforward connectionist models. *IEEE International Conference on Artificial Neural Networks*, pages 881–886, 1998.

[100] M. Yacoub and Y. Bennani. A neural network methodology for machine's class identification. *IEEE International Joint Conference on Neural Networks*, pages 322–325, 1998.

[101] M. Yacoub and Y. Bennani. Discriminative feature extraction and selection applied to face recognition. *Accepté à International Joint Conference on Neural Networks*, 1999.

[102] M. Yacoub and Y. Bennani. Features selection and architecture optimization in connectionist systems. *soumis à International Journal on Neural Systems*, 1999.