Réseaux de neurones artificiels : application à la reconnaissance optique de partitions musicales

Philippe Martin

T H E S E

présentée par

Philippe MARTIN

pour obtenir le titre de DOCTEUR
de l'Université JOSEPH FOURIER - GRENOBLE I

(Arrêté ministériel du 5 juillet 1984)
(Spécialité Informatique)

Réseaux de Neurones Artificiels :

Application à la Reconnaissance Optique de Partitions Musicales

Thèse soutenue le 2 avril 1992 devant la commission d'examen composée de :

Georges STAMON	Président
Rolf INGOLD	Rapporteur
Christian JUTTEN	Rapporteur
Camille BELLISSANT	Examinateur
Jean-Marc CHASSERY	Examinateur
Annick MONTANVERT	Examinateur
François ROBERT	Examinateur

Equipe Reconnaissance de Formes et Microscopie Quantitative
Laboratoire TIM 3
USR CNRS n° B 00690 - Institut IMAG

AVANT-PROPOS

Avant toute chose, je tiens à remercier mon directeur de thèse, Mr Camille BELLISSANT, Professeur à l'Université Pierre Mendès France, pour la confiance qu'il m'a toujours accordée, pour le temps précieux qu'il m'a consacré, pour sa patience et son soutien constant, à tous les niveaux, au cours de ces années de travail. Qu'il veuille bien trouver ici l'assurance de ma gratitude, de ma plus profonde estime et de mon indéfectible amitié.

Je suis très reconnaissant à Mr Georges STAMON, Professeur à l'Ecole des Hautes Etudes Informatiques - Paris V, d'avoir bien voulu me faire l'honneur de présider ce jury.

Je remercie Mr Rolf INGOLD, Professeur à l'Université de Fribourg, pour ses encouragements chaleureux, pour l'intérêt qu'il a porté à mon travail et pour le plaisir qu'il me fait en acceptant d'en être le rapporteur.

Merci également à Mr Christian JUTTEN, Professeur à l'Université Joseph Fourier de Grenoble, pour avoir accepté de juger cette thèse. Je lui sais gré du temps qu'il m'a consacré avec gentillesse afin d'améliorer la version finale de ce document.

Je remercie Mr Jean-Marc CHASSERY, Directeur de recherche au CNRS, de l'intérêt qu'il a toujours manifesté pour mon travail, des conseils qu'il m'a donnés pour le mener à bien et du plaisir qu'il me fait finalement en acceptant de le juger.

Mes plus vifs remerciements vont également à Mme Annick MONTANVERT, Maître de Conférences à l'Université Pierre Mendès France, pour son aide constante au cours de cette thèse. Je lui suis redevable de multiples et précieux conseils et de minutieuses relectures. Son dynamisme professionnel fut pour moi un exemple (et le reste !). Qu'elle aussi croie à toute ma reconnaissance et à toute mon amitié.

Mr François ROBERT, Professeur à l'Institut National Polytechnique de Grenoble, me fait aussi l'honneur d'examiner ce travail ; je lui en suis extrêmement reconnaissant.

Merci à Mr Michel DECOUST de la Direction de la Musique du Ministère de la Culture dont le soutien a rendu possible ce travail.

Je remercie tous les membres de l'équipe RFMQ que j'ai cotoyés durant ces dernières années ; travailler avec eux fut un réel plaisir. Merci particulièrement à tous ceux avec qui j'ai eu des échanges privilégiés, que ce soit sur le plan scientifique, culturel, sportif, ou tout simplement humain (merci aux amis en somme), et à Mr Guy BOURREL, Ingénieur au CNRS, pour sa disponibilité, son aide, toujours efficace, jamais comptée, et son humour chaleureux.

Un grand merci à Mr Guy DAVIN, Safman devant l'éternel, pour m'avoir accordé, à ses risques et périls, un peu de temps pour finir cette thèse.

Enfin j'adresserai mon dernier remerciement, mais non le moindre, à toute ma famille pour son indispensable soutien et pour tout le reste...

TABLE DES MATIERES

INTRODUCTION

La motivation "originelle" du travail présenté dans ce document est la volonté de résoudre un problème de reconnaissance de formes peu traité jusqu'alors : celui des partitions musicales. Au moment où nous nous sommes attelés à cette tâche, une discipline en plein essor tentait de s'imposer dans plusieurs domaines, dont justement celui de la reconnaissance des formes, en affirmant sa supériorité sur les méthodes dites classiques. Le connexionnisme, il s'agit bien sûr de lui, était alors fréquemment présenté selon la perspective "romantique" du mimétisme neuro-biologique et il était donc tentant de lui prêter des capacités similaires à celles du cerveau humain dont on connaît les prouesses, en particulier dans le domaine de la reconnaissance visuelle. Cet excès d'optimisme s'est rapidement résorbé, sans que cesse pour autant l'activité intense qui s'était développée autour des réseaux de neurones formels et de leurs applications.

L'une des raisons de ce "retour au calme" est d'abord la prise de conscience du faible degré de neuro-mimétisme de la grande majorité des techniques regroupées sous le label connexionniste. Ce constat provient d'une part de la confrontation avec le domaine biologique, la complexité des systèmes vivants contrastant avec le schématisme des systèmes artificiels simulés, et d'autre part de la mise en évidence des étroites relations liant les réseaux de neurones aux techniques classiques desquelles ils prétendaient se démarquer. Faut-il pour autant en déduire qu'ils sont dénués d'intérêt "pratique" ? Nous n'avons pas l'ambition de répondre à cette question ici. La part dévolue aux réseaux de neurones dans ce document témoigne en tout cas de la curiosité qu'ils peuvent susciter chez le profane qui découvre ce domaine.

Le premier chapitre de ce document est consacré à cette découverte. Il tente de présenter les principaux modèles de réseaux de neurones de manière homogène, synthèse qu'il n'est pas toujours facile de réaliser lorsque l'on jette un regard non initié sur la littérature foisonnante consacrée au sujet depuis quelques années. Nous essayerons également de justifier le choix du modèle sur lequel nous avons particulièrement porté notre intérêt : celui des réseaux multi-

couches.

C'est à ces derniers qu'est dévolu le deuxième chapitre. Nous tenterons à nouveau de présenter de manière synthétique les connaissances les plus utiles à la mise en œuvre pratique de ces réseaux et de l'algorithme d'apprentissage qui les accompagne le plus souvent : la rétro-propagation du gradient. Nous proposerons une méthode de synthèse de l'architecture développée dans le cadre de notre problème de reconnaissance de formes. Nous nous intéresserons également à un type particulier de réseaux multi-couches : les réseaux d'automates à seuil. Nous montrerons l'équivalence qui existe entre ces réseaux et d'autres méthodes de classification et proposerons en particulier un algorithme d'apprentissage hiérarchique qui leur est dédié.

Enfin, le troisième et dernier chapitre sera consacré à notre préoccupation initiale : la reconnaissance des partitions musicales. Un système de reconnaissance "bas-niveau" sera présenté ; nous proposerons différentes solutions pour résoudre les problèmes d'analyse d'image et de classification qui se posent dans un tel système, certaines de ces solutions intégrant naturellement les outils connexionnistes présentés dans les chapitres précédents.

CHAPITRE 1

MODELES CONNEXIONNISTES POUR LA CLASSIFICATION

Parmi les applications spectaculaires ayant contribué à l'engouement actuel pour les réseaux de neurones artificiels, la reconnaissance des formes occupe une place privilégiée. Que ce soit en reconnaissance de la parole ou en reconnaissance optique de caractères, les capacités de résistance au bruit, la souplesse et la généricité sont les qualités qui, alliées à une relative simplicité de mise en œuvre, ont motivé l'emploi des différents types de réseaux, essentiellement à des fins de classification automatique.

Nous nous attacherons à décrire, dans les paragraphes suivants, les principaux modèles connexionnistes parmi lesquels le concepteur d'un système de reconnaissance de formes peut avoir à choisir. Nous avons pris l'option de ne présenter ces réseaux que selon la perspective qui nous intéresse, celle de la classification. Nous nous sommes en effet intéressés aux réseaux de neurones pour résoudre la tâche suivante : on considère une population de formes divisée en classes (ou catégories) ; une forme est décrite par un vecteur de mesures. Comment construire, en partant d'un ensemble de formes dont les classe sont connues, une procédure de décision capable de prédire la classe d'une forme inconnue ?

Ce problème de la classification (parfois appelée classification automatique, analyse discriminante) est bien sûr central en reconnaissance de formes, à tel point que nous restreindrons parfois cette dernière à ce seul problème en appelant abusivement "problème de reconnaissance de formes" ce qui n'est que de l'ordre de la classification. Deux grands types d'approches ont jusque là été employés pour le résoudre : les approches statistiques et les approches syntaxiques [FU 82]. Toutes deux sont largement décrites dans la littérature et appliquées depuis longtemps. L'émergence d'une troisième voie dite "neuronale", ou

"connexionniste", est observée depuis le début des années 80 (bien que son origine soit nettement antérieure). Elle regroupe un ensemble de méthodes diverses dont les champs d'application vont bien au-delà de la reconnaissance des formes. Néanmoins, elle propose surtout un large éventail de techniques de classification dont l'étude semble maintenant incontournable avant toute construction d'un système de reconnaissance. Cette étude sera le sujet de ce premier chapitre qui se veut représentatif mais absolument pas exhaustif, tant est vaste la littérature publiée depuis quelques années sur ce sujet.

1.1 NOTIONS DE BASE

Sous le vocable "réseaux de neurones" se cache en fait une grande diversité d'architectures de réseaux et d'algorithmes d'apprentissage. Dans ces conditions, les aspects communs aux différents modèles qui permettraient de construire une description générique d'un réseau de neurones, sont très réduits. Hecht-Nielsen propose une définition [HECHT-NIELSEN 90] que l'on peut résumer de la manière suivante :

> Un réseau de neurones est une structure de traitement de l'information parallèle et distribuée, constituée d'unités de calcul interconnectées par des canaux unidirectionnels appelés *connexions*. Chaque unité de traitement n'a qu'une seule connexion de sortie qui peut être dupliquée en autant d'exemplaires que désiré, les duplicata transportant tous le même signal. Le traitement effectué par chaque unité peut être défini de manière arbitraire pourvu qu'il soit complètement local, c'est-à-dire qu'il ne dépende que des valeurs courantes des signaux arrivant à l'unité par ses connexions entrantes et du contenu de la mémoire locale attachée à cette unité.

Cette définition très générale est la seule qui permette de regrouper tous les modèles de réseaux de neurones et, comme le fait remarquer son auteur, elle ne fait que décrire un type particulier d'architecture parallèle MIMD (Multiple Instruction Multiple Data) qui n'a *a priori* pour seul intérêt particulier que son analogie avec la structure du "cerveau biologique". Car si les réseaux de neurones *artificiels* constituent maintenant un outil de calcul parmi tant d'autres pour l'ingénieur, leur origine provient en fait d'une tentative de modélisation du neurone *biologique* proposée par McCulloch et Pitts en 1943.

1.1.1 Le neurone artificiel

La diversité des modèles connexionnistes se retrouve au niveau microscopique des

briques de base qui les constituent : les neurones artificiels (ou neurones formels, cellules, unités, ... tous ces termes seront indifféremment employés par la suite).

Le neurone formel de McCulloch et Pitts [McCULLOCH 43] est un automate à seuil qui, recevant **n** signaux x_i sur ses connexions entrantes, calcule sa sortie (ou état) **y** en appliquant une fonction seuil à la combinaison linéaire des x_i définie par les *poids* des connexions w_i :

$$y = \begin{cases} 1 \text{ si } \sum_{i=1}^{n} w_i.x_i \geq \theta \\ 0 \text{ sinon} \end{cases} \qquad (1.1)$$

Dans le modèle original, les signaux d'entrée x_i devaient, tout comme le signal de sortie y, prendre leur valeur sur {0,1}, les poids w_i et le seuil θ étant quant à eux définis sur $\mathbb{R}$.

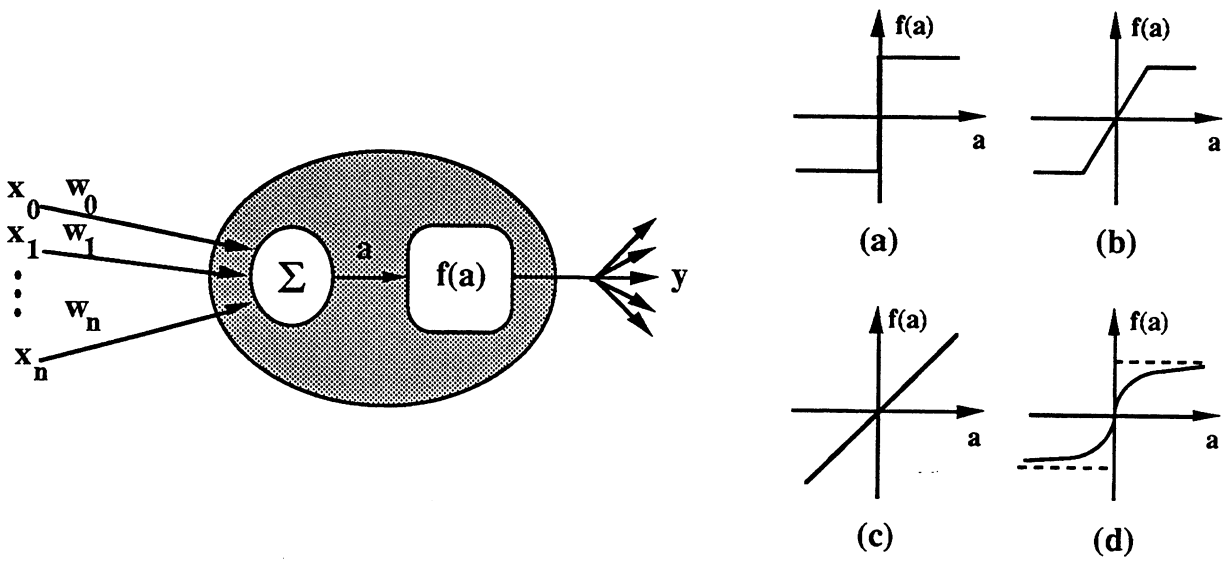


Fig. 1.1. Automate linéaire généralisé et fonctions de transition à seuil (a), à saturation (b), linéaire (c) et sigmoïdale (d).

Plus généralement, on définit un automate linéaire généralisé (Fig. 1.1), ou automate quasi-linéaire [FOGELMAN SOULIE 89], comme étant l'application d'une fonction **f**, dite *fonction de transition,* au produit scalaire entre le vecteur d'entrée $X=(1, x_1, x_2, ...,x_n)^t$ et le vecteur des poids $W=(w_0, w_1, w_2, ...,w_n)^t$:

$$y = f(a) \text{ avec } a = W.X, \quad W,X \in \mathbb{R}^{n+1}, y \in \text{Codomaine}(f) \qquad (1.2)$$

On notera au passage que l'on peut exprimer une combinaison affine en combinaison linéaire, et

ainsi se "débarrasser" de la notion de seuil θ, en considérant que ce seuil est l'opposé de la pondération d'une connexion d'indice 0 recevant un signal constant de valeur 1.

La fonction de transition **f** pourra être, selon les modèles de réseaux rencontrés, à seuil, à saturation, linéaire, sigmoïdale,...

Si cette définition d'un neurone artificiel est la plus répandue, il existe néanmoins certains modèles de réseaux utilisant des neurones dont la fonction de transition ne s'applique non pas au produit scalaire entre le vecteur d'entrée X et le vecteur des poids W, mais par exemple à la distance euclidienne entre ces deux vecteurs, le terme vecteur des *poids* perdant dans ce cas de son sens (cf § 1.4).

Deux "philosophies" s'opposent alors devant cette situation. On peut soit considérer que ces modèles s'écartent du domaine "neuro-mimétique", la notion de pondération d'une connexion ayant une forte analogie biologique (l'efficacité d'une synapse [PERSONNAZ 88]), et ne pas leur attribuer le label "réseau de neurones" ; ou l'on considère au contraire qu'il faut appréhender les réseaux de neurones en tant que modèles de calcul, originellement inspirés d'un phénomène physique, mais qui n'ont d'autre but que de réaliser efficacement une tâche donnée, même s'il doivent pour cela abandonner une certaine plausibilité biologique. C'est cette dernière approche qui semble la plus prisée dans le domaine des applications.

Dans cette optique, on pourra définir un neurone artificiel comme étant un élément de calcul dont la sortie **y** est le résultat d'une fonction de transition **f** appliquée au vecteur d'entrée X et au contenu d'une mémoire locale au neurone ML : y = f(X,ML) (Fig 1.2).

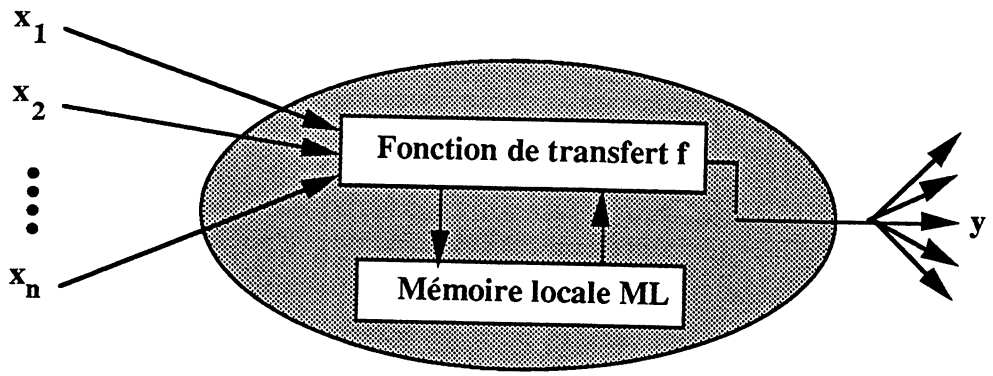


Fig. 1.2. Neurone formel générique [HECHT-NIELSEN 90].

On peut noter que dans ce cas les poids ne sont plus directement associés aux connexions

mais sont considérés comme étant une information locale au neurone. Cette définition permet d'unifier tous les types de neurones rencontrés et semble, en première analyse, assez "naturelle" en vue d'une mise en œuvre informatique, parallèle ou séquentielle.

Une première taxonomie des réseaux de neurones pourrait ainsi être réalisée selon la nature des cellules qu'ils utilisent. L'architecture de ces réseaux peut constituer un second critère de discrimination.

1.1.2 Architectures de réseaux

L'architecture d'un réseau décrit la manière dont sont interconnectées les cellules qui le composent. Plus précisément, l'architecture d'un réseau est entièrement spécifiée, à un instant donné, par :

- le nombre de cellules
- la nature des cellules (c'est-à-dire leur fonction de transition, en général identique pour toutes les cellules)
- le graphe d'interconnexion des cellules
- les "relations" entre le réseau et le monde extérieur

Ces paramètres sont en général fixés *a priori*, certains étant cependant susceptibles d'être modifiés au cours de la "vie" du réseau, certains modèles faisant par exemple varier le nombre de cellules et/ou de connexions.

Les relations entre le réseau et l'extérieur définissent, à l'intérieur du réseau, trois types de cellules (Fig. 1.3) :

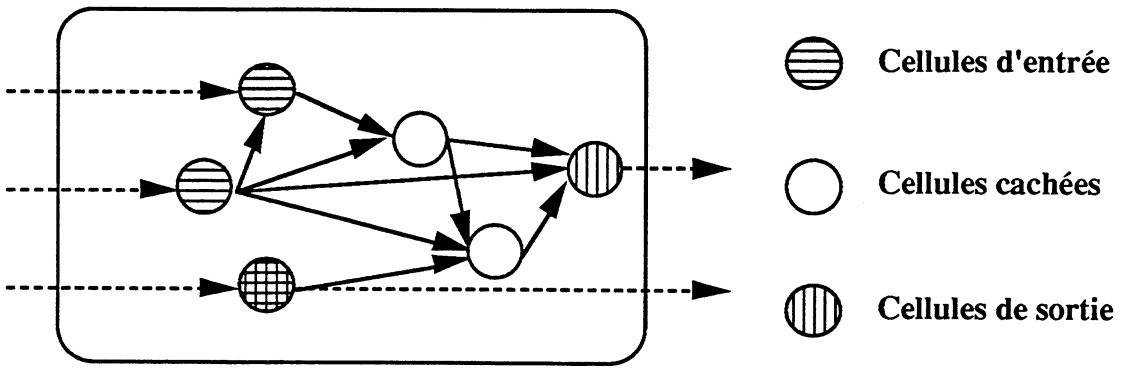


Fig. 1.3. Graphe d'interconnexion et types des cellules.

• les *cellules d'entrée* sont des cellules qui disposent d'une connexion entrante reliée non pas à la sortie d'une autre cellule du réseau, mais à l'extérieur. C'est par l'intermédiaire de telles cellules que l'on pourra agir sur le fonctionnement du réseau.
• les *cellules de sortie* font partie de l'ensemble des cellules dont on pourra consulter la valeur de sortie (on peut considérer qu'il existe une connexion depuis ces cellules vers l'extérieur). Ce sont elles qui indiqueront la "réponse" du réseau.
• les *cellules cachées* (sous entendu, au monde extérieur) sont, par opposition aux cellules d'entrées et de sortie, des cellules dont les connexions entrantes ne proviennent que de cellules du réseau, et dont la sortie ne peut être consultée.

Certaines cellules peuvent être à la fois cellules d'entrée et de sortie.

Le graphe d'interconnexion est un graphe orienté dont les sommets représentent les cellules du réseau, les arcs représentant les connexions "intra-réseau" (les connexions *de* ou *vers* l'extérieur ne sont pas prises en compte dans ce graphe).

Pour clarifier ce qui suit, nous rappelons ici quelques notions élémentaires concernant les graphes :
• Un graphe orienté $G : (S,A)_\varphi$ est défini par l'ensemble de ses sommets S (ici, l'ensemble des cellules du réseau), par l'ensemble de ses arcs A (les connexions du réseau) et par une application φ qui à chaque arc $\mathbf{a} \in A$ associe le couple de ses extrémités : $\varphi(a) = (u,v)$, $\mathbf{u} \in S$ étant l'extrémité d'origine de **a** et $\mathbf{v} \in S$ son extrémité d'arrivée.
• Le degré entrant d'un sommet **u**, $d^-(u)$, est le nombre d'arcs de **a** ayant **u** pour extrémité d'arrivée : $d^-(u) = | \{ a \in A / \varphi(a) = (s,u), s \in S \} |$, où |E| dénote le cardinal d'un ensemble E.
• Enfin, il existe un chemin de longueur **p** reliant deux sommets **u** et **v** de S s'il existe une suite d'arcs $a_1, a_2, ..., a_p$, avec $a_i \in A$, telle que :

1. $\varphi(a_1) = (u,s_1)$, $s_1 \in S$
2. $\varphi(a_i) = (s_{i-1},s_i)$, $s_i \in S$, $2 \leq i \leq p-1$
3. $\varphi(a_p) = (s_{p-1},v)$

Munis de ces notions, nous pouvons caractériser précisément les deux grands types de réseaux que l'on distingue habituellement : les réseaux *à couches* et les réseaux *dynamiques*.

1.1.2.1 Réseaux à couches

Formellement, un réseau à couches est un réseau dont le graphe d'interconnexion, G : $(S,A)_\varphi$ peut être *mis en niveaux*, au sens donné en théorie des graphes à cette expression. Ceci est possible pour tout graphe orienté **sans circuit**, c'est-à-dire pour tout graphe ne contenant aucun chemin reliant un sommet de S à lui même. Dans un tel graphe, on peut en effet associer à chaque sommet $\mathbf{u} \in S$ un niveau $\nu(u)$ défini ainsi :

$\nu(u) = 0$ si $d^-(u)=0$

$\nu(u) = p$, $p>0$, si le plus long chemin se terminant à **u** est de longueur **p**

Le terme de *niveau* se traduit, dans l'univers connexionniste, par le terme de *couche*. Une couche est composée de cellules ayant le même niveau dans le graphe d'interconnexion. La propriété principale d'un tel réseau est que les cellules d'un même niveau **p** ne peuvent avoir de connexions entrantes provenant de cellules de niveau supérieur ou égal à **p** (Fig. 1.4).

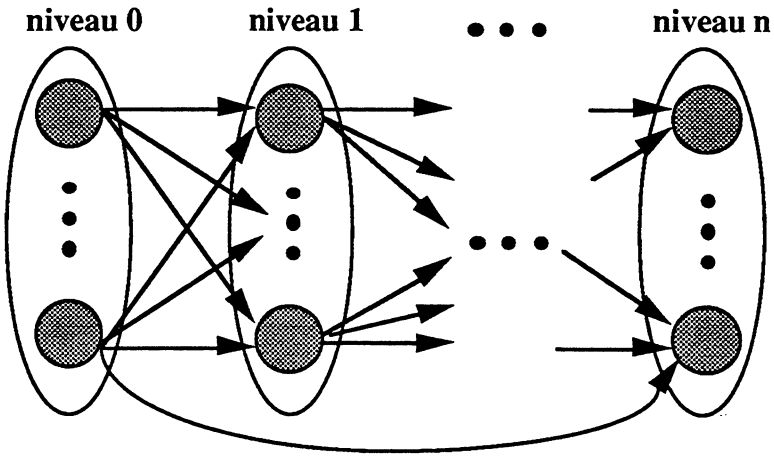


Fig. 1.4. Graphe d'interconnexion d'un réseau à n+1 couches.

En général, toutes les cellules d'entrée appartiennent au niveau 0 (leurs seules connexions entrantes proviennent de l'extérieur) et toutes les cellules de sortie sont regroupées dans le dernier niveau.

Puisque la sortie d'une cellule de niveau **p** ne dépend que des cellules de niveau **q**, avec $q<p$, l'évaluation de l'état du réseau, lorsque celui-ci est soumis à de nouvelles "excitations extérieures", doit se faire couche par couche, par niveau croissant, les cellules d'une même couche pouvant travailler indépendamment et donc en parallèle. La réponse du réseau se lira sur les cellules de sortie lorsque la dernière couche aura été mise à jour. C'est à cette propagation du

flot d'information de couche en couche, dans une seule direction, que ces réseaux doivent leur appellation de *feedforward* (littéralement, "passe aux avants"). On les qualifie également de réseaux *one-shot-computing* (calcul en un coup) car la réponse du réseau est disponible après que chaque cellule se soit mise à jour une seule fois, contrairement aux réseaux dynamiques que nous allons aborder maintenant.

1.1.2.2 Réseaux dynamiques

Par opposition aux réseaux à couches, le graphe d'interconnexion d'un réseau dynamique contient des circuits (Fig 1.5). La sortie d'une cellule appartenant à un tel circuit dépend donc d'elle-même. Pour pouvoir l'évaluer, il est nécessaire de définir un temps discret au cours duquel les cellules du réseau vont évoluer par itérations successives selon une dynamique fixée.

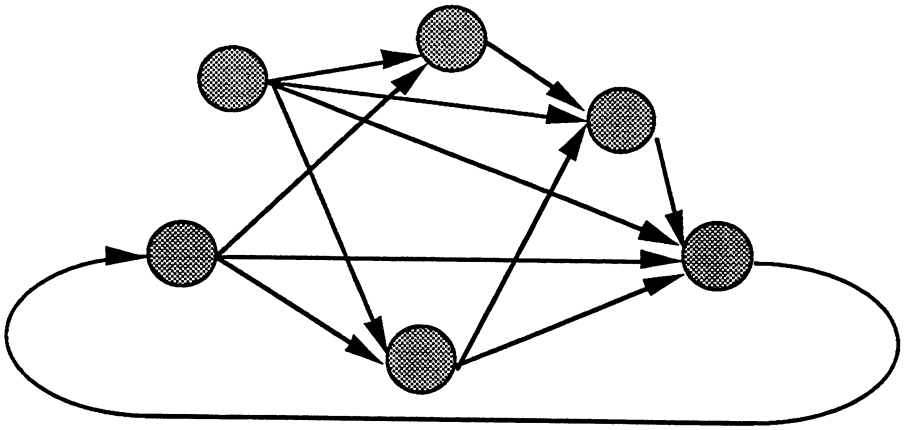


Fig. 1.5. Le graphe d'interconnexion d'un réseau dynamique contient des circuits.

On considère un réseau contenant **n** cellules ; la sortie $y_i(t)$ de la $i^{ème}$ cellule dépend de son vecteur d'entrée $X_i(t)$ (les composantes de ce vecteur pouvant être des sorties de cellules $y_j(t)$, $1 \leq j \leq n$, ou des valeurs provenant de l'extérieur). La mise à jour de la sortie des cellules peut se faire selon différentes itérations [ROBERT 86] :

- Itération *parallèle* ou *synchrone* : lors d'une itération, toutes les sorties $y_i(t+1)$ sont mises à jour simultanément en fonction des $y_j(t)$.
- Itération *séquentielle* ou *asynchrone* : pendant une itération, les cellules sont visitées une par une dans un ordre fixé par une permutation sur $\{1, 2, ..., n\}$. La sortie d'une cellule $y_i(t+1)$ dépend donc de $y_j(t+1)$ si **j** précède **i** dans l'itération, et de $y_j(t)$ si **j** suit **i** ou si **j=i**.

• Itération *séquentielle par blocs* : c'est une combinaison des deux modes précédents. Une partition en **b** blocs est définie sur l'ensemble des cellules du réseau. La visite des blocs se fait séquentiellement, dans un ordre fixé par une permutation sur {1,2, ..., b}. Les cellules à l'intérieur d'un même bloc sont mises à jour en parallèle.
• Itération *aléatoire* : une seule cellule est mise à jour à la fois, comme dans le mode séquentiel, mais ici les cellules sont visitées dans un ordre aléatoire.

Dans un réseau dynamique, une fois les valeurs fixées sur les connexions provenant de l'extérieur, le réseau va évoluer librement selon la dynamique que l'on s'est fixée. La réponse du réseau sera alors indiquée par l'état des cellules de sortie lorsque le réseau aura atteint un état stable, c'est-à-dire lorsqu'au cours d'une itération, aucune cellule n'aura changé d'état : $y_i(t+1) = y_i(t)$, $\forall i$, $1 \leq i \leq n$. Ceci peut bien sûr prendre plusieurs itérations ou même ne pas se produire du tout. Le fonctionnement d'un réseau dynamique est donc considérablement plus complexe que celui d'un réseau en couches et *a fortiori* plus lent, à tailles de réseaux égales.

1.1.3 Réseaux et reconnaissance de formes

En quoi un réseau de neurones peut-il nous aider à résoudre un problème de reconnaissance de formes ?

Un réseau admettant **n** connexions entrantes et **p** cellules de sortie (dont les fonctions de transition sont à valeurs sur ξ) peut être considéré comme une fonction $F_W : \mathbb{R}^n \rightarrow \xi^p$, fonction paramétrée par les poids des connexions W (ou pour reprendre la notation du § 1.1.1, par l'ensemble des mémoires locales à chaque cellule). Ainsi, un réseau peut être appréhendé comme une "boîte noire" capable de produire une forme (un vecteur de ξ^p) en réponse à une autre forme (un vecteur de $\mathbb{R}^n$) qu'on lui a présentée (Fig. 1.6). Peu importe en l'occurrence que cette réponse soit le résultat d'une propagation de signaux de couche en couche ou l'état final de la relaxation d'un système dynamique.

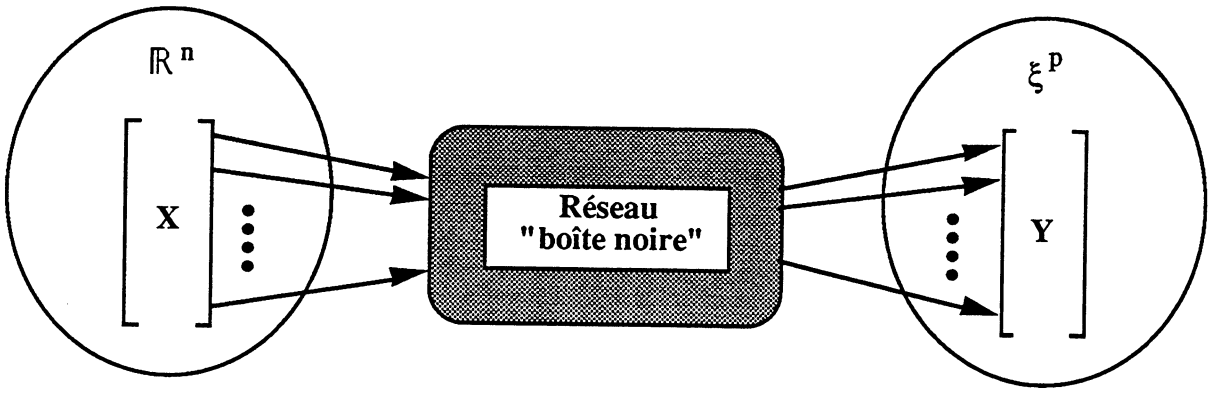


Fig. 1.6. Un réseau est une fonction de mise en correspondance.

Un réseau capable de mettre en correspondance un ensemble de paires de formes donné est appelé, dans le jargon neuronal, *mémoire associative.*

Tout le problème consiste bien sûr, étant donné un ensemble E de N paires $\{(X_i,Y_i)\}$, $1 \le i \le N$, $X_i \in \mathbb{R}^n$, $Y_i \in \xi^p$, à construire un réseau F_W capable de mettre ces formes en correspondance, c'est-à-dire tel que $F_W(X_i)=Y_i$, $\forall i$, $1 \le i \le N$. Ceci passe tout d'abord par le choix de l'architecture du réseau, telle que nous l'avons définie au § 1.1.2. Ce choix est loin d'être simple, même si quelques critères existent pour le guider. L'option de l'architecture étant prise, il reste à déterminer, au cours d'une phase dite d'*apprentissage*, un ensemble de poids W permettant à cette architecture de réaliser la tâche associative demandée (c'est-à-dire "d'apprendre" l'ensemble E appelé *ensemble d'apprentissage.*). Les méthodes ou *algorithmes d'apprentissage* permettant de calculer W sont très variées et en général liées à l'architecture du réseau choisi.

L'apprentissage présenté ainsi est qualifié de *supervisé.* Dans un apprentissage *non supervisé*, on ne demande pas explicitement au réseau d'associer un Y_i à chaque X_i. Les formes X_i sont simplement présentées au réseau et celui-ci doit s'*auto-organiser* afin de produire des sorties proches en réponse à des formes d'entrées proches, ces deux notions de proximité restant bien sûr à définir.

Dans un problème de reconnaissance de formes, un réseau fonctionnant en mémoire associative, avec apprentissage supervisé, pourra être utilisé à des fins de classification. Dans ce cas, une forme d'entrée X_i sera un vecteur de primitives décrivant l'objet à reconnaître, et l'on demandera au réseau d'associer à cet objet une forme Y_i qui pourra être soit sa classe (représentée par un vecteur de ξ^p), soit un prototype de sa classe (lui-même par exemple). On parle d'*hétéro-association* dans le premier cas (on met en correspondance deux formes de

natures différentes, $Y_i \neq X_i$) ou d'*auto-association* dans le second cas ($Y_i = X_i$).

Un réseau pourra alors réellement classifier si, après avoir appris à associer objets et classes de l'ensemble d'apprentissage E, il est capable de prédire avec succès la classe d'un objet n'appartenant pas à E. Obtenir un réseau ayant cette capacité de *généralisation* est en fait un problème beaucoup plus complexe que celui de l'apprentissage, pour la simple raison que l'on est en général incapable de caractériser précisément la généralisation désirée. De plus, cette capacité dépend de beaucoup de paramètres (architecture du réseau, algorithme d'apprentissage, ensemble d'apprentissage, ...), chacun d'entre eux devant faire l'objet d'un choix précis en fonction de l'application envisagée.

Nous allons, au cours des paragraphes suivants, décrire quelques-uns des principaux modèles de réseaux rencontrés en reconnaissance de formes, "modèle de réseau" signifiant ici une paire (architecture, algorithme d'apprentissage). Nous avons regroupé ces modèles en trois catégories (bien que d'autres taxonomies soient possibles) : les réseaux à couches qui présentent des architectures telles que celle du § 1.1.2.1, les réseaux dynamiques correspondant au § 1.1.2.2 et enfin les réseaux à prototypes, à architectures hybrides, dont le point commun est d'utiliser des cellules afin de stocker explicitement les formes à apprendre.

1.2 RESEAUX A COUCHES

Les réseaux à couches occupent une place particulière dans l'histoire des réseaux de neurones car c'est à un modèle à couches, le Perceptron, que l'on doit le premier essor, dans les années 1950, des méthodes connexionnistes. C'est ensuite à ce même modèle que l'on doit le "déclin" (médiatique surtout) de la recherche dans ce domaine, avant qu'un autre modèle, le réseau multi-couche à rétro-propagation du gradient, ne vienne relancer spectaculairement, au début des années 80, l'intérêt de la communauté scientifique pour ce sujet (le modèle de Hopfield y étant également pour beaucoup).

De plus, les réseaux à couches ont pratiquement toujours été conçus à des fins de reconnaissance de formes, même s'ils ont parfois été utilisés à d'autres tâches. Nous commencerons ici par décrire le précurseur historique de ces modèles, le Perceptron.

1.2.1 Le Perceptron

1.2.1.1 Architecture

Le perceptron est un réseau à trois couches proposé par Rosenblatt en 1957 [ROSENBLATT 57]. Chaque couche est composée de cellules différentes (Fig. 1.7) et joue un rôle précis :

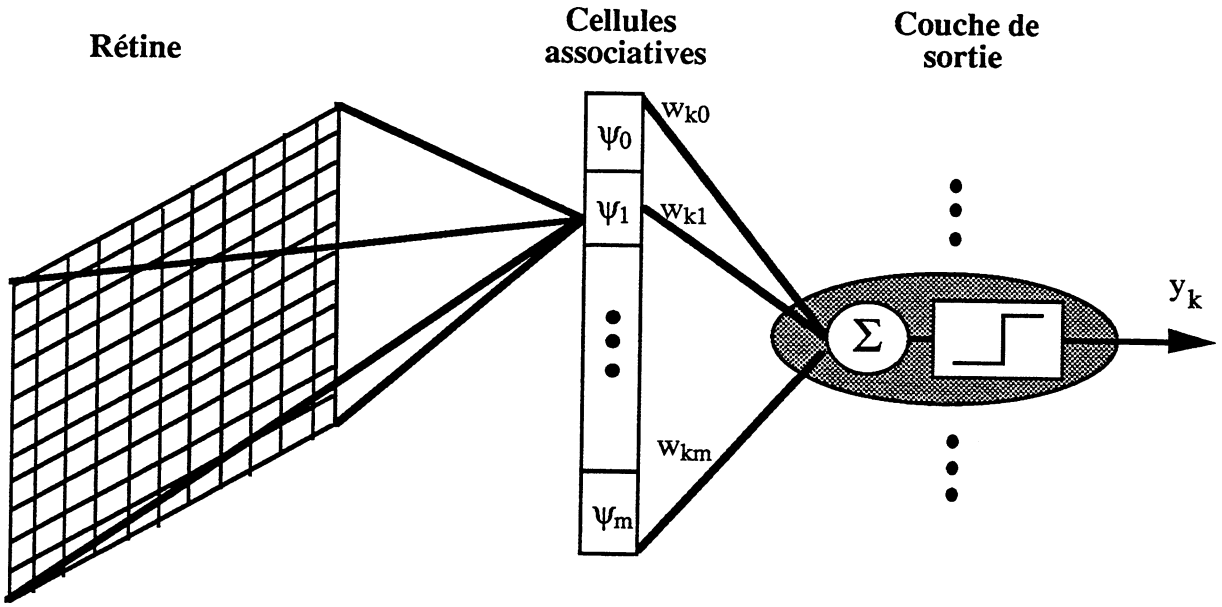


Fig. 1.7. Architecture du perceptron.

• La première couche (niveau 0), appelée *rétine*, contient **n** cellules d'entrée, chaque cellule se contentant de recopier la valeur qu'elle reçoit de l'extérieur sur sa sortie. A l'origine, ces cellules avaient une organisation matricielle à deux dimensions, tout comme les pixels d'une image.

• La seconde couche (niveau 1) est composée de **m+1** cellules dites *associatives*. Chaque cellule a des connexions entrantes pouvant provenir de toutes ou d'une partie des cellules de la rétine. Les fonctions de transition de ces cellules, $\psi_i : \mathbb{R}^n \rightarrow \{0,1\}$, sont fixées *a priori* et sont en général différentes d'une cellule à l'autre. Les connexions des cellules associatives ne sont pas pondérées (la cellule n'a pas de mémoire locale) et cette couche n'est donc pas *adaptative* : aucun apprentissage ne peut intervenir à ce niveau du réseau.

• La dernière couche (niveau 2) est composée de **p** automates à seuil, chaque automate étant connecté à toutes les sorties de la couche précédente.

Soit $X \in \mathbb{R}^n$ le vecteur présenté à la rétine, la sortie du $k^{ème}$ automate vaudra :

$$y_k(X) = \mathbb{1}(W_k.\Psi(X)) \tag{1.3}$$

où $W_k=(w_{k0}, w_{k1}, w_{k2}, ..., w_{km})^t \in \mathbb{R}^{m+1}$ est le vecteur poids de l'automate **k** (w_{ki} est la pondération de la connexion reliant la sortie de la $i^{ème}$ cellule associative à l'automate **k**), où $\Psi(X)=(1,\psi_1(X), ...,\psi_m(X))^t \in \mathbb{R}^{m+1}$ est l'image de X dans l'espace des cellules associatives, et où $\mathbb{1}(x)$ désigne la fonction seuil $\mathbb{1} : \mathbb{R} \rightarrow \{-1,1\}$ telle que $\mathbb{1}(x)=1$ si $x \geq 0$, -1 sinon. Comme nous l'avons déjà dit, le seuil de l'automate est l'opposé du poids de la connexion provenant de la cellule associative "0" qui est telle que $\psi_0(X)=1, \forall X \in \mathbb{R}^n$.

1.2.1.2 Apprentissage

L'algorithme du perceptron permet un apprentissage supervisé en adaptant les poids des connexions de la dernière couche. Considérons le problème consistant à reconnaître **p** types d'objets différents, un objet étant décrit par un vecteur de $\mathbb{R}^n$ (composé par exemple des n valeurs des pixels de son image).

Soit $E=\{(X_i,Y_i)\}$, $1 \leq i \leq N$, $X_i \in \mathbb{R}^n$, $Y_i \in \{-1,1\}^p$, l'ensemble d'apprentissage. Par convention, si X_i est de classe **j**, $1 \leq j \leq p$, alors on souhaite lui associer $Y_i=(y_{i1}, y_{i2}, ..., y_{ip})^t$ tel que $y_{ik}=1$ si $k=j$, $y_{ik}=-1$ si $k \neq j$. Ainsi, une forme X présentée au réseau se verra attribuer la classe **k** si la sortie du $k^{ème}$ automate vaut 1 et si les sorties de tous les autres automates valent -1. Dans tous les autres cas, on décidera, par convention, que la forme est *rejetée* (on ne sait quelle classe lui attribuer).

L'apprentissage du Perceptron se fait selon le processus itératif suivant :

Etape 0 :	$t=0$
	Initialiser aléatoirement les poids $W_k(t)$
Etape 1 :	Choisir aléatoirement un couple d'apprentissage (X_i,Y_i)
	Présenter X_i en entrée sur la rétine, puis calculer la sortie du réseau : $Y(X,t)=(y_1(X,t), ..., y_p(X,t))^t$
Etape 2 :	Pour tout automate k tel que $y_k(X_i,t) \neq y_{ik}$ modifier ses poids par : $W_k(t+1) = W_k(t) + y_{ik}.\Psi(X_i)$
Etape 3 :	$t = t+1$
	Si condition d'arrêt non remplie, aller à l'étape 1

Une itération de l'algorithme consiste donc à présenter une forme X_i de l'ensemble d'apprentissage au réseau, à observer la sortie produite $Y(X_i,t)$ (étape 1) et à modifier les poids du réseau si cette sortie produite n'est pas égale à la sortie désirée Y_i (étape 2).

Soit $W_k(t)$ le vecteur des poids d'un automate **k** dont la réponse n'est pas bonne à l'itération t : $y_k(X_i,t) \neq y_{ik}$. Ce vecteur est modifié par :

$$W_k(t+1) = W_k(t) + y_{ik}.\Psi(X_i) \tag{1.4}$$

Après modification, la sortie de cet automate vaudra :

$$y_k(X_i,t+1) = \mathbb{1}(W_k(t+1).\Psi(X_i)) = \mathbb{1}(W_k(t).\Psi(X_i)+y_{ik}.\|\Psi(X_i)\|^2) \tag{1.5}$$

Le principe de l'algorithme consiste donc à ajouter à la valeur seuillée par l'automate "fautif" la quantité : $y_{ik}.\|\Psi(X_i)\|^2$. Cette quantité est positive (resp. négative) si la valeur seuillée est négative (resp. positive) alors qu'elle aurait dû être positive (resp. négative). Ainsi, la direction de la correction sur cette valeur tend à inverser la décision de l'automate. Ce principe de correction peut être assimilé à une minimisation de coût (ici d'un coût d'erreur de classification) et est commun à de nombreux algorithmes d'apprentissage connexionnistes, comme nous aurons l'occasion de le voir par la suite.

1.2.1.3 Capacités et limites du Perceptron

Considérons un automate **k** réalisant la tâche demandée : la sortie de cet automate vaut 1 pour toutes les formes de classe **k**, et -1 pour toutes les autres formes : $y_k=\mathbb{1}(W_k.\Psi(X_i))=y_{ik}$, $\forall\ (X_i,Y_i) \in E$. Le vecteur des poids de cet automate définit un hyperplan dans l'espace des cellules associatives. Son équation est :

$$w_{k1}.\psi_1(X) + w_{k2}.\psi_2(X) + ... + w_{km}.\psi_m(X) + w_{k0} = 0 \tag{1.6}$$

Cet automate assignera donc la classe **k** à toute forme X dont l'image dans l'espace des cellules associatives se trouve dans le demi-espace positif ($W_k.\Psi(X) \geq 0$) défini par l'hyperplan et il rejettera toute forme dont l'image est dans le demi-espace négatif (Fig. 1.8).

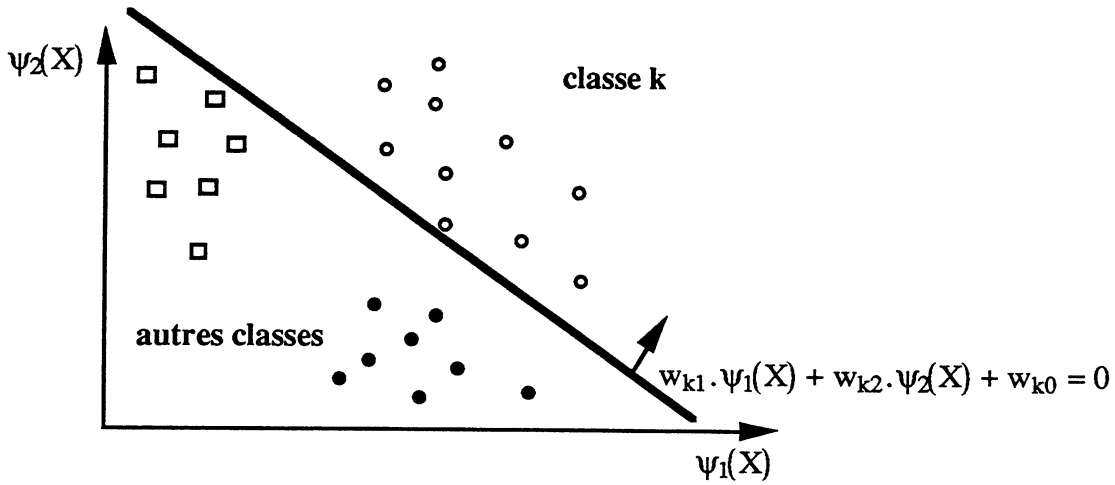


Fig. 1.8. Un automate à seuil réalise une séparation linéaire dans l'espace des cellules associatives (m=2, p=3).

Ainsi, lorsque l'apprentissage est achevé, chaque automate de la couche de sortie réalise une séparation linéaire, dans l'espace des cellules associatives, entre les formes d'une classe et le reste des formes de E.

Minsky et Papert ont démontré [MINSKY 69] que si les formes d'une classe **k** sont, dans l'espace des cellules associatives, linéairement séparables des autres formes, alors l'algorithme du perceptron trouvera en un nombre fini d'itérations les poids de l'automate **k** capable de réaliser cette séparation, et ce quelles que soient les valeurs initiales données à ces poids et quelle que soit la manière de choisir les formes lors de l'étape 1 (*Perceptron convergence theorem*). Dans ce cas, l'algorithme se termine avec succès (étape 3) lorsque les poids de tous les automates ne sont pas modifiés au cours d'une itération.
Remarque : ce théorème se généralise au cas où les fonctions de transition des cellules d'association ψ_i sont à valeurs réelles.

Un Perceptron peut donc apprendre un problème de classification dès lors que l'on est capable de déterminer un ensemble de cellules associatives qui rendent chaque classe linéairement séparable de toutes les autres. Tout le problème consiste en fait à déterminer les **m** fonctions $\{\psi_j\}$ "ad-hoc", et le Perceptron ne propose aucune méthode automatique pour réaliser cette tâche. A cet égard, le lien fonctionnel proposé par Pao ([PAO 89],[KLASSEN 89]) doit être considéré comme une tentative de réponse à ce problème plutôt que comme un modèle de réseau à part entière. Il consiste à construire un espace des cellules associatives qui soit celui des cellules d'entrée, $\mathbb{R}^n$, augmenté de termes dits "d'ordre supérieur" obtenus en multipliant les composantes des formes d'entrée deux à deux ($\Rightarrow m > n$).

En fait, la couche associative du Perceptron étant "figée", elle apparaît comme étant une phase d'extraction de primitives qui peut être déconnectée du réseau lui-même et le Perceptron sera alors considéré comme un réseau à **deux** couches : une couche d'entrée et une couche de cellules adaptatives qui est en fait la seule réellement concernée par l'algorithme d'apprentissage. Trouver les bonnes primitives à soumettre à un système dans l'optique d'une classification est un autre problème que l'on peut dissocier de celui de l'apprentissage, même si certaines approches tentent de les intégrer tous deux dans le cadre des réseaux de neurones, (toujours par souci de "neuro-mimétisme").

Ayant choisi un ensemble de **m** primitives $\{\psi_j\}$, comment nous assurer que ces primitives permettront effectivement au Perceptron d'apprendre l'échantillon E ainsi codé ? On peut certes vérifier, pour chaque classe, que le nuage de points qu'elle constitue dans l'espace des primitives est linéairement séparable du nuage des autres points. On sait en effet que deux nuages sont linairement séparables si leurs enveloppes convexes ont une intersection vide. Cette vérification nécessitera la résolution de **p** programmes linéaires à **m** variables et N contraintes, tâche très lourde dès lors que **m** est supérieur à trois [PREPARATA 85]. Mais peut-on soumettre un problème d'apprentissage au Perceptron si l'on n'est pas assuré *a priori* que les classes sont linéairement séparables ? Deux cas se présentent :

> Cas 1 : les classes **sont** lineairement séparables. Alors, nous l'avons vu, l'algorithme convergera et déterminera les poids des automates de sortie en un temps fini.
>
> Cas 2 : les classes **ne sont pas** linéairement séparables. L'algorithme ne convergera pas mais on peut néanmoins déterminer un critère d'arrêt dans ce cas. En effet, Minsky et Papert ont démontré [MINSKY 69] que l'ensemble des vecteurs de poids d'un automate donné que l'algorithme pouvait "visiter" au cours de ses itérations était fini (*Perceptron cycling theorem*). Ainsi, l'algorithme se terminera avec échec (étape 3) si, au cours de l'itération **t**, en modifiant l'automate **k**, on obtient un vecteur $W_k(t)$ tel que $\exists$ t', t'<t / $W_k(t')=W_k(t)$. L'algorithme peut donc éventuellement servir à déterminer si chaque classe est linéairement séparable des autres, mais ceci nécessitera un nombre d'itérations important et le stockage de tous les vecteurs poids de chaque automate à chaque itération...

Ainsi, l'algorithme du perceptron, appliqué à n'importe quelle situation d'apprentissage, se terminera toujours. Cependant, en cas d'échec, on pourrait souhaiter que l'algorithme fournisse une solution qui soit "la meilleure possible", c'est-à-dire un réseau qui associerait la

bonne classe au plus grand nombre de formes possible (une solution de moindre coût en termes d'erreurs de classification). Or, tel quel, l'algorithme ne garantit pas une telle solution.

Gallant a proposé [GALLANT 86] une modification simple permettant de résoudre ce problème : l'*algorithme de la poche* (pocket algorithm). Il consiste à garder pour vecteur de poids final de chaque automate, le vecteur qui n'a pas été modifié pendant le plus grand nombre d'itérations successives (vecteur que l'on aura mis de côté, dans sa *poche*). Le nombre de vecteurs visités étant fini, la probabilité pour que la solution finale soit optimale (nombre de classifications correctes maximal) tend vers 1 lorsque le nombre d'itérations augmente. Le problème consiste ici à déterminer le nombre d'itérations au terme duquel il faut s'arrêter...

Une autre méthode permettant de trouver des solutions de moindre pénalité en cas de non-séparabilité linéaire avait été proposée auparavant par un modèle de réseau à couches au principe d'apprentissage plus général : l'*Adaline*.

1.2.2 L'Adaline

Le terme Adaline ne désigne pas en fait un modèle de réseau, mais un type de cellules, Adaline signifiant : ADAptive LINear Element. Ces cellules sont des automates linéaires généralisés dont la fonction de transition est l'identité. Soit $W=(w_0, w_1, w_2, ..., w_n)^t$ le vecteur des poids d'une telle cellule (on conserve la même convention concernant le seuil) ; sa sortie en réponse au vecteur $X=(1, x_1, x_2, ..., x_n)^t$ est réelle et vaut :

$$y(X) = W.X = w_1.x_1 + w_2.x_2 + ... + w_n.x_n + w_0 \qquad (1.7)$$

Ces unités jouent un rôle identique à celui des automates de la dernière couche du Perceptron. Connectées aux cellules de la couche d'entrée, dans un schéma à deux couches, elles vont tenter de séparer chaque classe des autres par un hyperplan, directement dans l'espace des formes dans le cas présent. A nouveau, cet apprentissage est réalisé par un processus itératif visant à adapter les poids de chaque cellule afin que la réponse du réseau aux formes d'apprentissage corresponde aux réponses désirées. Mais contrairement au Perceptron, le principe de cette adaptation, appelé *règle de Widrow-Hoff*, est exprimé ici **explicitement** comme une minimisation d'erreur.

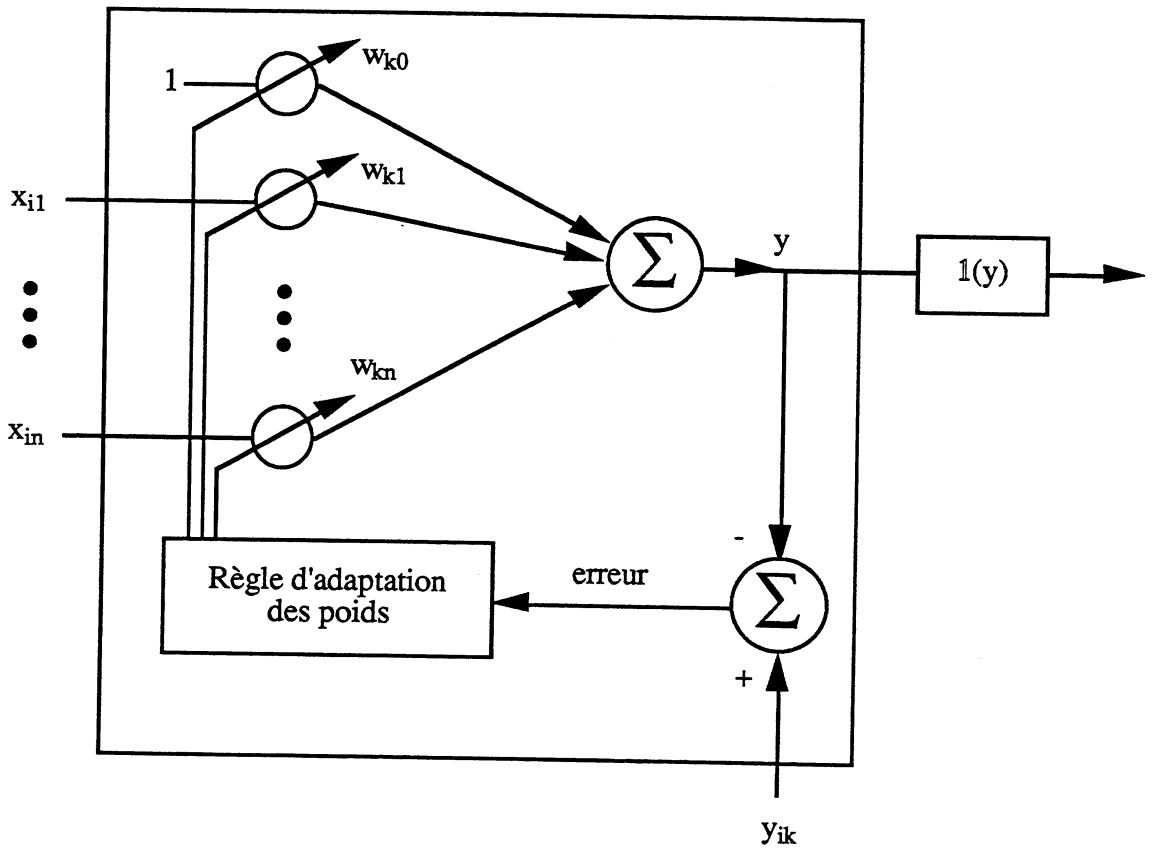


Fig. 1.9. Schéma conceptuel d'un ADALINE.

1.2.2.1 Apprentissage par minimisation d'erreur

Soit E l'ensemble d'apprentissage tel que nous l'avons défini auparavant. On considère un réseau dont la couche de sortie comporte **p** cellules de type Adaline. Soit $W_k(t)$ les poids de la $k^{ème}$ cellule à un instant donné. L'erreur commise par cette cellule sur l'ensemble E, $C(W_k(t))$, est définie comme étant la moyenne des erreurs quadratiques commises sur chaque forme X_i, $C_i(W_k(t))$:

$$C(W_k(t)) = \frac{1}{N}\sum_{i=1}^{N} C_i(W_k(t)) = \frac{1}{N}\sum_{i=1}^{N} (y_k(X_i,t) - y_{ik})^2 \qquad (1.8)$$

où $y_k(X_i,t)$ est la sortie de la $k^{ème}$ cellule au temps **t** en fonction de l'entrée X_i (expression 1.7).

Cette fonction coût (ou erreur) devrait bien sûr être nulle si la cellule pouvait complètement "apprendre" l'ensemble E. Ceci est en général impossible dans une situation de classification ($y_{ik} \in \{-1,1\}$), même si la classe **k** est linéairement séparable des autres. Mais en

fait, pour interpréter la sortie y d'une cellule Adaline afin de classifier une forme, il est nécessaire de la seuiller et de prendre par exemple les mêmes conventions que pour le Perceptron ; finalement, à la sortie de ce processus *discret* de décision, l'échantillon peut être appris même si le coût *continu* C n'est pas nul. L'apprentissage consiste donc simplement à rechercher un vecteur poids W_k^* qui minimise C, sans forcément l'annuler. De plus, dans le cas de non séparabilité linéaire, un coût minimal conduira probablement à une solution de moindre pénalité en termes d'erreurs de classification.

On peut démontrer que $C(W_k(t))$ est une surface paraboloïde qui admet un ou plusieurs minima (tous de même valeur) et il est possible de déterminer analytiquement un tel minimum W_k^*. Mais ceci nécessite le calcul de la pseudo-inverse d'une matrice de taille nxN et ce calcul est difficilement envisageable lorsque l'espace des formes est de grande dimension et l'ensemble d'apprentissage important, ce qui est typiquement le cas en reconnaissance de formes.

La méthode proposée par Widrow et Hoff pour résoudre ce problème consiste à adapter une méthode d'optimisation bien connue : la minimisation par *descente en gradient.* Le principe en est simple : l'opposé du gradient de C par rapport aux poids en $W_k(t)$, $-\nabla_W C(W_k(t))$, pointe dans la direction dans laquelle la diminution de $C(W_k)$ est maximale ; en modifiant le vecteur poids, par itérations successives, dans la direction opposée au gradient, on peut espérer aboutir à un minimum de $C(W_k)$ (Fig. 1.10).

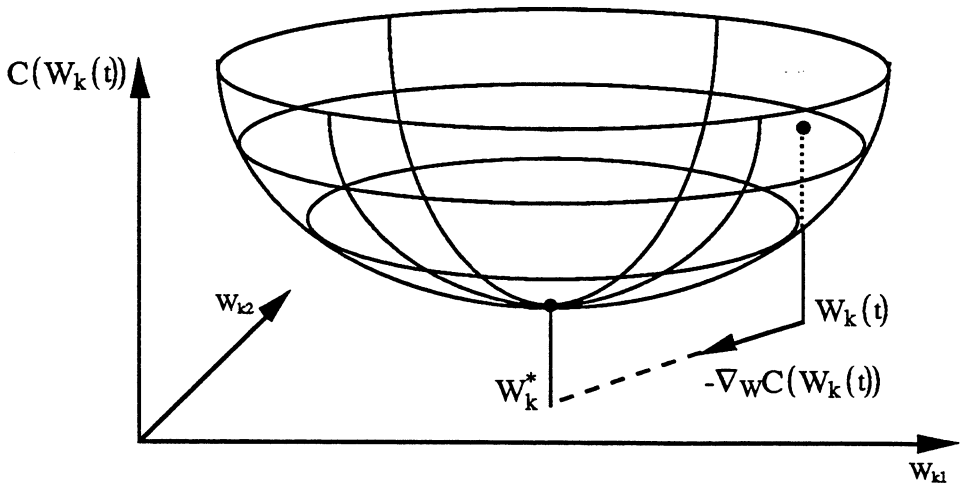


Fig. 1.10. Minimisation de la fonction coût par descente en gradient.

Le gradient de la fonction coût par rapport à W_k se calcule aisément par :

$$\nabla_W C(W_k(t)) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W C_i(W_k(t)) = \frac{2}{N}\sum_{i=1}^{N} (y_k(X_i,t) - y_{ki}).X_i \qquad (1.9)$$

Et l'algorithme d'apprentissage par descente en gradient est donc le suivant :

Etape 0 :	$t = 0$
	Initialiser aléatoirement les poids $W_k(t)$
Etape 1 :	Pour chaque automate k, calculer l'erreur commise sur l'ensemble d'apprentissage E : $C(W_k(t))$
Etape 2 :	Pour chaque automate k, modifier les poids par : $W_k(t+1) = W_k(t) - \alpha.\nabla_W C(W_k(t))$
Etape 3 :	$t = t+1$
	Si condition d'arrêt non remplie, aller à l'étape 1.

Le paramètre α, appelé *taux d'apprentissage*, influe sur la longueur du déplacement que l'on va réaliser dans le sens opposé au gradient. Pour garantir la convergence, α doit être positif et inférieur à 2 divisé par le rayon spectral de la matrice de covariance R. Bien sûr, cette valeur n'est pas calculée en pratique et le choix de α (et éventuellement, la manière de la faire varier au cours de l'apprentissage) est une des difficultés de l'utilisation de la descente en gradient. Une autre difficulté consiste à choisir un critère d'arrêt (étape 3). Celui-ci peut être par exemple fondé sur une valeur seuil pour la fonction coût (ou pour la variation de cette fonction entre deux itérations), sur le nombre de formes bien classées de E, ...

1.2.2.2 Règle de Widrow-Hoff

L'algorithme décrit ci-dessus minimise donc l'erreur quadratique moyenne commise sur l'ensemble d'apprentissage. Cette erreur est la somme des erreurs commises sur chaque forme (1.8) et de la même manière, la correction apportée aux poids (1.9) peut être vue comme une somme de corrections émanant de chaque forme. La règle de Widrow-Hoff telle qu'on la connaît est en fait légèrement différente : elle consiste à ne considérer, à une itération donnée, qu'une seule forme X_i au lieu de tout l'ensemble d'apprentissage. La modification des poids se fait alors selon le gradient de l'erreur commise sur la forme considérée :

$$\nabla_W C_i(W_k(t)) = 2.(y_k(X_i,t) - y_{ki}).X_i \qquad (1.10)$$

et l'algorithme se réécrit alors sous une forme proche de celle du Perceptron :

Etape 0 :	t=0
	Initialiser aléatoirement les poids $W_k(t)$
Etape 1 :	Choisir le couple d'apprentissage $(X_i,Y_i) \in E$ avec $i = (t \bmod N)+1$.
	Pour chaque automate k, calculer l'erreur commise sur la paire (X_i,Y_i) :
	$C_i(W_k(t))$
Etape 2 :	Pour chaque automate k, modifier les poids par :
	$W_k(t+1) = W_k(t) - \alpha.\nabla_W C_i(W_k(t))$
Etape 3 :	$t = t+1$
	Si condition d'arrêt non remplie, aller à l'étape 1.

Cette solution ne minimise plus explicitement le coût moyen C selon son propre gradient, mais selon les gradients "locaux à chaque forme" que l'on peut considérer comme des versions approchées du véritable gradient ; d'où la dénomination de *stochastique* attachée à cette procédure de descente en gradient [FOGELMAN SOULIE 87].

1.2.3 De l'utilité des unités cachées

Comme nous venons de le voir, les algorithmes d'apprentissage du Perceptron et de l'Adaline consistent tous deux à adapter les poids de cellules d'une seule couche, la couche de sortie (en fait, on peut démontrer que l'algorithme du Perceptron n'est qu'une version "discrète" de la règle de Widrow-Hoff correspondant à une autre fonction coût). Dans les deux cas, l'apprentissage consiste à trouver un hyperplan par classe de formes, chaque hyperplan devant séparer, dans l'espace d'entrée $\mathfrak{E}$ des cellules que l'on adapte, "sa" classe de toutes les autres. Ces deux algorithmes construisent donc des classifieurs linéaires dans $\mathfrak{E}$. Les limites de tels classifieurs sont bien connues ; ils ne peuvent évidemment fonctionner correctement que si les classes sont linéairement séparables, ou "presque". Lorsque le problème est fortement non linéaire, même une solution approchée ne peut être satisfaisante.

Ceci est généralement illustré par le problème du XOR qui consiste à classer les quatre sommets du cube binaire $B^2=\{0,1\}^2$ en deux classes selon leur valeur par la fonction logique "ou exclusif", ce qui correspond à l'ensemble d'apprentissage $E=\{((0,0)^t,-1), ((1,1)^t,-1), ((0,1)^t,1), ((1,0)^t,1)\}$. Les deux classes ainsi définies ne sont évidemment pas linéairement séparables et ne peuvent être apprises ni par l'algorithme du Perceptron, ni par celui de l'Adaline. Néanmoins, ces deux méthodes proposent des solutions pour contourner cette écueil.

Le Perceptron peut ainsi résoudre le problème du XOR moyennant une couche associative judicieusement choisie. En effet, on vérifie facilement que :

$$y = XOR(x_1,x_2) = \mathbb{1}(x_1 + x_2 - 2x_1.x_2 - 1) \qquad (1.11)$$

En utilisant trois cellules associatives ψ_1, ψ_2, ψ_3 telles que $\psi_1(x_1,x_2)=x_1$, $\psi_2(x_1,x_2)=x_2$ et $\psi_3(x_1,x_2)=x_1.x_2$, on construit un espace $\mathcal{E}=\{0,1\}^3$dans lequel le problème du XOR est résoluble par l'algorithme du perceptron (Fig. 1.11).

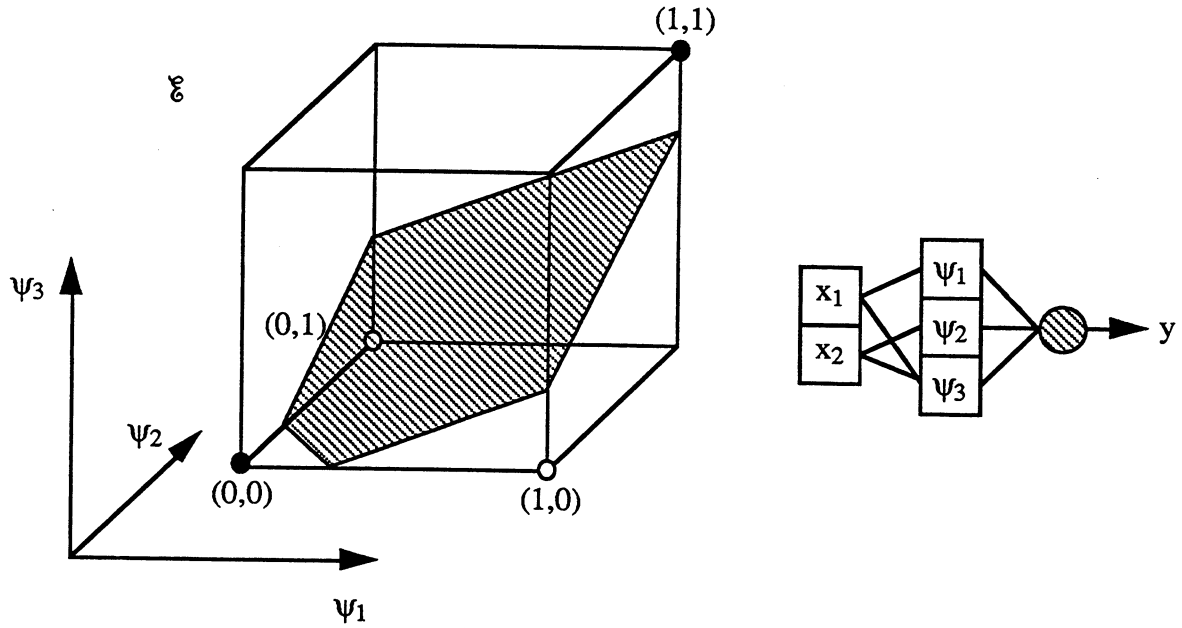


Fig. 1.11. Le Perceptron construit une séparation linéaire dans un espace transformé, celui de la couche cachée des cellules associatives qui est construite "manuellement" pour rendre les formes séparables.

La solution proposée par l'Adaline est légèrement différente ; elle consiste à ajouter une couche au dessus de la couche adaptative, et non pas en dessous comme le fait le Perceptron. Ces modèles combinant plusieurs Adalines pour résoudre des tâches plus complexes sont appelés Madaline (Multiple Adaline). Dans le cas du XOR, le problème de l'apprentissage est décomposé en deux sous-problèmes : $E_1=\{((0,0)^t,-1), ((1,1)^t,-1), ((0,1)^t,1), ((1,0)^t,-1)\}$ et $E_2=\{((0,0)^t,-1), ((1,1)^t,-1), ((0,1)^t,-1), ((1,0)^t,1)\}$. Deux Adalines sont utilisées pour apprendre ces ensembles dans $\mathcal{E}=\{0,1\}^2$ et leurs sorties y_1 et y_2 sont combinées, dans une couche supérieure (Fig 1.12), par un autre automate à seuil dont la sortie y fournit les réponses désirées aux formes de E :

$$y = \mathbb{1}(y_1 + y_2) \qquad (1.12)$$

Les poids de ce dernier automate sont bien évidemment fixés "manuellement", tout comme le sont les fonctions de transition ψ_i des cellules associatives du Perceptron.

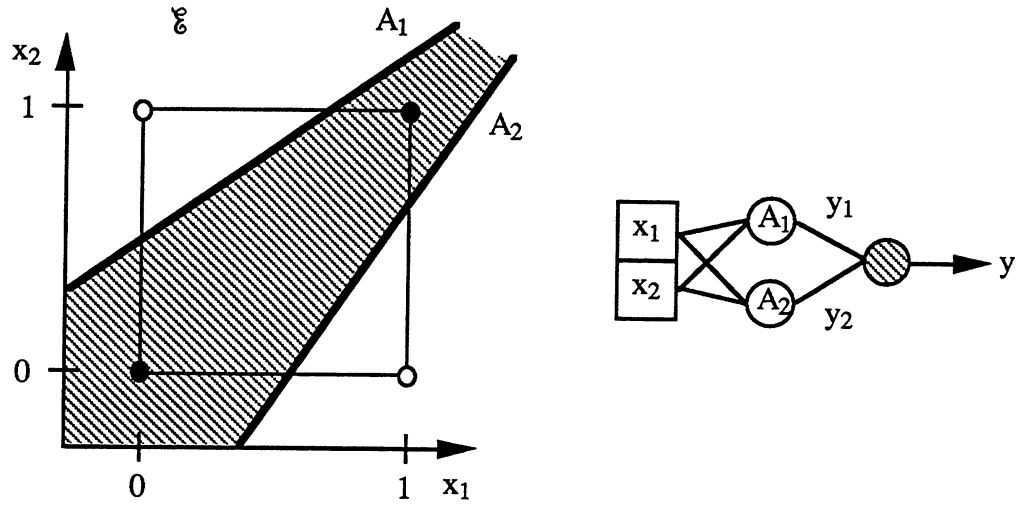


Fig 1.12. Le réseau Madaline construit des séparations linéaires directement dans l'espace des formes grâce à une couche cachée d'Adalines. Puis des régions de décision plus complexes sont "assemblées" dans la couche de sortie, construite "manuellement".

Ces deux exemples illustrent bien la nécessité de disposer de réseaux à plusieurs couches, au moins dans le cas de réseaux d'automates quasi-linéaires. Ils montrent aussi les deux différents angles sous lesquels le rôle des couches cachées peut être appréhendé.

On peut en effet considérer que les réseaux multi-couches élaborent, de couche cachée en couche cachée, des représentations de plus en plus discriminantes des formes qu'ils ont à apprendre. La dernière couche n'est alors qu'une couche de décision permettant de passer d'un espace où les classes sont clairement séparées à l'espace final, celui des classes [PAO 89]. C'est un peu l'approche du Perceptron dont la couche associative a pour but de trouver les "bonnes" primitives.

On peut également considérer qu'un réseau multi-couche "travaille" directement dans l'espace des formes qui lui sont soumises. Les automates quasi-linéaires ne permettant que de construire des frontières de décision simples (des hyperplans), il est alors nécessaire de disposer de plusieurs couches afin d'assembler ces frontières en frontières plus complexes seules capables de s'adapter aux régions de classe de l'espace des formes, régions qui peuvent être *a priori* quelconques (non convexes, non connexes, ...) [LIPPMAN 87]. C'est l'approche Madaline pour résoudre le XOR.

Ces deux vues sont évidemment aussi valides l'une que l'autre (et l'on peut passer indifféremment de l'une à l'autre). Mais bien que l'Adaline et le Perceptron aient intégré chacun à leur manière la notion de couche cachée, ni l'un ni l'autre n'ont proposé de méthode automatique pour gérer ces multiples couches. Il aura fallu attendre plusieurs années avant que ne soit proposé un algorithme générique permettant l'apprentissage dans une architecture en couches quelconque. Et si la vraie paternité de cet algorithme est très disputée, sa filiation avec l'algorithme de Widrow-Hoff est elle, en revanche, indiscutable : la *rétro-propagation du gradient* dans les réseaux multi-couches est une généralisation de la règle du delta.

1.2.4 Réseaux multi-couches et rétro-propagation du gradient

L'écueil sur lequel ont buté les tentatives de conception d'une méthode d'apprentissage dans les réseaux multi-couches est celui du *credit assignment* [MINSKY 69] : quel critère utiliser pour adapter les poids des cellules cachées ? On dispose, dans une situation d'apprentissage supervisé, de valeurs cibles que les cellules de sortie doivent atteindre en réponse aux formes d'entrée, et l'apprentissage consiste, nous l'avons vu pour le Perceptron et l'Adaline, à minimiser l'écart entre les sorties réelles des cellules et ces valeurs cibles. Mais quel objectif se fixer pour des cellules cachées dont on ne peut *a priori* prédire dans quelle mesure leur état est utile à la reconnaissance de telle ou telle forme ?

Ce problème a été résolu par l'algorithme dit de "rétro-propagation du gradient" qui fournit des valeurs cibles aux cellules cachées en propageant un "signal d'erreur" des cellules de sortie vers les cellules cachées.

1.2.4.1 Architecture multi-couches

Nous nous situons toujours dans le cas d'un problème de classification à **p** classes où l'on tente d'apprendre au réseau à associer, à chaque forme $X_i \in \mathbb{R}^n$ d'un ensemble d'apprentissage E de cardinalité N, une forme $Y_i \in \{-1,1\}^p$ telle que $y_{ik}=1$ si X_i est de classe **k** et $y_{ik}=-1$ sinon.

Nous disposons pour cela d'un réseau d'automates linéaires généralisés à **n** cellules d'entrée (comme auparavant, chaque cellule d'entrée **i** ne reçoit de l'extérieur qu'une seule valeur, la $i^{ème}$ composante de la forme X présentée en entrée, et se contente de recopier cette valeur sur sa connexion sortante) et à **p** cellules de sortie sur lesquelles la réponse Y(X) du réseau sera lue. Hormis ces contraintes, l'architecture du réseau peut être tout à fait quelconque, pourvu que :

- Le graphe d'interconnexion puisse être mis en niveaux (§ 1.1.2.1).
- Les fonctions de transition de tous les automates soient dérivables.

Soit L le nombre de couches du réseau (niveaux 0 à L-1 du graphe d'interconnexion) ; on supposera pour plus de clarté que toutes les cellules d'entrée appartiennent à la couche 1 et que toutes les cellules de sortie appartiennent à la couche L. On notera A_l l'ensemble des automates de la couche l, w_{kj} le poids d'une connexion allant de la cellule **j** à la cellule **k**, CE_k l'ensemble des automates dont la sortie est reliée aux entrées de l'automate **k** et CS_k l'ensemble des automates recevant la sortie de **k**.
On supposera également l'existence d'une cellule 0, appartenant à A_1, connectée à tous les automates des couches supérieures ($0 \in CE_k, \forall k \in A_l \ / \ l > 1$) et dont la sortie y_0 est constante et vaut 1 (les poids des connexions émanant de cette cellule joueront le rôle de seuil pour les automates "receveurs").
Une forme X_i étant présentée au réseau, on définit l'entrée "nette"[1] d'une cellule **k**, $k \in A_l$, $l > 1$, comme étant la somme pondérée des signaux arrivant sur ses connexions entrantes, $net_k(X_i)$:

$$net_k(X_i) = \sum_{j \in CE_k} w_{kj} . y_j(X_i) \tag{1.13}$$

et la sortie de cette cellule, $y_k(X_i)$, vaut :

$$y_k(X_i) = f_k(net_k(X_i)) \tag{1.14}$$

où f_k est la fonction de transition, dérivable, de l'automate **k**. Pour les cellules de la couche d'entrée, on a simplement $y_0(X_i)=1$ et $y_k(X_i)=x_{ik}$ pour $k \in A_1$, $k>0$.

Comme nous l'avons déjà précisé, le calcul des sorties des automates, lorsqu'une forme est présentée au réseau, doit se faire séquentiellement entre couches, mais peut être réalisé en parallèle à l'intérieur d'une même couche :

Pour l variant de 1 à L faire :
Pour tout $k \in A_l$ faire (en parallèle) : évaluer y_k

---

[1] "nette" par opposition à "brut", c'est-à-dire l'entrée globale.

1.2.4.2 Apprentissage par rétro-propagation du gradient

Le principe d'apprentissage dans un tel réseau est strictement identique à celui proposé pour l'Adaline : il s'agit de minimiser l'erreur commise par le réseau sur l'ensemble d'apprentissage E en adaptant les poids des connexions entre cellules, W. Cette optimisation est à nouveau réalisée itérativement par une descente en gradient sur une fonction coût C(W) qui n'est autre que l'erreur quadratique moyenne de l'Adaline, moyenne des erreurs $C_i(W)$ commises sur chaque forme de E :

$$C(W) = \frac{1}{N}\sum_{i=1}^{N} C_i(W) = \frac{1}{N}\sum_{i=1}^{N} \| Y(X_i) - Y_i \|^2 = \frac{1}{N}\sum_{i=1}^{N} \left[\sum_{k \in A_L} (y_k(X_i) - y_{ik})^2 \right] \qquad (1.15)$$

L'algorithme d'apprentissage est donc identique à celui déjà présenté (§ 1.2.1.1), toute la difficulté consistant ici à calculer le gradient de C par rapport aux poids des cellules cachées. Ce problème semble avoir été résolu indépendamment par plusieurs personnes, à différentes époques, la plus large diffusion de ce résultat pouvant être attribuée à Rumelhart, Hinton et Williams à travers l'un des livres de référence du connexionnisme, *Parallel Distributed Processing* [RUMELHART 86].

Considérons un poids élémentaire d'une cellule **k**, w_{kj}. Pour corriger ce poids selon le gradient de C, il faut calculer la dérivée partielle de C par rapport à w_{kj}. Ce calcul est simple pour les cellules de sortie mais s'avère complexe pour les cellules cachées, sauf si l'on passe par le gradient de C par rapport à l'entrée "nette" de la cellule, net_k. On obtient ainsi :

$$\frac{\partial C}{\partial w_{kj}} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial C_i}{\partial w_{kj}} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial C_i}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}} \qquad (1.16)$$

D'après (1.13) on déduit :

$$\frac{\partial net_k}{\partial w_{kj}}(X_i) = y_j(X_i) \qquad (1.17)$$

Pour une cellule de sortie, $k \in A_L$, il est facile de voir d'après (1.15) que :

$$\frac{\partial C_i}{\partial net_k} = 2.(y_k(X_i) - y_{ik}).\frac{\partial f_k}{\partial net_k} = 2.(y_k(X_i) - y_{ik}).f_k'(net_k(X_i)) \qquad (1.18)$$

Pour une cellule cachée, $k \in A_l$, $1 < l < L$, le terme $\partial C_i/\partial net_k$ peut être calculé en l'exprimant en fonction des termes $\partial C_i/\partial net_s$ où $s \in CS_k$:

$$\frac{\partial C_i}{\partial net_k} = \sum_{s \in CS_k} \frac{\partial C_i}{\partial net_s} \frac{\partial net_s}{\partial net_k} \tag{1.19}$$

et d'après (1.13) et (1.14), on obtient :

$$\frac{\partial net_s}{\partial net_k} = \frac{\partial net_s}{\partial y_k} \frac{\partial y_k}{\partial net_k} = w_{sk}.f_k'(net_k(X_i)) \tag{1.20}$$

Finalement, en associant (1.19) et (1.20), on peut déterminer $\partial C_i/\partial net_k$ pour toute cellule cachée **k** :

$$\frac{\partial C_i}{\partial net_k} = f_k'(net_k(X_i)) \sum_{s \in CS_k} w_{sk}.\frac{\partial C_i}{\partial net_s} \tag{1.21}$$

La correction d'un poids w_{kj} pour une forme X_i est donc proportionnelle au terme $\delta_{ik}=\partial C_i/\partial net_k$ (expression 1.16). Ce terme se calcule simplement pour le poids d'une cellule de sortie (expression 1.18), et il s'exprime, pour les poids d'une cellule cachée **k**, en fonction des δ_{is} associés aux cellules **s** auxquelles la sortie de **k** est connectée (expression 1.21). Ainsi, les poids de la cellule **k** ne peuvent être corrigés qu'après que toutes les cellules **s** de CS_k aient vu leurs poids adaptés et par construction, ces cellules ne peuvent appartenir qu'à des couches de niveau supérieur à celui de **k**.

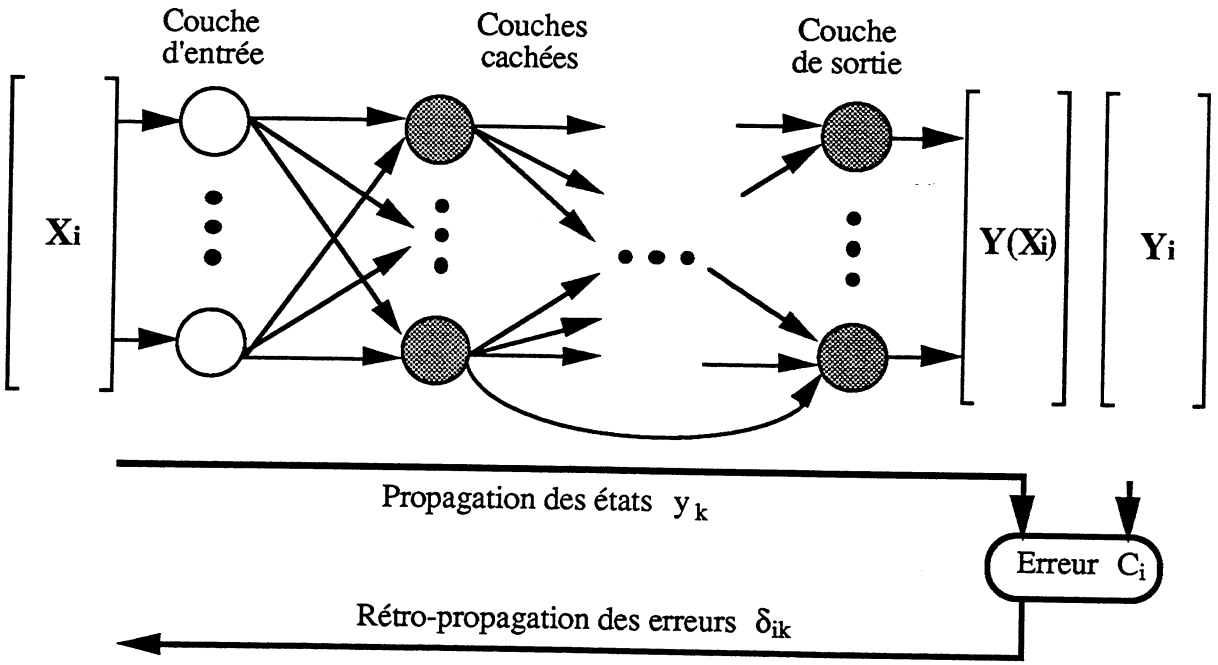


Fig. 1.13. Flots d'information dans un réseau en couches

La correction des poids peut donc s'effectuer si l'on commence par les poids des cellules de sortie pour poursuivre, couche après couche, dans l'ordre décroissant des niveaux. Alors qu'en phase normale de fonctionnement, les états des cellules y_k sont propagés dans le sens croissant des niveaux, pendant la phase de modification des poids, les signaux de correction δ_{is} sont eux *rétro-propagés* dans le sens inverse (Fig. 1.13)... d'où la dénomination de l'algorithme, algorithme que l'on peut résumer ainsi :

Etape 0 :	$t = 0$
	Initialiser aléatoirement les poids $w_{kj}(t)$
Etape 1 :	$\Delta w_{kj}(t) = 0$
	Pour i variant de 1 à N faire
	/* Propagation */
	Calculer la réponse du réseau à la forme X_i, $Y(X_i)$
	/* Rétro-propagation */
	Calculer la correction de chaque poids $w_{kj}(t)$ due à la forme X_i : $\partial C_i/\partial w_{kj}$ et l'ajouter aux corrections précédentes : $\Delta w_{kj}(t) = \Delta w_{kj}(t) + \partial C_i/\partial w_{kj}$
	Fin pour
Etape 2 :	Modifier chaque poids par : $w_{kj}(t+1) = w_{kj}(t) - \alpha(t).\Delta w_{kj}(t)/N$
Etape 3 :	$t = t+1$
	Si condition d'arrêt non remplie, aller à l'étape 1.

Comme dans le cas de l'Adaline, il existe une version stochastique de l'algorithme de descente en gradient, puisqu'ici aussi la correction apportée aux poids est une moyenne des corrections émanant de chaque forme de E (1.16). Dans cette version, les poids sont modifiés après chaque présentation d'une forme X_i selon la correction due à celle-ci ($\partial C_i/\partial w_{kj}$), au lieu d'être modifiés après présentation de tout l'ensemble E selon la moyenne des corrections.

1.2.4 Les réseaux multi-couches, ultime solution ?

La rétro-propagation du gradient, en fournissant un algorithme général d'apprentissage pour réseaux multi-couches, a permis d'envisager le développement d'applications qui semblaient jusque-là interdites aux réseaux de neurones par les limitations des modèles à deux couches, Adaline ou Perceptron. Ceci a bien sûr été illustré par le problème académique du XOR qui a pu enfin être résolu "automatiquement" [RUMELHART 86] par un réseau à trois couches "entraîné" par rétro-propagation. Ce succès est venu relancer l'intérêt pour les méthodes connexionnistes que le livre de Minsky et Papert [MINSKY 69] avait mis pour un

temps en sommeil.

Ceci explique sans doute le fait qu'un nombre important d'applications, particulièrement en reconnaissance des formes, aient utilisé de tels réseaux dont le caractère générique les rend aptes à s'accomoder de différentes situations. Mais ce caractère générique est aussi une source de problèmes et la mise en œuvre pratique d'un réseau multi-couche peut ne pas être aussi directe qu'elle n'y paraît... Nous aborderons ce sujet dans le second chapitre.

Les capacités de ces réseaux ne sauraient non plus masquer l'existence d'autres solutions "neuronales", de nature différente, mais dont les succès sont également à l'origine du renouveau connexionniste. Les réseaux dynamiques de Hopfield en sont un exemple.

1.3. RESEAUX DYNAMIQUES

Comme nous l'avons déjà dit, les réseaux dynamiques sont des réseaux dont les graphes d'interconnexion contiennent des circuits et qui de ce fait ont un fonctionnement plus complexe que les réseaux en couches que nous venons de voir. Cette complexité autorise en contrepartie un éventail de comportements plus large et on leur attribue en conséquence des capacités plus grandes [WASSERMAN 89], tout en omettant de caractériser précisément cette prétendue supériorité. Dans le domaine qui nous intéresse, celui de la reconnaissance de formes, ces réseaux sont en général peu utilisés ; les problèmes d'optimisation combinatoire seront plus volontiers attaqués par de tels réseaux. Néanmoins, c'est souvent par une application de reconnaissance de caractères que l'on illustre l'un des modèles dynamiques les plus connus : celui de Hopfield.

1.3.1 Réseau de Hopfield

1.3.1.1 Architecture et fonctionnement

Un réseau de Hopfield est composé de **n** cellules, totalement interconnectées les unes avec les autres : chaque cellule **k** émet sa sortie y_k vers toutes les cellules **j** du réseau à travers une connexion de poids $w_{jk} \in \mathbb{R}$ (Fig. 1.14).

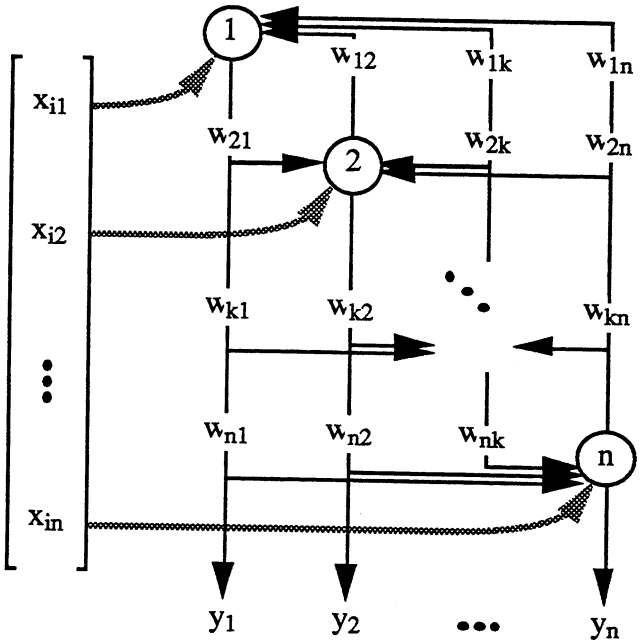


Fig. 1.14. Architecture d'un réseau de Hopfield

Les poids du réseau peuvent ainsi être représentés par une matrice W de dimension (nxn). Dans le modèle original [HOPFIELD 82] que nous présentons ici, les cellules sont des automates à seuil et un automate **k** ne peut être connecté à lui-même (la diagonale de W est nulle). Sa sortie y_k vaut donc :

$$y_k(t+1) = \mathbb{1}\left(\sum_{j \neq k} [w_{kj}.y_j(t)] - \theta_k\right) \tag{1.22}$$

où $\theta_k \in \mathbb{R}$ est le seuil de l'automate k. D'autres versions du réseau de Hopfield [HOPFIELD 84] utilisent des cellules à états continus (la fonction seuil est remplacée par une fonction de transition sigmoïdale).

Toutes les cellules sont à la fois d'entrée et de sortie. Une forme d'entrée X doit être constituée ici d'un vecteur de $\{-1,1\}^n$. Présenter cette forme au réseau consiste à imposer la valeur de sortie des cellules au temps 0 : $y_k(0) = x_k$. La réponse à cette forme sera également un vecteur de $\{-1,1\}^n$, Y(X), composé des sorties $y_k(t^*)$, où t^* correspond au temps mis par le réseau pour atteindre un état stable ($y_k(t^*)=y_k(t^*- 1)\ \forall k$) au terme de sa relaxation selon la dynamique choisie.

Un tel réseau sera donc utilisé comme auto-associateur. L'apprentissage consiste ici à construire un réseau qui puisse associer à elle-même chacune des N formes $X_i \in \{-1,1\}^n$ d'un

ensemble d'apprentissage E. On espère également que le réseau construit pourra associer à un exemplaire "bruité" d'une forme de E, $\underline{X_i}$, son prototype original, X_i. Ceci permettra au réseau de fonctionner en classifieur, une forme inconnue se voyant attribuer la classe du prototype de E que le réseau lui a associé, ou éventuellement le label "non reconnue" si la réponse du réseau ne correspond pas à un prototype de E. Dans tous les cas, ceci exigera du réseau qu'il ait fourni une réponse, c'est-à-dire convergé vers un état stable (ou attracteur, point fixe). L'apprentissage doit donc trouver l'ensemble des poids W^* et la dynamique de fonctionnement qui assureront au réseau le comportement décrit ci-dessus.

1.3.1.2 Convergence

Hopfield a montré [HOPFIELD 82] qu'un tel réseau convergeait toujours vers un état stable si :

- La matrice des poids W était symétrique : $w_{kj} = w_{jk}\ \forall\ j \neq k$
- Les cellules étaient mises à jour selon une dynamique séquentielle aléatoire (cf § 1.1.2.2), toutes les cellules étant équiprobables.

Ceci tient au fait que l'état du réseau à un instant **t** peut être caractérisé par une fonction réelle H(t) (appelée énergie par analogie avec les verres de spin de la physique) dont on peut démontrer, sous les hypothèses précédentes, qu'il s'agit d'une fonction de Lyapunov. Ceci se traduit par : il existe une fonction H admettant un minimum absolu et qui ne peut que décroître entre deux itérations du réseau. Cette fonction est ici la suivante :

$$H(t) = -\sum_{k=1}^{n}\sum_{j=1}^{n} w_{kj}.y_k(t).y_j(t) + 2\sum_{k=1}^{n} \theta_k.y_k(t) \qquad (1.23)$$

Si au cours d'une itération, une cellule **k** change d'état ($y_k(t+1) = -y_k(t)$), la variation de H qui peut être imputée à ce changement, ΔH_k, vaut :

$$\Delta H_k(t) = 2.(y_k(t) - y_k(t+1)).\left[\sum_{j=1}^{n} w_{kj}.y_j(t) - \theta_k\right] \qquad (1.24)$$

La quantité entre crochets dans (1.24), net_k, est par définition (1.22), du même signe que $y_k(t+1)$, alors que $(y_k(t) - y_k(t+1))$ est de signe opposé. La variation de H est donc strictement

négative[1] si la cellule élue par la dynamique change d'état, et elle est bien sûr nulle si cette cellule reste inchangée.

Des résultats de même nature ont été démontrés dans le cas de dynamiques différentes [GOLES 87].

Un réseau de Hopfield va donc converger, en mode séquentiel aléatoire, quel que soit son état initial (c'est-à-dire quelles que soient les valeurs $y_k(0)$) et les états stables qu'il pourra atteindre correspondront aux minima de H. Tout le problème consiste donc à trouver une matrice des poids W symétrique qui rende la fonction H minimale pour les formes de l'échantillon d'apprentissage.

1.3.1.3 Apprentissage Hebbien

La méthode la plus simple pour déterminer la matrice W des poids d'un réseau de Hopfield consiste à prendre la matrice X de dimension (nxN) dont les colonnes sont les N formes de l'ensemble d'apprentissage E, à la multiplier par sa transposée, X^t, et à annuler les termes diagonaux [KOHONEN 84] :

$$w_{kj} = \begin{cases} \sum_{i=1}^{N} x_{ik}.x_{ij} & \text{si } k \neq j \\ 0 \text{ sinon} \end{cases} \qquad (1.25)$$

Les poids θ_k peuvent être pris nuls.

Cette solution peut être appréhendée de deux manières. Informellement, elle correspond à la *loi de renforcement synaptique* proposée par Hebb [HEBB 49] comme modèle de l'adaptation des connexions entre neurones *biologiques*. Celle-ci se formule, dans notre cadre, de la manière suivante :
Si deux cellules **k** et **j** doivent avoir le même état ($y_k=y_j$), alors les poids qui les relient (w_{kj} et w_{jk}, égaux) devraient être positifs afin que chaque cellule "attire" l'autre vers un état identique au sien.
Au contraire, si les cellules doivent être "opposées", leurs connexions devraient porter des poids négatifs pour que les deux cellules se "repoussent" vers des états différents.

---

[1] Il est toujours possible, en jouant sur les poids θ_k, de rendre net_k non nulle sans pour autant modifier la sortie de la cellule **k** [GOLES 87].

Pour une forme X_i, on va donc adapter les poids selon la règle $w_{kj}=w_{kj}+x_{ik}.x_{ij}$, $x_{ik}.x_{ij}$ tendant à rendre w_{kj} positif si $x_{ik}=x_{ij}$ et négatif sinon (car x_{ik} et x_{ij} appartiennent à $\{-1,1\}$). Ceci correspond bien, en partant de poids nuls et après prise en compte de toutes les formes de E, à la solution exprimée en (1.25).

Une justification plus "mathématique" peut être tirée de la théorie de l'apprentissage linéaire dans laquelle nous n'entrerons pas en détail (voir [KOHONEN 84]). De manière abrupte, le problème que l'on s'y pose est celui de la résolution de l'équation matricielle :

$$Y = WX \tag{1.26}$$

où les colonnes de X sont les formes d'apprentissage X_i et les colonnes de Y sont les réponses à associer aux X_i, Y_i. Dans le cas de l'auto-association dans lequel se situent les réseaux de Hopfield, Y=X.

Une telle association linéaire peut être mise en œuvre par un réseau à deux couches ; un réseau de Hopfield revient quant à lui à itérer ce processus d'association, avec le mécanisme de seuillage en plus, ce qui rend l'analogie assez vague...

Néanmoins, la solution à ce problème permet "d'éclairer" l'apprentissage Hebbien sous un autre angle. En effet, dans le cas de l'auto-association, une solution à (1.26) est [KOHONEN 84] :

$$W = XX^{+} \tag{1.27}$$

où X^{+} est la pseudo-inverse de X. Or quand les X_i sont orthonormés ($X_i.X_j=\delta_{ij}$), on a justement $X^{+} = X^{t}$ et l'apprentissage linéaire coïncide alors avec la procédure d'apprentissage de Hopfield, à la contrainte de la diagonale nulle près. La règle empirique de Hebb a donc de lointaines fondations mathématiques, même si celles-ci ne permettent pas directement de démontrer la validité de la procédure d'apprentissage du modèle de Hopfield, celui-ci n'étant pas une mémoire associative linéaire au sens strict du terme.

1.3.1.4 Capacités d'un réseau de Hopfield

Après avoir déterminé les poids selon le processus décrit précédemment, le réseau doit avoir pour états stables les formes d'apprentissage X_i. Ceci n'est en fait vérifié que si deux conditions sont remplies.

La première impose que le nombre N de formes que l'on veut apprendre ne soit pas trop important en regard du nombre **n** de cellules du réseau. Hopfield a montré, en utilisant des formes générées aléatoirement et sous certaines conditions, que N ne devait pas excéder 0.14**n**. Pratiquement, cette restriction est plutôt gênante car, même en supposant que l'on arrive à caractériser chaque classe par un seul prototype X_i, le nombre de cellules nécessaires pour "mémoriser" ces formes, et donc la dimension des vecteurs X_i, devra être environ sept fois celle du nombre de classes. Si l'on désire par exemple reconnaître les 26 caractères d'un alphabet romain, environ 180 cellules seront nécessaires, toutes interconnectées par plus de 32000 liaisons pondérées. Ceci est à comparer par exemple aux 26 cellules qui seraient nécessaires à un Perceptron pour remplir la même tâche, pour peu que les formes qui lui sont soumises soient linéairement séparables.

C'est justement sur la nature des formes à apprendre que porte la seconde limitation du réseau de Hopfield tel qu'il est présenté ici. Si celles-ci ne doivent pas être linéairement séparables, elles devront en revanche être orthogonales deux à deux pour que le réseau puisse les "mémoriser". Cette contrainte est à relier à l'explication de la loi de Hebb en termes d'apprentissage linéaire : ces deux méthodes ne coïncident que lorsque les formes sont orthonormées...

Lorsque ces deux obligations ne sont pas respectées, les formes mémorisées par le réseau risquent de ne pas correspondre exactement aux formes d'apprentissage. De plus, la capacité de généralisation du réseau est également compromise par de telles violations.

Cette capacité de généralisation est d'ailleurs intrinsèquement limitée. Une forme $\underline{X_i}$ ne faisant pas partie de l'ensemble d'apprentissage ne sera reconnue que si elle se trouve dans le *bassin d'attraction* d'une forme apprise X_i (Fig. 1.15). Ce bassin, de forme imprévisible, est en général limité à un petit voisinage, au sens de la distance de Hamming[1], autour de X_i. Si la forme $\underline{X_i}$ est "très" éloignée de X_i, elle risque de ne pas être reconnue, même si elle est "beaucoup" plus proche de X_i que de tous les autres $X_j \in E$. Dans ce cas, le réseau lui associera une forme qui ne correspondra pas à un prototype de E, mais simplement à un minimum de l'énergie H, un "faux attracteur" dans le bassin duquel se trouvait $\underline{X_i}$.

---

[1] La distance de Hamming entre deux vecteurs de $\{-1,1\}^n$ est le nombre de composantes par lesquelles ces vecteurs diffèrent.

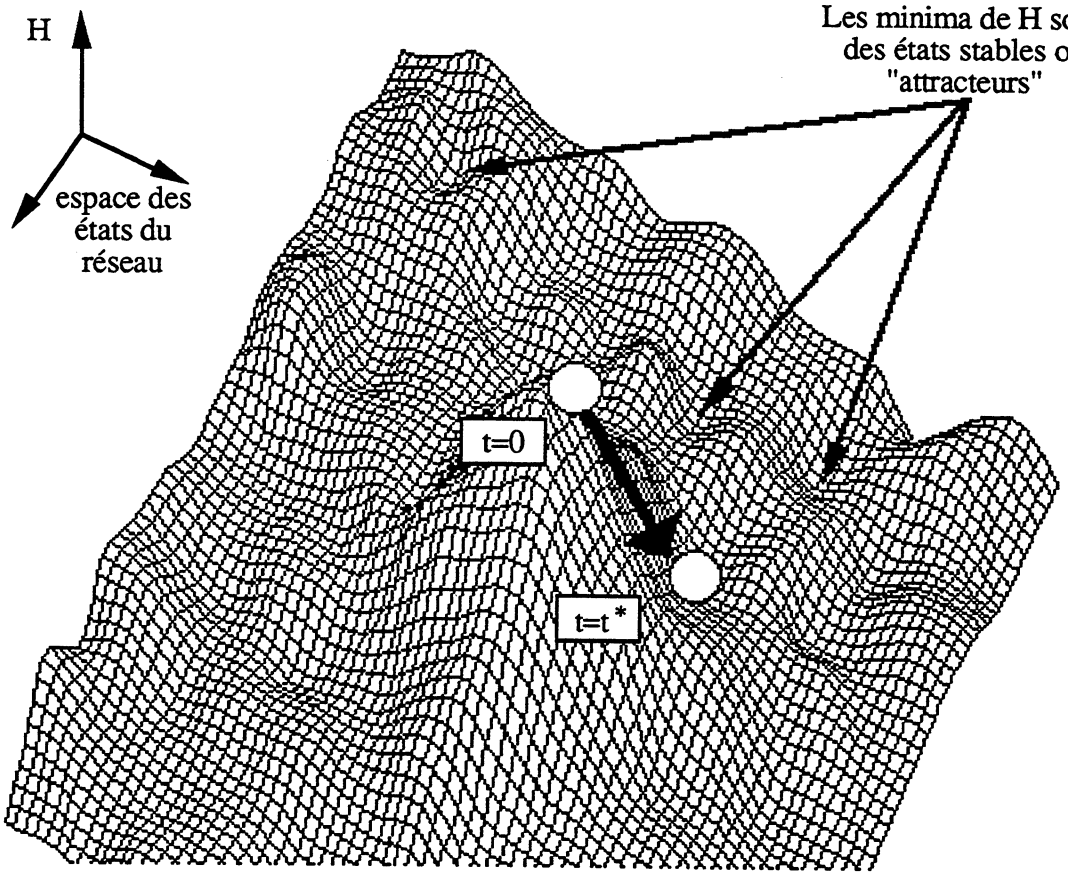


Fig 1.15. L'état du réseau peut être vu comme un ballon posé sur un relief d'énergie. La dynamique du réseau minimisant cette énergie, le ballon va, depuis l'endroit où il a été initialement posé, rouler vers le minimum du bassin d'attraction dans lequel il se trouve. L'apprentissage consiste à "creuser" des bassins dans le relief aux endroits correspondant aux formes d'apprentissage.

Mais bien que certaines solutions permettent de s'affranchir de ces difficultés, les modèles de Hopfield, même plus sophistiqués que celui présenté ici, restent peu adaptés à la reconnaissance de formes. En effet, si la capacité de généralisation recherchée pour une application se limite simplement à des notions de proximité par rapport à un ensemble de formes de référence, alors il existe d'autres modèles connexionnistes capables d'implémenter plus efficacement un schéma de classification du type "plus proche voisin". Il s'agit des réseaux à prototypes que nous verrons par la suite. Des réseaux de Hopfield peuvent néanmoins être utilisé pour faire de la correction d'erreur en amont de la classification.

Mais l'aspect le plus intéressant des réseaux dynamiques réside en fait dans leur capacité à converger rapidement vers un minimum d'énergie. C'est cette propriété qui est, en pratique, mise à contribution pour résoudre des problèmes d'optimisation. Toute la difficulté consiste alors à "coder" le problème en termes de réseau. Les variables doivent correspondre aux états

des cellules, et les contraintes, ainsi que la fonction coût à optimiser, doivent être codées dans les poids de telle sorte qu'un état d'énergie minimale corresponde à une solution de faible coût respectant les contraintes. La traduction d'un problème d'optimisation "en réseau de neurones" est un mécanisme bien maîtrisé qui a déjà été appliqué à plusieurs "classiques" de l'optimisation combinatoire : voyageur de commerce [HOPFIELD 85], emploi du temps, ... (on pourra trouver une synthèse dans [PERETTO 90]).

Toujours dans le domaine qui nous intéresse, cette technique a par exemple été utilisée pour résoudre le problème de la mise en correspondance de deux ensembles de segments de droites représentant des caractères [LEUNG 88]. Une cellule est utilisée pour chaque appariement de deux segments pouvant être réalisé entre les deux ensembles. Sa sortie (continue, ici) indique la probabilité de l'appariement. Le poids entre deux cellules permet de coder la compatibilité des deux appariements qu'elles représentent. Le réseau, en se relaxant, sert à trouver un ensemble de correspondances ayant une incompatibilité minimale. Ceci pour affirmer encore qu'un réseau de Hopfield ne peut être utilisé dans un problème de reconnaissance de formes que si ce dernier est formulé en termes d'optimisation...

1.3.2 Machine de Boltzmann

Il est difficile d'évoquer les réseaux dynamiques en passant sous silence la *machine de Boltzmann* proposée par Hinton et Sejnowski en 1983 [HINTON 83]. Cette appellation est d'abord apparue dans le cadre d'un algorithme d'apprentissage avant de désigner de manière plus générale des réseaux dynamiques stochastiques implémentant la méthode d'optimisation combinatoire du *recuit simulé* [SAMUELIDES 89].

1.3.2.1 Relaxation stochastique et recuit simulé

Nous n'entrerons pas dans les détails de ces machines complexes (voir [HINTON 86]). En bref, il s'agit de réseaux dynamiques composés d'automates dont la sortie, binaire, est le résultat d'un processus stochastique. La dynamique qui anime un tel réseau est évidemment particulière. Une des plus utilisées est celle de Métropolis que nous décrivons ici.

Tout comme pour un réseau de Hopfield, une fonction d'énergie H est définie sur le réseau. Une cellule **k**, élue par la dynamique asynchrone à un instant **t**, peut changer d'état si la variation d'énergie ΔH_k qui en résulte est négative ($H(t+1) < H(t)$). Dans le cas contraire, la cellule peut également changer d'état avec une probabilité p_k définie ainsi :

$$p_k = \exp\left(\frac{-\Delta H_k}{T(t)}\right) \tag{1.28}$$

où T(t) dénote la "température" du système à l'instant **t**. Encore une fois, ce terme de température provient d'une analogie avec la physique, tout comme le processus de recuit simulé [KIRKPATRICK 83] qui est utilisé pendant la relaxation du réseau.

En effet, le recuit simulé est une méthode utilisée en métallurgie pour obtenir des matériaux très purs. Elle consiste à chauffer un cristal et à diminuer lentement sa température, ce qui permet d'atteindre un état d'énergie minimale de la matière.

De la même manière, la température T(t) sera lentement réduite au cours des itérations d'une machine de Boltzmann. A haute température, la probabilité p_k est importante, et l'énergie du système est donc susceptible d'augmenter, ce qui permettra éventuellement de s'échapper d'un bassin d'attraction correspondant à un minimum local, pour aller vers un "meilleur" minimum. Au fur et à mesure que la température décroît, cette probabilité diminue et finalement le réseau reste captif d'un minimum de H que l'on espère global, ou proche d'un minimum global (en fait, il a été démontré que la probabilité pour que ce minimum soit global tendait vers 1 lorsque le nombre d'itérations tendait vers l'infini...). Le terme "machine de Boltzmann" provient de ce que la probabilité p_α d'un état stable α est une distribution de *Gibbs-Boltzman* : à la température T, pour deux états stables α et β, d'énergies respectives H_α et H_β, on a [SAMUELIDES 89] :

$$\frac{p_\alpha}{p_\beta} = \exp\left(-\frac{H_\alpha - H_\beta}{T}\right) \tag{1.29}$$

Le recuit simulé est donc une technique qui permet au réseau d'atteindre des états d'énergie minimale proches d'un minimum global. Ceci est évidemment très important dans les problèmes d'optimisation pour lesquels les réseaux de Hopfield, qui ne garantissent que des minima locaux, peuvent être insuffisants.

1.3.2.2 Apprentissage dans une machine de Boltzmann

En reconnaissance de formes, la machine de Boltzmann est également utilisée et un algorithme d'apprentissage spécifique a été proposé par Hinton et Sejnowski [HINTON 86].

Contrairement aux réseaux de Hopfield, une machine de Boltzmann peut contenir des

unités cachées. Les autres unités, visibles, sont soit d'entrée, soit de sortie, permettant ainsi de réaliser des tâches d'hétéro-association : après apprentissage, une forme est présentée en fixant l'état des cellules d'entrée et la réponse du réseau est fournie, après relaxation, par les états des cellules de sortie.

L'architecture du réseau peut être quelconque mais si une cellule **k** est connectée à une cellule **j**, par un poids w_{kj}, alors la connexion symétrique doit exister et $w_{jk}=w_{kj}$.

L'apprentissage de ces poids se fait en deux étapes et repose sur un modèle probabiliste : on désire que les cellules du réseau s'activent selon une loi de probabilité prédéterminée (par l'ensemble d'apprentissage). Pour cela, en partant de poids aléatoires, on observe le réseau pendant une phase dite "contrainte" où les états des cellules d'entrée et de sortie sont fixés (par les couples entrée/sortie de l'ensemble d'apprentissage). Les mêmes observations sont réalisées avec le réseau en phase dite "libre" où les états des cellules de sortie ne sont plus imposés. Le principe est de minimiser une mesure de distance G (mesure de Küllback issue de la théorie de l'information) entre la loi de probabilité des unités visibles en phase contrainte, P^+, et la même loi en phase libre P^- :

$$G(P^+,P^-) = \int P^+(\alpha).\mathrm{Log}\left(\frac{P^+(\alpha)}{P^-(\alpha)}\right) d\alpha \qquad (1.30)$$

Cette minimisation est, ici encore, réalisée par une descente en gradient sur les poids et il est montré [HINTON 86] que, quelles que soient deux cellules **k** et **j** reliées (cachées ou visibles), on a :

$$\frac{\partial G}{\partial w_{kj}} = -\frac{1}{T}\left(p^+_{kj} - p^-_{kj}\right) \qquad (1.31)$$

où p_{kj}^+ est la probabilité, mesurée en moyenne sur tout l'ensemble d'apprentissage, pour que les sorties des cellules **k** et **j** soient simultanément à l'état 1 lorsque le réseau a atteint un état stable en phase contrainte, à la température finale T du recuit simulé ; p_{kj}^- est la même probabilité correspondant à la phase libre.

Il est donc nécessaire d'estimer les valeurs de p_{kj}^+ et p_{kj}^- en faisant fonctionner le réseau plusieurs fois en phases contrainte et libre. Or si l'ensemble d'apprentissage comporte N formes, chacune présentée M fois pour chaque phase libre et contrainte, une seule itération de l'algorithme pour modifier les poids requièrera 2NM recuits simulés... autant dire que les temps

d'apprentissage sont prohibitifs.

Une machine de Boltzmann, légèrement différente de celle présentée ici, a été utilisée pour la reconnaissance de silhouettes de navires [AZENCOTT 90]. Les temps d'apprentissage rapportés sont de l'ordre de 2 heures CPU... sur une *connection machine* composée de 16384 processeurs !

Les principes de la machine de Boltzmann ont également été employés en restauration d'images [GEMAN 84].

1.4 RESEAUX A PROTOTYPES

Les réseaux que nous avons vus jusqu'alors ont pour dénominateur commun d'avoir une représentation distribuée de l'information. En effet, un neurone aide à définir un fragment de frontière de décision dans un réseau à couches, ou bien participe à la construction d'une fonction d'énergie dont les minima "stockent" les formes à reconnaître. Mais il est impossible d'attacher un rôle précis à telle ou telle cellule : l'information est répartie sur l'ensemble du réseau, chaque cellule coopérant avec les autres pour donner au réseau un comportement global susceptible de résoudre la tâche d'apprentissage qui lui a été soumise.

Au contraire, les modèles à prototypes que nous allons aborder maintenant sont composés d'éléments aux rôles bien distribués, contrairement à l'information qui est, elle, localisée. Autre particularité de ces réseaux : les cellules y entrent en compétition plus qu'elles n'y coopèrent.

Un exemple archétypique d'un tel réseau est le réseau de Hamming [LIPPMAN 87] ou MAXNET [PAO 89], qui n'est en fait qu'un cas particulier de ce que l'on pourrait appeler une implémentation neuronale de la conventionnelle classification au(x) plus proche(s) voisin(s).

1.4.1 Réseaux "plus proche voisin"

Le principe de ces réseaux est de stocker chaque forme de l'ensemble d'apprentissage dans les poids d'une des cellules du réseau. Lorsqu'une forme inconnue X est présentée, chaque cellule "prototype" calcule alors une mesure de similarité entre ses poids (c'est-à-dire le prototype stocké) et X. On assignera alors à la forme inconnue la classe du prototype stocké dans la cellule dont la sortie indique la plus forte ressemblance. Pour désigner cette cellule, un processus de compétition est mis en œuvre qui permet "d'élire", parmi un ensemble de cellules, celle dont la sortie est maximale .

Ceci est réalisé par un réseau dont l'architecture peut être qualifiée d'hybride. Celle-ci se compose en effet de deux sous-réseaux (Fig. 1.16):

1/ Un réseau à deux couches. La première est une classique couche d'entrée qui diffuse les composantes des formes présentées au réseau vers la seconde couche, seconde couche dont les cellules "prototype" vont calculer la similarité entre les formes qu'elles mémorisent dans leurs poids et la forme d'entrée.

2/ Un réseau dynamique. Celui-ci comporte autant de cellules qu'il y a de cellules "prototypes". Elles sont complètement interconnectées, à la manière d'un réseau de Hopfield, et vont entrer en compétition de telle sorte qu'après un certain nombre d'itérations, seule la cellule dont la valeur initiale était maximale possèdera une sortie non nulle. Les valeurs initiales des cellules sont bien sûr les mesures de similarité fournies par les cellules "prototypes" du sous-réseau en couches.
Un réseau dynamique ayant un tel comportement est appelé *winner-take-all* (difficilement traduisible... *le gagnant rafle tout* ?).

Nous allons maintenant détailler le fonctionnement de ces sous-réseaux.

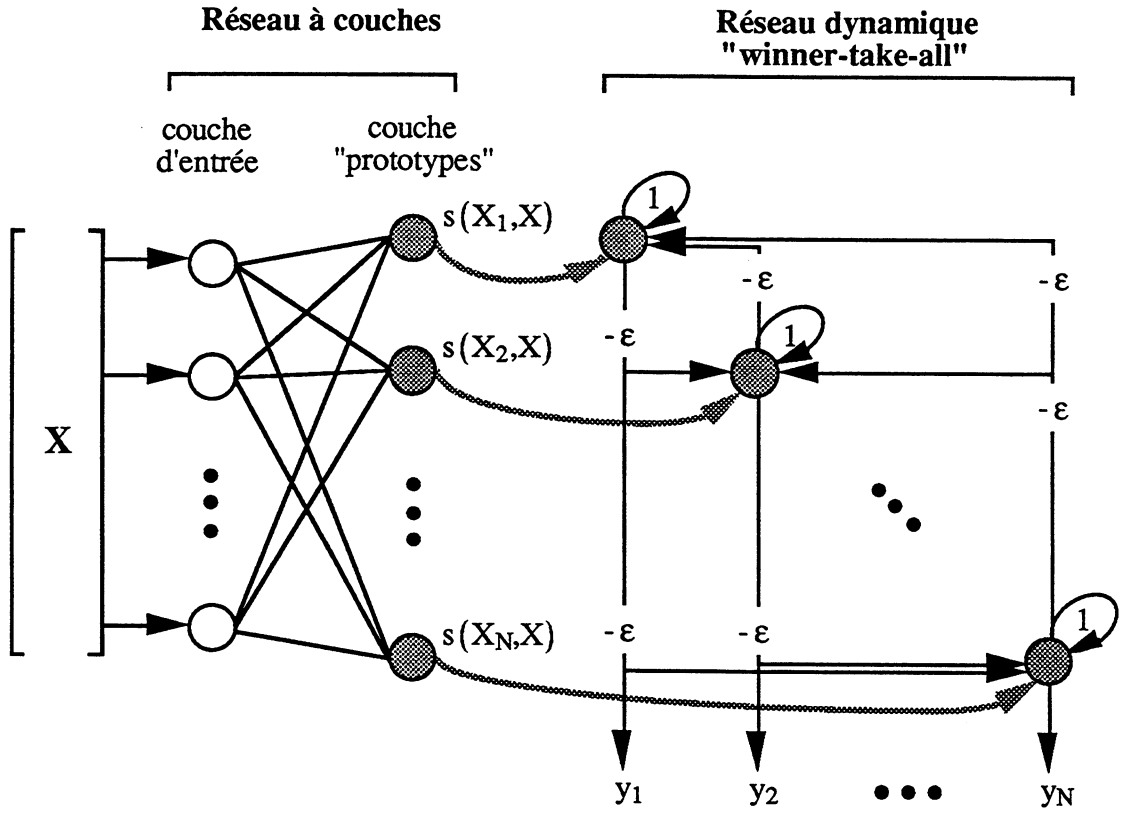


Fig. 1.16. Architecture d'un réseau "plus proche voisin"

1.4.1.1 Cellule "prototype" et mesures de similarité

Considérons toujours notre problème de classification à **p** classes. L'ensemble d'apprentissage E est formé de N formes X_i dans un espace de dimension **n**. On note c_i la classe d'un élément X_i, $1 \leq c_i \leq p$. Le sous-réseau se compose d'une couche de **n** cellules d'entrée et d'une couche de N cellules "prototypes", toutes complètement reliées à la couche d'entrée.

Les cellules que nous avons rencontrées jusqu'alors ont toujours été des automates linéaires généralisés qui appliquent une fonction de transition **f** au produit scalaire W.X, où W représente le vecteur de pondération des connexions de l'automate par lesquelles arrivent les signaux incidents représentés par le vecteur X. Si l'on désire conserver ce schéma de fonctionnement, seuls deux types de classification au plus proche voisin sont implémentables.

Distance de Hamming

Si les formes que l'on manipule sont binaires, le produit scalaire entre deux vecteurs W et

X de $\{-1,1\}^n$ est un entier compris entre -n et n :

$$W.X = n - 2.d_H(W,X) \tag{1.32}$$

où $d_H(W,X)$ dénote la distance de Hamming entre W et X.
Cette quantité varie entre **-n** (W et X diffèrent par toutes leurs composantes) et **n** (W=X). En définissant le terme s(W,X) de la manière suivante :

$$s(W,X) = n - d_H(W,X) = \frac{W.X + n}{2} \tag{1.33}$$

on obtient un indice de similarité[1] [DIDAY 82], $0 \leq s \leq n$, tel que la forme W_i la plus similaire à X selon cet indice est la forme la plus proche de X au sens de la distance de Hamming (s représente le nombre de composantes identiques entre les deux vecteurs).

De cette manière, si chaque cellule "prototype" **i** est un automate linéaire généralisé dont la fonction de transition **f** est l'identité, dont le vecteur poids W_i vaut $X_i/2$ et dont le seuil θ_i vaut -n/2, alors la sortie d'une cellule **i**, lorsqu'une forme X est présentée au réseau, vaut $s(X_i,X)$ et le réseau dynamique va alors désigner l'indice du prototype X_i qui est le plus proche de X au sens de la distance de Hamming... une classification au plus proche voisin est ainsi réalisée.

Distance euclidienne

Si les formes manipulées ne sont plus binaires mais réelles, le produit scalaire entre deux vecteurs W et X de $\mathbb{R}^n$ ne permet de définir un indice de similarité que lorsque tous les vecteurs sont de même norme L. En effet, on a dans ce cas :

$$W.X = \frac{\|W\|^2 + \|X\|^2 - \|W-X\|^2}{2} = L^2 - \frac{d_E(W,X)^2}{2} \tag{1.33}$$

où $d_E(W,X)$ dénote la distance euclidienne entre W et X, $0 \leq d_E(W,X) \leq 2L$. On peut alors définir un indice s positif de la manière suivante :

$$s(W,X) = W.X + L^2 \tag{1.34}$$

---

[1] s est un indice de similarité si :
1. $s \geq 0$
2. $\forall$ W,X, $s(W,X) = s(X,W)$
3. $\forall$ W,X, $W \neq X$, $s(W,W) = s(X,X) \geq s(W,X)$

Chaque cellule "prototype" **i** sera donc un automate ayant pour poids $W_i=L.X_i/\|X_i\|$, pour seuil $\theta_i=-L^2$ et la cellule dont la sortie $s(W_i,X)$ sera maximale désignera la forme X_i la plus proche de X au sens de la distance euclidienne (dans $\mathbb{R}^n$), si X est de norme L. Dans ce cas, le produit scalaire entre W_i et X correspond tout simplement au cosinus entre les deux vecteurs, cosinus que l'on translate pour obtenir un indice positif...

Autres distances

Bien d'autres mesures de dissimilarité sont utilisées en reconnaissance de formes : distances de Chebychev, de Camberra, de Minkowsky, de Mahalanobis... [DIDAY 80] ; toutes ne sont pas calculables à l'aide du produit scalaire d'un automate linéaire généralisé. Pour pouvoir les implémenter dans un cadre connexionniste, il faut s'autoriser, comme nous l'avons fait au § 1.1.1., à élargir la notion de neurone formel à celle d'un élément disposant d'une mémoire locale pouvant stocker toutes les informations nécessaires au calcul de la distance entre le prototype stocké et la forme présentée en entrée à l'élément.

Dans ces conditions, tous les types de mesure peuvent être utilisés ; on notera au passage que si les cellules "prototypes" calculent des indices de dissimilarité, et non de similarité, le réseau dynamique devra désigner la valeur la plus faible. Nous allons décrire dans ce qui suit un réseau désignant la valeur la plus élevée, ce qui suppose que les cellules "prototypes" calculent un indice de similarité (donc positif ou nul) ; la modification nécessaire pour réaliser l'opération inverse est triviale.

1.4.1.2 Réseau dynamique "winner-take-all"

Ce réseau est donc un réseau de type Hopfield, composé de N cellules totalement interconnectées les unes avec les autres (Fig. 1.16), y compris avec elles-mêmes. Chaque cellule **k** reçoit une entrée extérieure qui permet de fixer son état initial $y_k(0)$; celui-ci est égal à la sortie de la $k^{ème}$ cellule "prototype". Une fois ces valeurs initiales obtenues, le réseau va ensuite se relaxer selon une dynamique **parallèle** (synchrone), l'état de chaque cellule à l'instant (t+1) étant calculé en fonction des états au temps **t** :

$$y_k(t+1) = f\left(\sum_{j=1}^{N} w_{kj}.y_j(t)\right) \qquad (1.36)$$

où **f**, la fonction de transition des cellules, est définie ainsi :

$$f(x) = \begin{cases} 0 \text{ si } x \leq 0 \\ x \text{ si } x > 0 \end{cases} \tag{1.37}$$

et où le poids reliant la sortie de la cellule **j** à la cellule **k**, w_{kj}, vaut :

$$w_{kj} = \begin{cases} 1 \text{ si } j=k \\ -\varepsilon \text{ si } j \neq k,\ \varepsilon < \frac{1}{N} \end{cases} \tag{1.38}$$

D'après (1.36) et (1.38), on voit que le comportement de chaque cellule est de maintenir sa valeur de sortie, tout en essayant de supprimer celle des autres par un processus appelé "inhibition latérale".

On constate aisément que l'état y_k de chaque cellule est une fonction décroissante du temps. On observe également que, à l'instant t, la diminution de la cellule k_0 ayant l'état maximal est strictement inférieure à celle de toutes les autres cellules et est également inférieure à $y_{k0}(t)$. Par conséquent, la cellule dont la valeur initiale était la plus élevée conserve sa suprématie à chaque itération du réseau et, de par la nature de la fonction de transition (1.37), au bout d'un nombre fini (et en général faible) d'itérations,t^*, le réseau dynamique a convergé[1] vers un état stable tel que $y_{k0}(t^*)>0$ et $y_k(t^*)=0\ \forall\ k \neq k_0$.

Cet état stable du réseau dynamique nous indique alors la réponse du réseau "global" à la forme X présentée : celle-ci se verra attribuer la classe c_{k0} du prototype X_{k0} de E dont elle était la plus proche selon l'indice de similarité calculé par le réseau en couches.

Nous venons donc de voir qu'il était possible de réaliser un classifieur au plus proche voisin à l'aide de réseaux de neurones à architectures hybrides ; nous montrerons dans le second chapitre que de simples réseaux en couches peuvent également accomplir cette tâche.

Le réseau "plus proche voisin" décrit précédemment peut aussi se généraliser en réseau "k plus proches voisins". Il suffit pour cela de modifier le réseau dynamique afin que celui-ci extrait les k plus grands indices de similarité qui lui sont soumis ; on parle alors de réseau *k-winner-take-all* [MAJANI 89].

---

[1] La convergence n'est garantie que si une seule cellule possède l'état initial maximal. Dans le cas contraire, toutes les cellules d'état maximal vont évoluer de la même manière et mettre un temps infini pour décroître vers 0.

Cependant cette approche peut paraître bien coûteuse lorsque l'ensemble d'apprentissage E dont on dispose est grand. Pour le "représenter", N cellules "prototypes" sont en effet nécessaires, plus N cellules interconnectées dans le réseau dynamique, soit N(n+N) connexions pondérées... Or justement la classification au plus proche voisin nécessite un nombre important de prototypes pour généraliser correctement.

Une solution consiste alors à déterminer un ensemble d'apprentissage E' de N' formes, N'<<N, tel qu'une classification au plus proche voisin d'après E' fournisse des résultats aussi proches que possible de ceux d'une classification d'après l'ensemble original E. Ce problème de *quantification vectorielle* a été abordé dans un cadre connexionniste par Kohonen à qui l'on doit les célèbres *cartes topologiques* et les algorithmes *LVQ* que nous allons aborder maintenant.

1.4.2 Cartes topologiques et quantification vectorielle

Les *cartes topologiques auto-organisatrices* (self-organizing topological maps [KOHONEN 84]) ont été proposées par Kohonen au début des années 80. Elles sont inspirées d'un principe biologique affirmant que l'information est organisée spatialement à l'intérieur du cerveau (en *cartes*) et que des neurones voisins à l'intérieur d'une même zone se "projettent" vers d'autres zones du cerveau en conservant leurs relations de proximité (*rétinotopie*).

Les réseaux de neurones formels dérivés de ce principe sont constitués de cellules "prototypes", identiques à celles que nous venons de voir, sur lesquelles sont définies des relations de voisinage (une topologie). Les poids de ces cellules sont calculés, par une procédure non supervisée, de telle sorte que deux cellules voisines **dans le réseau** mémorisent des prototypes voisins **dans l'espace des formes**. De plus, l'ensemble des prototypes ainsi déterminé est représentatif des formes qui ont été présentées durant l'apprentissage : les cellules s'auto-organisent pour "recouvrir" l'ensemble d'apprentissage en essayant de "préserver" leur topologie.

On obtient ainsi un mécanisme de quantification vectorielle susceptible de résoudre le problème que nous évoquions plus haut. Ce mécanisme a déjà été largement mis en pratique en reconnaissance de la parole, robotique, contrôle de processus... On trouvera dans [KOHONEN 90] une liste d'applications type ainsi qu'une synthèse complète de ce modèle que nous ne décrirons que brièvement ici.

1.4.2.1 Architecture d'une carte topologique

Une carte de Kohonen est donc un réseau à deux couches, similaire au sous-réseau en couches des réseaux "plus proche voisin". La première couche est composée de **n** cellules d'entrée, la seconde comporte N' cellules "prototypes". La sortie d'une cellule **k**, y_k, est définie comme étant la distance euclidienne entre la forme X présentée en entrée et le vecteur des poids W_k :

$$y_k(X) = d_E(W_k,X) = \|W_k - X\| \tag{1.39}$$

D'autres métriques peuvent bien sûr être utilisées. On notera au passage qu'une telle cellule n'entre pas dans la catégorie des automates linéaires généralisés.

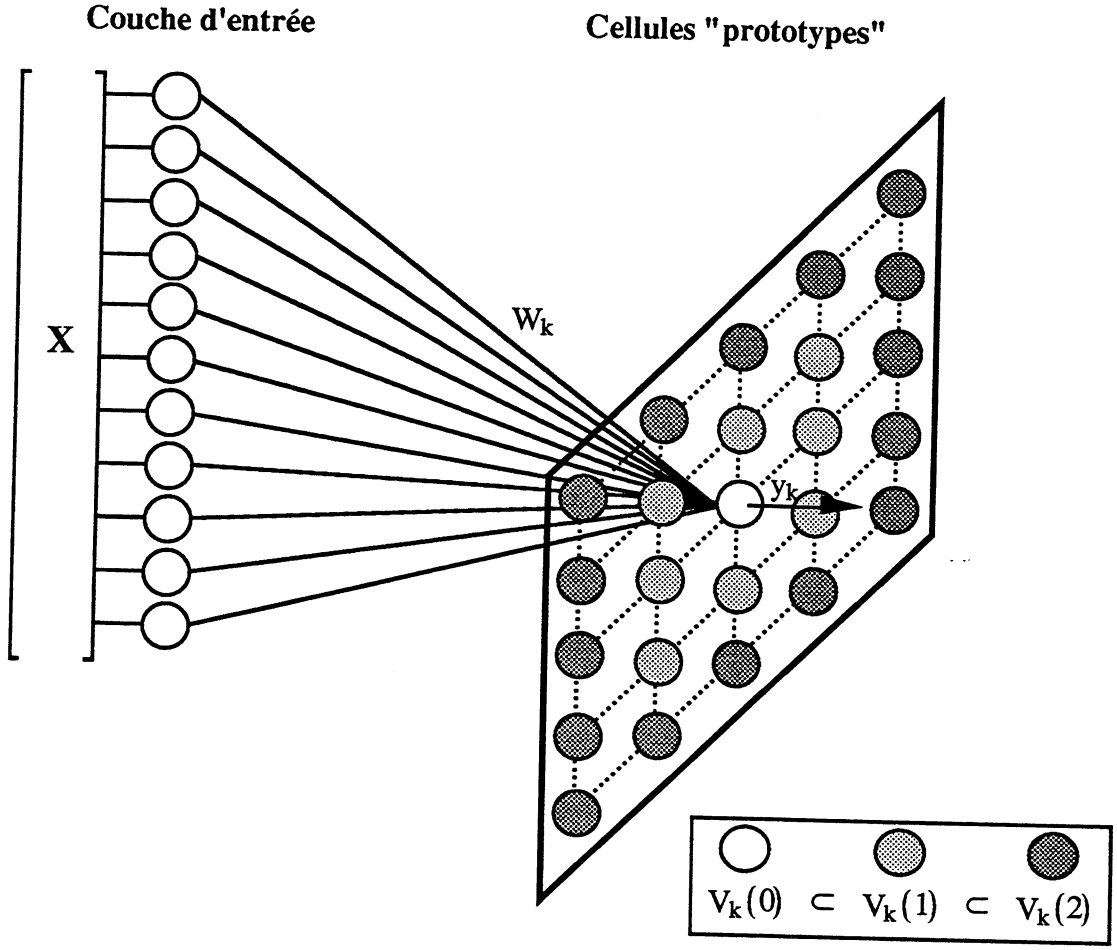


Fig. 1.17. Architecture d'une carte topologique de Kohonen et relations de voisinage entre cellules "prototypes".

Sur ces N' cellules, une topologie est définie *a priori*. Elle permet d'associer à chaque

cellule **k** un voisinage de rayon **r**, $V_k(r)$, qui représente l'ensemble des cellules se trouvant dans la boule de rayon **r** centrée sur la cellule **k** (et donc **k** elle-même). En général, on considère que les cellules sont disposées dans le plan, sur un maillage carré muni de la distance de l'échiquier (Fig. 1.17), mais ici aussi, d'autres choix sont possibles (cellules organisées en 1D, en 3D, sur un maillage hexagonal en 2D, ...).

1.4.2.2 Apprentissage non supervisé

On considère toujours un ensemble E composé de N formes $X_i \in \mathbb{R}^n$ que l'on cherche à représenter "au mieux" par N' prototypes $W_k \in \mathbb{R}^n$, N'<<N. L'apprentissage est dit non supervisé car les prototypes W_k sont déterminés en fonction des X_i, mais indépendamment de leur classe (qui peut ne pas être connue dans certaines applications) ; il consiste à présenter itérativement chaque forme X_i, à chercher le prototype k_0 le plus proche de X_i (c'est-à-dire trouver la cellule ayant la sortie y_{k0} minimale, à l'aide d'un réseau type "winner-take-all" par exemple) et à modifier les vecteurs poids de la cellule k_0 et de ses voisines afin de les rendre plus proches de X_i. L'algorithme est le suivant :

Etape 0 :	t=0
	Initialiser aléatoirement les poids $W_k(t)$
Etape 1 :	Choisir une forme $X_i \in E$, la présenter au réseau.
	Soit k_0 la cellule / $y_{k0}(X_i,t) = \min_{1\le k\le N'} \{y_k(X_i,t)\}$
Etape 2 :	Pour toute cellule $k \in V_{k0}(r(t))$, modifier ses poids par :
	$W_k(t+1) = W_k(t) + \alpha(t).[X_i - W_k(t)]$
Etape 3 :	t = t+1
	Si condition d'arrêt non remplie, aller à l'étape 1.

La taille du voisinage dans lequel les poids sont modifiés, r(t), tout comme le "gain d'adaptation" $\alpha(t)$, $0<\alpha(t)<1$, doit être une fonction décroissante du temps. Le gain $\alpha(t)$ peut également ne pas être constant sur tout le voisinage, mais représenter une sorte de "noyau" gaussien, α étant maximal pour le prototype "gagnant" k_0, puis de plus en plus faible à mesure que l'on s'éloigne de la cellule k_0.
La condition d'arrêt de l'étape 3 porte en général sur le nombre d'itérations effectuées qui doit être "raisonnablement grand" [KOHONEN 90]...

Un exemple d'application de cet algorithme est montré (Fig. 1.18). Il illustre comment les prototypes calculés tentent d'approcher la distribution des formes d'apprentissage. On peut en effet considérer, sous certaines conditions [KOHONEN 90], que cet algorithme minimise, par

descente en gradient, une erreur de "reconstruction" commise lorsque l'on utilise N' prototypes W_k pour représenter une distribution continue de vecteurs de fonction de densité p(X). Cette erreur, C, est définie par :

$$C = \int_{\mathbb{R}^n} || W_{k_0} - X ||^2 . p(X)\, dX \tag{1.40}$$

où W_{k0}, fonction de X, est le prototype tel que $||W_{k0} - X|| = \min_{1 \le k \le N'} \{||W_k - X||\}$.

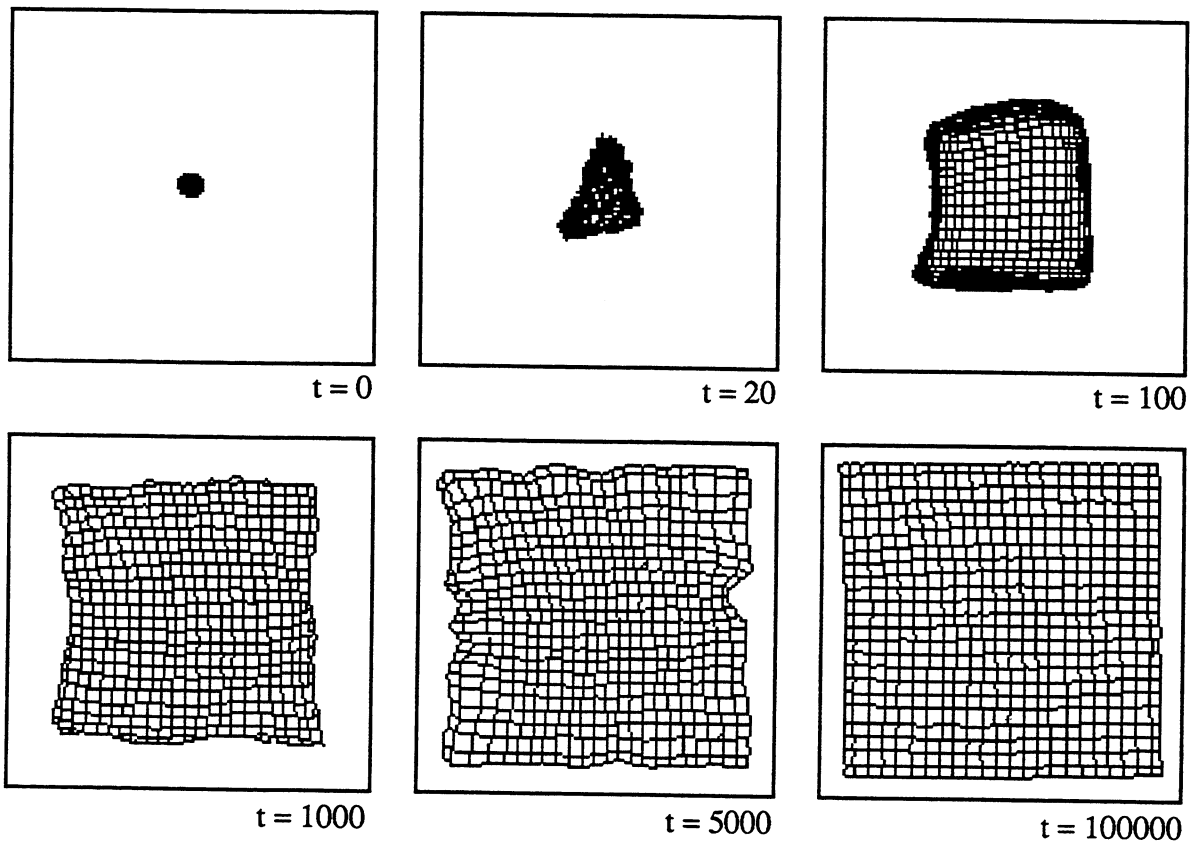


Fig. 1.18. Un exemple d'apprentissage extrait de [KOHONEN 90]. Les formes à apprendre sont des points du plan distribués uniformément sur un carré. On a représenté, dans cet espace des formes, les prototypes mémorisés par chaque cellule et les relations de voisinage entre ces cellules. On voit qu'au cours du temps, la grille de 24x24 cellules s'auto-organise pour représenter au mieux la distribution carrée des formes d'apprentissage.

L'algorithme ayant fourni, de manière non supervisée, un ensemble E' de formes prototypes représentant au mieux l'ensemble d'apprentissage E, il reste alors à déterminer à quelle classe attribuer chaque prototype W_k afin de pouvoir classifier une forme inconnue au plus proche voisin d'après ce nouvel ensemble E'.

Pour cela, une solution consiste à rechercher pour chaque prototype W_k l'ensemble E_k des formes de E qui sont plus proches de W_k que de tous les autres prototypes. On affectera alors à W_k la classe qui est la plus représentée dans E_k (ou la classe du vecteur de E_k le plus proche de W_k). En fait, on réalise une classification de E' aux plus proches voisins d'après E.

Cependant, le classifieur ainsi construit n'est pas réellement adapté à la tâche de classification que nous visons. En effet, le sous-ensemble E' a été déterminé d'après un critère (1.40) qui n'est pas réellement celui recherché, à savoir trouver un ensemble E' tel qu'une classification au plus proche voisin[1] d'après E' approche au mieux la même classification d'après E. Pour obtenir cela, on préfèrera une solution qui positionne les prototypes W_k autour des frontières de décision définies par E de telle sorte que les frontières de décision de E' s'en approchent le plus fidèlement possible. Peu importe alors que la distribution de E' soit très représentative de celle de E... (Fig. 1.19).

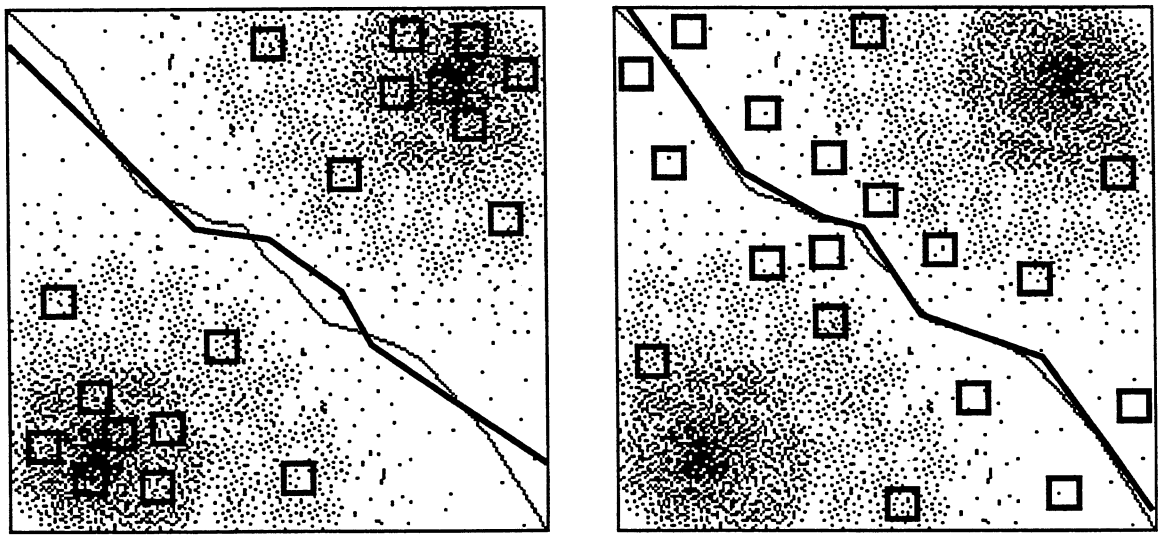


Fig. 1.19. L'apprentissage non supervisé (à gauche) fournit un ensemble de prototypes (les carrés) approchant la distribution des formes d'apprentissage. Pour une tâche de classification, on préfèrera un ensemble de prototypes (à droite) qui génère des frontières de décision (en gras) proches de celles de l'ensemble d'apprentissage.

Dans ce but, Kohonen a proposé plusieurs méthodes pour adapter "finement" les prototypes trouvés par l'apprentissage non supervisé en tenant compte maintenant de la tâche de

[1] Les méthodes de décision bayesiennes semblent plus adaptées pour réaliser une classification en aval d'une carte topologique puisque cette dernière constitue une approximation de la distribution des formes.

classification qu'ils doivent remplir. Ce sont les algorithmes LVQ 1, 2 et 3 [KOHONEN 90] de *quantification vectorielle supervisée* (learning vector quantization) dont nous ne présenterons, à titre d'exemple, que le premier.

1.4.2.3 Quantification vectorielle supervisée

On suppose donc que l'on dispose de N' prototypes W_k calculés par l'algorithme non supervisé et que l'on a "étiquetés" par la méthode décrite précédemment ; soit c_k la classe de W_k. L'algorithme LVQ 1 consiste encore à présenter itérativement les formes X_i de l'ensemble d'apprentissage, mais en tenant compte de leur classe c_i. Le prototype le plus proche d'un vecteur X_i est alors rapproché de celui-ci s'ils ont même classe, et éloigné dans le cas contraire :

Etape 0 :	t=0
	Déterminer les poids $W_k(t)$ par l'algorithme non supervisé
Etape 1 :	Choisir une forme $X_i \in E$, de classe c_i ; la présenter au réseau.
	Soit k_0 la cellule / $y_{k0}(X_i,t) = \min_{1\leq k\leq N'} \{y_k(X_i,t)\}$
Etape 2 :	Si $c_{k0} = c_i$ alors
	$W_{k0}(t+1) = W_{k0}(t) + \alpha(t).[X_i - W_{k0}(t)]$
	Sinon
	$W_{k0}(t+1) = W_{k0}(t) - \alpha(t).[X_i - W_{k0}(t)]$
Etape 3 :	t = t+1
	Si condition d'arrêt non remplie, aller à l'étape 1.

Comme auparavant, $\alpha(t)$, $0<\alpha(t)<1$, sera une fonction décroissante du temps, de valeur initiale faible, et l'algorithme s'achèvera (étape 3) après un nombre "raisonnablement grand" d'itérations.

On remarquera que dans cet algorithme, les relations de voisinage entre cellules ne sont plus utilisées lors de la modification d'un prototype (étape 2). Ces relations ne seront pas plus utilisées si l'on se contente, après apprentissage supervisé, de classifier une forme inconnue seulement selon la classe du prototype le plus proche. Cette information peut cependant être employée en dehors de la phase d'apprentissage non supervisé [MAGGIONI 91].

En effet, deux prototypes voisins dans le réseau représentant des vecteurs proches dans l'espace des formes, il est possible de considérer ce voisinage dans le réseau pour appliquer une règle du type "k plus proches voisins". On attribuera alors à une forme inconnue, X, la classe la plus représentée dans un voisinage $V_{k0}(r)$ du prototype le plus proche, W_{k0}. Tout l'intérêt

réside dans le fait que ce voisinage n'a pas à être calculé.
Un mécanisme semblable a été mis en œuvre dans un problème de vision où l'on souhaitait associer à l'image d'un objet 3D les paramètres de sa prise de vue. L'ensemble d'apprentissage était constitué de couples (image, paramètres). L'algorithme non supervisé a été utilisé pour obtenir un ensemble de couples prototypes. Les paramètres d'une image inconnue X étaient alors calculés en interpolant les paramètres des prototypes pris dans un voisinage autour du prototype dont l'image était la plus proche de X [MAGGIONI 91].

1.4.3 Autres réseaux à prototypes

Nous venons de voir comment il était possible d'implanter de manière complètement "neuronale" des techniques de classification fondées sur la comparaison, à l'aide d'une métrique donnée, d'une forme inconnue avec un ensemble de prototypes étiquetés. Nous avons présenté une méthode permettant de déterminer un tel ensemble de prototypes ; d'autres algorithmes de quantification vectorielle plus simples ont été proposées dans le cadre des réseaux "plus proche voisin" [ALPAYDIN 90].

Pour clore ce paragraphe consacré aux réseaux à prototypes, nous mentionnerons également l'existence de deux modèles célèbres fondés sur la mémorisation d'un ensemble de formes de référence : les réseaux R.C.E (Restricted Coulomb Energy) et A.R.T (Adaptative Resonance Theory).

Le réseau R.C.E [REILLY 82] est un réseau à trois couches stockant un ensemble de prototypes. A chaque prototype est associé un rayon, ce qui définit une "hypersphère d'influence" dans l'espace des formes muni de la distance euclidienne. Toute forme inconnue se voit alors attribuer la classe du prototype dans l'hypersphère duquel elle tombe. L'apprentissage, supervisé, consiste dans ce modèle à déterminer un sous-ensemble de prototypes (quantification vectorielle) et à leur associer le rayon adéquat. La méthode de classification qui en résulte est similaire à la classique technique des fenêtres de Parzen [DUDA 73].

Les réseaux A.R.T (ART 1 pour les formes binaires et ART 2 pour les formes réelles, voir [CARPENTER 87] pour une synthèse) ont été développés essentiellement dans le but de proposer un modèle réaliste du mécanisme de la reconnaissance dans le cerveau humain. Le principe de base en est toujours la mémorisation d'exemples représentatifs, par une procédure d'apprentissage non supervisé. Contrairement aux autres modèles, les réseaux ART sont entièrement spécifiés en termes de réseaux dynamiques, tant dans la phase de classification que

dans la phase d'apprentissage, ce qui rend complexe toute tentative de description brève et complète de leur fonctionnement.

1.5 A L'HEURE DU CHOIX

Les motivations qui peuvent présider au choix de l'utilisation d'un modèle connexionniste sont bien sûr nombreuses. Au premier rang (historiquement) de celles-ci, se trouve l'ambition, très légitime, de construire des systèmes "neuro-mimétiques", c'est-à-dire des modélisations de systèmes nerveux vivants ; ce n'est évidemment pas la nôtre. On peut cependant noter que cet argument de plausibilité biologique est souvent avancé pour justifier de l'usage d'un modèle connexionniste ; sa validité paraît pourtant fragile dans bien des cas. On aura pu juger de cette fragilité au cours de ce chapitre, tant les algorithmes qui y étaient présentés semblaient plus relever des sciences de l'ingénieur que de celles de la nature.

Ne restent alors, pour motiver l'emploi de ces méthodes, que des considérations beaucoup plus pragmatiques : les réseaux de neurones qui nous intéressent se présentent comme des méthodes de classification, pour certaines originales, formulées de manière *parallèle*. L' intérêt de ces différents points doit alors être discuté selon chaque modèle et la situation dans laquelle on envisage de l'utiliser.

Réseaux à prototypes

Les réseaux à prototypes reposent, nous l'avons vu, sur un principe parfaitement connu et que l'on rattache en général aux méthodes statistiques : la classification au plus proche voisin. Leur seule originalité réside alors dans leur formulation parallèle de cette technique. Ceci n'a évidemment d'intérêt que si l'on peut tirer parti de ce parallélisme. Or la plupart des applications de réseaux de neurones ont jusqu'à présent été réalisées sur des machines séquentielles, celles-ci étant pour l'instant nettement plus répandues que les machines parallèles. De plus, la majorité des machines parallèles disponibles ont une granularité importante (à base de Transputers par exemple) peu compatible avec la granularité fine des réseaux de neurones, ce qui rend la programmation de ces réseaux non "triviale". On peut dès lors se demander s'il ne serait pas plus "rentable" de concevoir des implantations des "plus proche voisin" adaptées aux machines disponibles, plutôt que de passer par le formalisme des réseaux de neurones qu'il devient alors bien difficile de justifier.

Cette justification est encore plus hasardeuse lorsque ces réseaux sont simulés sur machine séquentielle ; dans ce cas, il existe en effet des méthodes plus efficaces de programmer

une recherche de plus proche voisin (les arbres k-dimensionnels par exemple [KELLY 91], [PREPARATA 85]) que celle consistant à calculer un indice de similarité avec chaque prototype puis à simuler la relaxation d'un réseau "winner-take-all" pour déterminer l'indice maximal !

Finalement, on retiendra essentiellement de ces réseaux les algorithmes de quantification vectorielle qu'ils ont suscités, mais que l'on pourrait tout à fait décrire dans un cadre non connexionniste. Les cartes topologiques de Kohonen se sont en effet avérées très efficaces dans de nombreuses applications [KOHONEN 90]. De par le caractère stochastique de leur apprentissage, elles seront toutefois réservées aux cas où l'ensemble d'apprentissage est de taille conséquente.

Réseaux dynamiques

Les réseaux dynamiques sont, nous l'avons déjà dit, peu adaptés à la reconnaissance de formes. Les réseaux de Hopfield ont des capacités de "stockage" et de généralisation trop faibles pour pouvoir être utilisés dans des applications autres que "scolaires". Les machines de Boltzmann requièrent quant à elles de telles capacités de calcul pendant l'apprentissage que leur simulation sur une simple station de travail paraît difficilement envisageable. Le fait que, pour ces modèles, il faille attendre la relaxation d'un réseau dynamique pour pouvoir classifier une forme inconnue, les rend en tout état de cause peu aptes aux applications type OCR (Optical Character Recognition) où un grand nombre de formes doivent être classifiées en un minimum de temps (à moins bien sûr de disposer, comme cela existe maintenant à l'état de prototype dans les laboratoires, de machines "neuronales" implémentant spécialement ces types de réseaux [BLAYO 89], [JUTTEN 90]).

Réseaux à couches

Les réseaux à couches sont incontestablement les plus utilisés dans le domaine de la reconnaissance de formes. Leur mode de fonctionnement (calcul en un coup) en fait des outils adaptés à la classification.

Les réseaux à deux couches fournissent des classifieurs fondés sur un principe très connu en classification non paramétrique : la construction de fonctions discriminantes linéaires dans l'espace des formes ; la procédure de prise de décision qui en résulte est compacte et rapide, mais ne peut s'adapter à toutes les situations.

Les réseaux multi-couches ne connaissent pas ces limitations. Ils peuvent théoriquement

s'accomoder de n'importe quel ensemble d'apprentissage. On sait cependant encore peu de choses sur la nature des classifieurs produits par l'algorithme de rétro-propagation du gradient. Pourtant, les nombreuses expériences rapportées dans la littérature font état de résultats prometteurs, au moins de l'ordre des meilleures méthodes de classification statistiques, que ce soit en reconnaissance de la parole [WAIBEL 89] ou de caractères [GUYON 89]. C'est donc vers ce type de réseaux que s'est porté notre choix, tant ils semblaient adaptés à l'application visée ; le second chapitre est consacré à leur étude détaillée.

CHAPITRE 2

RESEAUX MULTI-COUCHES

Nous avons justifié notre intérêt pour les réseaux multi-couches au cours du premier chapitre. De nombreuses applications en reconnaissance de formes ont été basées sur ces réseaux et sur l'algorithme d'apprentissage qu'on leur associe en général : la rétro-propagation du gradient.

Pourtant, la mise en œuvre de cet algorithme n'est pas directe et les performances qu'il peut fournir ne sont pas toujours prévisibles. Cette incertitude provient bien sûr de l'absence de résultats théoriques concernant ce modèle de réseaux, absence de théorie qui a provoqué des réactions différentes selon les deux publics qui la constataient. Le premier, celui des rigoristes, y a vu une raison suffisante pour ne pas employer une technique "entachée" d'empirisme. Le second, celui des pragmatiques qui ne voulaient pas se priver d'une approche potentiellement performante, aussi peu justifiée fût-elle, en a conclu à la nécessité d'études empiriques afin de fournir des outils permettant de ne pas aborder "démuni" ces réseaux. Ce sont ces outils, qualifiés de "recettes de cuisine" par les premiers et "d'heuristiques" par les seconds, que nous détaillerons dans le premier paragraphe de ce chapitre. Nous apporterons notre petite pierre à cet édifice expérimental en proposant une méthode d'aide à la génération de l'architecture d'un réseau.

Au cours du second paragraphe, nous nous intéresserons à un sous-ensemble particulier de ces réseaux : les réseaux multi-couches d'automates à seuil, ou *Gamba Perceptrons* selon la terminologie de Minsky et Papert. Les capacités de ces réseaux pour la reconnaissance de formes sont beaucoup plus connues que celles des réseaux multi-couches "généraux" et nous montrerons qu'ils peuvent supporter différents schémas de classification. Nous proposerons en particulier un algorithme d'apprentissage hiérarchique basé lui aussi sur une minimisation par descente en gradient.

2.1 CONCEPTION D'UN RESEAU A RETRO-PROPAGATION DU GRADIENT

Le problème que l'on se pose est toujours celui de l'apprentissage d'un ensemble E composé de N couples (X_i, Y_i), où X_i, vecteur de dimension **n**, représente une forme à reconnaître et où Y_i, vecteur de dimension **p**, représente sa classe. On souhaite donc construire un réseau multi-couche capable d'associer à chaque X_i de E, sa réponse "désirée", Y_i.

Les données de ce problème sont très réduites et les seules contraintes qu'elles induisent portent sur le nombre de cellules d'entrée et de sortie que doit comporter le réseau, respectivement **n** et **p**. Hormis cela, il reste un nombre important de paramètres que le candidat à la conception du réseau a *a priori* toute latitude pour fixer. Ces décisions peuvent être divisées en deux ensembles selon les étapes de la construction du réseau pendant lesquelles elles doivent être prises. Nous considérerons d'une part les choix à faire pendant la phase de conception de l'architecture du réseau et d'autre part les choix à faire lors de la phase suivante, celle de l'apprentissage, que nous allons tout d'abord présenter.

2.1.1 Apprentissage

Cette étape n'intervient qu'après que "l'opérateur" ait fixé, selon des critères que nous aborderons plus tard, l'architecture du réseau : nombre de couches, nombre de cellules par couche, graphe d'interconnexion. Cette architecture permet de définir le réseau comme une fonction, F_W, définie de l'espace des cellules d'entrée sur l'espace des cellules de sortie et paramétrée par l'ensemble des pondérations des connexions, W. L'apprentissage, ou évaluation de paramètres, consiste à rechercher un ensemble W^* tel que $F_{W^*}(X_i) = Y_i \ \forall i, 1 \leq i \leq N$. Comme nous l'avons vu au § 1.2.4, cet apprentissage est traduit en un problème d'optimisation : la minimisation de l'erreur quadratique moyenne C(W) commise par le réseau sur l'ensemble E. Ceci est réalisé par un processus répétitif, qui, en partant d'un *ensemble de poids initiaux*, va modifier ceux-ci selon une *règle d'adaptation* jusqu'à la satisfaction d'un *critère d'arrêt* indiquant que l'apprentissage est achevé.

Le principe de cette minimisation est de "descendre" la surface d'erreur C(W) en suivant la direction opposée au gradient de C par rapport à W. Selon ce principe, chaque itération de l'algorithme doit donc entraîner une diminution de C(W). Cependant, si cette "promenade" sur une surface peut certes conduire à un minimum global, elle peut également conduire à un minimum local, si un tel minimum existe. Ceci n'est évidemment pas souhaitable.

D'autre part, si la pente menant à un minimum est très faible, l'algorithme risque de nécessiter un nombre très important d'itérations pour atteindre ce minimum, ce qui n'est pas souhaitable non plus. Or, si l'on connaît peu l'allure générale des surfaces d'erreur C(W), trois points sont cependant acquis [HECHT-NIELSEN 90] :

1/ Ces surfaces comportent des minima locaux.

2/ Ces surfaces contiennent une multitude de "plateaux", zones où les pentes sont faibles dans plusieurs dimensions.

3/ Ces surfaces comportent une multitude de minima globaux.

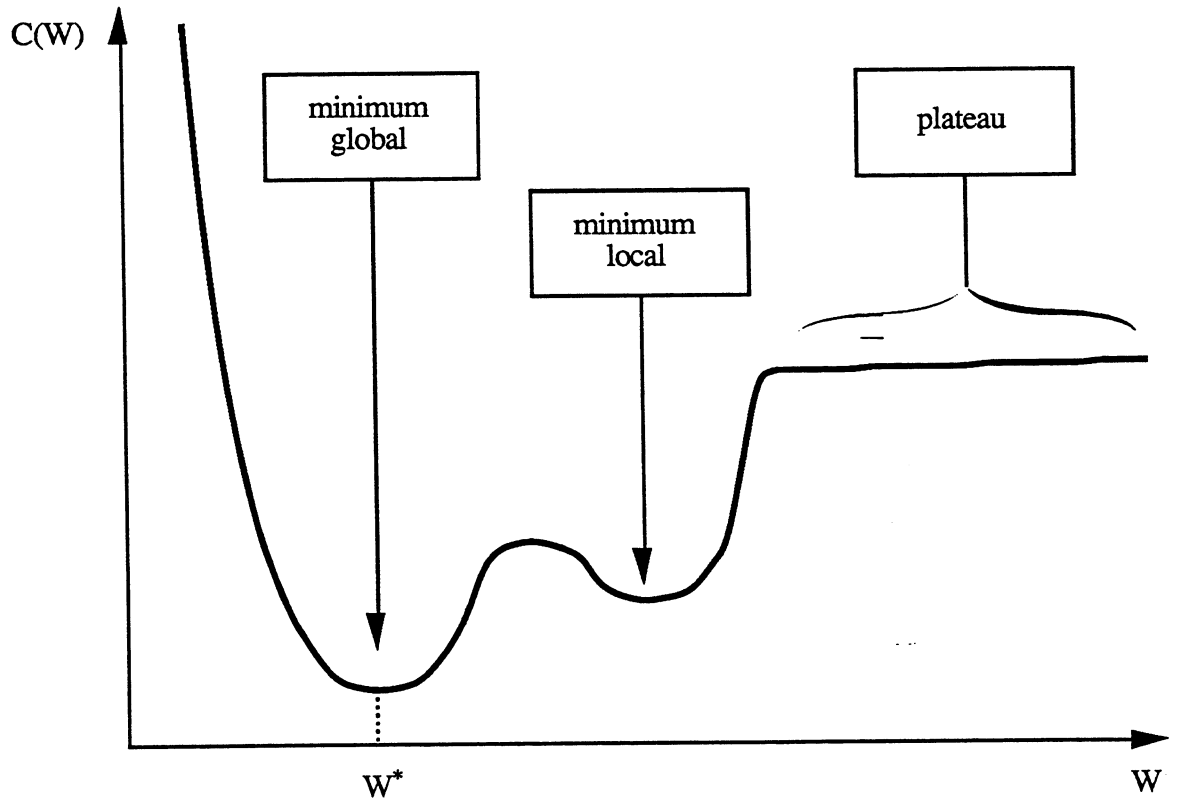


Fig. 2.1. Surface d'erreur d'un réseau à rétro-propagation du gradient

Si ce dernier point rend raisonnable la probabilité de voir l'algorithme converger vers une bonne solution, les deux premiers nous révèlent cependant que cette convergence peut être très longue, et n'est, en tout état de cause, pas garantie. Pourtant, plusieurs précautions d'ordre pratique peuvent aider à éviter les "pièges" que constituent plateaux et minima locaux. La première de ces précautions réside dans le choix des valeurs initiales des poids, choix conditionné en fait par la nature de la fonction de transition des automates du réseau.

2.1.1.1 Fonction de transition

L'adoption d'une fonction de transition relève de la conception de l'architecture plus que de l'apprentissage. Néanmoins c'est sur ce dernier que ce choix a le plus d'impact et c'est pourquoi nous le décrivons ici.

Dans une situation de classification, on souhaite pouvoir interpréter facilement la réponse du réseau à une forme en termes de classes. La convention est, dans un problème à **p** classes, d'utiliser **p** cellules de sortie à deux états, **a** et **b**, la réponse désirée à une forme X_i de classe **q** devant alors consister en un vecteur Y_i de $\{a,b\}^p$ dont seule la $q^{ème}$ composante vaut **a**. Ceci est obtenu en utilisant pour fonction de transition une fonction seuil, comme nous l'avons vu pour le Perceptron. Cependant, une telle fonction n'est pas dérivable, comme le requiert l'algorithme de rétro-propagation, et l'on est donc obligé d'utiliser des "versions" continues et dérivables de la fonction seuil : les fonctions sigmoïdales. La plus répandue et que l'on appelle en général "la" fonction sigmoïde, est :

$$f(x) = \frac{2}{1 + e^{-t_f.x}} - 1 \qquad (2.1)$$

où $t_f/2$ représente la pente de la sigmoïde en 0, fixée *a priori*.
D'autres choix sont bien sûr possibles (la tangente hyperbolique par exemple [BOTTOU 88]). On veillera cependant à ne prendre que des fonctions impaires [FOGELMAN SOULIE 91].

La fonction **f** ainsi définie est à valeur sur]-1,1[et dans ces conditions, l'habitude couramment rapportée dans la littérature veut que l'on prenne **a** = 1, **b** = -1 et des valeurs de poids aléatoirement tirées sur]-1,1[. Mais de telles valeurs risquent justement de "conduire" l'apprentissage vers des zones de "plateaux", ces plateaux étant dus au comportement asymptotique de la sigmoïde dès que l'on s'éloigne trop de l'origine. En effet, en reprenant les notations du § 1.2.4, la sortie d'une cellule **k** vaut $y_k = f(net_k)$, où net_k, l'entrée "nette" de la cellule **k** vaut :

$$net_k = \sum_{j \in CE_k} w_{kj} y_j \qquad (2.2)$$

Si les cellules **j** de CE_k sont toutes des cellules cachées, tous les y_j sont à valeurs sur]-1,1[et si les poids w_{kj} sont initialement à valeurs sur]-1,1[, alors net_k peut décrire l'intervalle $]-n_k,n_k[$, où $n_k = |CE_k|$ est le degré entrant de la cellule **k**.
Ainsi, à la première itération de l'algorithme, la sortie des cellules dont le degré entrant est important risque d'être "propulsée" sur les branches asymptotiques de la sigmoïde, zone dans

laquelle la dérivée $f' = t_f.(1 - f^2)/2$ est presque nulle. Or le gradient de C par rapport aux poids w_{kj} de la cellule **k** étant proportionnel à $f'(net_k)$ (expressions 1.18 et 1.21), "fréquenter" les branches asymptotiques de la sigmoïde revient donc à positionner le réseau sur un plateau de C, plateau qu'il ne pourra quitter qu'après de (trop) nombreuses itérations.

Pour éviter cela, on aura donc intérêt à s'assurer que l'entrée "nette" de chaque cellule est initialement dans l'intervalle central $[-w_0,w_0]$ sur lequel la sigmoïde a une forte pente (Fig. 2.2). Les poids w_{kj} seront donc initialisés en fonction du degré entrant de leur cellule par une valeur choisie aléatoirement sur $[-w_0/n_k,w_0/n_k]$, w_0 étant l'abscisse positive du point de courbure maximale de la sigmoïde. Pour que cet intervalle ne soit pas trop étroit, la sigmoïde devra être choisie suffisamment "penchée", mais sans pour autant trop s'approcher d'une courbe linéaire dans l'intervalle de fonctionnement, ce qui risquerait de transformer notre réseau multi-couche en "vulgaire" Adaline. Dans nos différentes expériences, nous avons en général fixé la valeur du paramètre t_f à 4 ; *a posteriori* cette valeur semble satisfaisante.

On notera également que pour que l'entrée "nette" des cellules connectées aux cellules d'entrée soit dans le bon intervalle, il faut que les composantes des formes présentées au réseau prennent leur valeur sur [-1,1]. Ceci nécessitera éventuellement un pré-traitement des formes de l'ensemble d'apprentissage lorsque les X_i n'appartiennent pas d'emblée à $[-1,1]^n$.

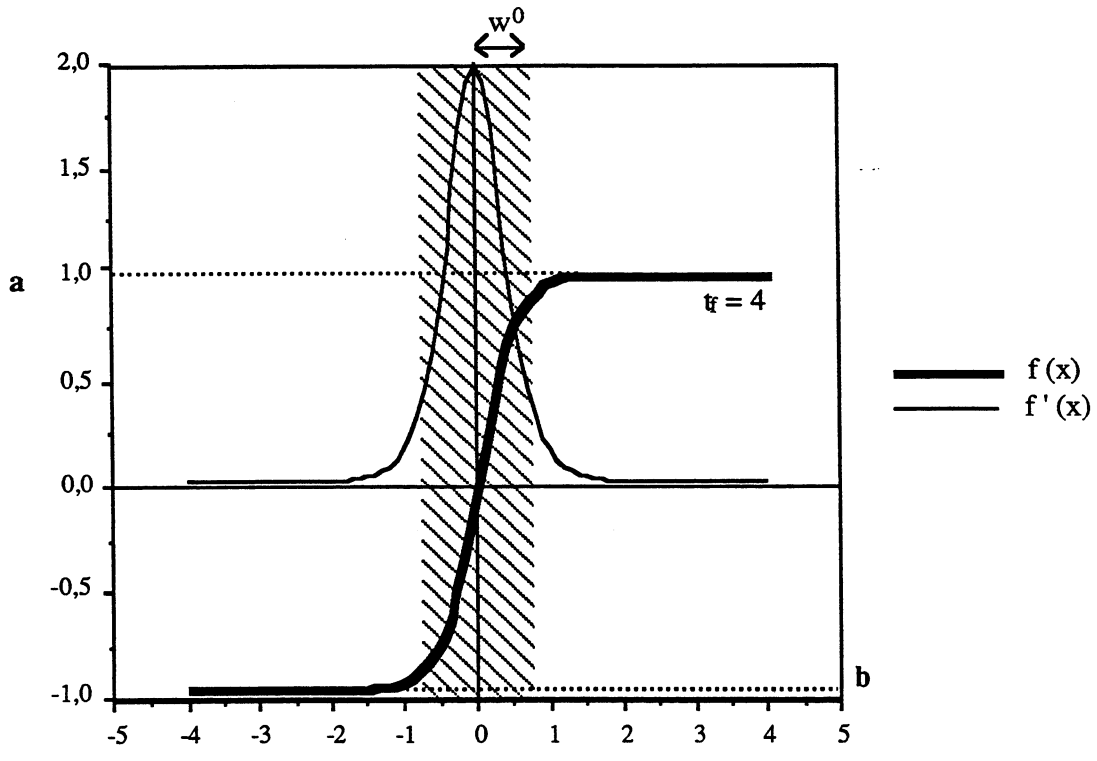


Fig. 2.2. Fonction de transition sigmoïde.

Ces deux précautions étant prises, il y a de fortes chances pour que les premières itérations de l'algorithme "rencontrent" des gradients assez forts entraînant une décroissance rapide de l'erreur C. Néanmoins, la convergence vers un minimum risque d'être fort longue si les valeurs cibles des cellules de sortie, **a** et **b**, sont égales aux valeurs des asymptotes, +1 et -1 respectivement. Afin de ne pas perdre un temps infini à tenter de rejoindre, en fin d'apprentissage, ces cibles inaccessibles, on positionnera plutôt **a** et **b** au "début" des branches de l'asymptote : $\mathbf{a} = f(w_0+\varepsilon)$, $\mathbf{b} = -\mathbf{a}$ (Fig. 2.2).

2.1.1.2 Règle d'adaptation

L'algorithme de rétro-propagation du gradient procède donc en modifiant tous les poids à chaque itération **t** :

$$w_{kj}(t+1) = w_{kj}(t) + \Delta w_{kj}(t) \tag{2.3}$$

Si l'on veut réellement minimiser l'erreur moyenne, C, commise sur tout l'ensemble d'apprentissage, la variation d'un poids w_{kj}, Δw_{kj}, doit se faire dans la direction opposée au gradient de C par rapport au poids. Ceci donne la règle de descente en gradient "standard" :

$$\Delta w_{kj}(t) = -\alpha \frac{\partial C}{\partial w_{kj}} = -\alpha \frac{1}{N} \sum_{i=1}^{N} \frac{\partial C_i}{\partial w_{kj}} \tag{2.4}$$

Une "vraie" descente en gradient nécessite que les variations de poids à chaque étape soient infinitésimales. Le taux d'apprentissage α doit donc être choisi faible (typiquement 0.01 ou moins), faute de quoi les poids oscilleront. Mais les temps d'apprentissage sont d'autant plus longs que le taux α est faible. Une solution permettant de garder un taux élevé, tout en évitant les oscillations, consiste à prendre en compte dans la modification d'un poids à l'instant **t**, sa modification à l'instant précédent. Ceci conduit à la règle de "l'inertie" (momentum) décrite dans [RUMELHART 86] :

$$\Delta w_{kj}(t) = -\alpha \frac{\partial C}{\partial w_{kj}} + \beta . \Delta w_{kj}(t-1) \tag{2.5}$$

où $0 \leq \beta < 1$ (typiquement $\beta = 0.9$). L'effet de ce terme supplémentaire est d'augmenter le "pas" de l'adaptation lorsque deux modifications successives ont même direction (cas d'une descente le long d'une pente très faible), alors que ce pas est réduit lorsque les deux directions sont opposées (cas du passage d'un versant à l'autre autour d'un "ravin" profond).

Différentes études ont essayé de mettre à jour des relations liant la vitesse d'apprentissage et les paramètres α et β ([FAHLMAN 88], [SATO 91] entre autres). Les résultats y sont en

général observés sur de petits problèmes d'apprentissage et se généralisent souvent mal aux problèmes de dimensions plus réelles. De même, les avis divergent sur la manière, éventuellement automatique, de faire varier ces paramètres au cours de l'algorithme. Aucune méthode infaillible n'existe à ce jour (à notre connaissance) et le réglage manuel reste souvent l'ultime recours en cas d'apprentissage délicat, rendant de ce fait l'algorithme par rétro-propagation non systématique. Si ceci peut être gênant dans certaines situations, ce ne l'est pas dans le cas de la classification où l'apprentissage est en général une étape différée qui peut prendre du temps et nécessiter l'intervention d'un opérateur, pourvu que le classifieur finalement obtenu soit efficace et rapide en phase de reconnaissance.

Descente en gradient stochastique

Une autre variation de la règle d'adaptation est la descente en gradient dite "stochastique" où les poids ne sont plus modifiés qu'en fonction de l'erreur C_i commise sur une forme X_i choisie pour l'itération courante (i=i(t)):

$$\Delta w_{kj}(t) = - \alpha \frac{\partial C_i}{\partial w_{kj}} \tag{2.6}$$

Cette version est la généralisation de la règle de Widrow-Hoff (règle du delta) vue pour l'Adaline. Plusieurs arguments plaident en faveur de l'utilisation de cette règle au détriment de la descente en gradient "standard".

Au premier rang de ceux-ci se trouve la possibilité offerte par cette règle de s'échapper des minima locaux de l'erreur moyenne C. En effet, la descente se faisant ici suivant le gradient de C_i et non selon celui de C, il est possible qu'au cours d'une itération la modification des poids selon $\partial C_i/\partial w_{kj}$ résulte en un nouvel ensemble de poids pour lesquels l'erreur C a augmenté. Si l'on se trouvait alors dans le bassin d'attraction d'un minimum local de C, cette correction non conforme à la règle du gradient standard pourra éventuellement permettre au réseau de s'échapper de cette situation dangereuse (Fig. 2.3). Ce phénomène est à rapprocher de la relaxation stochastique par recuit simulé que nous avons vue au § 1.3.2.1 [FOGELMAN SOULIE 91].

Le second argument provient de ce que cette règle d'adaptation, bien qu'ayant à un instant donné un comportement différent de celui de la règle standard, réalise quand même globalement une descente en gradient sur l'erreur quadratique moyenne C. Ce résultat, qui avait été démontré pour la règle de Widrow-Hoff, est donc également valide pour la règle du delta généralisée [HECHT-NIELSEN 91].

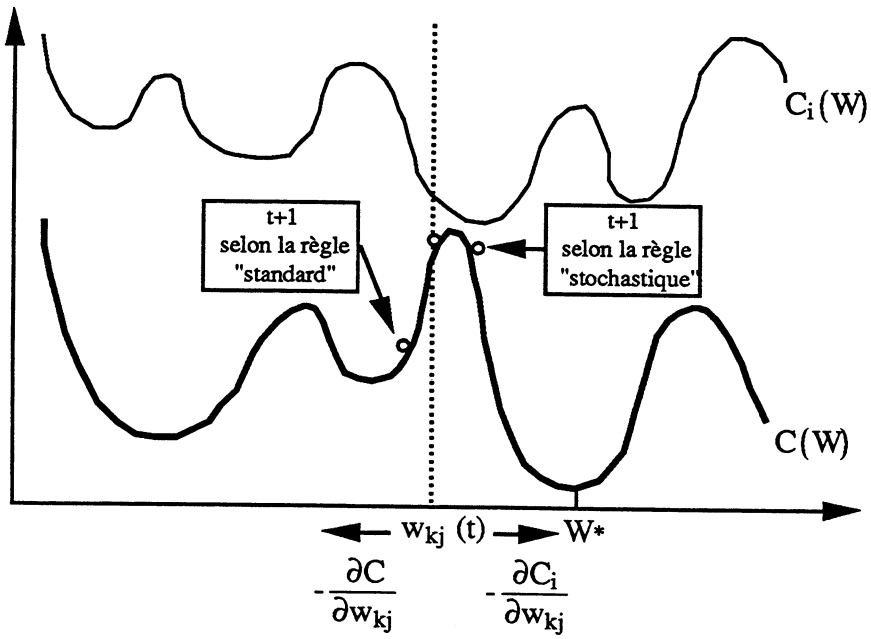


Fig. 2.3. La descente en gradient stochastique peut permettre de sauter des "barrières" de l'erreur moyenne C et de s'échapper ainsi du bassin d'attraction d'un minimum local.

Enfin cette règle respecte mieux l'idée que l'on peut se faire d'un système adaptatif, idée qui voudrait que le réseau s'adapte instantanément à la forme qui lui est présentée. Ceci semble plus naturel que la règle généralisée qui, si l'on pense en termes d'architecture physique, nécessiterait que chaque connexion puisse "mémoriser" une suite de corrections avant de finalement adapter sa pondération.

Puisque la règle stochastique modifie les poids après chaque présentation d'une forme, l'ordre dans lequel ces présentations sont effectuées prend ici de l'importance. La stratégie communément utilisée consiste à présenter toutes les formes de l'ensemble d'apprentissage, les unes après les autres, dans un ordre aléatoire, mais de telle sorte que des formes de même classe ne soient jamais présentées successivement. En effet, si cette contrainte n'est pas respectée, l'algorithme risque d'être attiré vers un minimum local correspondant à une situation où toutes les formes d'une même classe sont parfaitement apprises au détriment des autres classes.

Si l'ensemble d'apprentissage n'est pas représentatif de la distribution des classes, on pourra dupliquer dans l'ensemble E les formes des classes que l'on estime insuffisamment représentées.

Autres règles d'apprentissage

Beaucoup d'autres algorithmes d'apprentissage ont été développés depuis l'apparition de la rétro-propagation du gradient. Certains ont été inspirés de considérations "heuristiques", comme par exemple la version avec "inertie" vue précédemment, ou l'algorithme "Quickprop" [FAHLMAN 88]. D'autres proviennent de l'adaptation de techniques bien connues en optimisation : ce sont toutes les méthodes dites "du second ordre", méthodes qui reposent de manière plus ou moins lointaine sur la technique itérative de Newton. Des exemples de tels algorithmes peuvent être trouvés dans [WATROUS 87], [BECKER 88], [WANG 89]. Si ces algorithmes nécessitent effectivement moins d'itérations pour converger, le surcroît de calcul que nécessite chaque itération annule, en général, cet avantage [FOGELMAN SOULIE 91].

Dans les expériences que nous rapportons au chapitre 3, la règle d'adaptation utilisée est toujours la descente en gradient stochastique, sans "inertie". Le taux d'apprentissage α a toujours été adapté "manuellement". Avec l'initialisation des poids que nous avons décrite, les gradients sont en général importants au début de l'apprentissage et l'on utilise donc une valeur initiale de α faible, valeur que l'on augmente au fur et à mesure, les gradients devenant faibles en fin d'apprentissage (typiquement α varie de 0.05 à 1). Dans ces conditions, nous n'avons que rarement rencontré de problèmes de convergence, sauf bien sûr lorsque l'architecture du réseau n'était pas adaptée à la tâche demandée.

2.1.1.3 Critère d'arrêt

La procédure d'apprentissage recherchant itérativement un minimum de l'erreur C, l'algorithme doit théoriquement se terminer lorsqu'un tel minimum est atteint, c'est-à-dire lorsque le gradient de C par rapport aux poids est nul. En pratique, cette situation n'est jamais rencontrée, ne serait-ce que pour des raisons de précision numérique, et il est donc nécessaire de déterminer un critère d'arrêt plus réaliste. Une méthode fréquemment rencontrée consiste à arrêter l'apprentissage dès que l'erreur commise sur l'ensemble d'apprentissage devient suffisamment petite (un seuil est fixé *a priori*). Cette stratégie est pourtant à déconseiller pour deux raisons.

Tout d'abord, elle risque de conduire à des temps d'apprentissage inutilement longs. En effet, dans la tâche de classification qui nous intéresse, l'objectif est d'apprendre au réseau à associer sa classe à chaque forme de E. Nous l'avons vu, la classe d'une forme X_i est codée par un vecteur Y_i de $\{a,b\}^p$. Or la réponse du réseau à une forme est un vecteur $Y(X_i)$ de $]-1,1[^p$, vecteur qu'il va falloir interpréter en termes de classes. Cette interprétation peut être

correcte même si $Y(X_i)$ est différent de Y_i, c'est-à-dire même si l'erreur C n'est pas nulle ou proche de l'être. Le critère pour apprécier le degré d'avancement de l'apprentissage doit donc porter sur les résultats du réseau en termes de **classification.**

La seconde raison pour ne pas rechercher à tout prix une erreur minimale est qu'une telle recherche peut être parfois nuisible aux capacités de généralisation du réseau. Ce phénomène, appelé "sur-apprentissage" (overtraining), a été souvent mis en évidence et il est analysé dans [CHAUVIN 90]. Nous y reviendrons plus tard dans le paragraphe consacré à la généralisation.

Finalement, on utilisera de préférence un critère d'arrêt portant sur la qualité de la classification atteinte. Celui-ci consistera par exemple à attendre que le taux de reconnaissance atteigne un pourcentage suffisant. Pour être rigoureux, ce taux devra être mesuré sur un ensemble de validation E' distinct de l'ensemble d'apprentissage (validation croisée). Pour calculer ce taux, nous devons bien sûr préciser comment doit être interprétée la réponse du réseau à une forme.

Interprétation de la sortie du réseau

La réponse théorique souhaitée pour une forme de classe **q** est un vecteur V_q de $\{a,b\}^p$ dont seule la $q^{ème}$ composante vaut **a**, toutes les autres valant **b**. Le principe de l'apprentissage étant de minimiser (en moyenne) la distance entre la réponse du réseau à une forme de classe **q** et le vecteur V_q, on assignera tout naturellement à une forme inconnue X la classe **c** dont le vecteur représentant V_c est le plus proche de la réponse du réseau, Y(X) :

$$\|Y(X)-V_c\| = \min_{1\leq q\leq p} \{\|Y(X)-V_q\|\} \qquad (2.7)$$

On vérifiera aisément que lorsque **a** est supérieur à **b**, alors **c** est l'indice de la composante de Y(X) qui est maximale. La classe d'une forme est donc indiquée par la cellule de sortie ayant l'état maximal. Ceci se comprend d'autant mieux si l'on sait que, après apprentissage par minimisation de l'erreur C, l'état d'une cellule de sortie constitue une approximation de la probabilité *a posteriori* de la classe de X, p(c|X) : la règle de la sortie maximale s'interprète bien dans le cadre de la classification Bayesienne [FOGELMAN SOULIE 91].

Cependant, dans un problème de reconnaissance, il est souvent préférable de rejeter une forme plutôt que de commettre une erreur de classification. On peut alors définir un critère de rejet de différentes manières. Par exemple, le critère proposé dans [GUYON 91] consiste à considérer une forme comme non reconnue si l'état de la cellule de sortie maximale ne dépasse

pas un seuil θ_{max} ou (non exclusif) si la différence entre les deux plus grandes cellules de sortie n'excède pas un second seuil θ_{diff}. Ces deux seuils pourront être ajustés, en fonction de l'ensemble de validation E', de manière à fournir la meilleure classification possible (taux de reconnaissance maximal et taux d'erreur minimal).

2.1.2 Architecture

Nous venons de présenter quelques "heuristiques" susceptibles d'aider à la réussite de l'apprentissage par rétro-propagation du gradient. Cette réussite reste cependant conditionnée au choix de l'architecture du réseau, choix qu'aucune méthode ne permet de réaliser systématiquement.

2.1.2.1 Capacités d'apprentissage

La première question que l'on peut se poser face à un problème d'apprentissage est celle de l'existence : étant donné un ensemble E, existe-t-il une architecture de réseau définissant une fonction F_W telle que l'on puisse trouver un ensemble de poids W^* pour lesquels $F_{W^*}(X_i)=Y_i$ pour tout couple (X_i,Y_i) de E ? La seconde question est celle de la construction : comment déterminer une telle architecture ?

Une réponse positive a été apportée à la première question. Cette réponse est dérivée du théorème de Kolmogorov que l'on peut présenter ainsi :

> Toute fonction continue $f : [0,1]^n \rightarrow \mathbb{R}^p$ peut être implémentée par un réseau à trois couches : une couche d'entrée classique de **n** cellules, une couche cachée de (2n+1) cellules toutes complètement connectées aux cellules d'entrée, et une couche de sortie de **p** cellules totalement connectées aux cellules cachées.

Cependant les cellules cachées et de sortie qui sont évoquées ici peuvent être de nature quelconque et l'on ne peut hélas déduire de ce théorème qu'un réseau multi-couche d'automates linéaires généralisés comportant une couche cachée de (2n+1) éléments peut apprendre n'importe quelle tâche associative. Un résultat concernant ce type de réseaux a cependant été démontré [HECHT-NIELSEN 90] :

> Pour toute fonction f de $\mathcal{C}^2$, $f : [0,1]^n \rightarrow \mathbb{R}^p$, il existe un réseau d'automates linéaire généralisé à 3 couches, F_W, qui peut approcher f au sens des

moindres carrés avec n'importe quelle précision ε.

Dans notre cas, la fonction f n'est connue que par les couples (X_i, Y_i) de l'ensemble d'apprentissage E. Ce théorème nous permet donc de savoir que pour tout $\varepsilon>0$, il existe un réseau F_W à une couche cachée pour lequel il existe des poids W* tels que $C(W^*)<\varepsilon$. Mais ce théorème n'est pas constructif : il ne nous dit pas combien de cellules la couche cachée doit comporter. Il n'indique pas non plus comment déterminer W*.

On pourrait cependant en déduire que la bonne stratégie pour concevoir l'architecture d'un réseau consiste à n'utiliser qu'une seule couche cachée comportant un nombre "suffisamment" important de cellules, toutes totalement connectées à la couche d'entrée. Si cette stratégie peut effectivement permettre de remplir l'objectif d'apprentissage, il y a néanmoins de fortes chances pour que le réseau qui en résulte n'ait que de piètres capacités de généralisation.

2.1.2.2 Capacités de généralisation

En effet, Baum a démontré [BAUM 89], dans le cas de réseaux d'automates à seuil à une couche cachée totalement connectée, que la capacité de généralisation d'un réseau était liée au rapport entre la taille de ce réseau et le nombre de formes N qui lui étaient présentées pendant l'apprentissage. Ce résultat peut être formulé ainsi :

> Le taux de reconnaissance obtenu sur un ensemble de test par un réseau comportant |W| poids risque, avec une probabilité fixe, de ne pas dépasser (1-ε) si le nombre de formes utilisées pendant l'apprentissage, N, est inférieur à $\Omega(|W|/\varepsilon)$[1] (en supposant que l'ensemble de test et l'ensemble d'apprentissage sont issus d'une même distribution).

Des résultats similaires peuvent être attendus pour les réseaux d'automates linéaires généralisés qui nous intéressent et l'on peut les résumer informellement de la manière suivante : pour obtenir une bonne généralisation, le nombre de formes d'apprentissage doit être d'autant plus grand qu'est grande la taille du réseau. Inversement, pour un ensemble d'apprentissage donné, on recherchera un réseau qui soit le plus petit possible.

Ce résultat s'admet volontiers si l'on pense à l'apprentissage dans un réseau en termes

---

[1] $\Omega(f(N))$ désigne l'ensemble des fonctions qui sont supérieures à $k.f(N)$, $k>0$.

d'interpolation. L'apprentissage consiste en effet à interpoler par une fonction F_W une autre fonction que l'on ne connaît que par un ensemble de N points (X_i,Y_i). Le nombre de poids |W| représente alors le nombre de paramètres libres de la fonction d'interpolation F_W (pas tout à fait en réalité... voir [CHAUVIN 90]). Si ce nombre de paramètres est trop important en regard du nombre de points, on arrivera toujours à construire une fonction F_W qui passe exactement par tous les points, mais l'interpolation risque de ne pas être satisfaisante : il y a *overfitting* (sur-adéquation), c'est-à-dire, pour un réseau, mauvaise généralisation (Fig. 2.4).
C'est également de cette manière qu'il faut appréhender le problème du "sur-apprentissage" que nous évoquions précédemment : tenter de rendre l'erreur moyenne C nulle signifie essayer de faire passer exactement la fonction d'interpolation par tous les points, toujours au risque d'obtenir une mauvaise généralisation.

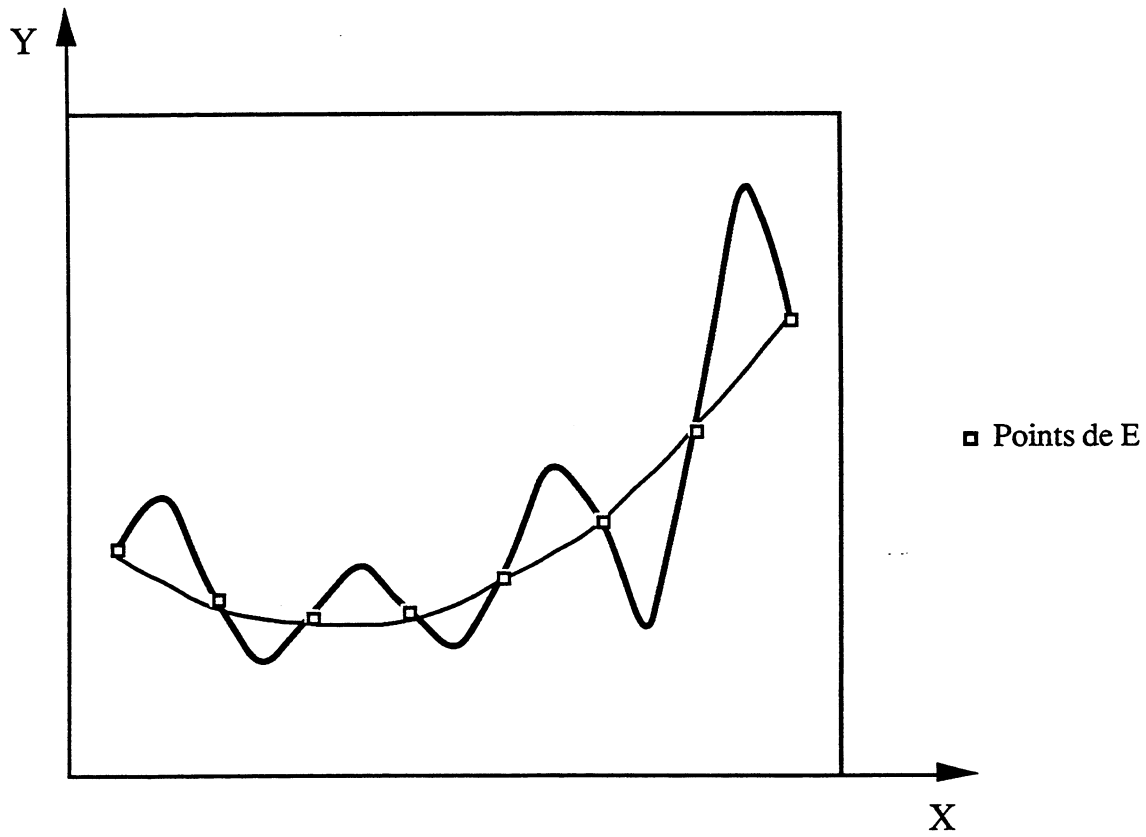


Fig. 2.4. Si on utilise un polynôme de degré important (trait gras), la fonction d'interpolation passe exactement par tous les points de E mais ne se comporte pas bien en dehors de ces points. En revanche, en interpolant par un polynôme du second degré (trait fin), la solution obtenue est satisfaisante bien qu'elle ne passe pas par tous les points.

Tout le problème consiste donc, pour un ensemble E donné, à construire un réseau qui soit suffisamment petit pour pouvoir être "entraîné" avec E à bien généraliser, mais qui soit

également suffisamment grand pour pouvoir apprendre E (une interpolation par une droite ne donnerait pas non plus de bon résultats dans l'exemple de la Fig. 2.4).

Certes, il est toujours possible de procéder par essai/erreur en réalisant plusieurs apprentissages sur des réseaux de taille croissante, jusqu'à ce que l'on trouve enfin un réseau qui satisfasse ces deux exigences contradictoires. Mais cette approche risque d'être coûteuse en temps et elle n'est en tout état de cause pas très satisfaisante "intellectuellement" parlant.

La démarche généralement adoptée consiste alors à tenter "d'incorporer" dans le réseau les quelques connaissances que l'on peut avoir *a priori* sur le problème d'apprentissage à résoudre. Ces connaissances peuvent en effet être utilisées pour contraindre l'architecture du réseau et ainsi réduire le nombre de paramètres libres de celui-ci.

2.1.2.3 Architectures contraintes

Contraindre l'architecture du réseau consiste à spécialiser les cellules en restreignant leur connectivité. Dans une architecture "générale", le réseau est construit à partir de cellules *totalement connectées*, c'est-à-dire que toute cellule d'une couche l, $l>1$, reçoit des connexions entrantes de toutes les cellules de la couche précédente l-1 (ou même de manière plus générale, de toutes les cellules de toutes les couches m précédentes, $m < l$).

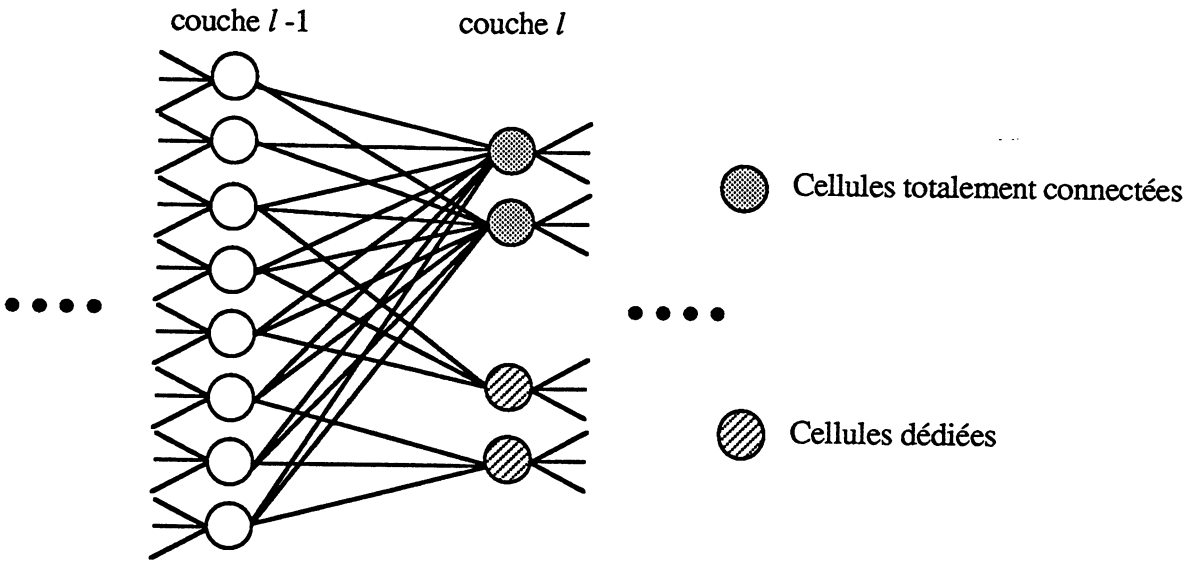


Fig. 2.5. Cellules dédiées et cellules totalement connectées.

A l'inverse, une architecture contrainte est composée de cellules dites *dédiées* qui ne

reçoivent de connexions entrantes que d'un ensemble limité de cellules de la couche précédente : une cellule dédiée a un *champ récepteur* restreint (Fig. 2.5). Bien sûr, ces champs récepteurs vont différer d'une cellule à l'autre.

Une cellule dédiée possède naturellement moins de poids qu'une cellule totalement connectée. A nombre de cellules égal, une architecture contrainte a donc moins de paramètres libres qu'une architecture générale et est ainsi susceptible de fournir une meilleure généralisation.

Ce phénomène a été analysé de manière probabiliste dans [SOLLA 89,90]. De manière informelle, cette analyse peut se résumer de la manière suivante. Pour une architecture de réseau donnée, il existe différentes configurations de poids pour lesquelles l'apprentissage peut être considéré comme accompli. Parmi celles-ci, seules un certain nombre produisent une généralisation satisfaisante. Une "bonne" architecture est une architecture pour laquelle la probabilité d'avoir une généralisation satisfaisante, lorsque l'apprentissage est réussi, est grande. Avec une architecture contrainte, cette probabilité est en principe plus importante qu'avec une architecture générale car le nombre de configurations correspondant à un apprentissage réussi est plus faible. Si l'architecture est "bien" contrainte, la plupart de ces configurations ne pourront produire que des généralisations similaires et satisfaisantes.

Une architecture contrainte, en offrant un ensemble de solutions réduit, augmente donc les chances qu'a l'algorithme d'apprentissage de trouver une "bonne" configuration de poids. Encore faut-il que cet ensemble ne soit pas vide. Aussi, on ne saurait concevoir de manière arbitraire une telle architecture en espérant qu'elle permettra d'une part d'accomplir l'apprentissage et d'autre part de bien généraliser. C'est à ce niveau que l'on va tenter d'utiliser des connaissances extérieures à l'ensemble d'apprentissage proprement dit, afin de construire des cellules dédiées de manière "intelligente".

Construction de cellules dédiées

La connaissance mise à contribution pour la construction de cellules dédiées porte généralement sur la nature des primitives utilisées pour la classification. Si l'on sait que les composantes des formes soumises au réseau constituent une collection de primitives de même nature, organisées selon une topologie, la stratégie consiste à construire, dans la première couche cachée, des cellules dédiées dont les champs récepteurs seront focalisés sur des groupes de primitives contiguës selon cette topologie.

Ceci est particulièrement utile lorsque les primitives "vues" par le réseau sont le résultat d'un processus d'échantillonnage d'un signal temporel (un son) ou d'un signal spatial (une image). Dans ce cas, les cellules dédiées de la première couche cachée deviennent des sortes d'extracteurs de primitives locales de "plus haut niveau" que celles qui sont soumises au réseau. Ces cellules étant à leur tour munies d'une topologie du fait de leur focalisation, il est alors possible de construire des cellules dédiées de la même manière dans la seconde couche cachée, cellules qui vont représenter des primitives de niveau encore plus élevé... et ainsi de suite.

De cette manière, chaque couche cachée peut se construire en fonction de la topologie de la couche précédente et l'on obtient, en quelque sorte, une suite "multi-échelle" de primitives, la classification étant réalisée par la couche de sortie dans l'espace des primitives de la dernière couche cachée. On retrouve dans cette façon de concevoir l'architecture l'une des deux manières d'appréhender le fonctionnement d'un réseau multi-couche, à savoir qu'un réseau est censé construire, de couche cachée en couche cachée, des représentations de plus en plus discriminantes, jusqu' à ce qu'enfin la séparation des classes soit possible dans la dernière couche.

Ce principe a été perfectionné et utilisé en reconnaissance de la parole : ce sont les réseaux à retard ou *TDNN* (Time Delay Neural Network) [WAIBEL 89]. Les architectures de ces réseaux sont contraintes non seulement à l'aide de cellules dédiées, mais également en introduisant la notion de *poids partagés*.
Une version de ces réseaux a été adaptée à la reconnaissance de caractères, ce sont les réseaux à *convolution* que nous allons décrire plus particulièrement maintenant.

2.1.2.4 Réseaux à convolution

Le réseau à convolution a été développé pour la reconnaissance optique de chiffres manuscrits provenant de codes postaux [LE CUN 90]. Les formes présentées au réseau sont ici des "imagettes" en niveaux de gris représentant le caractère à reconnaître (28x28 pixels).

La première couche cachée du réseau est organisée en *régiments*. Chaque régiment regroupe un ensemble de cellules (24x24) dont les champs récepteurs, limités à un petit voisinage (5x5), recouvrent toute l'image. Les sorties des cellules d'un même régiment représentent ainsi le résultat d'une convolution de l'image avec un masque composé des poids des connexions. Pour que ceci soit vraiment une convolution, il faut que toutes les cellules d'un régiment "voient" leur voisinage à travers le même masque de poids (Fig. 2.6). Ceci est réalisé en modifiant légèrement l'algorithme de rétro-propagation du gradient pour gérer cette notion de

poids partagés. La modification est la suivante :
soit $\{k_1j_1, k_2j_2, ..., k_sj_s\}$ un ensemble de connexions devant partager le même poids. A tout moment il faut donc avoir $w_{k1j1} = w_{k2j2} = ... = w_{ksjs}$. Pour cela, tous les poids sont initialisés avec la même valeur et à chaque itération de l'algorithme d'apprentissage, ils sont modifiés de la même manière : la variation Δw de ces poids est définie comme étant la moyenne des modifications qu'ils devraient subir s'ils n'étaient pas partagés :

$$\Delta w(t) = \frac{1}{s} \sum_{i=1}^{s} \Delta w_{k_i j_i}(t) \tag{2.8}$$

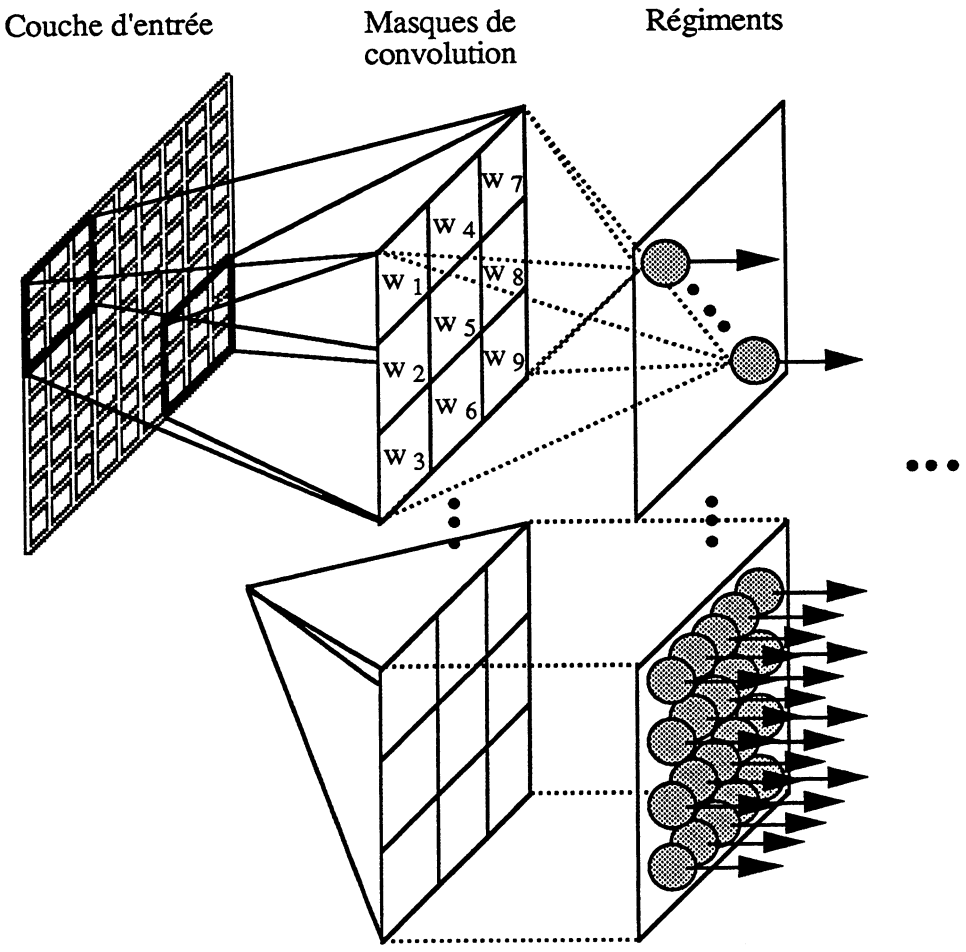


Fig. 2.6. Les cellules sont organisées en régiments. Chaque régiment implémente une convolution de l'image d'entrée avec un masque de poids partagés.

La seconde couche cachée du réseau comporte le même nombre de régiments que la première, chaque régiment étant chargé de réaliser un "sous-échantillonnage" de l'un des

régiments de la première couche cachée. Pour cela, les cellules ont des champs récepteurs qui définissent, sur le régiment à sous-échantillonner, une partition en voisinages (2x2). Le masque de poids (2x2) est bien sûr partagé pour toutes les cellules d'un même régiment et pour que ce masque réalise une sorte de moyenne, les quatre poids qui le composent sont contraints à être égaux (de la même manière que précédemment).

Le réseau est encore composé d'un enchaînement de deux autres couches cachées réalisant successivement convolution et sous-échantillonnage. La couche de sortie comporte classiquement autant de cellules que de classes et est totalement connectée à la dernière couche cachée (Fig. 2.7).

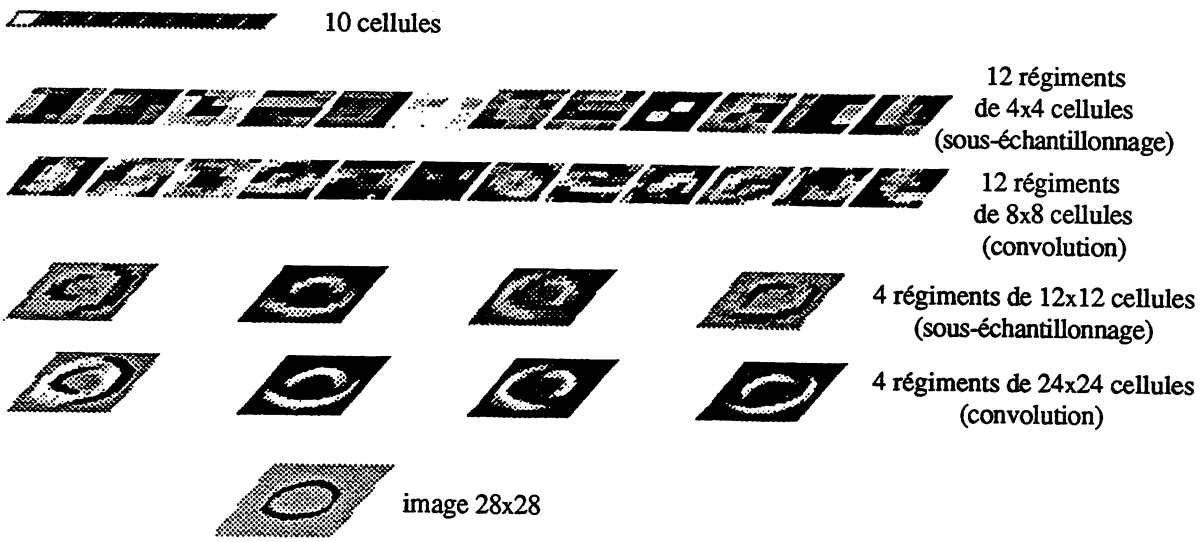


Fig. 2.7. Architecture du réseau à 4 couches cachées implémentant convolutions et sous-échantillonnages (extrait de [LE CUN 90]).

L'architecture ainsi obtenue intègre donc des connaissances issues de l'analyse d'images. Le réseau implémente dans ses couches cachées une chaîne de traitements que l'on effectue généralement en amont de l'étape de classification. Ici, les opérateurs d'extraction de primitives sont directement inclus dans le classifieur lui-même. L'intérêt majeur de cette approche réside dans le fait que l'algorithme d'apprentissage va alors déterminer les paramètres de ces opérateurs (c'est-à-dire les noyaux de convolution) de manière optimale selon un critère de **classification**.

En ce qui concerne l'apprentissage, on voit que celui-ci est fortement "guidé" par l'architecture, le nombre de paramètres libres étant restreint non seulement parce que des

cellules dédiées sont utilisées, ce qui réduit le nombre de connexions, mais également parce que ces connexions partagent leur poids. Le réseau décrit par [LE CUN 90] possède ainsi 4635 cellules mais seulement 98442 connexions pour lesquelles seules 2578 valeurs de poids indépendantes doivent être déterminées pendant l'apprentissage. Dans ces conditions, si l'on désirait construire un réseau à une couche cachée totalement connectée possèdant environ le même nombre de paramètres libres, celui-ci ne pourrait comporter que 4 cellules cachées et serait certainement incapable d'accomplir la même tâche.

Les performances rapportées concernant ce réseau sont comparables aux résultats des meilleures méthodes de reconnaissance de caractères manuscrits. Nous avons également expérimenté avec succès un réseau à convolution pour la reconnaissance de symboles musicaux (cf. chapitre 3).

2.1.2.5 Synthèse de l'architecture par "pré-apprentissage"

Nous venons de voir comment il est possible de contraindre l'architecture d'un réseau lorsque l'on connaît la structure des données qu'il doit manipuler. Cependant une telle information n'est pas toujours disponible.

Dans un problème de reconnaissance de formes, il peut arriver que l'on souhaite explicitement contrôler les primitives par lesquelles un objet sera décrit. Dans ce cas, on ne soumettra pas directement au réseau la forme "brute" (image ou son), mais on préfèrera d'abord pré-traiter cette forme et extraire des caractéristiques que l'on espère discriminantes. Ce sont ces primitives que l'on va alors soumettre au réseau. Elles peuvent être tout à fait quelconques et constituer une collection hétérogène.

Par exemple, Weideman utilise quatre types de mesures différentes pour caractériser un chiffre manuscrit : moments géométriques, descripteurs de Fourier et deux types de primitives "topologiques" [WEIDEMAN 89].
Une autre approche très répandue en reconnaissance de caractères consiste à utiliser des variables binaires pour coder la présence/absence de différentes primitives "structurelles" (segments de droite, trous, ...) en différentes positions d'un symbole [KAHAN 87]. C'est en utilisant un codage de cette nature (cf. chapitre 3) que nous avons été amenés à nous poser la question suivante : comment trouver un critère permettant de construire des cellules dédiées, alors que plus aucune organisation topologique évidente n'existe entre les primitives soumises au réseau ? La réponse que nous proposons passe par un "pré-apprentissage" utilisant des méthodes non statistiques pour sélectionner et grouper les primitives en ensembles d'intérêt sur

lesquels les cellules dédiées pourront se focaliser [MARTIN 90, 91ab, 92].

Méthodes de classification non statistiques

On désigne généralement par méthode de classification non statistique, un processus permettant de sélectionner, au cours d'une phase d'apprentissage supervisé, un ensemble de primitives dites "pertinentes" qui vont définir une partition de l'espace des formes en régions de décision. La méthode est qualifiée de *non statistique* lorsque le critère permettant de sélectionner une primitive ne repose pas sur l'estimation de densités de probabilité. Cette estimation est en effet impossible lorsque l'échantillon d'apprentissage est trop petit et dans ce cas, les méthodes non statistiques s'imposent [BONNEFOY 87].

L'algorithme que nous avons employé est adapté à la sélection de primitives binaires. Il consiste à construire un arbre de décision, arbre qui est binaire lorsque les primitives le sont aussi. Chaque nœud non terminal de l'arbre "porte" alors une primitive qui sépare l'espace des formes en deux. Chaque feuille de l'arbre "porte" l'étiquette d'une ou de plusieurs classes (Fig. 2.8).

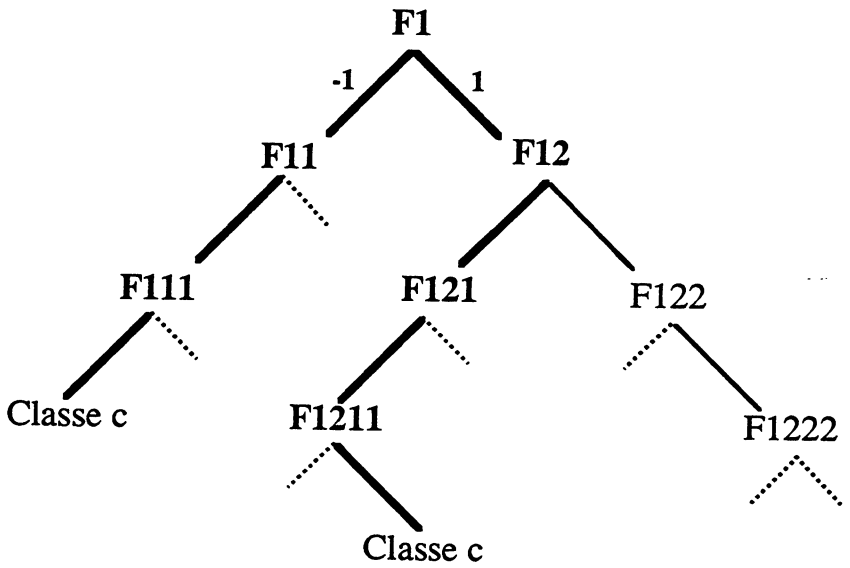


Fig. 2.8. Arbre de décision binaire.

Pour classifier une forme inconnue X avec un tel arbre, il faut, en partant de la racine, traverser chaque nœud à gauche ou à droite en fonction de la valeur que prend la primitive portée par le nœud courant pour la forme X (-1 ou 1 respectivement). Lorsque l'on atteint une feuille, la forme se voit attribuer la classe portée par cette feuille. Si plusieurs classes sont

possibles, on attribuera par exemple à X la classe de la forme de l'ensemble d'apprentissage qui "atterrit" dans la même feuille que X et qui est la plus similaire à X (selon un indice de similarité à choisir).
Nous allons maintenant détailler un algorithme de construction d'arbre, algorithme proposé dans [CHERIET 89]. Nous présenterons ensuite la manière dont nous l'utilisons.

Construction d'un arbre de décision binaire

On considère un ensemble d'apprentissage E de N formes X_i, formes décrites par **n** primitives binaires. Par convention, on suppose : $X_i = (x_{i1}, ..., x_{in})^t \in \{-1,1\}^n$. Soit c_i la classe de X_i, $1 \leq c_i \leq p$. L'arbre de décision est naturellement construit récursivement. A chaque nœud ν, on recherche parmi l'ensemble P_ν des primitives disponibles (c'est-à-dire parmi les primitives qui ne sont pas portées par un ancêtre de ν), la primitive pertinente d'indice F_ν qui sépare "au mieux" l'ensemble des formes E_ν ayant "atterri" dans ν. Cette recherche n'est effectuée que si l'ensemble C_ν des classes représentées dans E_ν contient au moins deux éléments. Dans le cas contraire, ν est étiqueté avec la seule classe de C_ν.

Si la recherche est effectuée mais n'aboutit pas, ν sera une feuille étiquetée avec toutes les classes de C_ν. Si elle aboutit, ν sera un nœud non terminal portant la primitive F_ν. Cette primitive sépare alors l'ensemble E_ν en deux ensembles $E_{\nu.1}$ et $E_{\nu.2}$ qui serviront à déterminer respectivement le fils gauche de ν, $\nu.1$, et le fils droit, $\nu.2$.
L'algorithme est donc le suivant :

```
Etape 0 :   /* Initialisation */
            ν = 1 /* racine */
            Eν = E ; Cν = {classes de Eν} ; Pν = {1, ..., n}

Etape 1 :   /* Construction récursive */
            Procédure TRAITER_NŒUD (ν, Eν, Cν, Pν)
              Si |Cν| = 1 alors début
                ν.type = feuille ; ν.classe = Cν  /* feuille non ambiguë */
              Fin si
              Sinon début
                Rechercher dans Pν la primitive pertinente Fν réalisant
                  la meilleure séparation de Eν
                Si une primitive Fν est trouvée alors début
                  ν.type = non terminal ; ν.primitive = Fν
                  Eν.1 = {Xi ∈ Eν / xiFν = -1 } ; Cν.1 = {classes de Eν.1}
                  Eν.2 = {Xi ∈ Eν / xiFν =  1 } ; Cν.2 = {classes de Eν.2}
                  Pν.1 = Pν.2 = Pν - {Fν}
```

```
            TRAITER_NŒUD (v.1, E_v.1, C_v.1, P_v.1)
            TRAITER_NŒUD (v.2, E_v.2, C_v.2, P_v.2)
        Fin si
        Sinon début
            v.type = feuille ; v.classe = C_v  /* feuille ambiguë */
        Fin sinon
    Fin sinon
Fin procédure
```

Recherche de la primitive pertinente

Pour un nœud donné de l'arbre, v, on recherche donc une primitive $F_v \in P_v$ qui est pertinente pour séparer l'ensemble E_v. Pour cela, toutes les primitives de P_v vont être évaluées selon deux critères : on va d'une part mesurer la stabilité (robustesse) de chaque primitive et d'autre part sa capacité à séparer de manière équilibrée l'ensemble E_v.

Une primitive $F_j \in P_v$ est dite *stable* pour une classe $c_k \in C_v$ si la valeur de cette primitive est identique pour toutes les formes $X_i \in E_v$ de classe c_k.

Une primitive $F_j \in P_v$ réalise une *séparation équilibrée* de E_v si le nombre de classes pour lesquelles F_j est stable et vaut -1, est environ égal au nombre de classes pour lesquelles F_j est stable et vaut 1.

Deux quantités sont calculées pour mesurer d'une part le degré de stabilité et d'autre part le pouvoir de séparation équilibrée. On définit tout d'abord pour chaque primitive $F_j \in P_v$ et pour chaque classe $c_k \in C_v$ la valeur ε_{jk} de la manière suivante :

$$\varepsilon_{jk} = \begin{cases} 1 \text{ si } x_{iF_j} = 1 \ \forall \ X_i \in E_v, X_i \text{ de classe } c_k \\ -1 \text{ si } x_{iF_j} = -1 \ \forall \ X_i \in E_v, X_i \text{ de classe } c_k \\ 0 \text{ sinon} \end{cases} \tag{2.9}$$

Le degré d'instabilité de F_j, β_j, est défini comme étant le nombre de classes pour lesquelles F_j n'est pas stable :

$$\beta_j = |C_v| - \sum_{c_k \in C_v} |\varepsilon_{jk}| \tag{2.10}$$

où $|C_v|$ dénote le cardinal de C_v et $|\varepsilon_{jk}|$ la valeur absolue de ε_{jk}. On a : $0 \leq \beta_j \leq |C_v|$.

Le degré de déséquilibre de la séparation réalisée par F_j, δ_j, est défini comme étant la différence (positive) entre le nombre de classes pour lesquelles F_j est stable et vaut -1 et le nombre de classes pour lesquelles F_j est stable et vaut 1 :

$$\delta_j = \left| \sum_{\alpha_k \in C_v} \varepsilon_{jk} \right| \qquad (2.11)$$

On a : $0 \leq \delta_j \leq | C_v |$.

La procédure proposée dans [CHERIET 89] consiste à retenir tout d'abord les primitives ayant le degré d'instabilité minimal si ce minimum est inférieur à un seuil β_0. On retient donc l'ensemble des primitives $\{F_s\}$ telles que $\beta_s = \min \{ \beta_j \leq \beta_0, F_j \in P_v \}$. Si plusieurs candidates sont retenues, on ne garde finalement que la primitive F_{s0} dont le degré de déséquilibre est minimal et inférieur à un seuil δ_0 : $F_v = F_{s0} / \delta_{s0} = \min \{ \delta_s \leq \delta_0, F_s \in \{F_s\} \}$.

Bien sûr, selon la manière dont on fixera les seuils β_0 et δ_0, la recherche de la primitive F_v fournira ou non une réponse. Si l'on souhaite développer l'arbre de décision de manière à ce que chaque feuille ne porte l'étiquette que d'une seule classe, les seuils seront alors fixés à leur valeur maximale : $\beta_0 = \delta_0 = | C_v |$. Dans ces conditions, l'arbre risque de grandir de manière importante si certaines classes ont de nombreux "traits communs". A l'extrême, l'arbre pourra se développer jusqu'à ce que chaque feuille isole une forme d'apprentissage particulière.

Au contraire, si l'on désire seulement obtenir une première décomposition grossière mais robuste de l'espace des formes, les seuils seront fixés à des valeurs inférieures. A titre d'exemple, les valeurs proposées dans [CHERIET 89] sont : $\beta_0 = | C_v | / 5$ et $\delta_0 = | C_v | / 2$. Dans ce cas, l'arbre ne pourra effectivement servir que de pré-classifieur et il sera nécessaire de développer pour chaque feuille ambiguë une procédure de décision permettant de séparer les différentes classes qu'elle contient.

Construction de cellules dédiées

La classification à l'aide d'un arbre de décision est intéressante car elle regroupe les primitives binaires en ensembles qui constituent des primitives discriminantes de plus haut niveau. Chaque feuille de l'arbre définit en effet un ensemble composé des primitives que l'on rencontre sur le chemin menant de la racine à cette feuille. Un tel ensemble constitue en quelque sorte un "trait caractéristique" de la classe (ou des classes) dont l'étiquette est portée par la feuille. Si aucune autre feuille ne porte l'étiquette de cette classe, ce trait caractéristique est même un invariant pour toutes les formes d'apprentissage de cette classe.

En construisant l'arbre binaire sur l'ensemble d'apprentissage, on peut donc obtenir un moyen de regrouper les primitives soumises au réseau en groupes pertinents, tout comme les pixels sont regroupés par voisinage dans un réseau à convolution. La différence réside ici dans le fait que ces groupes sont déterminés automatiquement là où il fallait précédemment l'intervention d'un opérateur doté de connaissances en traitement d'images.

L'architecture du réseau peut alors être simplement "synthétisée" en fonction de l'ensemble d'apprentissage de la manière suivante : une cellule dédiée est construite dans la première (et unique) couche cachée pour chaque feuille de l'arbre. Cette cellule ne reçoit de connexions entrantes que des cellules de la couche d'entrée qui correspondent aux primitives rencontrées en cheminant sur l'arbre de la racine à la feuille considérée. La sortie de la cellule n'est également connectée qu'aux cellules de sortie qui correspondent aux classes dont les étiquettes sont portées par cette feuille (Fig. 2.9).

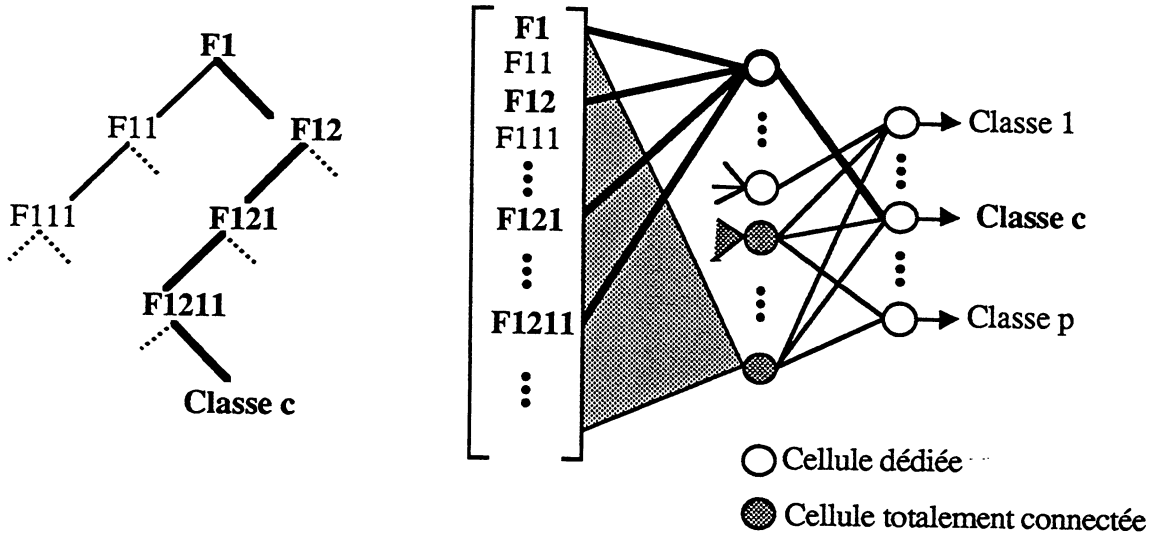


Fig. 2.9. L'architecture du réseau (à droite) est déduite de l'arbre de décision construit sur l'ensemble d'apprentissage (à gauche). Les cellules d'entrée ne sont pas représentées.

L'architecture que l'on construit ainsi est évidemment très contrainte, le nombre de connexions dans le réseau étant en général très faible en regard de la taille de l'ensemble d'apprentissage. On peut donc légitimement se demander d'une part si le réseau obtenu pourra effectivement apprendre cet ensemble d'apprentissage et d'autre part s'il sera capable de généraliser correctement.

Discussion

En fait, cette manière de contraindre l'architecture peut tout à fait être justifiée. L'arbre de décision que l'on a construit définit en effet l'appartenance à une classe sous la forme d'une fonction booléenne. Si l'on considère qu'une primitive binaire est une variable logique (vrai=1, faux=-1), l'appartenance à une classe va alors s'exprimer comme une forme normale disjonctive. En effet, une forme est de classe **c** si elle atterrit dans l'une des feuilles de l'arbre portant l'étiquette **c**, ce qui se traduit par une disjonction de termes décrivant l'appartenance à une feuille donnée. Ces termes sont quant à eux des conjonctions, chaque conjonction décrivant les valeurs que doivent prendre les primitives menant à la feuille considérée. Si l'on considère l'arbre binaire donné en exemple dans la figure 2.8, la classe **c** étant "portée" par deux feuilles, l'appartenance à cette classe s'écrit :

$$\text{Est\_de\_classe\_c} = F_1 \wedge \overline{F_{12}} \wedge \overline{F_{121}} \wedge F_{1211} \vee \overline{F_1} \wedge \overline{F_{11}} \wedge \overline{F_{111}}.$$

Or le calcul de formes normales disjonctives s'implémente très facilement à l'aide d'un réseau à trois couches d'automates à seuil. La couche cachée contient autant d'automates qu'il y a de termes conjonctifs, chaque automate calculant le ET logique de ses entrées. La couche de sortie contient pour sa part autant d'automates que de formes normales disjonctives, chaque automate calculant le OU logique de ses entrées.

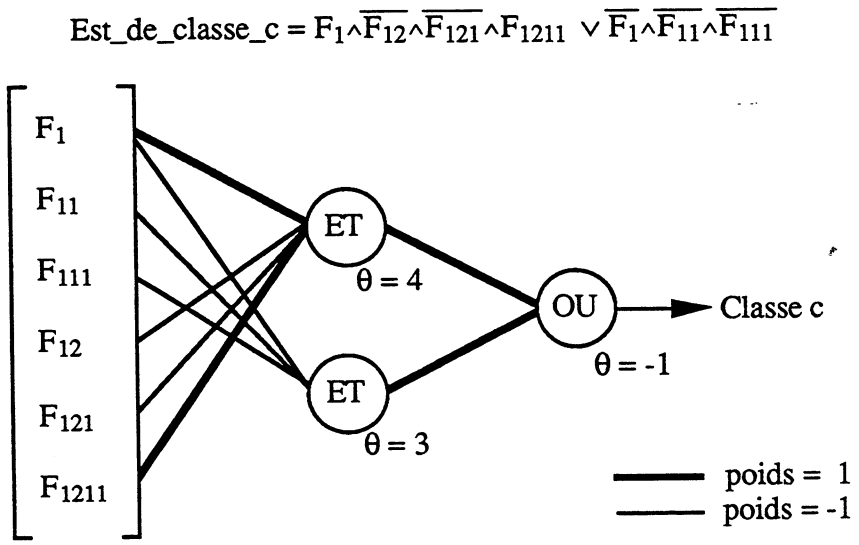


Fig 2.10. Implémentation d'une forme normale disjonctive à l'aide d'un réseau à trois couches d'automates à seuil.

Pour calculer un ET logique de v variables à l'aide d'un automate à seuil tel que celui vu au § 1.2.1.1, il suffit de fixer le seuil de cet automate à v. Le poids de la connexion entrante correspondant à une variable sera fixé à 1 si la variable n'apparaît pas inversée dans la conjonction (nœud traversé à droite), sinon il sera fixé à -1 (nœud traversé à gauche).
Pour calculer un OU logique, on fixera les poids de la même manière mais le seuil vaudra alors 1-v (puisque les variables prennent leur valeur sur {-1,1}) (Fig. 2.10).

L'architecture du réseau à rétro-propagation du gradient que nous construisons à partir de l'arbre est donc celle du réseau d'automates à seuil qui permettrait d'implémenter exactement le processus de classification de l'arbre binaire. Si l'arbre binaire réalise un taux de reconnaissance de 100% sur l'ensemble d'apprentissage (c'est-à-dire s'il n'y a pas de feuilles ambiguës, ce que l'on peut toujours obtenir), alors nous sommes sûrs qu'il existe au moins une configuration de poids correspondant à un apprentissage réussi : c'est la configuration des poids du réseau d'automates à seuil tel que nous venons de les définir (en jouant sur l'interprétation des cellules de sortie d'un réseau à unités sigmoïdales, il est tout à fait possible de lui donner le même comportement qu'un réseau d'automates à seuil).

La façon dont nous avons contraint l'architecture est donc valide puisqu'elle garantit l'existence d'une solution au problème de l'apprentissage. Pour autant, permet-elle une bonne généralisation ?

Pratiquement, nous nous sommes aperçus que l'architecture ainsi construite était tellement contrainte que l'apprentissage convergeait toujours vers une configuration de poids correspondant à l'implémentation directe de l'arbre binaire. Or un arbre binaire ne fournit pas, en général, une généralisation excellente. Cette méthode a surtout été développée pour faire de la pré-classification et non pas pour une étape de classification finale.

Pour permettre au réseau de produire une généralisation différente, nous avons dû en quelque sorte "relâcher" la contrainte architecturale. Pour cela, nous avons autorisé l'ajout, dans la couche cachée, de "quelques" cellules totalement connectées à côté des cellules dédiées (Fig. 2.9). Intuitivement, ceci permet d'élargir le domaine des configurations de poids correspondant à un apprentissage réussi (en ayant toujours la garantie que ce domaine n'est pas vide) et donc le domaine des fonctions que le réseau peut réaliser. On augmente ainsi la probabilité de tomber sur une solution qui fournisse une meilleure généralisation que celle de l'arbre.

Cette démarche est à l'évidence très empirique et, à nouveau, elle exige que l'opérateur fasse des choix guidés par sa seule intuition. Evidemment, le nombre de cellules totalement

connectées que l'on ajoute ne doit pas être trop important afin de ne pas trop augmenter le nombre de paramètres libres du réseau et perdre alors le bénéfice de la contrainte.

Expérimentalement, cette approche semble valide et l'on obtient des réseaux compacts qui ont des capacités de généralisation de l'ordre des meilleurs réseaux à une couche cachée totalement connectée, tout en étant nettement plus petits (cf. chapitre 3). La manière dont sont construites les cellules dédiées (focalisation sur des primitives stables et discriminantes pour une classe) est bien adaptée aux problèmes réels de reconnaissance des formes où il existe en général pour chaque classe des "invariants", invariants que de telles cellules permettent de détecter. Cette approche ne fonctionnera pas sur des problèmes plus artificiels (le problème de la parité par exemple) où la classification repose sur la recherche de relations plus complexes entre primitives, relations que l'arbre de décision utilisé sera incapable de prédire.

2.1.3 Conclusion

Les réseaux multi-couches à rétro-propagation du gradient sont des modèles très puissants pratiquement capables de s'accommoder de n'importe quel problème d'associations entrées-sorties. Pour une architecture donnée, il existe en général différentes configurations de poids qui permettent d'accomplir la tâche associative spécifiée par l'ensemble d'apprentissage. L'algorithme d'apprentissage, en explorant l'espace des poids, peut permettre de trouver une telle configuration. Mais toutes ces solutions ne sont pas satisfaisantes en termes de généralisation et rien ne garantit *a priori* que l'algorithme en exhibera une qui le soit.

On est donc amené à essayer de restreindre l'ensemble des solutions de l'apprentissage à un ensemble de solutions fournissant une bonne généralisation. Ceci est généralement réalisé[1] en tentant de contraindre l'architecture de manière pertinente. Pour cela, il est nécessaire de comprendre la nature de la généralisation qu'une architecture de réseau peut fournir. Cette analyse est beaucoup trop complexe si l'on considère les réseaux à unités sigmoïdales utilisés pour la rétro-propagation du gradient. En utilisant des réseaux composés d'unités plus simples, on peut espérer une meilleure compréhension des phénomènes et obtenir des méthodes permettant de guider la construction de réseaux à unités sigmoïdales.

En considérant des cellules à fonction de transition linéaire, il a ainsi été possible de

---

[1] D'autres types de contraintes ont parfois été utilisés pour améliorer la généralisation, comme la méthode des *suggestions* (hints) proposée par [SUDDARTH 90].

mettre en évidence les relations liant les réseaux multi-couches et l'analyse de données classique [GALLINARI 88], [RADOUI 90] ou les méthodes bayésiennes [THIRIA 89].

On peut également considérer des réseaux d'automates à seuil : c'est ce que nous avons commencé à faire dans le cadre d'un problème de classification de formes binaires. Nous allons maintenant poursuivre cette démarche et aborder le problème général de la classification à l'aide de réseaux multi-couches d'automates à seuil, ou *Gamba Perceptrons*.

2.2 SYNTHESE DE RESEAUX MULTI-COUCHES D'AUTOMATES A SEUIL

La généralisation fournie par un réseau à unités sigmoïdales, après apprentissage par rétro-propagation du gradient, est difficilement caractérisable, si ce n'est par le taux de classification qu'elle réalise sur un ensemble test de formes inconnues. En utilisant des réseaux multi-couches d'automates à seuil, il devient possible de caractériser précisément cette généralisation par l'intermédiaire des régions de décision qu'un tel réseau définit dans l'espace des formes d'entrée. En connaissant la nature de ces régions, il devient alors possible de concevoir des algorithmes d'apprentissage qui produisent de bonnes généralisations, c'est-à-dire des algorithmes qui "dessinent" explicitement les régions de décision désirées.

2.2.1 Caractérisation des régions de décision

2.2.1.1 Régions de décision d'un automate à seuil

Comme nous l'avons déjà vu au § 1.2.1.3, un automate à seuil **k** délimite dans l'espace de ses entrées deux régions de décision : l'ensemble H_k^+ des formes pour lesquelles sa sortie, y_k, vaudra +1, et l'ensemble H_k^- des formes pour lesquelles sa sortie vaudra -1. Ces deux régions sont parfaitement connues (Fig. 2.11) : ce sont les deux demi-espaces définis par l'hyperplan H_k dont l'équation est déterminée par les poids w_{kj} des connexions entrantes de l'automate et par son seuil θ_k (Fig. 2.11) :

$$\begin{aligned} H_k^+ &= \{X \in \mathbb{R}^n / y_k(X) = \mathbb{1}(net_k(X)) = 1\} \\ H_k^- &= \{X \in \mathbb{R}^n / y_k(X) = \mathbb{1}(net_k(X)) = -1\} \end{aligned} \qquad (2.12)$$

Comme précédemment, $net_k(X) = W_k.X - \theta$ et $\mathbb{1}(x)$ désigne la fonction seuil usuelle : $\mathbb{1}(x)=1$ si $x \geq 0$, -1 sinon.

Un réseau à deux couches (la couche d'entrée habituelle et une couche d'automates à seuil totalement connectés, un pour chaque classe) permet donc de définir des régions de décision

simples, simplicité dont nous avons évoqué les limites dans le paragraphe consacré au Perceptron et à l'Adaline. Que deviennent ces régions de décision si l'on ajoute des couches cachées à un réseau d'automates à seuil ?

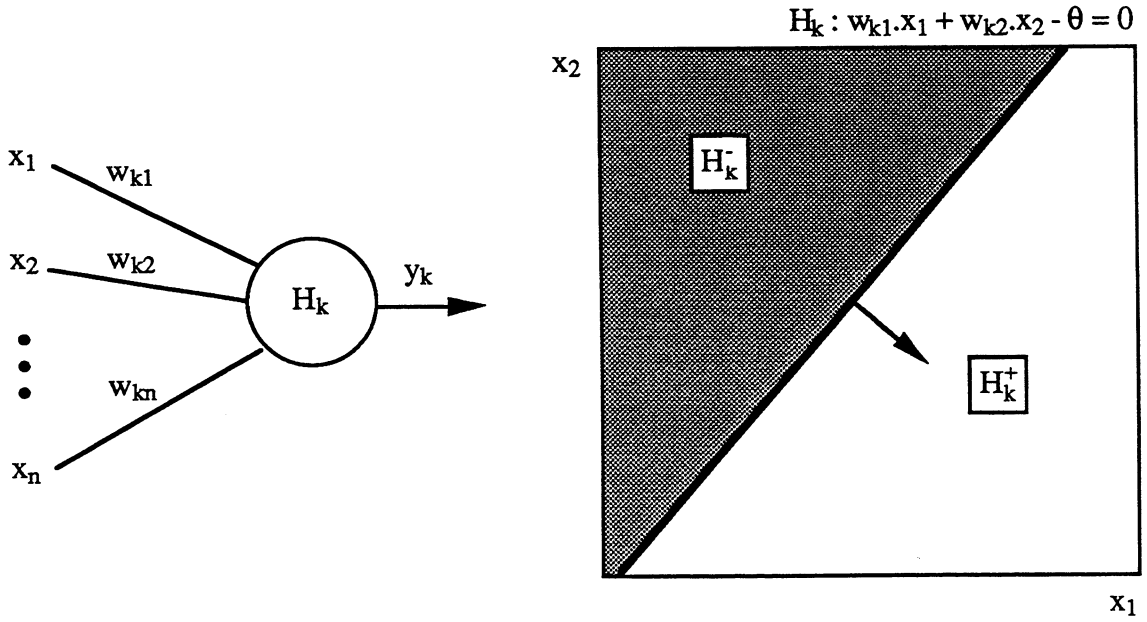


Fig. 2.11. Régions de décision d'un automate à seuil dans $\mathbb{R}^n$, n=2.

2.2.1.2 Régions de décision d'un réseau multi-couches

Considérons un réseau multi-couches dont la première couche cachée comporte **m** automates à seuil totalement connectés à la couche d'entrée. Ces **m** automates correspondent à **m** hyperplans dans l'espace des formes $\mathbb{R}^n$. Ces **m** hyperplans engendrent une partition de $\mathbb{R}^n$ en 2^m régions. Chaque région est composée de l'ensemble des points ayant même image dans l'espace $\{-1,1\}^m$ des états des automates de la première couche. Soit $B=(b_1, ...,b_m)^t$ un point de $\{-1,1\}^m$. La région ayant B pour étiquette, P_B, est définie de la manière suivante :

$$P_B = \{ X \in \mathbb{R}^n / (y_1(X), ...,y_m(X))^t = B \} \qquad (2.13)$$

Une telle région est définie comme étant l'une des intersections que l'on peut construire avec les demi-espaces définis par chaque automate **k**. C'est donc un polyèdre[1] (convexe) de $\mathbb{R}^n$,

---

[1] Pas tout à fait, car un polyèdre est une intersection de demi-espaces **fermés** [PREPARATA 85] , or seuls les H_k^+ sont fermés, les H_k^- étant ouverts... mais cette nuance est sans conséquence ici.

éventuellement vide :

$$P_B = \bigcap_{k=1}^{m} H_k^{Sgn(b_k)} \tag{2.14}$$

où $Sgn(b_k)$ désigne le signe (+ ou -) de b_k.

La première couche d'un réseau d'automates à seuil peut ainsi être vue comme une transformation qui, à chaque forme X de $\mathbb{R}^n$, associe l'étiquette B(X) du polyèdre convexe auquel X appartient : $P_{B(X)}$.

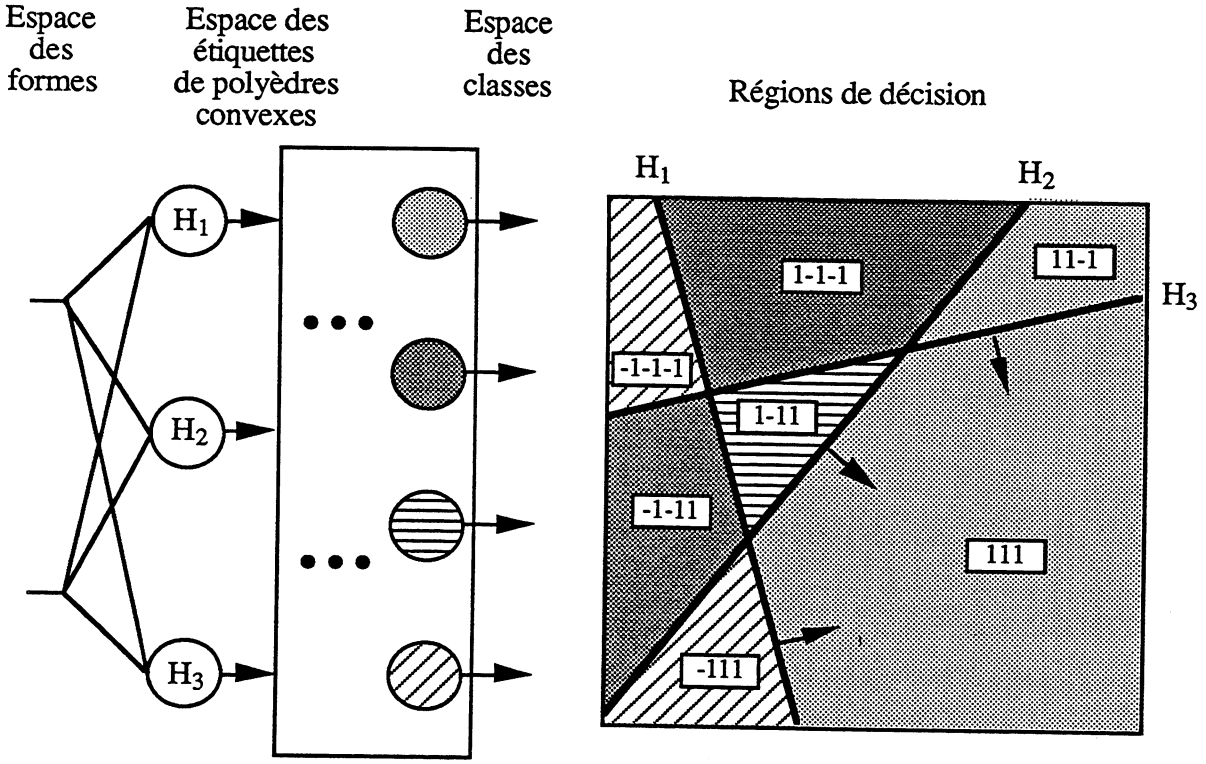


Fig 2.12. Les m=3 automates de la première couche cachée définissent, dans l'espace des formes $\mathbb{R}^n$ (n=2), sept polyèdres convexes ($P_{(-11-1)} = \varnothing$). Les p=4 régions de décision sont des unions de ces polyèdres.

Dans cette perspective, le seul rôle des couches supérieures du réseau consiste alors à réaliser une transformation de l'espace des étiquettes de polyèdre, $\{-1,1\}^m$, dans l'espace des cellules de sortie, $\{-1,1\}^p$, qui est l'espace des classes (Fig. 2.12).

Ainsi, une forme X se verra attribuer la classe **c** si l'image de son étiquette B(X) dans l'espace des cellules de sortie est le vecteur V_c représentant de cette classe (comme toujours, V_c sera un vecteur dont seule la $c^{ème}$ composante vaut 1). La région de décision d'une classe **c** est

donc constituée de l'union des régions P_{Bi} dont les étiquettes B_i ont pour image V_c.

2.2.1.3 Formulation géométrique de l'apprentissage

Un réseau d'automates à seuil comportant au moins une couche cachée permet donc de définir des régions de décision qui sont des unions de polyèdres convexes. Cette constatation permet de formuler le problème de l'apprentissage de manière géométrique :

> Etant donné notre ensemble E composé de N formes $X_i \in \mathbb{R}^n$, X_i de classe c_i, l'apprentissage supervisé consiste à déterminer une partition de $\mathbb{R}^n$ en polyèdres convexes que nous appellerons **homogènes**. Un polyèdre P est dit homogène vis-à-vis de E si tous les points de E qu'il contient sont de la même classe : $\forall X_i, X_j \in E$, si $X_i \in P$ et $X_j \in P$ alors $c_i = c_j$.

Le premier problème consiste bien sûr à trouver une telle partition.
Le second problème consiste à "synthétiser" un réseau d'automates à seuil tel que la région de décision d'une classe, dans ce réseau, soit bien égale à l'union des polyèdres de la partition qui contiennent des formes de cette classe. Si cette synthèse est possible, alors évidemment le réseau obtenu classifiera exactement toutes les formes de E. En fait, on peut toujours obtenir un tel réseau.

Synthèse du réseau

Pour cela, la première couche cachée de ce réseau devra comporter un automate pour chaque hyperplan nécessaire à la définition de la partition en polyèdres. Les poids de ces automates seront bien sûr déterminés par les équations des hyperplans. Cette première couche, une fois construite, définit dans son espace de sortie une étiquette pour chaque polyèdre.

Les couches suivantes doivent alors être construites de manière à associer à chaque étiquette la bonne classe, c'est-à-dire la classe des points de l'ensemble d'apprentissage qui appartiennent au polyèdre désigné par l'étiquette considérée. C'est en fait un problème de classification de formes binaires qui peut être résolu de différentes manières. La solution la plus directe est la suivante :

Soit **m** le nombre d'automates créés dans la première couche. Soit P_j, $1 \leq j \leq n_c$ les polyèdres de la partition qui contiennent des formes de classe **c**. L'étiquette d'un polyèdre P_j est notée B_j, $B_j = (b_{j1}, ..., b_{jm})^t \in \{-1,1\}^m$. Il faut donc que la $c^{ème}$ cellule de sortie du réseau réponde 1 lorsque les sorties des automates de la première couche produisent l'un des B_j, et -1

sinon. A nouveau, ceci peut être vu comme la réalisation d'une fonction booléenne à **m** variables :

$$\text{Est\_de\_classe\_c (B)} = \begin{cases} \text{vrai si } B \in \{B_1, ..., B_{n_c}\} \\ \text{faux sinon} \end{cases} \tag{2.15}$$

Or une fonction booléenne peut toujours se mettre sous sa forme normale disjonctive dite "principale" :

$$\text{Est\_de\_classe\_c (B)} = \bigvee_{j=1}^{n_c} \left(\bigwedge_{k=1}^{m} t_{jk} \right) \tag{2.16}$$

où $B = (b_1, ..., b_m)^t$, $t_{jk} = b_k$ si $b_{jk} = 1$ et $t_{jk} = \overline{b_k}$ si $b_{jk} = -1$.
Nous avons vu au § 2.1.2.5 qu'une forme normale disjonctive pouvait être implémentée par deux couches d'automates à seuil. On sait donc construire les couches supérieures du réseau de manière à associer la classe qui convient à chaque étiquette de polyèdre.

2.2.1.4 Conclusion

Notre problème est ainsi résolu : il est toujours possible de synthétiser un réseau d'automates à seuil qui produise les régions de décision voulues si celles-ci ont été exprimées sous forme d'unions de polyèdres convexes. Pour cela, le réseau doit en principe comporter trois couches (en plus de la couche d'entrée) :

- Dans la première couche cachée, chaque automate définit un hyperplan.
- Dans la seconde couche cachée, chaque automate définit un polyèdre convexe en intersectant les demi-espaces définis par la couche précédente. Pour cela, il réalise un ET logique des automates de la première couche qui correspondent aux hyperplans définissant la frontière du polyèdre considéré.
- Dans la couche de sortie, chaque automate définit une région de classe en faisant l'union des polyèdres définis par la couche précédente. Pour cela, il réalise un OU logique des automates de la seconde couche qui correspondent aux polyèdres contenant des formes d'apprentissage de la classe considérée.

On peut noter que, dans certains cas, moins de deux couches cachées sont nécessaires. Cependant, on peut montrer facilement qu'un réseau à une couche cachée ne peut définir n'importe quelle union de polyèdres convexes [GIBSON 90]. Pour s'en convaincre, il suffit de considérer l'exemple de la figure 2.13. Si le réseau ne comportait qu'une seule couche cachée, l'automate de la couche de sortie correspondant à la région de décision grisée devrait répondre 1

pour les étiquettes de polyèdre (1,1) et (-1,-1) et il devrait répondre -1 pour les deux autres étiquettes, (-1,1) et (1,-1). En d'autres termes, cet automate devrait réaliser un XOR, ce qui est, nous le savons, impossible.

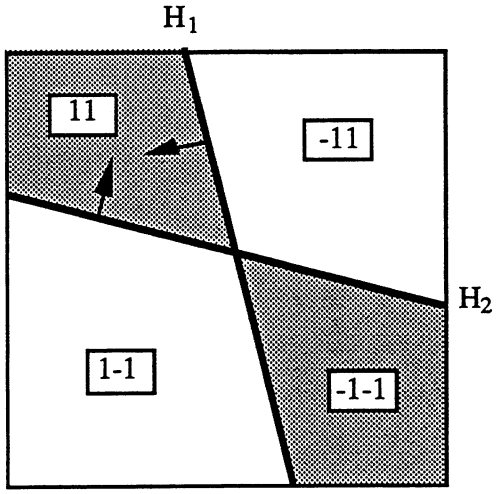


Fig. 2.13. Régions de décision qui ne peuvent être réalisées par un réseau à une seule couche cachée.

Il faut noter également que réaliser l'association entre étiquettes de polyèdres et classes en implémentant les formes normales disjonctives principales (FND) est coûteux : cela nécessite un automate pour chaque polyèdre (c'est-à-dire pour chaque terme conjonctif). Il existe des algorithmes permettant de simplifier ces FND (voir [TOMAN 91] par exemple). On pourrait également envisager d'employer l'un des algorithmes d'apprentissage qui permettent de générer des réseaux d'automates à seuil pour la classification exacte de formes binaires [NADAL 89], [FREAN 90]. Ces algorithmes utiliseront moins d'automates pour réaliser les associations voulues, mais plus de deux couches peuvent être nécessaires.

Finalement, la construction du réseau étant réalisable, le problème de l'apprentissage se réduit à la recherche d'une partition de l'espace des formes en régions de décision exprimées sous forme d'unions de polyèdres convexes. Plusieurs méthodes de classification définissent de telles régions et sont donc susceptibles d'être "traduites" à l'aide d'un réseau, à condition toutefois que l'on puisse explicitement calculer les régions produites.

La première méthode à laquelle on peut penser est la classification au plus proche voisin. Nous avons vu, lors du premier chapitre, des modèles de réseaux de neurones qui réalisaient une telle classification. Nous allons maintenant montrer que cette classification est également possible à l'aide de réseaux multi-couches d'automates à seuil (solution développée

indépendamment par [MURPHY 90] et [MARTIN 91cd]).

2.2.2 Réseaux multi-couches "plus proche voisin"

La région de décision d'une classe **c** produite par une classification au plus proche voisin selon un ensemble d'apprentissage E est constituée d'une union de régions élémentaires. Ces régions élémentaires sont constituées des points qui se trouvent plus proches de l'un des points de E de classe **c** que de n'importe quel autre point de E. Ces régions ont été étudiées et utilisées depuis fort longtemps en géométrie algorithmique [PREPARATA 85], ce sont les régions de Voronoï.

2.2.2.1 Pavage de Voronoï

Soit E un ensemble de N formes (ou points, germes) $X_i \in \mathbb{R}^n$. Le pavage de Voronoï défini par E est composé de N régions P_i définies ainsi :

$$P_i = \{ X \in \mathbb{R}^n / d(X,X_i) < d(X,X_j) \; \forall \; j \neq i, \; 1 \leq i,j \leq N \} \qquad (2.17)$$

où d(,) désigne la distance euclidienne usuelle dans $\mathbb{R}^n$.

P_i est donc l'ensemble des points qui sont plus proches de X_i que de n'importe quel autre point de E. La région de décision d'une classe **c** est donc l'union des P_i tels que $c_i=c$, c_i désignant la classe de X_i. Or les régions P_i ont la propriété, bienvenue en l'occurrence, d'être des polyèdres de $\mathbb{R}^n$, polyèdres qui sont bien évidemment homogène vis-à-vis de E puisque chacun d'entre eux ne contient qu'un seul point de E. Il est donc possible de construire un réseau multi-couche d'automates à seuil qui réalise une classification au plus proche voisin selon la distance euclidienne.

Une région de Voronoï P_i est un polyèdre (un polygone en 2D) défini comme étant l'intersection des demi-espaces qui contiennent X_i et qui sont délimités par les hyperplans médiateurs (les médiatrices en 2D) perpendiculaires aux segments joignant X_i à chaque point X_j de E qui est adjacent à X_i dans le maillage dual de Delaunay.

Le maillage de Delaunay défini sur E est constitué de simplexes (des triangles en 2D). Un simplexe est composé de (n+1) sommets, chaque sommet étant un point de E. Si deux points appartiennent à un même simplexe, c'est-à-dire si ils sont adjacents dans le maillage de Delaunay, alors leurs polyèdres de Voronoï partagent un hyperplan commun (Fig. 2.14).

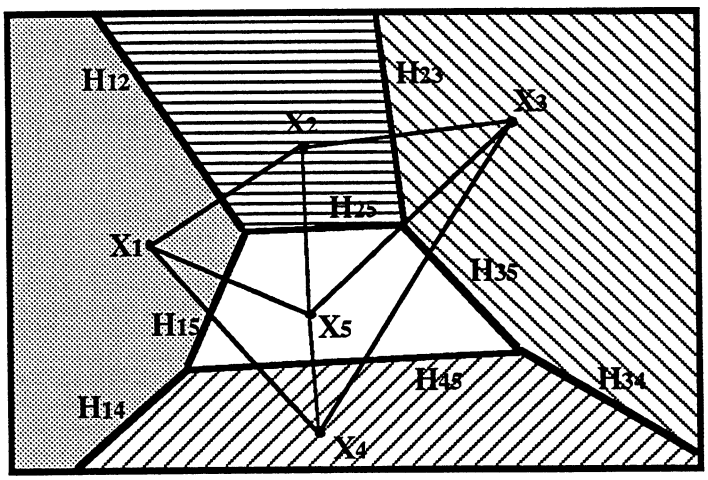


Fig. 2.14. Les régions de Voronoï dans le plan pour un ensemble E de 5 points. Les H_{ij} sont les hyperplans bissecteurs (traits gras) des couples de points (X_i, X_j) qui sont adjacents dans le maillage de Delaunay (trait fin).

Si l'on est capable de calculer les relations d'adjacence de Delaunay sur un ensemble E de points de $\mathbb{R}^n$, alors il est possible de synthétiser un réseau multi-couches réalisant une classification au plus proche voisin selon E.

2.2.2.2 Synthèse du réseau "plus proche voisin"

On considère donc un ensemble d'apprentissage E contenant N formes X_i de $\mathbb{R}^n$, X_i de classe c_i, $1 \leq c_i \leq p$. On suppose que l'on dispose des relations d'adjacence de Delaunay sur cet ensemble, relations qui peuvent être représentées par une matrice symétrique M de dimension (NxN) telle $M_{ij}=1$ si X_i et X_j sont adjacents, $M_{ij}=0$ sinon. Dans le cas de la figure 2.14, on a :

$$M = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

La première couche du réseau est composée de **n** cellules d'entrée classiques.

La première couche cachée comporte un automate H_{ij} pour chaque couple de points (X_i,X_j) tels que $M_{ij}=1$ et $1 \leq i < j \leq N$. Chaque automate H_{ij} est totalement connecté à la couche d'entrée. Le vecteur poids W_{ij} et le seuil θ_{ij} sont établis de manière à définir l'équation de l'hyperplan médiateur de X_i et X_j tel que la normale à cet hyperplan pointe dans la direction de

X_i :

$$W_{ij} = X_i - X_j \tag{2.18}$$

$$\theta_{ij} = \frac{(X_i - X_j).(X_i + X_j)}{2} \tag{2.19}$$

Ainsi, si une forme X est présentée au réseau, la sortie de l'automate H_{ij} vaudra 1 si $d(X,X_i) \leq d(X,X_j)$ et elle vaudra -1 si $d(X,X_i) > d(X,X_j)$.

La seconde couche cachée du réseau comporte un automate P_k pour chaque forme X_k de E. La sortie d'un automate H_{ij} de la première couche est alors connectée par un poids de 1 à l'automate P_i et par un poids de -1 à l'automate P_j. L'automate H_{ij} "envoie" ainsi une valeur positive à P_i ou à P_j selon le côté de l'hyperplan médiateur de X_i,X_j dans lequel "tombe" la forme d'entrée X.
On désire qu'un automate P_k ait une sortie à 1 si la forme X est dans le polyèdre de Voronoï de X_k, c'est-à-dire si elle appartient à l'intersection des demi-espaces contenant X_k. L'automate P_k doit donc calculer un ET logique de ses entrées et pour cela, nous l'avons vu, son seuil θ_k doit être fixé à n_k où n_k, le degré entrant de l'automate P_k, vaut :

$$\left| \{ H_{ij} / i=k \text{ ou } j=k \} \right|$$

Ainsi, l'automate P_k ne produira une sortie de 1 que lorsque la forme X se trouve du bon côté de tous les hyperplans qui définissent la frontière du polyèdre de Voronoï de X_k.

La couche de sortie du réseau est finalement composée de **p** automates C_r, $1 \leq r \leq p$. La sortie d'un automate C_r doit valoir 1 si et seulement si la sortie d'au moins[1] un automate P_k, tel que $c_k=r$, vaut 1. Pour cela, l'automate C_r n'est connecté, par un poids égal à 1, qu'aux automates P_k tels que $c_k=r$. L'automate doit réaliser un OU logique entre ses entrées et son seuil est donc fixé à $(1-n_r)$ où $n_r = \left| \{ P_k / k=r \} \right|$.

Le réseau ainsi construit (Fig. 2.15) réalise exactement une classification au plus proche voisin selon E : une cellule de sortie C_r ne vaut 1 que si la forme X soumise au réseau appartient à l'union des polyèdres de Voronoï associés aux points de E de classe **r.**

---

[1] En fait un seul P_k aura sa sortie à 1 puisque les polyèdres de Voronoï sont disjoints.

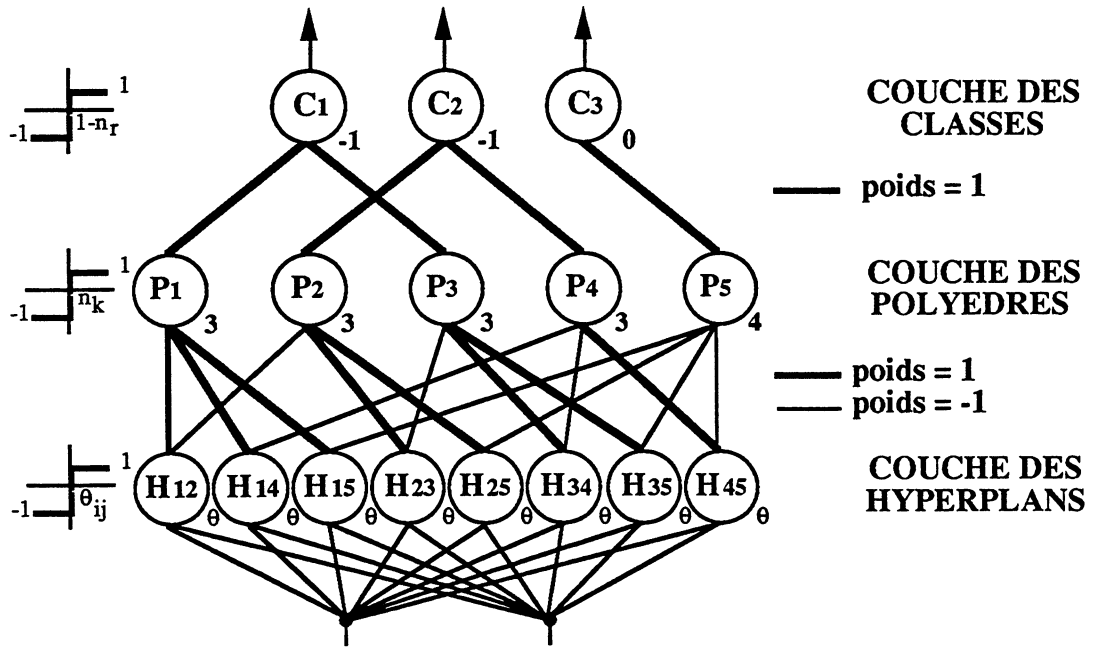


Fig. 2.15. Réseau multi-couche réalisant une classification au plus proche voisin selon l'ensemble des 5 points de la fig. 2.14. On suppose que X_1 et X_3 sont de classe 1, X_2 et X_4 de classe 2 et X_5 de classe 3.

"Traduire" la classification au plus proche voisin en termes de réseaux multi-couches est donc toujours possible dès lors que l'on sait calculer les relations d'adjacence à l'intérieur de l'ensemble d'apprentissage. Théoriquement, un maillage de Delaunay existe pour tout ensemble de points, bien qu'il ne soit pas forcément unique comme l'est le pavage de Voronoï. Nous allons rappeler comment le calculer.

2.2.2.3 Calcul des relations d'adjacence de Delaunay

Des algorithmes efficaces existent depuis longtemps pour calculer à la fois le pavage de Voronoï et le maillage de Delaunay d'un ensemble de points du plan. Dans des espaces de dimension supérieure, les algorithmes sont apparus plus tard [BOWYER 80], [WATSON 80]. C'est ce dernier algorithme que nous avons implémenté pour évaluer la validité de notre démarche. En voici brièvement le principe.

L'algorithme consiste à construire incrémentalement le maillage de Delaunay d'un ensemble de points en utilisant le critère de la sphère vide. En effet, le maillage de Delaunay a la propriété suivante : la *circonsphère* de tout simplexe du maillage de Delaunay de E (c'est-à-dire l'hypersphère à la surface de laquelle se trouvent les (n+1) sommets du simplexe) ne contient

aucun point de E. L'algorithme de Watson construit donc le maillage en insérant un point de E après l'autre de manière à ce que le maillage vérifie à chaque instant cette propriété.

La méthode suppose que le point que l'on insère dans le maillage courant est contenu dans l'enveloppe convexe des points déjà insérés. On part donc d'un maillage initial composé d'un simplexe de (n+1) points "fictifs", XF_j, choisis de manière à ce que leur enveloppe convexe contienne tous les points de E. Les points de E sont alors insérés l'un après l'autre dans ce maillage, une insertion d'un point X_i consistant en trois étapes :

Etape 1 : Parmi l'ensemble DS de tous les simplexes du maillage courant, trouver l'ensemble NEC des simplexes dont la circonsphère contient X_i (on a préalablement calculé le centre et le rayon de la circonsphère de chaque simplexe de DS).

Etape 2 : Trouver l'ensemble SOF des (n-1)-faces qui n'apparaissent qu'une seule fois parmi les simplexes de NEC (chaque simplexe est un (n+1)-uplet de points et est donc composé de **n** (n-1)-faces, chaque face étant l'un des n-uplet que l'on peut construire à partir d'un (n+1)-uplet).

Etape 3 : Oter de DS tous les simplexes appartenant à NEC et y ajouter les nouveaux simplexes construits en ajoutant X_i à chaque n-uplet de SOF. Pour chaque nouveau simplexe, calculer le centre de sa circonsphère (résolution d'un système linéaire nxn) afin de préparer la prochaine insertion d'un point.

Le principe de cette insertion est illustré par la figure 2.16.

Une fois que le diagramme de Delaunay est complètement construit, les relations d'adjacence entre les points de E sont disponibles : deux points X_i et X_j sont adjacents ($M_{ij} = M_{ji} = 1$) si ces deux points appartiennent à un même simplexe, c'est-à-dire s'ils apparaissent dans un même n-uplet du DS final. Ces relations étant disponibles, il est alors possible de construire le réseau.

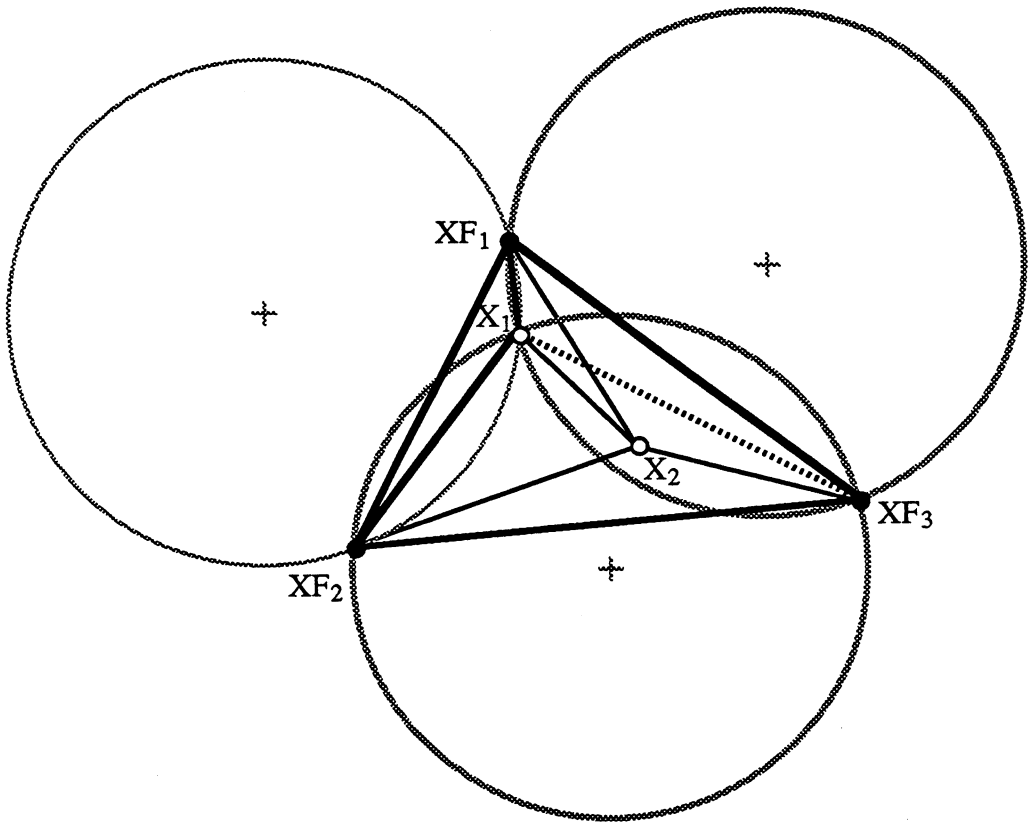


Fig. 2.16. Insertion du point X_2 dans le maillage de Delaunay contenant les germes fictifs XF_j et un premier point de E, X_1. Initialement DS={(XF_1,XF_2,X_1), (XF_1,XF_3,X_1), (XF_2,XF_3,X_1)}. Le point X_2 est contenu dans deux circonsphères (trait gras, grisé), donc NEC={(XF_1,XF_3,X_1), (XF_2,XF_3,X_1)}. Ces deux simplexes ont une seule face commune : (XF_3,X_1) (trait gras en pointillé), donc SOF = {(XF_1,X_1), (XF_2,X_1), (XF_1,XF_3), (XF_2,XF_3)}. Finalement, les nouvelles relations d'adjacence (trait fin) donnent le nouveau diagramme : DS = {(XF_1,XF_2,X_1), (XF_1,X_1,X_2), (XF_2,X_1,X_2), (XF_1,XF_3,X_2), (XF_2,XF_3,X_2)}.

Des problèmes apparaissent lors de l'insertion d'un point si l'on ne peut déterminer avec précision si, oui ou non, celui-ci appartient à une circonsphère donnée (limites de la précision numérique ou existence de (n+2) points de E cocirculaires). Si ce problème est mal géré, il peut créer des incohérences dans la structure de Delaunay et le réseau que l'on construit alors ne réalise plus la classification souhaitée. Pour notre part, nous nous sommes contentés de rejeter de tels points "critiques", en gardant à l'esprit que dans un problème de classification, l'occurrence de telles situations est improbable et que dans ce cas, la suppression d'un point de l'ensemble d'apprentissage ne porte pas à conséquence si cet ensemble est suffisamment grand.

2.2.2.5 Résultats

Pour illustrer le fait que des réseaux multi-couches d'automates à seuil peuvent ainsi être synthétisés de manière à définir des régions de décision arbitrairement complexes (non convexes, non connexes), nous avons choisi deux problèmes artificiels de classification de formes bidimensionnelles.

Les spirales imbriquées

Le premier problème choisi est celui de la classification de deux spirales imbriquées. C'est une tâche que les réseaux à rétro-propagation du gradient arrivent difficilement à résoudre si une architecture particulière n'est pas employée [LANG 88].

Les formes de l'ensemble d'apprentissage sont divisées en deux classes. Les points de classe 1 sont générés de la manière suivante : $X_i = (\rho_i.\cos(\beta_i), \rho_i.\sin(\beta_i))^t$ avec $1 \leq i \leq N/2$, $\theta_i = 12\pi i/N$, $\rho_i = 10.\theta_i$. Les points de la classe 2 sont obtenus par une symétrie par rapport à l'origine : $X_i = -X_{i-N/2}$, $1+N/2 \leq i \leq N$. Dans l'exemple présenté, N=192 (Fig. 2.17a).

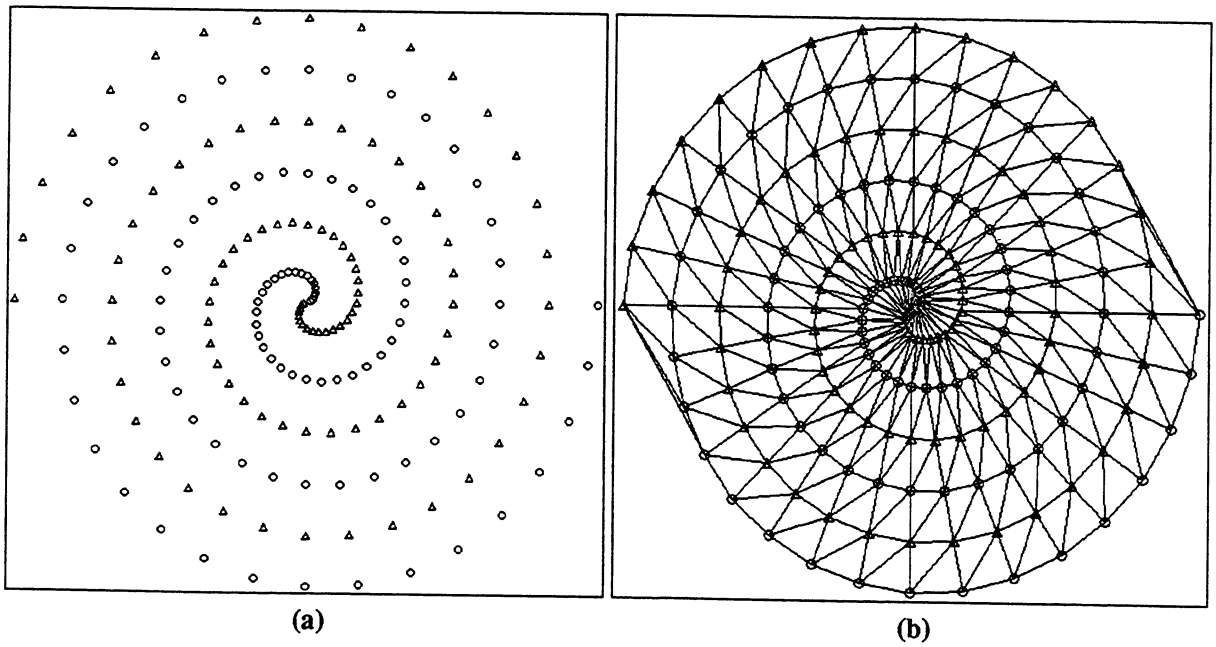


Fig. 2.17. (a) Problème des deux spirales imbriquées (les cercles représentent les points de classe 1, les triangles les points de classe 2). (b) Relations d'adjacence de Delaunay sur l'ensemble d'apprentissage.

Après calcul des relations d'adjacence de Delaunay (Fig. 2.17b), un réseau multi-couche a été généré. Ce réseau comportait ici 545 automates dans la première couche cachée, un pour chaque hyperplan du pavage de Voronoï (Fig. 2.18a) et bien sûr 192 automates dans la seconde couche cachée, un pour chaque point de E. Les régions de décision produites (Fig. 2.18b) correspondent à une généralisation très "naturelle" : ce sont deux spirales imbriquées, connexes. L'approche "plus proche voisin" est assez efficace pour ce problème alors que la généralisation produite par un réseau à rétro-propagation du gradient est souvent mauvaise dans ce cas (régions de décision non connexes).

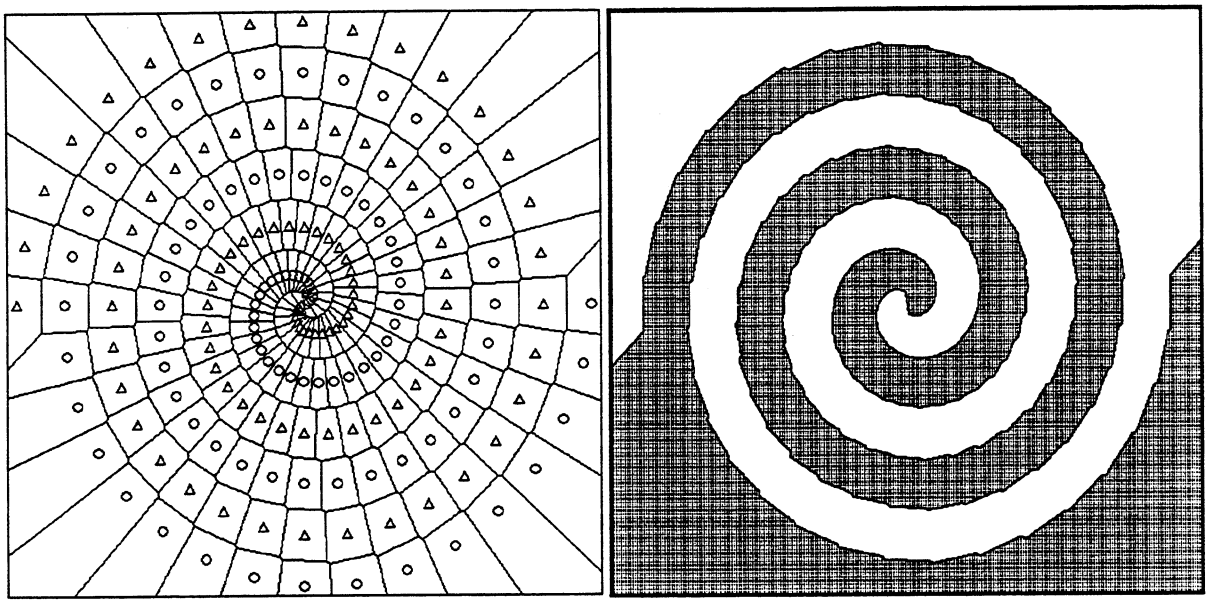
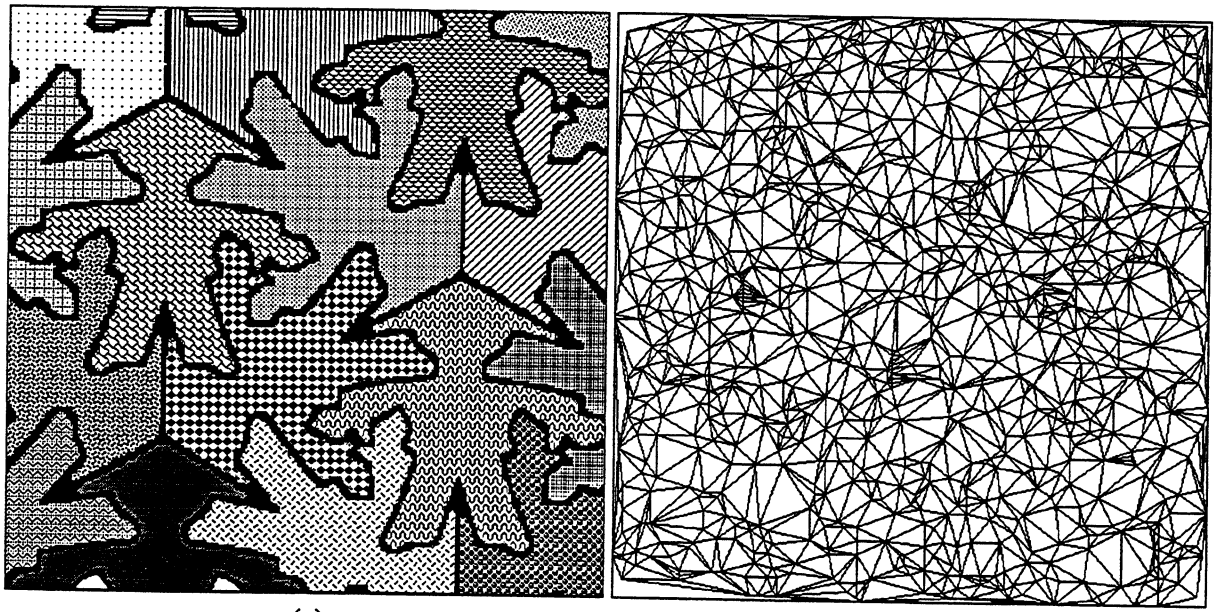


Fig. 2.18. (a) Polygones de Voronoï. (b) Régions de décision (classe 1 en gris, classe 2 en blanc).

Les chinois d'Escher

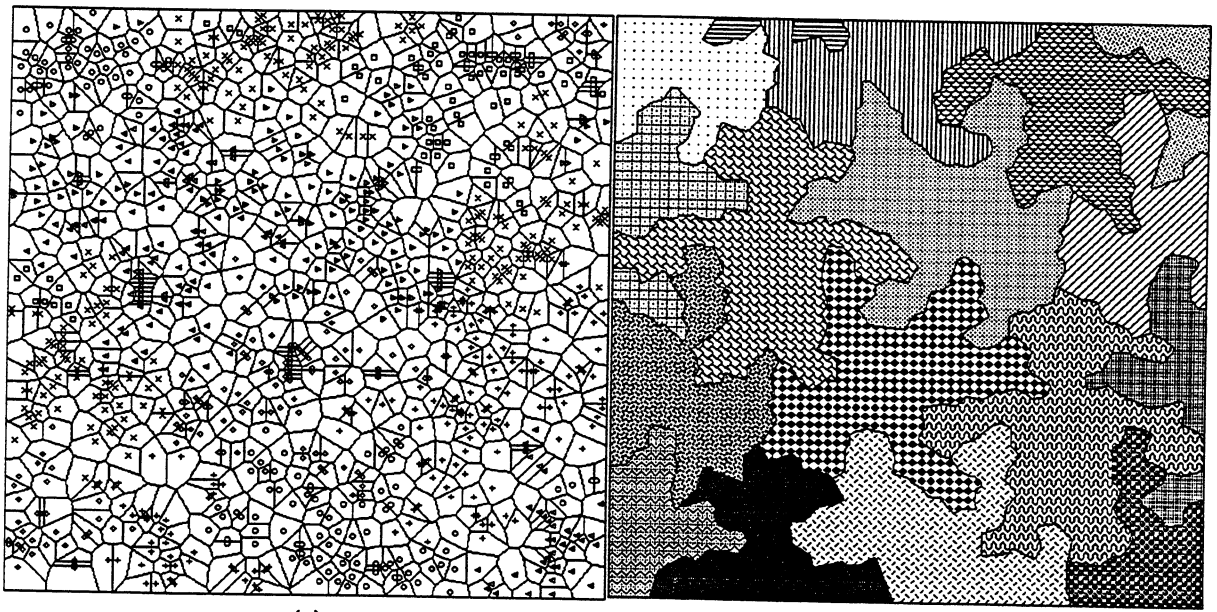
Le second problème traité est obtenu en choisissant N points du plan tirés aléatoirement selon une distribution uniforme sur un carré. Chaque point se voit attribuer une classe en fonction de la région du carré dans laquelle il tombe. Le carré est en effet partitionné en tesselles régulières, chaque tesselle représentant un personnage chinois stylisé et donc une classe particulière (21 classes en tout). Il faut noter que certaines tesselles, sur les bords de l'image, ne sont pas connexes, et que toutes les tesselles sont non convexes (Fig. 2.19a). Les régions de décision à approcher sont donc très complexes.



(a) (b)

Fig 2.19. (a) Les chinois d'Escher. (b) Maillage de Delaunay des N=1000 points de l'ensemble d'apprentissage.

En utilisant N=1000 points, après calcul des relations de Delaunay (Fig. 2.19b), on obtient un réseau comportant 2965 automates dans la première couche cachée (Fig. 2.20a) et 1000 automates dans la seconde couche cachée. Les régions de décision produites (Fig. 2.20b) approchent assez bien les régions de décision que l'on souhaite atteindre (les tesselles).



(a) (b)

Fig. 2.20. (a) Pavage de Voronoï. (b) Régions de décision.

2.2.2.6 Implémenter le rejet

Il faut noter que les régions de décision ainsi obtenues ne sont pas exactement des unions de polyèdres de Voronoï. En effet, pour avoir une vraie partition de $\mathbb{R}^n$, nous avons arbitrairement choisi d'attribuer les points d'une face commune à deux polyèdres P_i,P_j à la classe du point X_i de plus petit indice. Les régions P_i définies dans le réseau sont alors les suivantes :

$$P_i = \{ X \in \mathbb{R}^n / d(X,X_i)<d(X,X_j)\ \forall j<i \text{ et } d(X,X_i)\leq d(X,X_j)\ \forall j>i,\ 1\leq i,j\leq N \} \quad (2.20)$$

Puisque l'on a une partition, toute forme de l'espace $\mathbb{R}^n$ se verra attribuer une classe par le réseau. Or il peut être parfois préférable de rejeter une forme plutôt que de la classifier de manière erronée. En particulier, les points sur les frontières de décision, ou proches d'elles, peuvent être considérés comme ambigus : ils sont aussi proches d'une forme de E que d'une autre. Si ces deux voisins sont aussi proches l'un que l'autre et ne sont pas de même classe, quelle catégorie leur attribuer ? Dans ce cas la forme devra plutôt être rejetée.

Pour pouvoir "implémenter" cette notion de rejet dans le réseau, il faut s'autoriser à utiliser des automates dont la fonction de transition n'est plus la fonction seuil usuelle, mais une fonction tri-modale à valeurs sur {-1,0,1}. De tels automates vont permettre de définir des sortes de "no man's land" autour des frontières de décision (Fig. 2.21).

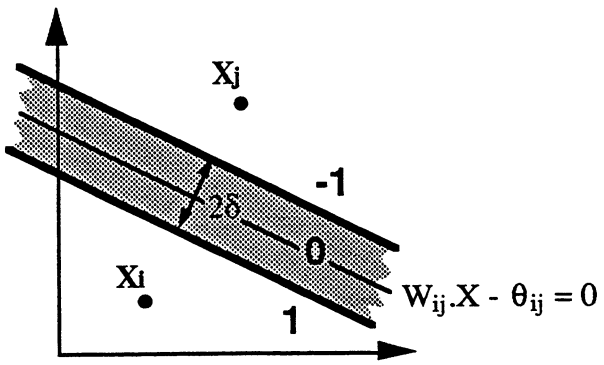


Fig. 2.21. Définition d'une zone de rejet autour de l'hyperplan médiateur de deux formes.

En effet, l'entrée nette d'un automate H_{ij}, $net_{ij}(X)$, est proportionnelle à la distance qui sépare la forme d'entrée, X, de l'hyperplan codé par les poids et le seuil de H_{ij}. Le signe de cette quantité indique le côté de l'hyperplan (positive si X est dans le demi-espace de X_i, négative sinon) et sa valeur absolue est égale à la distance perpendiculaire, si toutefois

l'équation de l'hyperplan a été normalisée, c'est-à-dire si le vecteur poids et le seuil dans (2.18) et (2.19) sont remplacés par :

$$W_{ij} = \frac{X_i - X_j}{||X_i - X_j||} \tag{2.21}$$

$$\theta_{ij} = \frac{(X_i - X_j).(X_i + X_j)}{2.||X_i - X_j||} \tag{2.22}$$

Si on définit la fonction de transition tri-modale de la manière suivante,

$$1_{bis}(net_{ij}(X)) = \begin{cases} -1 \text{ si } net_{ij}(X) < -\delta \\ 1 \text{ si } net_{ij}(X) > \delta \\ 0 \text{ sinon} \end{cases} \tag{2.23}$$

alors en utilisant cette fonction pour les automates H_{ij} qui définissent un hyperplan médiateur de deux points de classes différentes ($c_i \neq c_j$), on crée des régions de largeur 2δ autour de chaque hyperplan, régions pour les points de laquelle la sortie de l'automate vaudra 0. Ces régions seront alors considérées comme n'appartenant à aucun polyèdre de Voronoï et donc à aucune région de classe (tous les automates de la couche de sortie vaudront -1). On aura ainsi défini une région de rejet de largeur constante 2δ autour des frontières de décision (Fig 2.22a). De manière plus naturelle, on peut définir la largeur de la bande de rejet autour d'un hyperplan proportionnellement à la distance qui sépare les deux points de E qui génèrent l'hyperplan. Il suffit pour cela de remplacer (2.21) et (2.22) par :

$$W_{ij} = \frac{X_i - X_j}{||X_i - X_j||^2} \tag{2.24}$$

$$\theta_{ij} = \frac{(X_i - X_j).(X_i + X_j)}{2.||X_i - X_j||^2} \tag{2.25}$$

Le paramètre δ de la fonction trimodale représente alors pour chaque H_{ij} le pourcentage de la distance entre X_i et X_j qui constituera la largeur de la zone de rejet de part et d'autre de l'hyperplan (Fig. 2.22b, la largeur proportionnelle de la zone de rejet s'observe au centre de la spirale).

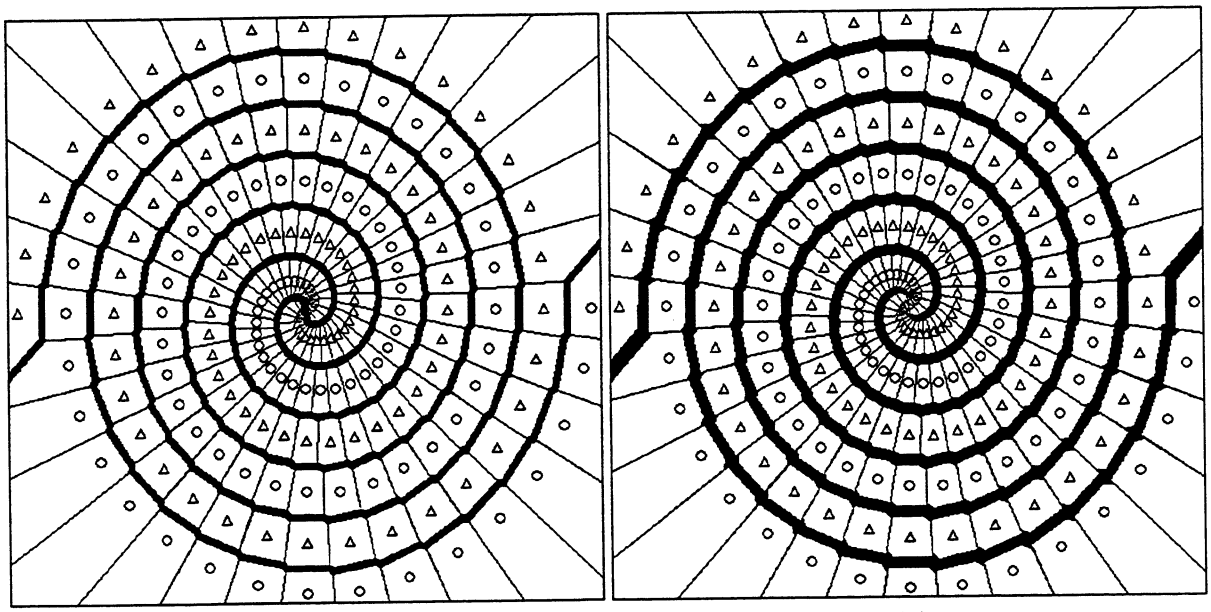


Fig. 2.22. (a) Région de rejet de largeur constante (δ=2). (b) Région de rejet de largeur proportionnelle (δ=10%).

2.2.2.7 Discussion

Nous venons de montrer comment il était possible de synthétiser un réseau multi-couche d'automates à seuil réalisant une classification au plus proche voisin. Comme nous l'avons fait précédemment pour les implémentations neuronales du type "winner-take-all", nous nous sommes interrogés sur l'intérêt d'une telle démarche. Cet intérêt est essentiellement académique. Cette étude montre qu'un réseau à deux couches cachées est capable de définir des régions de décision arbitrairement complexes puisqu'il est capable de définir celles des plus proches voisins. Elle illustre également comment il est possible d'avoir le contrôle de la généralisation fournie par un réseau en construisant les régions de décision (et de rejet) voulues.

Mais les régions de Voronoï que nous avons utilisées pour cela ne constituent certainement pas une solution efficace au problème de la synthèse d'un réseau. Ceci est naturellement dû à la complexité de l'algorithme "d'apprentissage", c'est-à-dire en l'occurrence de l'algorithme de construction du maillage de Delaunay. En effet, le calcul d'un maillage de Delaunay de N points en dimension **n**, **n**>2, a une complexité en temps de $O(N^{\lceil (n+1)/2 \rceil})$ et des besoins en espace mémoire en $O(N^{\lceil n/2 \rceil})$ [EDELSBRUNER 87]. Nos mesures expérimentales sont relativement conformes à ces valeurs (Fig. 2.23).

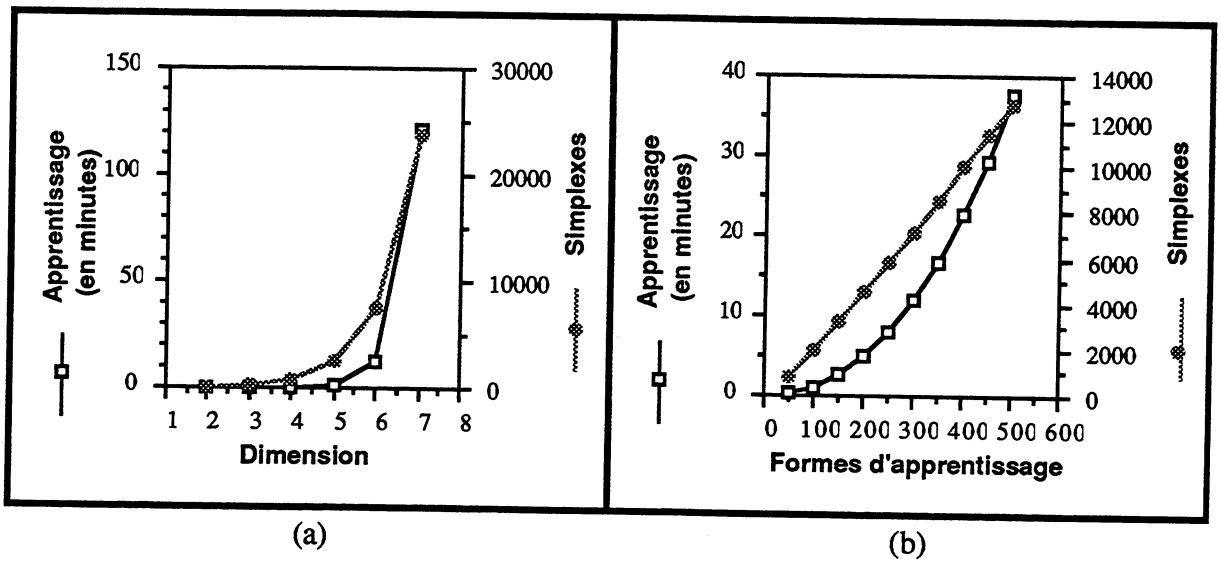


Fig. 2.23. (a) Temps de calcul du maillage de Delaunay et nombre de simplexes obtenus en fonction de la dimension (n) des formes. L'ensemble d'apprentissage est composé de N=2x25 points appartenant à deux classes distribuées selon des gaussiennes de dimension n, 2≤n≤7. (b) Les mêmes mesures pour n=4 et 50≤N≤500.

De même, le nombre d'hyperplans nécessaires à la construction d'un pavage de Voronoï, et donc le nombre d'automates dans la première couche du réseau, est en $O(N^2)$. Si l'on voulait vraiment utiliser un tel réseau pour un problème réel de classification, où la dimension de l'espace des formes est fréquemment de l'ordre de la centaine, il serait alors beaucoup plus rapide de créer directement un réseau comportant dans sa première couche les N.(N-1)/2 automates nécessaires au calcul des hyperplans médiateurs de **toutes** les paires de points de E. On éviterait ainsi le calcul des relations d'adjacence de Delaunay et finalement, le réseau créé ne serait pas tellement plus grand.

En tout état de cause, on admettra volontiers que les régions de décisions qui sont ainsi définies sont composées de beaucoup plus de polyèdres convexes qu'il n'est nécessaire pour obtenir une généralisation satisfaisante.

Ce nombre de polyèdres, c'est-à-dire N, peut bien sûr être réduit en appliquant, en amont de la construction du réseau, un processus de quantification vectorielle (les cartes de Kohonen et algorithmes LVQ vus au § 1.4.2 ou l'algorithme "d'édition" de l'ensemble d'apprentissage de [WILSON 72], par exemple).

On pourrait également penser restreindre ce nombre de régions en construisant des pavages de Voronoï d'ordre k, k>1. Les polyèdres qui composent un tel pavage correspondent aux régions des "k-plus proches voisins". Mais la construction de ces pavages est également

très complexe.

Pour conclure, nous répèterons ici encore que les réseaux de neurones, en couches ou dynamiques, ne constituent certainement pas une méthode efficace pour implémenter une classification exacte au plus proche voisin. Contrairement à [MURPHY 90], nous n'affirmeront pas que la méthode présentée *constitue une alternative, fonctionnant en temps polynomial* (mais quel polynôme !), *à l'apprentissage par rétro-propagation du gradient.* L'intérêt de cette étude est, nous l'avons dit, plutôt "anecdotique".

Dans leur livre *Perceptrons* [MINSKY 69], Minsky et Papert faisaient deux constatations intéressantes : ils remarquaient tout d'abord (p. 192) le lien existant entre le schéma "plus proche voisin" et l'opération élémentaire que permet de réaliser un automate à seuil (le seuillage d'un produit scalaire). A quelques pages d'intervalle, ils constataient (p. 229) l'absence de méthode générale d'apprentissage concernant les réseaux multi-couches d'automates à seuil. La méthode de construction de réseau que nous venons de montrer a pour seul intérêt de réaliser la connexion (!) entre ces deux remarques. Elle ne constitue pas un résultat intéressant, tout comme comme ne l'est pas, d'après Minsky, le résultat établissant qu'un réseau multi-couche d'automates à seuil peut implémenter n'importe quelle fonction booléenne (puisque toute fonction booléenne peut se mettre sous forme normale disjonctive). Dans les deux cas, les réseaux, considérés en termes de mémoire associative, sont moins efficaces que la méthode consistant à stocker toutes les associations de l'ensemble d'apprentissage.

On remarquera tout de même que si dans un avenir plus ou moins lointain, des machines parallèles "neuronales" comportant des milliers de cellules sont disponibles, alors il sera possible de construire un classifieur au plus proche voisin qui classifiera une forme en temps constant (3 étapes, car 3 couches d'automates à seuil) quelle que soit la taille de l'ensemble d'apprentissage...

En attendant cette heure, on peut toujours tenter de synthétiser des réseaux qui comportent un nombre plus raisonnable d'automates tout en produisant une généralisation satisfaisante. Il faut pour cela chercher à construire des pavages de l'espace des formes qui remplissent les conditions voulues tout en comportant moins de polyèdres que le pavage de Voronoï. Ce sera le sujet du paragraphe suivant.

2.2.3 Apprentissage hiérarchique

Dans l'optique de restreindre le nombre de polyèdres convexes nécessaires à la définition des régions de décision, une approche naturelle consiste à tenter de construire la partition de manière "incrémentale", c'est-à-dire en ne rajoutant qu'un hyperplan à la fois jusqu'à ce que tous les polyèdres ainsi engendrés deviennent homogènes vis-à-vis de l'ensemble d'apprentissage (cf § 2.2.1.3).

Pour réaliser cela, une stratégie hiérarchique s'impose elle aussi tout naturellement dès lors que l'on considère le rôle joué par un hyperplan (c'est-à-dire un automate de la première couche cachée) dans le processus de classification. En effet, un premier hyperplan "jeté" dans l'espace des formes définit une partition en deux polyèdres simples : des demi-espaces. Si ces deux polyèdres sont déjà homogènes, comme ce peut être le cas si l'on traite un problème à deux classes linéairement séparables, la partition recherchée est trouvée. Si tel n'est pas le cas, alors on peut considérer les points appartenant à l'un des polyèdres non homogènes comme un sous-ensemble d'apprentissage et le sous-problème va consister à trouver un hyperplan permettant de décomposer le polyèdre considéré en deux polyèdres homogènes... et ainsi de suite.

La dichotomie réalisée par un hyperplan amène donc naturellement à tenter de construire la partition en polyèdres homogènes à l'aide d'une hiérarchie (un arbre binaire) d'hyperplans. La recherche d'une réponse à la formulation géométrique du problème de l'apprentissage dans un réseau d'automate nous a ainsi ramenés à des techniques de classification bien connues : les méthodes de segmentation [CELEUX 91] ou arbres de classification [BREIMAN 84].

2.2.3.1 Equivalence entre méthodes de segmentation et réseaux multi-couches d'automates à seuil

"Par segmentation on désigne les méthodes de discrimination qui construisent des arbres de décision binaires" [CELEUX 91]. Le principe de ces méthodes est identique à celui de l'arbre de décision que nous avons présenté au § 2.1.2.5. Dans cet arbre, chaque nœud divisait en deux l'espace des formes binaires en posant une question relative à la valeur (+1 ou -1) de l'une des composantes. Une forme inconnue cheminait ainsi dans l'arbre en fonction des réponses obtenues et la feuille dans laquelle elle "atterrissait" permettait de lui assigner une classe.

Ceci se généralise facilement au cas des formes non binaires (vecteurs réels ou à composantes qualitatives). Il suffit pour cela que les questions que l'on pose à chaque nœud de l'arbre n'admettent que deux réponses possibles. Pour des formes réelles, les questions posées consistent généralement [GELFAND 91] à comparer l'une des composantes de la forme à un seuil ; si la composante est supérieure ou égale au seuil, la forme est "dirigée" vers le fils droit du nœud considéré, sinon elle est dirigée vers le fils gauche (on peut d'ailleurs considérer que l'interrogation d'une composante d'une forme binaire n'est qu'un cas particulier de ce seuillage).

Un tel seuillage réalise une partition à l'aide d'un hyperplan orthogonal à l'une des dimensions de l'espace des formes. De manière plus générale, la partition pourra être réalisée par un hyperplan quelconque si le seuillage porte non plus sur une composante particulière, mais sur une combinaison linéaire de toutes les composantes de la forme. Un arbre de classification consiste donc bien en une hiérarchie d'hyperplans telle que nous voulons la construire. Cette hiérarchie peut-elle être traduite par un réseau multi-couche d'automates à seuil ?

Régions de décision d'un arbre de classification

Considérons un chemin de l'arbre $v_1, v_2, ..., v_m$ menant de la racine v_1 à une feuille v_m portant la classe **c**. Soit $H_1, H_2, ..., H_{m-1}$ les hyperplans correspondant aux m-1 premiers nœuds du chemin. Une forme inconnue X cheminera le long de ce chemin jusqu'à la feuille v_m si à chaque nœud **i**, $1 \leq i \leq m-1$, X appartient au demi-espace positif (resp. négatif) défini par H_i si v_{i+1} est le fils droit (resp. gauche) de v_i. La région de décision P_m associée à la feuille v_m est donc une intersection de demi-espaces, c'est-à-dire un polyèdre convexe :

$$P_m = \bigcap_{i=1}^{m-1} H_i^{sgn(i)} \qquad (2.26)$$

où sgn(i) = "+" si v_{i+1} est fils droit de v_i, "-" sinon.

La région de décision d'une classe **c** est naturellement composée de l'union des régions de décision associées à chaque feuille de l'arbre portant la classe **c** : c'est une union de polyèdres convexes (Fig. 2.24). La classification réalisée par un arbre est donc tout à fait calculable par un réseau d'automates à seuil à deux couches cachées.

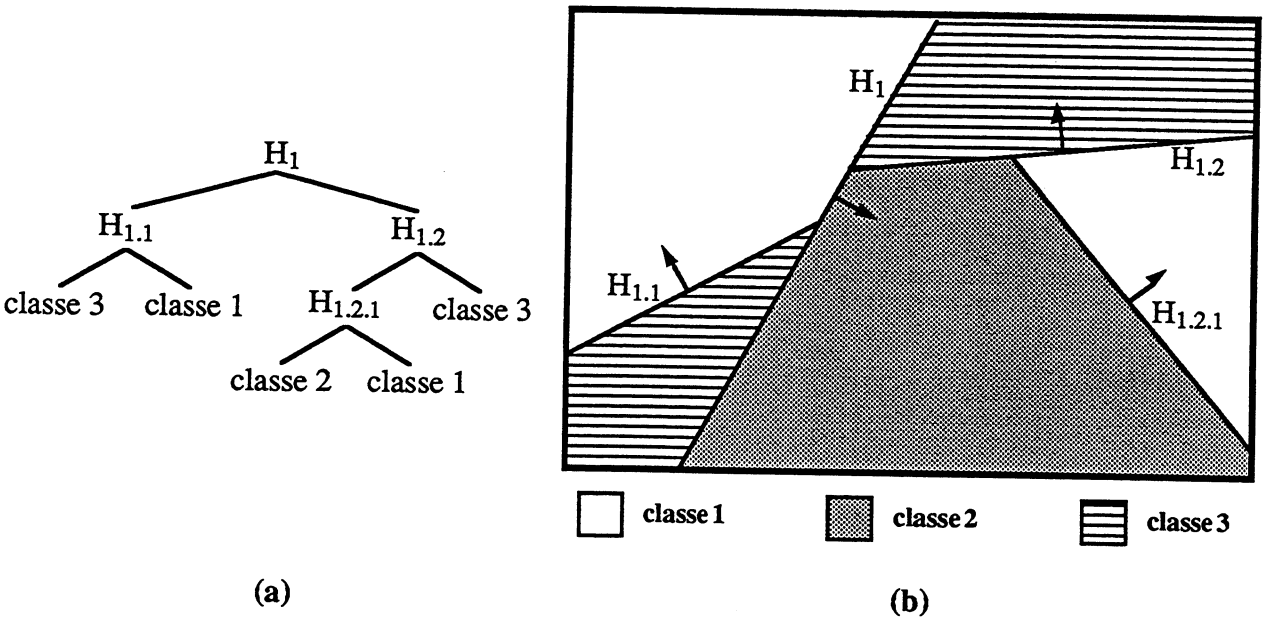


Fig. 2.24. (a) Arbre de classification dans $\mathbb{R}^2$. (b) Régions de décision correspondantes.

Synthèse du réseau

La construction d'un réseau implémentant un arbre de classification est directe.
La première couche du réseau est classiquement composée de **n** cellules d'entrée si l'on traite des formes de $\mathbb{R}^n$.
La première couche cachée comporte un automate H_i pour chaque nœud v_i non feuille de l'arbre. Le vecteur poids et le seuil d'un automate H_i sont évidemment établis de manière à ce que l'automate réponde "1" lorsque la forme d'entrée appartient au demi-espace positif associé à v_i, et "-1" sinon.
La seconde couche cachée comporte un automate P_j pour chaque feuille v_j de l'arbre (cet arbre étant binaire, le nombre d'automates dans la seconde couche cachée est donc égal au nombre d'automates dans la première couche cachée plus un). Un automate P_j n'est connecté qu'aux automates H_i tels que v_i est un ancêtre de v_j. La pondération reliant v_i à v_j vaut "1" si v_i est un fils droit, "-1" sinon. Le seuil de P_j est fixé de manière à ce que l'automate calcule un ET logique de ses entrées (intersection de demi-espaces).
La couche de sortie du réseau comporte pour sa part un automate C_k par classe, $1 \leq k \leq p$. Un automate C_k n'est connecté qu'aux automates P_j correspondant à des feuilles v_j porteuses de la classe k. La pondération de toutes les connexions est égale à "1" et le seuil de C_k est tel que l'automate calcule un OU logique de ses entrées (union de polyèdres).

Si une forme X se voyait assigner la classe c par l'arbre de classification, alors le réseau ainsi construit (Fig. 2.25) produira bien, en réponse à cette même forme, une sortie telle que $C_k=1$ si k=c, $C_k=-1$ sinon.

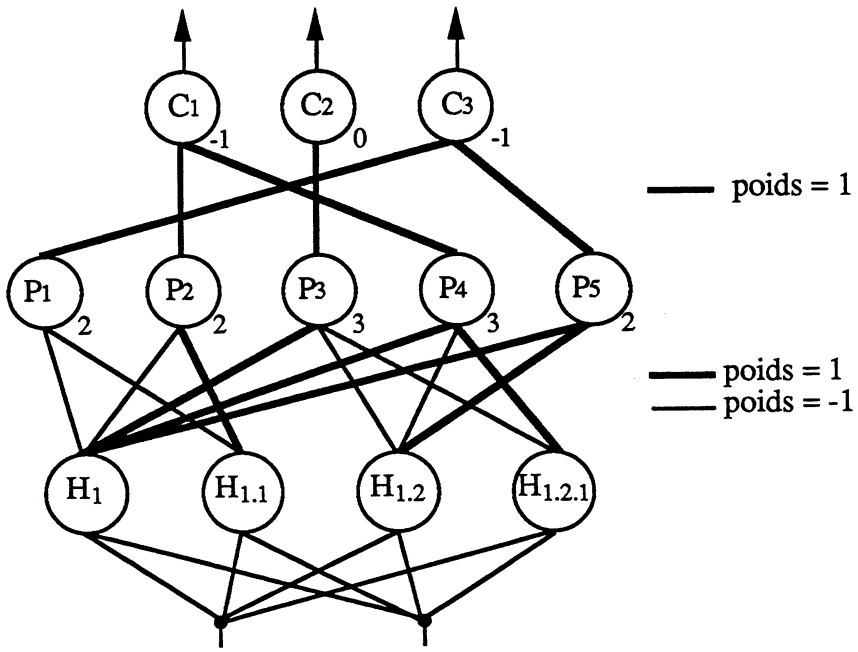


Fig. 2.25. Réseau d'automates à seuil implémentant l'arbre de classification de la figure 2.24.

L'apprentissage dans un réseau multi-couche d'automates à seuil peut donc être traduit en un problème de construction d'arbre de classification. Le paragraphe suivant propose une méthode permettant de résoudre ce problème.

2.2.3.2 Construction d'un arbre de classification

L'algorithme récursif de construction de l'arbre proprement dit est identique à celui de l'arbre de décision binaire que nous avons vu au § 2.1.2.5. En partant de l'échantillon d'apprentissage initial, on recherche un hyperplan qui segmente cet échantillon en deux et le processus est réitéré récursivement sur les deux sous-ensembles ainsi obtenus jusqu'à ce que chaque sous-échantillon obtenu ne contienne que des représentants d'une même classe.

Si l'on considère notre ensemble d'apprentissage habituel E, composé de N formes X_i de $\mathbb{R}^n$, X_i de classe c_i, $1 \leq i \leq p$, la construction d'un arbre de classification sur E se fait de la manière suivante :

```
Etape 0 :   /* Initialisation */
            ν = 1 ; /* racine */
            Eν = E ; Cν = {classes de Eν} = {1,2,...,p} ;

Etape 1 :   /* Construction récursive */
            Procédure TRAITER_NŒUD (ν, Eν, Cν)
              Si |Cν| = 1 alors début
                 ν.type = feuille ; ν.classe = Cν
              Fin si
              Sinon début
                 ν.type = non terminal ;
                 Rechercher un hyperplan Hν qui segmente "au mieux" Eν ;
                 ν.hyperplan = Hν ;
                 Eν.1 = {Xi ∈ Eν / Xi ∈ Hν- } ; Cν.1 = {classes de Eν.1}
                 Eν.2 = {Xi ∈ Eν / Xi ∈ Hν+ } ; Cν.2 = {classes de Eν.2}
                 TRAITER_NŒUD (ν.1, Eν.1, Cν.1) ;
                 TRAITER_NŒUD (ν.2, Eν.2, Cν.2) ;
              Fin sinon
            Fin procédure
```

L'arbre ainsi construit va "pousser" jusqu'à ce que chaque feuille ne contienne plus que des formes de même classe, c'est-à-dire jusqu'à ce que le polyèdre correspondant soit homogène. D'autres critères d'arrêt peuvent être utilisés, ou des techniques d'élagage de l'arbre peuvent être mises en œuvre [BREIMAN 84] [GELFAND 91]. Nous nous sommes pour notre part intéressés à l'autre point critique d'un tel algorithme : le choix de l'hyperplan réalisant la segmentation.

2.2.3.3 Recherche d'un hyperplan pour la segmentation

La plupart des méthodes pour la recherche d'un tel hyperplan sont explicitement fondées sur la minimisation d'un critère d'impureté [CELEUX 91] à caractère statistique : critère de Gini ou fonction d'impureté liée à l'entropie. Le critère que nous avons développé est plutôt de nature géométrique et il nous apparaît intuitivement plus proche des algorithmes d'apprentissage connexionnistes que les méthodes évoquées ci-dessus.

Ce critère est en quelque sorte une généralisation aux formes réelles de la méthode non statistique dédiée aux formes binaires vue au § 2.1.2. Il repose sur les notions de stabilité et d'équilibre de la segmentation réalisée par un hyperplan.
Nous dirons qu'un hyperplan réalise une séparation stable pour une classe si tous les points de

l'ensemble d'apprentissage de cette classe sont d'un même côté de l'hyperplan.
Cette séparation sera équilibrée si le même nombre de classes pour lesquelles la segmentation est stable se trouvent de part et d'autre de l'hyperplan.

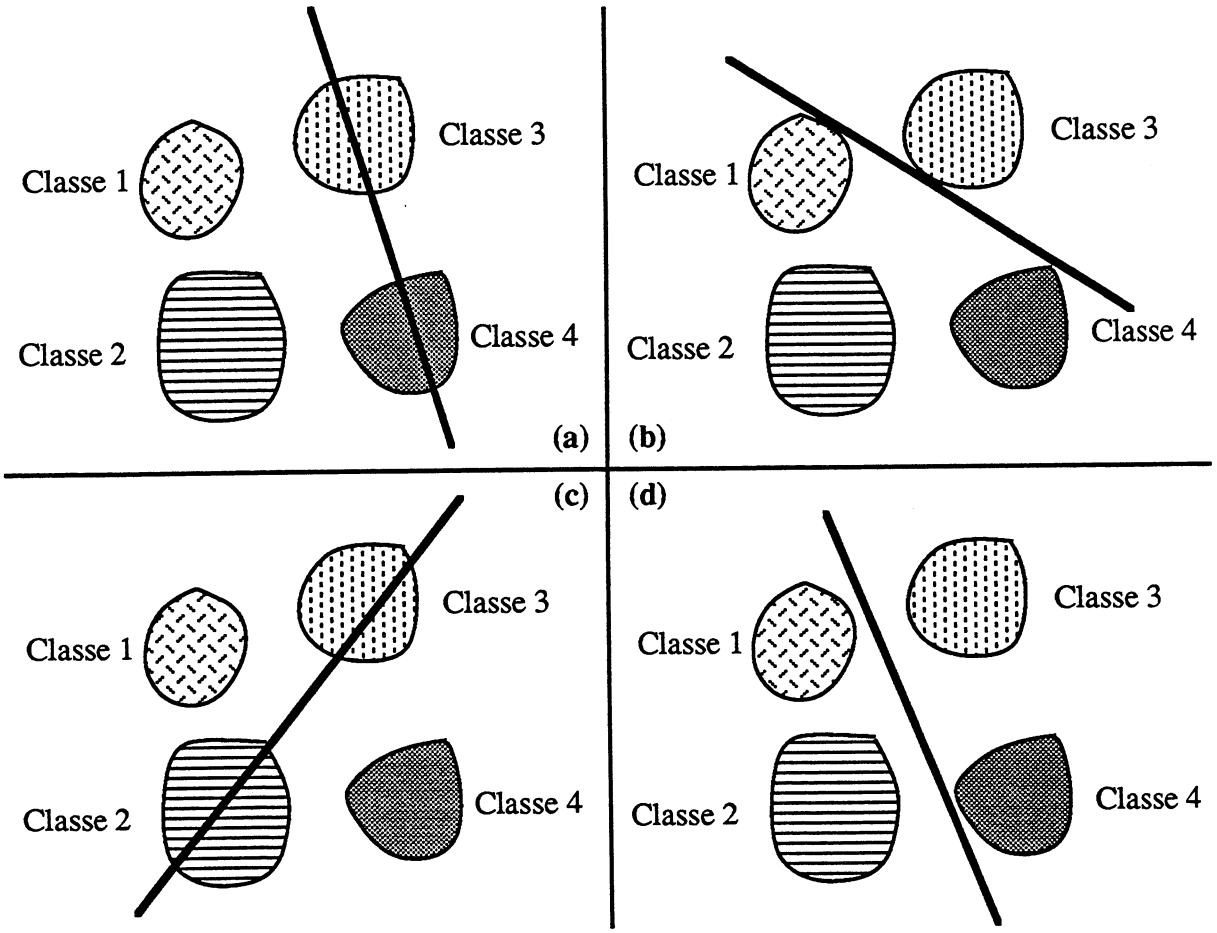


Fig. 2.26. (a) La séparation n'est pas stable pour les classes 3 et 4 et n'est pas équilibrée. (b) La séparation est stable pour toutes les classes mais n'est pas équilibrée (trois classes d'un côté, une de l'autre). (c) La séparation est équilibrée mais n'est pas stable pour les classes 2 et 3. (d) La séparation est stable et équilibrée.

Ces notions sont illustrées par la figure 2.26. On recherchera naturellement un hyperplan qui combine au mieux stabilité et équilibre.

Soit E l'ensemble d'apprentissage que l'on considère à un nœud donné de l'arbre ν et soit H l'hyperplan que l'on recherche (pour plus de clarté, on omettra ici l'indice du nœud sur E et H). Soit nbc le nombre de classes représentées dans E ($nbc \geq 2$) ; on considère la partition de E en nbc ensembles de formes de même classe :

$$E = \bigcup_{k=1}^{nbc} E_k \tag{2.27}$$

où $E_k = \{ X_i \in E / c_i = k \}$. On note nbf le cardinal de E, nbf_k le cardinal de E_k.
On décrira H par un vecteur de poids $W=(w_0, w_1,.., w_n)$, $W \in \mathbb{R}^{n+1}$ (w_0 représente l'opposé du seuil), et l'on considère les vecteurs de formes "augmentés" : $X_i=(1, x_{i1},.., x_{in})$. Une forme X_i appartient à H^+ si $\mathbb{1}(W.X)=1$, à H^- si $\mathbb{1}(W.X)=-1$.

Nous définissons le degré d'instabilité de la séparation par H pour la classe **c**, IS_c, de la manière suivante :

$$IS_c = 1 - \frac{1}{nbf_c} \left| \sum_{X_i \in E_c} \mathbb{1}(W.X_i) \right| \tag{2.28}$$

Ce degré atteint son minimum (égal à 0) lorsque toutes les formes de E_c sont du même côté de l'hyperplan ($\mathbb{1}(W.X_i)=\mathbb{1}(W.X_j)\ \forall\ X_i,X_j \in E_c$) et il est maximum (égal à 1) lorsque le même nombre de formes de E_c se trouvent de part et d'autre de l'hyperplan (lorsque $|\{X_i \in E_c / X_i \in H^+\}| = |\{X_j \in E_c / X_j \in H^-\}|$).
L'instabilité de la séparation réalisée par H, IS, est naturellement définie comme étant la moyenne des instabilités pour chaque classe :

$$IS = \frac{1}{nbc} \sum_{k=1}^{nbc} IS_k \tag{2.29}$$

Pour évaluer l'équilibre de la séparation, nous définissons tout d'abord ce que l'on pourrait appeler le "signe de la majorité" d'une classe **c**, SM_c :

$$SM_c = \frac{1}{nbf_c} \sum_{X_i \in E_c} \mathbb{1}(W.X_i) \tag{2.30}$$

Cette valeur varie de -1 (lorsque toutes les formes de E_c appartiennent à H^-) à 1 (lorsque toutes les formes appartiennent à H^+) en passant par 0 lorsque la séparation n'est pas stable pour **c**.
Le degré de déséquilibre de la séparation réalisée par H, DS, est alors défini de la manière suivante :

$$DS = \frac{1}{nbc} \left| \sum_{k=1}^{nbc} SM_k \right| \tag{2.31}$$

On vérifie aisément que ce degré est minimal (égal à 0) lorsque la séparation est équilibrée (ou

lorsque la séparation est instable pour toutes les classes) et maximal (égal à 1) lorsque toutes les classes sont dans le même demi-espace.

Nous venons donc de définir deux mesures permettant de quantifier l'instabilité et le déséquilibre de la segmentation réalisée par un hyperplan. Il nous faut maintenant rechercher l'hyperplan qui minimise ces deux mesures : nous avons une fois encore ramené l'apprentissage à un problème d'optimisation et, comme souvent en connexionnisme, nous allons le résoudre par une descente en gradient.

Pour cela, il nous faut tout d'abord remplacer les fonctions *seuil* et *valeur absolue* par des fonctions ayant des comportements similaires tout en étant partout dérivables. De manière habituelle, la fonction seuil sera remplacée par "la" fonction sigmoïde et la valeur absolue par une élévation au carré. Ceci nous amène à réécrire les mesures IS et DS de la manière suivante :

$$IS = \frac{1}{nbc} \sum_{k=1}^{nbc} \left[1 - \frac{1}{nbf_k^2} \left(\sum_{X_i \in E_k} f(W.X_i) \right)^2 \right] \tag{2.32}$$

$$DS = \frac{1}{nbc^2} \left(\sum_{k=1}^{nbc} \left[\frac{1}{nbf_k} \sum_{X_i \in E_k} f(W.X_i) \right] \right)^2 \tag{2.33}$$

où la fonction sigmoïde **f** est définie ainsi :

$$f(W.X) = \frac{2}{1 + e^{tr.(W.X)}} - 1 \tag{2.34}$$

La recherche de l'hyperplan consiste alors à minimiser une fonction coût C qui est une somme pondérée des deux mesures d'instabilité et de déséquilibre :

$$C = \alpha.IS + \beta.DS \tag{2.35}$$

Cette mimisation est réalisée par une itération sur les poids :

$$W(t+1) = W(t) - \varepsilon.\nabla_W C = W(t) - \varepsilon.(\alpha.\nabla_W IS + \beta.\nabla_W DS) \tag{2.36}$$

où ε dénote l'habituel taux d'apprentissage.
Pour information, les dérivées partielles de IS, DS et **f** par rapport à la $j^{ème}$ composante de W s'écrivent :

$$\frac{\partial IS}{\partial w_j} = \frac{-2}{nbc} \sum_{k=1}^{nbc} \left[\frac{1}{nbf_k^2} \left(\sum_{X_i \in E_k} \frac{\partial f}{\partial w_j}(W.X_i) \right) \left(\sum_{X_i \in E_k} f(W.X_i) \right) \right] \tag{2.37}$$

$$\frac{\partial DS}{\partial w_j} = \frac{2}{nbc^2} \left(\sum_{k=1}^{nbc} \left[\frac{1}{nbf_k} \sum_{X_i \in E_k} \frac{\partial f}{\partial w_j}(W.X_i) \right] \right) \left(\sum_{k=1}^{nbc} \left[\frac{1}{nbf_k} \sum_{X_i \in E_k} f(W.X_i) \right] \right) \tag{2.38}$$

$$\frac{\partial f}{\partial w_j}(W.X) = \frac{t_f.x_j.(1 - f(W.X)^2)}{2} \tag{2.39}$$

Choix des paramètres

L'hyperplan initial peut être choisi de manière aléatoire mais l'on peut gagner beaucoup de temps pendant la descente en gradient en proposant une initialisation "raisonnable". Ainsi, dans la plupart de nos expériences, W(0) était l'hyperplan médiateur des centres de gravité de deux classes choisies au hasard parmi les nbc classes possibles.

La condition d'arrêt de la descente en gradient portait sur la valeur de l'erreur C ou sur la différence entre deux valeurs consécutives de C.

Les pondérations α et β des deux mesures IS et DS semblent, par expérience, avoir une certaine influence sur la rapidité de la convergence, mais il est difficile d'élaborer une méthode pour les fixer automatiquement, aussi les avons-nous toujours prises toutes deux égales à 1.

Le choix de la pente de la sigmoïde, t_f, revêt une grande importance. Si elle est trop faible, la sigmoïde ne constitue plus une approximation satisfaisante de la fonction seuil et les mesures IS et DS perdent de leur sens. Si au contraire elle est trop forte, la dérivée de la sigmoïde n'est plus "assez continue" et des phénomènes d'oscillation apparaissent.
Pour obtenir une descente en gradient qui ne soit pas trop chaotique, la méthode consiste à fixer t_f de telle sorte que seuls les points les plus proches de l'hyperplan aient vraiment une influence sur la modification de W (on peut voir d'après (2.37) et (2.38) que ces points sont les points X_i pour lesquels $f'(W.X_i)$ n'est pas proche de zéro). Pour cela, il faut tout d'abord s'assurer que la quantité $W.X_i$ représente bien la distance perpendiculaire de X_i à l'hyperplan. Dans ce but, après chaque modification de l'hyperplan, l'équation de celui-ci est normalisée par :

$$W(t) = W(t).\frac{1}{\sqrt{\sum_{i=1}^{n} w_i(t)^2}} \qquad (2.40)$$

Ensuite, t_f est calculé une fois pour toutes de manière à ce que l'intervalle "utile" (cf § 2.1.1.1) de la sigmoïde corresponde à une bande autour de l'hyperplan, bande dont la largeur devra être déterminée en fonction de l'ensemble d'apprentissage (Fig. 2.27). Nous avons pris celle-ci égale à deux fois la plus petite distance séparant deux points de l'ensemble d'apprentissage total, distance que l'on calcule une seule fois avant de construire de l'arbre.

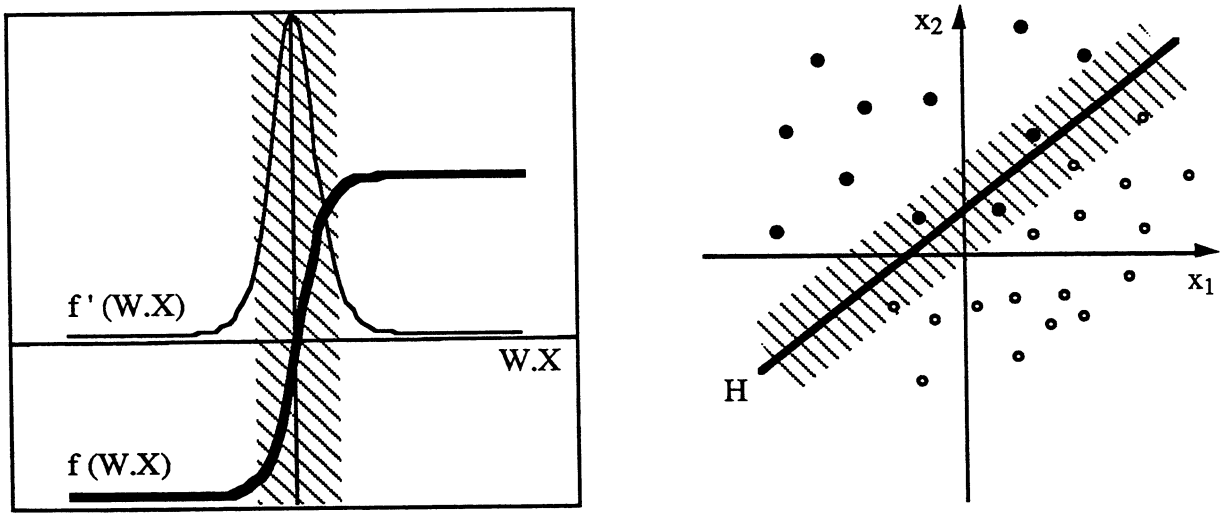


Fig. 2.27. La pente de la sigmoïde est calculée en fonction d'une "zone d'influence" autour de l'hyperplan, zone dont on fixe la largeur en fonction de l'ensemble d'apprentissage.

2.2.3.4 Cas particuliers

La recherche de l'hyperplan réalisant la meilleure segmentation peut être améliorée dans deux cas : si l'ensemble d'apprentissage que l'on traite à un nœud donné ne comporte que deux classes (nbc=2), ou s'il comporte un nombre impair de classes (nbc=2p+1).

Cas nbc=2

Dans ce cas, il est préférable, plutôt que d'essayer de minimiser notre complexe fonction C, d'utiliser l'un des algorithmes employés pour les réseaux à deux couches : Perceptron (algorithme de la poche) ou Adaline (cf. § 1.2). Nous avons obtenu les meilleurs résultats en utilisant une version modifiée de la règle de Widrow-Hoff. Cette modification consiste à utiliser

une fonction coût où l'erreur quadratique commise sur un élément X_i est pondérée de manière inversement proportionnelle à l'importance de la classe de X_i dans E. On s'assure ainsi que l'hyperplan trouvé réalise bien une séparation même si l'une des deux classes est beaucoup moins représentée que l'autre.
Donc lorsque nbc=2, le coût à minimiser C (expression 2.35) est remplacé par le coût C_2 défini ainsi :

$$C_2 = \sum_{k=1}^{2} \left[\frac{1}{nbf_k} \sum_{X_i \in E_k} ((2k-3) - f(W.X_i))^2 \right] \tag{2.41}$$

où (2k-3) représente, par rapport à la règle de Widrow-Hoff, la "sortie désirée" pour la classe **k**. Le gradient de C_2 par rapport à la $j^{ème}$ composante de W s'écrit alors :

$$\frac{\partial C_2}{\partial w_j} = -2 \sum_{k=1}^{2} \left[\frac{1}{nbf_k} \sum_{X_i \in E_k} \left[\frac{\partial f}{\partial w_j}(W.X_i).((2k-3) - f(W.X_i)) \right] \right] \tag{2.42}$$

Cas nbc=2p+1

Lorsque le nombre de classes représentées dans E est impair, il est évidemment impossible de réaliser une séparation parfaitement équilibrée au sens défini précédemment. Dans ce cas, la séparation idéale est celle qui place **p** classes d'un côté de l'hyperplan et **p+1** classes de l'autre (Fig. 2.28b).

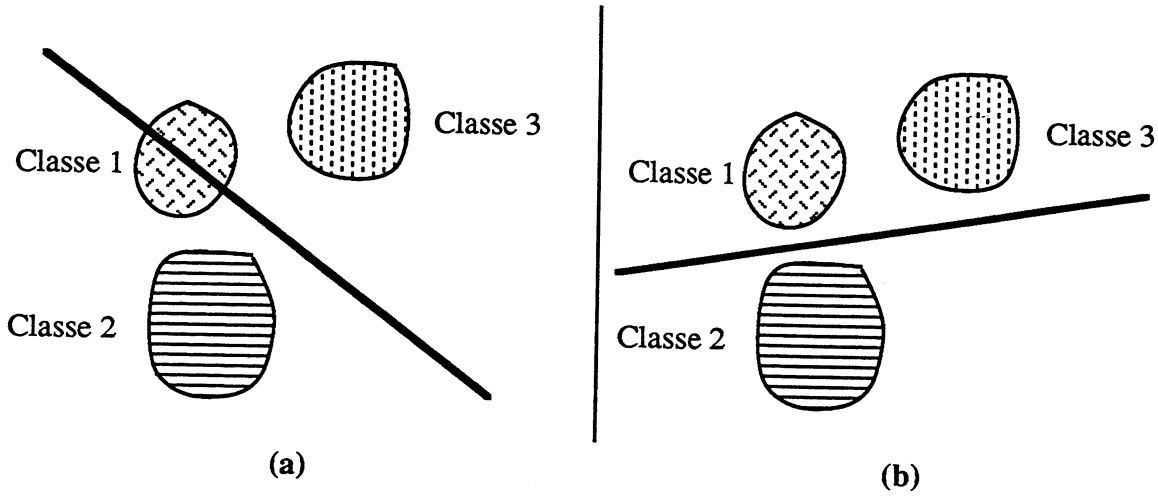


Fig. 2.28. (a) Une segmentation vers laquelle on tend si l'on minimise le coût C "standard". (b) Une segmentation que l'on aimerait obtenir.

Or une telle situation ne correspond pas à un minimum de la mesure DS (expression 2.31) qui prend alors la valeur 1/nbc. Pour que DS prenne la valeur 0, il faudrait

que la séparation soit instable pour l'une des classes ($\exists$ k / IS_k=1) et stable pour toutes les autres, ce qui n'est pas souhaitable (Fig. 2.28a).

Dans le cas idéal , la somme des "signes de la majorité" n'est pas nulle comme on aimerait qu'elle le soit, ce qui correspondrait à un minimum de DS, mais égale à 1 si (p+1) classes sont dans H^+ ou égale à -1 si (p+1) classes sont dans H^-. Pour annuler cette somme il faut donc lui retrancher 1 ou -1 selon son signe et la laisser inchangée quand elle est nulle, c'est-à-dire redéfinir la mesure DS (2.31) de la manière suivante :

$$DS_2 = \frac{1}{nbc}\left|\sum_{k=1}^{nbc} SM_k - \mathbb{1}_{bis}\left(\sum_{k=1}^{nbc} SM_k\right)\right| \tag{2.43}$$

où $\mathbb{1}_{bis}(x)$ est la fonction trimodale :

$$\mathbb{1}_{bis}(x) = \begin{cases} -1 \text{ si } x < 0 \\ 1 \text{ si } x > 0 \\ 0 \text{ sinon} \end{cases} \tag{2.44}$$

Pour les besoins de la descente en gradient, ce nouvel indice est approché en retranchant à la somme des "signes de la majorité" sa valeur par un substitut de la fonction signe $\mathbb{1}_{bis}$, c'est-à-dire une sigmoïde g de forte pente. Dans (2.33) la somme des signes de la majorité, SSM, valait :

$$SSM = \sum_{k=1}^{nbc}\left[\frac{1}{nbf_k}\sum_{X_i \in E_k} f(W.X_i)\right] \tag{2.45}$$

et l'on redéfinit donc, lorsque nbc=2p+1, l'indice de déséquilibre dans (2.33) par :

$$DS_2 = \frac{1}{nbc^2}(SSM - g(SSM))^2 \tag{2.46}$$

où g est donc une sigmoïde :

$$g(x) = \frac{2}{1 + e^{t_g.x}} - 1 \tag{2.47}$$

avec une pente t_g que nous avons fixée dans nos essais à 10.
La dérivée partielle de DS_2 par rapport à la $j^{ème}$ composante de W s'écrit alors :

$$\frac{\partial DS_2}{\partial w_j} = 2\left(\frac{\partial SSM}{\partial w_j}(1 - g'(SSM))\right)(SSM - g(SSM)) \tag{2.48}$$

avec :

$$\frac{\partial SSM}{\partial w_j} = \sum_{k=1}^{nbc} \left[\frac{1}{nbf_k} \sum_{X_i \in E_k} \frac{\partial f}{\partial w_j}(W.X_i) \right] \tag{2.49}$$

et :

$$g'(x) = \frac{t_g.(1 - g(x)^2)}{2} \tag{2.50}$$

Les deux améliorations que nous venons de présenter permettent de réduire de manière sensible le nombre d'hyperplans nécessaires à la segmentation de l'ensemble d'apprentissage, comme nous allons le voir maintenant.

2.2.3.5 Résultats

Nous avons appliqué l'algorithme décrit précédemment aux deux problèmes bidimensionnels présentés au § 2.2.2.5.

Les spirales imbriquées

Pour ce problème à deux classes et 192 formes d'apprentissage, l'arbre de décision a généré une partition en 48 polyèdres homogènes à l'aide de 47 hyperplans (Fig. 2.29a), là où la méthode des plus proches voisins (PPV) produisait une partition en 192 polyèdres à l'aide de 545 hyperplans (Fig. 2.18). Si l'on n'utilisait pas l'amélioration décrite au paragraphe précédent (nbc=2), l'arbre contiendrait alors 81 hyperplans.
On peut voir (Fig. 2.29b) que l'on obtient deux régions de décision connexes qui approchent assez bien les spirales recherchées. Ainsi, en se servant de moins de 9% des hyperplans requis par les PPV, l'arbre de classification produit des régions de décision qui coïncident à 89% avec celles des PPV.

On pourra comparer ce résultat à ceux obtenus par Cios et Liu [CIOS 91] (Fig. 2.30). Ces auteurs utilisent également un arbre de décision pour synthétiser un réseau multi-couche. L'arbre, construit par minimisation d'un critère d'entropie, sert à déterminer les poids des automates des premières couches cachées (plusieurs couches cachées peuvent être générées, chaque couche étant totalement connectée à toutes les couches qui la précèdent). La dernière couche cachée et la couche de sortie de leur réseau sont quant à elles synthétisées à l'aide d'un

second algorithme d'apprentissage (programmation linéaire) qui permet de décrire l'appartenance à une classe sous forme de règles portant sur la valeur des automates de la dernière couche précédente (ces règles remplacent les Formes Normales Disjonctives que nous utilisons directement).

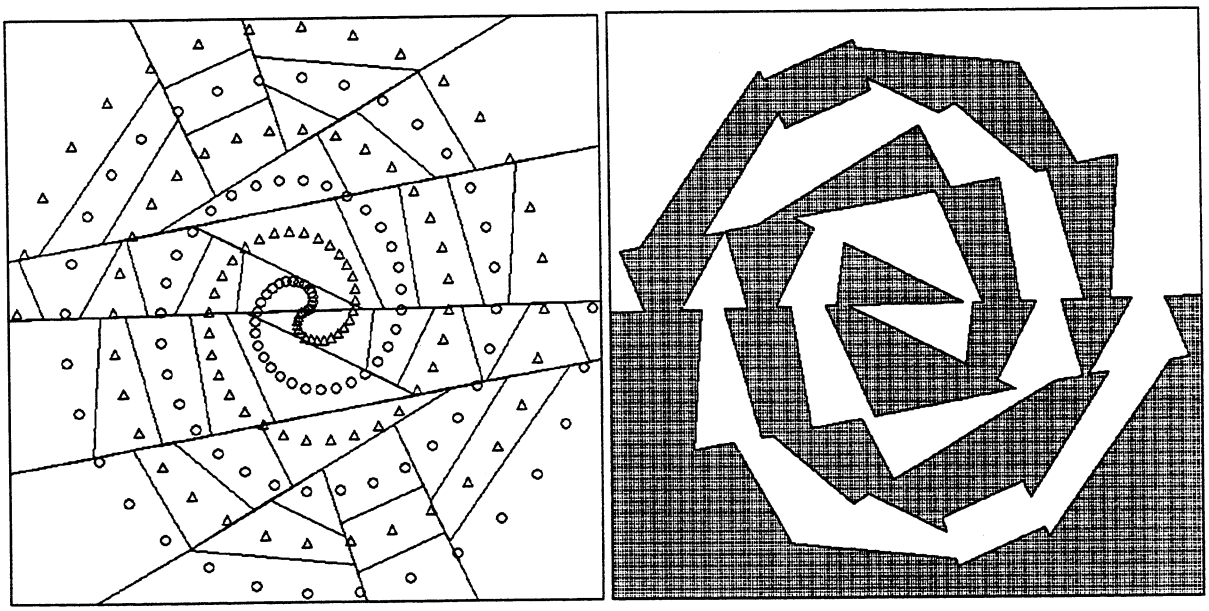


Fig. 2.29. Problème des deux spirales imbriquées. (a) La partition en polyèdres homogènes réalisée par l'arbre de classification. (b) Les régions de décision correspondantes.

Les régions de décision qu'ils produisent ainsi apparaissent beaucoup plus "bruitées" que les régions que nous obtenons (elles ne sont pas connexes en particulier) bien que les réseaux qu'ils génèrent soient à peu près de la même taille que le nôtre (en nombre de poids).

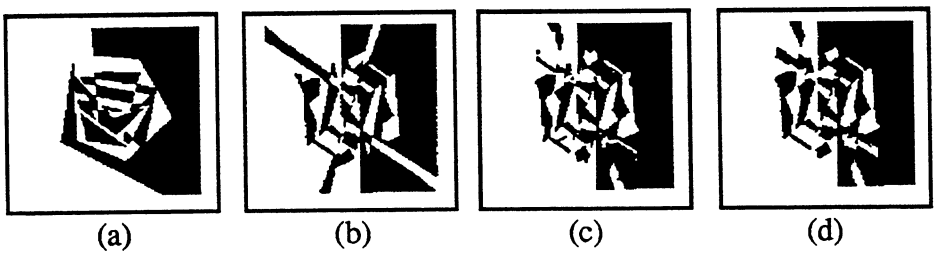
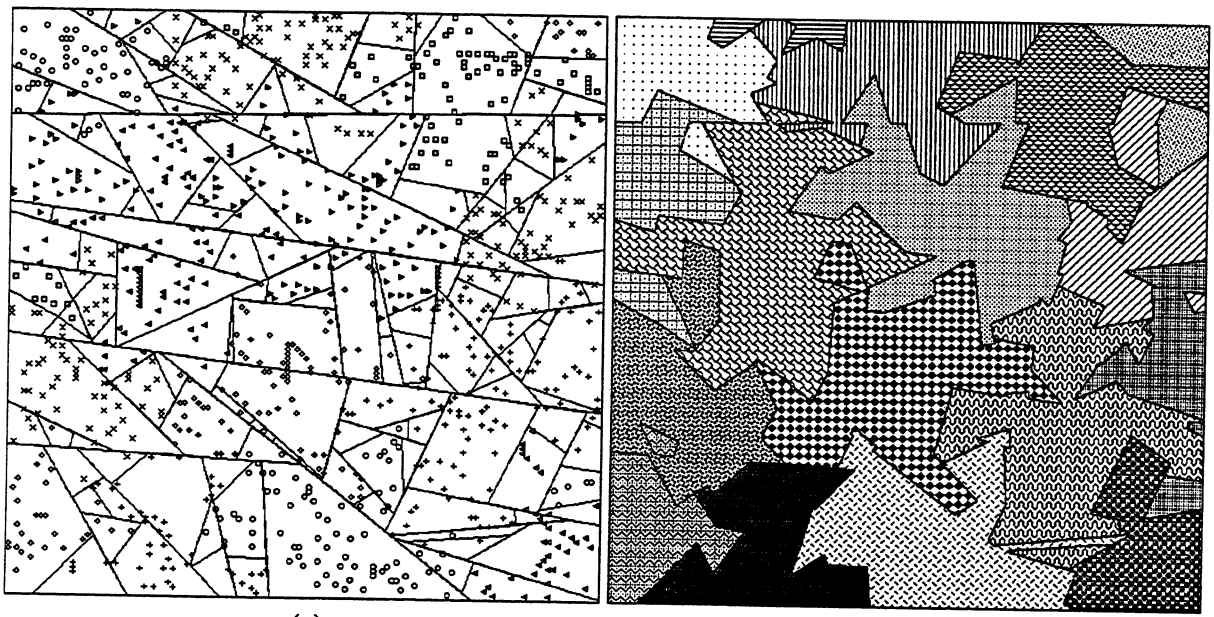


Fig. 2.30. Les régions de décisions obtenues par Cios et Liu (extrait de [CIOS 91]) avec des réseaux ayant 2 (a), 3 (b), 4 (c) et 5 (d) couches cachées.

Les Chinois d'Escher

Dans ce problème l'ensemble d'apprentissage est composé de 1000 formes et 21 classes. L'arbre construit une partition en 150 polyèdres homogènes à l'aide de 149 hyperplans. Si l'on n'utilisait que l'amélioration "nombre de classes impair", l'arbre contiendrait 175 hyperplans, et si aucune des deux améliorations n'était employée, il en contiendrait 205...

Les régions de décision obtenues coïncident à 90% avec les tesselles originales (Fig. 2.19a) et à 88% avec les régions des PPV, en n'utilisant pourtant que 5% des hyperplans utilisés par les PPV.



(a) (b)

Fig. 2.31. Problème des chinois d'Escher. (a) La partition en polyèdres homogènes réalisée par l'arbre de classification proposé. (b) Les régions de décision correspondantes.

Nous avons également expérimenté cet algorithme dans des espaces de dimension plus réaliste que celles du plan (cf § 3.3.2.2).

2.2.3.5 Discussion

Comme nous venons de le voir, l'arbre de classification proposé permet de "dessiner" des régions de décision de complexité quelconque tout en utilisant beaucoup moins d'hyperplans que la méthode des plus proches voisins. Cet algorithme est donc un bon candidat pour la

synthèse de réseaux multi-couches. Comment se comporte-t-il en termes de temps d'apprentissage et de taille du réseau généré ?

Temps d'apprentissage.

Il est difficile d'évaluer la complexité d'un tel algorithme. Une descente en gradient est effectuée à chaque nœud de l'arbre. Les temps d'apprentissage peuvent donc être comparés à ceux d'un réseau à rétro-propagation du gradient comportant autant de cellules qu'il y a d'automates dans la première couche cachée du réseau que nous générons par l'arbre. On notera cependant que le coût que l'on minimise est plus complexe (au moins dans les cas nbc>2 et nbc impair) mais que la taille de l'ensemble d'apprentissage considéré à chaque nœud de l'arbre décroît avec la profondeur de celui-ci. Finalement, ceci compensant sans doute cela, l'expérience montre que la complexité en temps de la construction de l'arbre est effectivement équivalente à une rétro-propagation du gradient.

La méthode souffre également des mêmes défauts : sensibilité au choix des paramètres et existence de minima locaux. Nous avons déjà évoqué le premier problème auquel des solutions similaires à celles de la rétro-propagation peuvent être apportées. Pour ce qui est des minima locaux, il n'est pas possible ici de développer une version "stochastique" de la descente en gradient. Mais les conséquences d'un minimum local sont moins graves : une "mauvaise" segmentation à un nœud de l'arbre n'empêche pas l'apprentissage mais conduit simplement à une solution comportant plus d'hyperplans que nécessaire.

Taille du réseau

En tout état de cause, la partition étant construite de manière hiérarchique, elle ne peut être optimale au sens du nombre d'hyperplans utilisés. Ainsi, si l'on considère un problème similaire à celui de la figure 2.26, quatre gaussiennes bidimensionnelles dont les moyennes sont situées sur les quatre sommets d'un carré, notre arbre de classification va construire une partition à l'aide de trois hyperplans (Fig. 2.32) là où visiblement deux seulement suffisent.

La partition étant définie par un arbre binaire, un arbre de classification devra utiliser au minimum (p-1) hyperplans, même lorsque $\lceil \log_2(p) \rceil$ suffisent. Il se peut alors que deux hyperplans, à des nœuds différents, soient identiques, ou presque. On pourrait imaginer un post-traitement de l'arbre qui rechercherait ces situations, c'est-à-dire la présence de deux hyperplans équivalents, chacun permettant de réaliser la segmentation de l'autre.

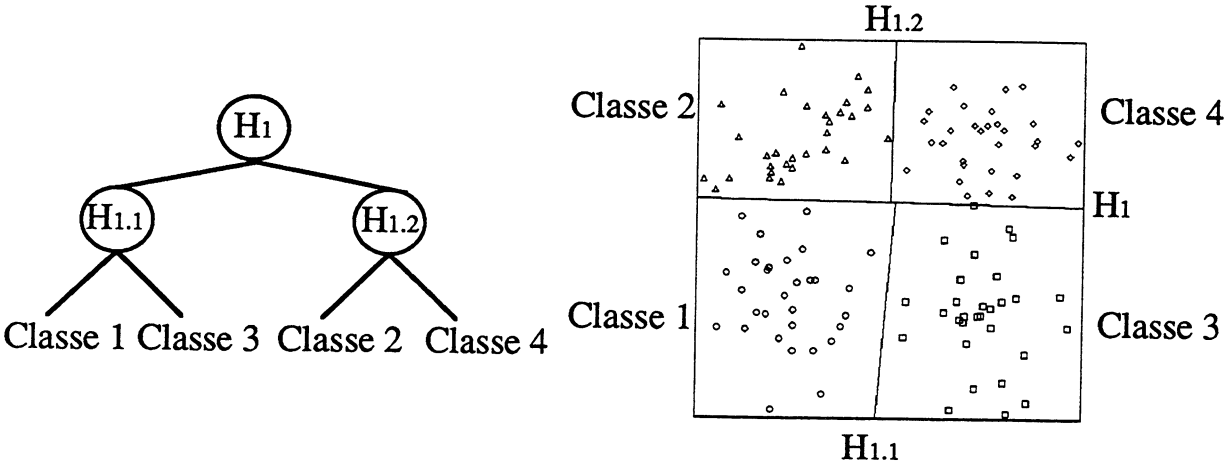


Fig. 2.32. L'arbre de classification définit une partition à l'aide de trois hyperplans alors que deux suffisent (H_{12} et H_{11} peuvent être remplacés par un hyperplan unique).

D'autres méthodes d'apprentissage permettent de s'affranchir de ce problème. Park et Sklansky proposent ainsi de construire la partition de manière non hiérarchique [PARK 89]. Ils recherchent un ensemble minimal d'hyperplans permettant de couper ce qu'ils appellent les "liens de Tomek", liens qui sont en fait les arêtes (segments) du graphe de Gabriel reliant deux points de l'ensemble d'apprentissage de classes différentes. Le graphe de Gabriel [PREPARATA 85] est un sous-graphe du graphe de Delaunay dont le calcul, contrairement à ce dernier, peut être entrepris même dans des espaces de dimension importante puisqu'il est en $O(N^3)$, et par conséquent indépendant de la dimension des données.
Cette méthode cherche donc explicitement à construire des frontières de décision qui approchent "une partie" des frontières de décision des PPV, partie qui est le sous-ensemble des frontières de Voronoï constitué par les hyperplans médiateurs des points de classes différentes qui sont adjacents dans le graphe de Gabriel.

Par cette approche, il est possible de trouver des partitions utilisant un minimum d'hyperplans (Fig. 2.33). Par contre, la méthode construisant des frontières de décision plutôt que des régions, il est nécessaire ensuite d'étiqueter chacun des polyèdres de la partition en recherchant quels points de l'ensemble d'apprentissage il contient (la correspondance entre "étiquette de polyèdre" et classe n'est pas mise à jour pendant la construction, comme c'est le cas pour les arbres de classification). Et bien sûr, on ne dispose pas de l'information, parfois intéressante, que constitue la structure hiérarchique d'un arbre. Par exemple, pour la synthèse des deux dernières couches du réseau, l'arbre permet d'obtenir des Formes Normales Disjonctives dans lesquelles chaque conjonction, qui correspond à un polyèdre, ne contient que

les termes utiles, c'est-à-dire ceux correspondant aux hyperplans qui définissent effectivement la frontière du polyèdre considéré et non pas tous les hyperplans de la partition.

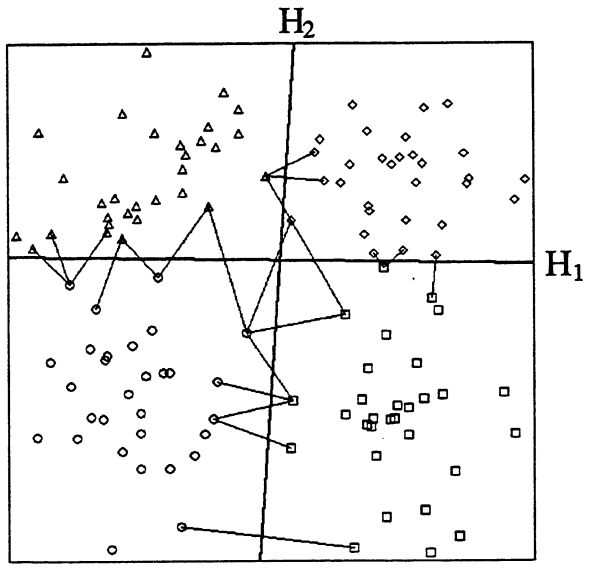


Fig. 2.33. Deux hyperplans suffisent pour couper tous les segments du graphe de Gabriel reliant deux points de classes différentes.

2.3 CONCLUSION

Nous avons vu au cours de ce chapitre deux types de réseaux multi-couches. Les réseaux à unités sigmoïdales munis de l'apprentissage par rétro-propagation du gradient (ou de l'un de ses dérivés) offrent une solution "standard" à tout problème de classification. Ils s'avèrent cependant relativement difficiles à mettre en œuvre. En particulier, le choix de l'architecture du réseau reste un problème crucial. Une solution partielle à ce problème consiste à contraindre l'architecture en s'appuyant sur les connaissances que l'on peut avoir concernant la tâche d'apprentissage considérée. Ces connaissances peuvent être inhérentes au domaine traité, c'est le cas des réseaux à convolution présentés au § 2.1.2.4, ou bien acquises de manière "automatique" comme nous le proposons au § 2.1.2.5. Cependant, aussi contrainte que soit l'architecture, il demeure toujours une part de "mystère" sur la transformation réalisée par un tel réseau, une fois l'apprentissage effectué. Car si la rétro-propagation du gradient a bien résolu l'aspect numérique du problème du *credit assignment*, elle n'en a pas pour autant éclairé l'aspect fonctionnel : quel rôle joue une cellule cachée dans le processus de classification ?

Afin de s'affranchir de ces incertitudes, il est nécessaire de se tourner vers des modèles plus simples de réseaux : les réseaux multi-couches d'automates à seuil. De nombreux

chercheurs semblent maintenant porter leur choix sur ce type de réseaux. Nous citerons entre autres [NADAL 89] et [FREAN 90] qui proposent des algorithmes pour la classification de formes binaires. Pour les formes réelles, nous avons montré comment des réseaux multi-couches d'automates à seuil pouvaient implémenter n'importe quelle technique de classification dont les régions de décisions sont formées par des unions de polyèdres convexes. Le rôle de chaque unité cachée est alors très bien défini dans un tel réseau et la généralisation produite est exactement celle de la méthode de classification implémentée. Cette méthode peut être celle des plus proches voisins (§ 2.2.2), ou bien un arbre de classification tel que celui proposé au § 2.2.3. D'autres algorithmes de construction d'arbre pour la synthèse d'un réseau ont été utilisés ([SETHI 90], [CIOS 91], ou encore [ATIYA 90] pour l'apprentissage non supervisé). On peut également envisager la combinaison de plusieurs méthodes [KNERR 91].

Les réseaux ainsi construits permettent de traiter des problèmes d'apprentissage aussi complexes que ceux abordés par les réseaux à rétro-propagation du gradient (cf les spirales imbriquées) tout en conservant la totale compréhension du classifieur obtenu. D'un point de vue pragmatique, on pourra toutefois s'interroger, comme nous l'avons fait au chapitre précédent à propos des réseaux à prototypes, sur l'intérêt qu'il y a à exprimer une technique de classification connue sous forme connexionniste. Ici encore, nous répèterons que la formulation "réseau de neurones", fût-il en couches, n'est sûrement pas la méthode la plus efficace, au moins sur machine séquentielle, pour calculer la classification réalisée par un arbre de décision. La mise en évidence des équivalences entre réseaux multi-couches et arbres de classification sert plutôt à rappeler, s'il en était besoin, qu'il existe des alternatives aux réseaux à rétro-propagation du gradient. Ces alternatives ont un fonctionnement proche de celui des réseaux mais elles permettent de contourner l'écueil du choix de l'architecture. Leurs performances semblent également assez proches de celles de ces réseaux avec, semble-t-il, un léger avantage, non décisif, à ces derniers [ATLAS 90]. Il est donc nécessaire, lorsque l'on aborde un problème réel, de comparer ces deux types de méthodes. C'est ce que nous allons faire au cours de ce troisième chapitre consacré à la reconnaissance de partitions musicales.

CHAPITRE 3

RECONNAISSANCE DE PARTITIONS MUSICALES

3.1 INTRODUCTION

3.1.1 Traitement et transformation de l'information musicale

On peut décrire toute activité musicale comme un cycle de transformations des données (musicales et physiques) par des processeurs ou transducteurs qui sont, dans l'ordre des transformations : le compositeur, l'interprète, l'instrument, la salle de concerts et enfin l'auditeur, le but final de ce cycle étant de provoquer chez ce dernier une émotion esthétique [MOORE 90] (Fig. 3.1). Le compositeur imagine et élabore son œuvre par référence à une base de connaissances musicales. Le matériau qu'il transmet à l'interprète est le plus souvent une représentation visuelle de l'œuvre à jouer : c'est la partition prescrivant les événements sonores qu'il a voulus. Les commandes temporelles exercées par l'interprète sur l'instrument synchronisent la production de sons caractérisés par leur timbre, leur hauteur, leur intensité et leur durée. La salle de concerts, par ses dimensions, ses matériaux, sa réverbération, spatialise le champ sonore. L'auditeur perçoit une partie de ce champ sonore et, par référence à des connaissances musicales qu'il partage en partie ou pas du tout avec le compositeur, est à même de recevoir son message et d'en éprouver une émotion.

Ainsi, ce cycle de transformations de données par des processeurs peut modéliser une forme de communication artistique qui est présente dans toutes les cultures. La courte histoire de l'informatique musicale a déjà tenté d'introduire l'ordinateur dans la transformation de ces données, voire même de substituer la machine à l'un des processeurs. C'est naturellement l'instrument qui, le premier, a été imité, étendu, modifié par l'ordinateur, devenu synthétiseur

[RISSET 69], [CHOWNING 73]. Puis la spatialisation des salles, à son tour, a été modélisée permettant des effets d'écho ou de réverbération [SCHRŒDER 80]. La pensée musicale peut également s'exprimer à l'aide de l'informatique, devenue outil de composition algorithmique ou stochastique [ROADS 79], [RODET 84].

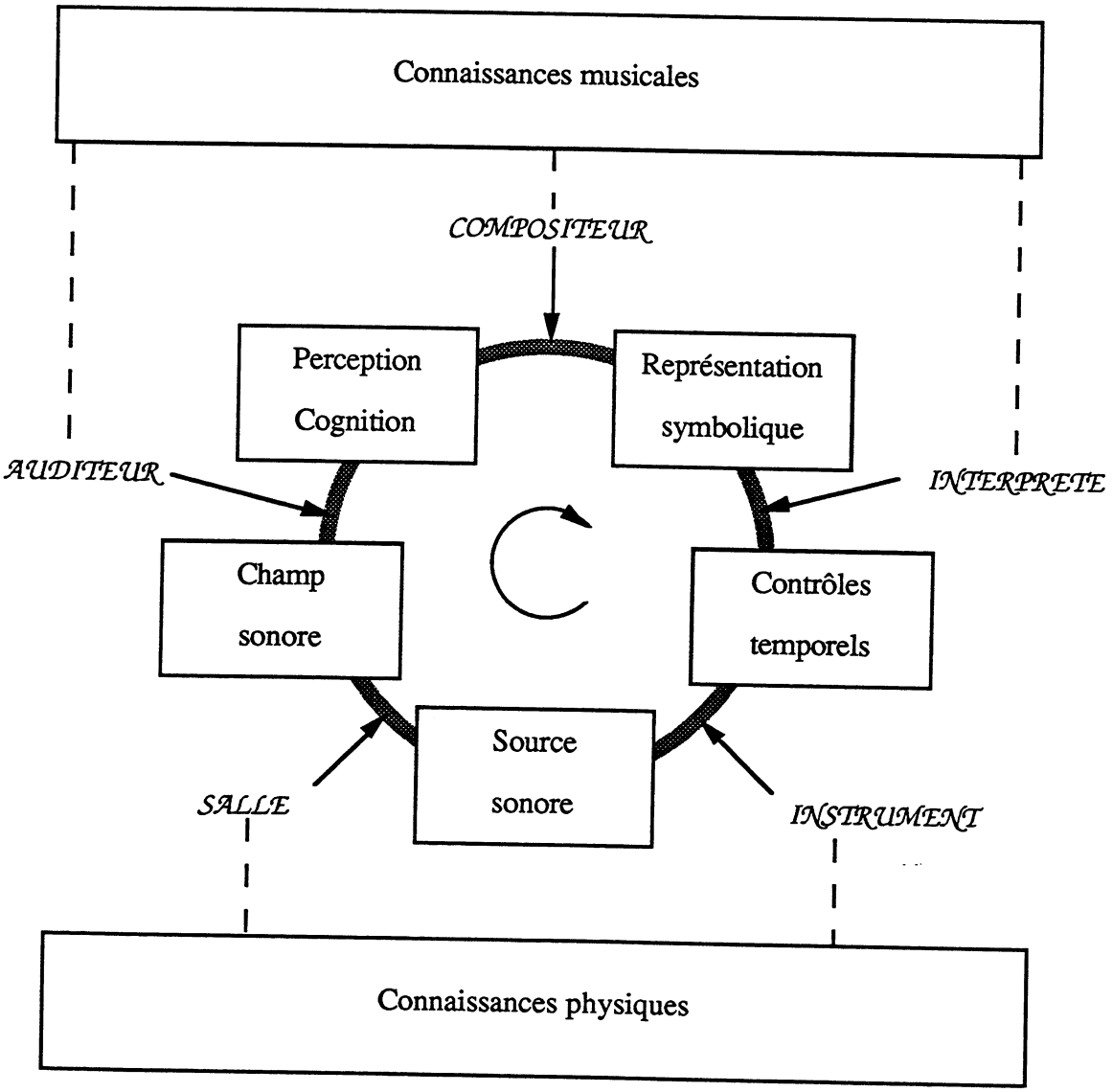


Fig. 3.1. Transformations de données musicales [MOORE 90].

Nous nous intéressons ici au processeur interprète et plus particulièrement à sa première activité, la lecture visuelle de la partition. L'interprète est lui-même auditeur de ce qu'il produit (Fig. 3.2), ce qui lui permet par rétroaction de contrôler son jeu issu de sa compétence notationnelle (compréhension de la partition), de sa compétence communicative (créativité et traditions d'exécution) et enfin de sa stratégie d'exécution (dextérité et comportement de

l'instrument et de la salle de concerts). La compétence notationnelle mesure la maîtrise plus ou moins achevée du lien logique existant entre la notation et l'univers sonore : une formule musicale n'est pas autre chose qu'une expression en notes s'apparentant à une phrase déclarative d'un langage naturel. Notre travail consiste à tenter de transposer dans un système artificiel cette compétence notationnelle acquise par tout musicien débutant, en vue de disposer d'une entrée visuelle de partitions dans un environnement d'informatique musicale [MARTIN 89, 90, 91abd, 92].

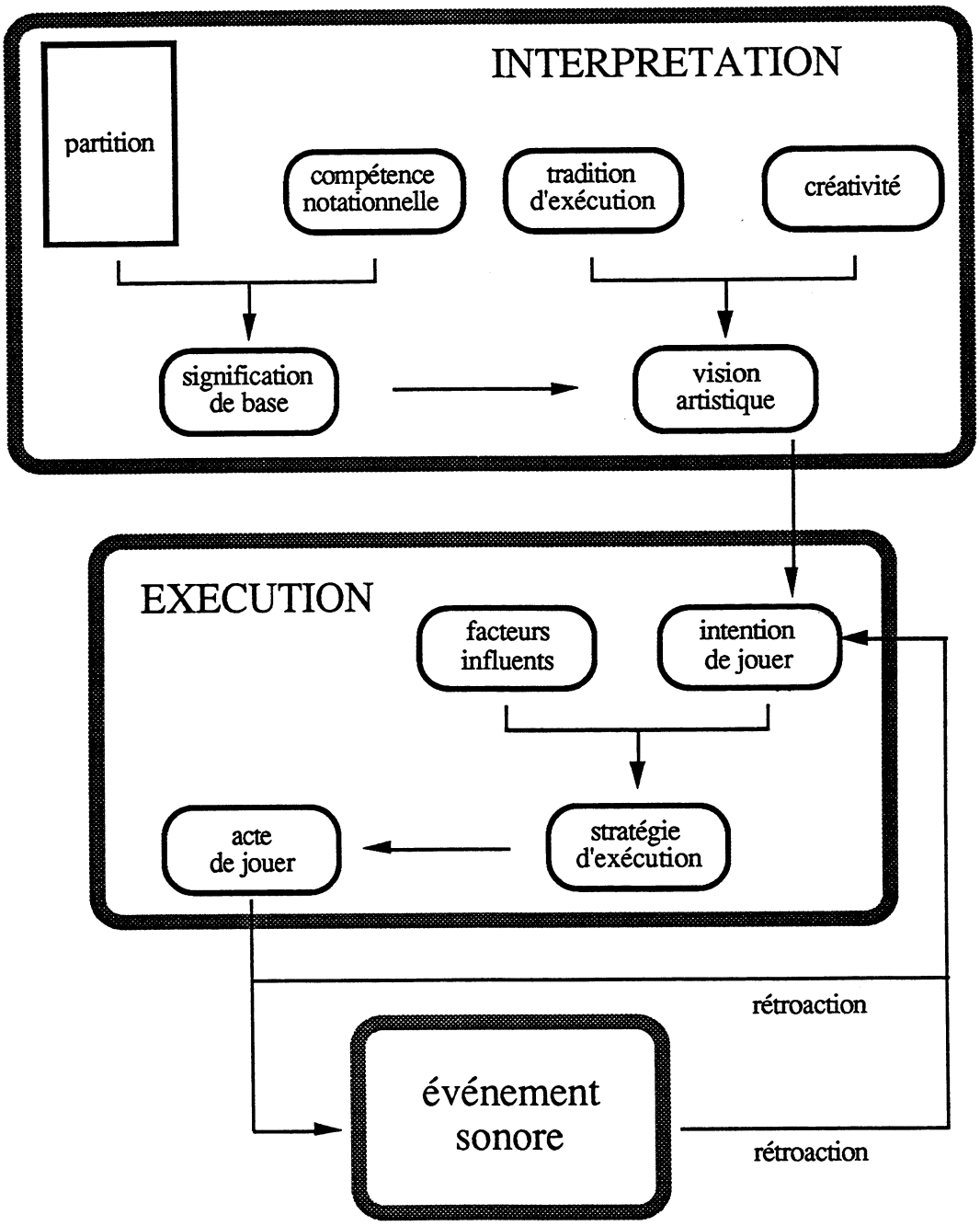


Fig. 3.2. Acte d'exécution d'une partition [KURKELA 88].

Cet environnement d'informatique musicale s'est largement étoffé grâce aux développements réalisés autour de la norme MIDI et des micro-ordinateurs peu coûteux capables de piloter des instruments à cette norme [LOY 85]. Une multitude de logiciels musicaux ont ainsi vu le jour : éditeurs de partitions, aides à la composition, "studios d'enregistrement", ... L'information manipulée par ces logiciels est dans la plupart des cas représentée à l'aide de la notation musicale classique : la partition. Cette information peut provenir soit d'un processus "manuel" (on décrit la partition symbole après symbole à l'aide d'une souris), soit d'une acquisition plus automatique par le biais d'un signal MIDI (le morceau est exécuté sur un synthétiseur, par exemple, et le signal résultant fournit une suite de codes décrivant les attributs des notes, hauteur et durée). Dans ce dernier cas, il est toujours nécessaire de corriger et compléter manuellement la partition qui ne peut évidemment être totalement déduite de son interprétation sur un instrument.

La reconnaissance optique des partitions, à l'instar de la reconnaissance optique des caractères (ROC) pour les traitements de texte, pourrait offrir une alternative utile à ces méthodes de saisie fastidieuses et lentes, et constituer ainsi une aide importante à l'édition musicale.

3.1.2 Stratégie de reconnaissance.

Le processus de reconnaissance d'une partition, comme celui de tout document graphique (plans, dessins techniques, ...) peut être décomposé en deux grandes étapes : une première étape d'identification de symboles, que l'on peut qualifier de bas-niveau, et une seconde étape consistant à extraire l'information qui réside dans la manière dont sont disposés ces symboles dans le plan du document. Cette décomposition n'a pas toujours été clairement soulignée dans les travaux que compte la littérature dans ce domaine : [PRERAU 71 et 75], [LUBAR 82], [TOJO 82], [BEASSE 87], [ROACH 88][1], [COUASNON 91] (on pourra trouver dans [BLOSTEIN 91] une synthèse bibliographique complète). Les systèmes qui en résultent, peu "formalisés", sont fondés sur des ensembles de règles ad-hoc combinant à un même niveau analyse d'image et analyse syntaxique. Cette caractéristique les rend sans doute peu évolutifs, alors même que les documents partition, qui utilisent un vocabulaire de symboles potentiellement très large et des conventions notationnelles très riches, exigent une certaine versatilité.

Si l'on veut bien admettre cette décomposition comme nécessaire, un certain nombre d'outils largement éprouvés s'offrent alors à nous. Ainsi, l'étape haut-niveau de la reconnaissance, faisant appel à la syntaxe musicale, pourra idéalement être "attaquée" par la

---

[1] Etude consacrée aux partitions manuscrites.

technique des grammaires de graphes, comme en témoignent les travaux de O. Fahmy [FAHMY 91]. Pour notre part, nous ne nous sommes intéressés qu'à la phase bas-niveau de la reconnaissance, en nous cantonnant au cas des partitions imprimées. Celle-ci s'apparente alors beaucoup aux problèmes de ROC auxquels de nombreuses solutions ont été apportées. Le système que nous allons décrire est classiquement composé, comme tout système de reconnaissance de caractères fondé sur l'analyse des primitives [CASH 87], d'une séquence de quatre opérations : pré-traitement, segmentation, extraction d'indices et classification.

Cependant certaines de ces opérations doivent être adaptées au type particulier de documents que nous traitons.

Pré-traitement

Cette phase, qui suit celle de l'acquisition de l'image, vise à corriger les distortions que celle-ci a pu produire. Elle consiste généralement en des améliorations d'image, un seuillage et un contrôle de l'horizontalité du document acquis. Ce dernier traitement pourra particulièrement être adapté au fait que l'image que l'on manipule ici contient un certain nombre de droites qui doivent être horizontales si le document l'est aussi : ce sont bien sûr les portées, ensembles de cinq lignes sur lesquelles sont disposés les symboles de la partition.

Segmentation

Si les lignes de portées facilitent la correction du biais, elles compliquent au contraire la segmentation, c'est-à-dire la localisation des symboles dans l'image. En reconnaissance de caractères, chaque symbole est identifié, dans le meilleur des cas, comme étant une composante connexe de l'image binaire. Dans le cas de partitions, l'image est principalement constituée d'une "grosse" composante connexe composée de tous les symboles reliés entre eux par les portées sur lesquelles ils ont été placés. La segmentation va alors consister à effectuer l'opération que réalise inconsciemment le musicien lisant sa partition, c'est-à-dire à séparer les objets musicaux du "fond" du document, i.e. des portées.

Extraction d'indices et classification

Les symboles du document ayant été isolés, ils doivent être caractérisés par un certain nombre de mesures, ou indices, au vu desquels la classification proprement dite pourra s'exercer. Indices et classifieur doivent naturellement être compatibles. La littérature décrit un grand nombre de méthodes de classification et une multitude plus vaste encore de types d'indices. Parmi ces derniers, on peut distinguer les indices à caractère fonctionnel des indices à

caractère structurel [COUEIGNOUX 81].

Les premiers tentent de décrire le symbole par des mesures qui n'ont pour seule justification que leur pouvoir "mathématique" de discrimination (moments géométriques [CASH 87], coefficients de la transformée de Fourier [WEIDEMAN 89] ou de Walsh-Hadamar [VASQUEZ 90], ...).

Les seconds vont décomposer les symboles en primitives élémentaires plus simples à reconnaître que l'entité qu'elles construisent : segments de droite, courbes, boucles, ...

Si rien ne permet en général d'opter *a priori* pour telle ou telle méthode, il apparaît dans notre cas que la nature même des symboles étudiés, ou tout au moins d'une partie d'entre eux, appelle une description structurelle. Il convient en effet de distinguer ici les signes musicaux iconiques des signes symboliques.

Les signes iconiques, les notes, sont caractérisés par le lien existant entre leur forme et leur signification. Les notes sont ainsi interprétées en fonction de la manière dont elles sont construites à partir de primitives élémentaires (ellipse, queue, crochet, barre), construction pouvant prendre une multitude de formes.

Les signes symboliques, c'est-à-dire tous les autres, peuvent réellement être qualifiés de symboles, au même titre qu'un caractère de l'alphabet, dans la mesure où leur forme est purement conventionnelle et donc sans lien direct avec le sens qui leur est attaché.

Cette distinction entre natures de signes musicaux nous a amené d'une part à choisir des indices de type structurel et d'autre part à réaliser une classification en deux étapes, chacune adaptée à une nature de signe particulière.

En effet, en ce qui concerne les signes iconiques, la situation est typiquement celle décrite par Fu [FU 80] : un ensemble quasi infini de signes obtenus par assemblage de primitives élémentaires. On ne peut dans ce cas considérer chaque description possible comme une catégorie que l'on va devoir apprendre à un classifieur. La reconnaissance doit plutôt consister à décrire chaque forme en fonction des primitives adéquates (extraction d'indices) et à vérifier que celles-ci sont assemblées de manière cohérente (classification). L'approche structurelle s'impose ici d'elle-même.

Les signes symboliques, quant à eux, peuvent être traités de la même manière que des caractères. Des indices structurels devant être extraits afin de traiter les notes, il semble logique d'employer cette information déjà disponible pour les autres symboles. Par contre la classification peut ici tirer parti du fait qu'un nombre fini (et raisonnable) de classes sont à distinguer. On pourra dès lors utiliser une technique fonctionnant selon un apprentissage par l'exemple. Nous nous tournerons naturellement pour cela vers les réseaux de neurones.

Nous avons donc opté pour une phase de classification en deux étapes dans laquelle, après une phase d'extraction d'indices commune à tous les signes, on tentera tout d'abord d'identifier chaque objet comme une note. On essaiera ensuite, dans une seconde étape,

d'assigner aux objets rejetés par la première, une classe de symbole (Fig. 3.3).
La suite de ce chapitre est consacrée à la description de ces différentes étapes.

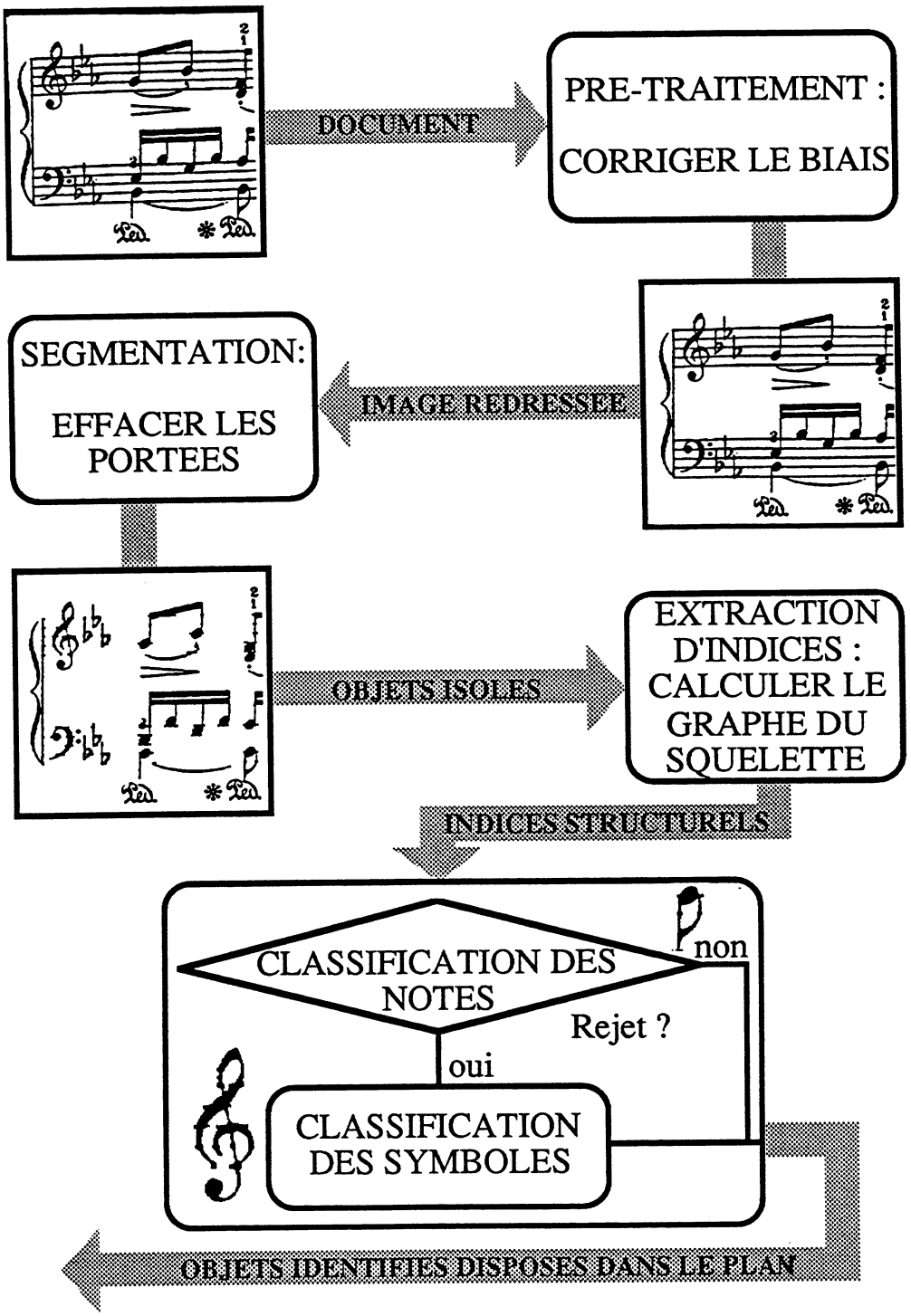


Fig. 3.3. Architecture du système "bas-niveau" de reconnaissance de partitions.

3.2 PRE-TRAITEMENT ET SEGMENTATION

Nous avons disposé pour notre étude d'un scanner optique "à plat" permettant d'acquérir des images de partitions au format A4 à une résolution de 300 points par pouce (≈12 pts/mm). Un tel dispositif fournit des images binaires de très bonne qualité et soulage ainsi la phase de pré-traitement d'opérations qui peuvent parfois s'avérer délicates : rehaussement du contraste, seuillage adaptatif, ... Le seul traitement qu'il nous ait semblé nécessaire d'effectuer est une fermeture morphologique binaire [SERRA 82] (dilatation suivie d'une érosion en 8-connexité[1]) afin de combler les petits trous accidentels pouvant apparaître au milieu d'une zone noire. Ce bruit "poivre et sel" est d'ailleurs plus souvent dû à la qualité du document numérisé qu'au dispositif de numérisation lui-même. Les tâches noires au milieu d'une zone blanche seront éliminées plus tard dans le processus, une fois que l'échelle du document aura pu être appréciée, ceci afin de n'éliminer que les tâches trop petites pour pouvoir être des points de la notation.

Cette amélioration étant apportée, l'essentiel du pré-traitement va consister à corriger l'éventuel biais du document. Ce biais, qui peut provenir d'un mauvais positionnement du document dans le scanner, ou d'une photocopie mal réalisée, peut s'avérer gênant par la suite, bien qu'étant en pratique toujours très faible (quelques degrés au maximum). En le corrigeant dès l'origine, on s'assure que la reconnaissance effectuée sera invariante en rotation sans laisser à la phase de classification la prise en charge de cette invariance.

3.2.1 Correction du biais

Plusieurs méthodes de correction du biais ont été développées jusque-là en ROC. Elles utilisent l'histogramme de la projection horizontale [CIARDELLO 88], la transformée de Hough [HINDS 90] voire même la transformée de Fourier dans le cas d'images en niveaux de gris [POSTL 86]. Toutes sont fondées sur l'existence de structures linéaires (des lignes de textes) dans l'image numérisée et sont donc applicables aux partitions qui, nous l'avons dit,

---

[1] Rappels :

3	2	1
4	P	0
5	6	7

4-voisinage de P = {0, 2, 4, 6}

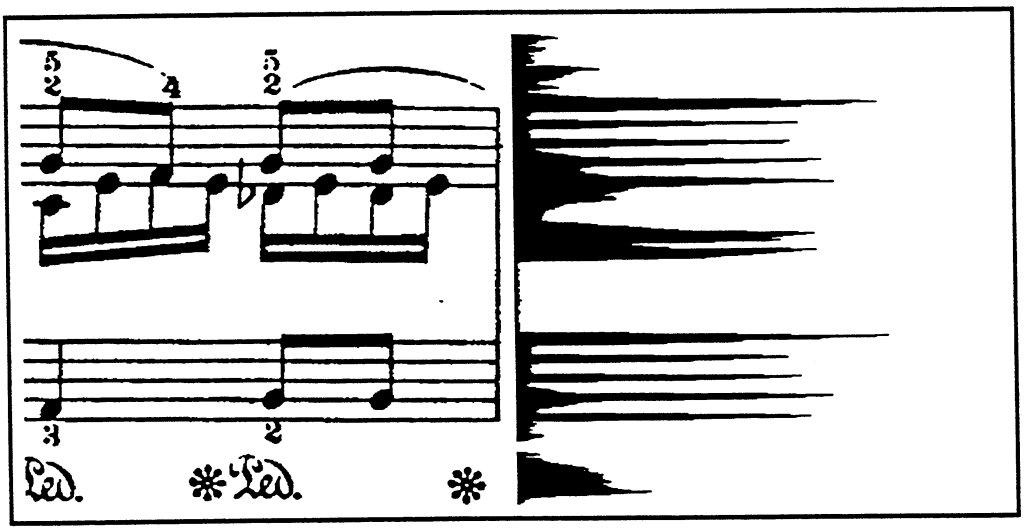
8-voisinage de P = {0, 1, 2, 3, 4, 5, 6, 7}

Lors d'une dilatation (resp. érosion) en d-connexité (d=4 ou 8), tout pixel fond (resp. objet) qui a un pixel objet (resp. fond) dans son d-voisinage devient pixel objet (resp. fond). Une fermeture est une dilatation suivie d'une érosion. Elle a pour effet de combler les petits trous et lisser les contours des objets.

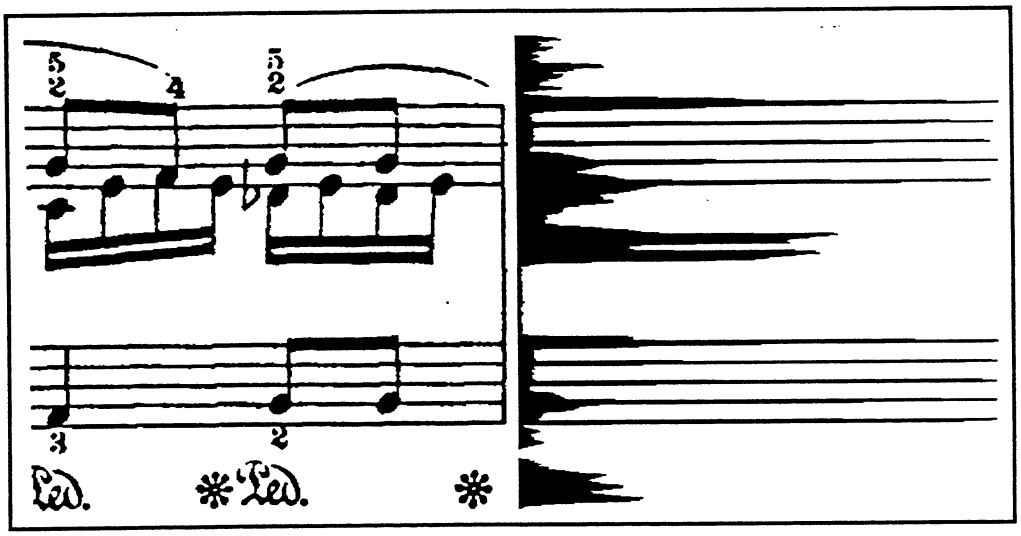
contiennent elles aussi de fort belles structures linéaires : les lignes de portées. Nous avons expérimenté les deux premières méthodes et en avons finalement développé une troisième plus adaptée à notre problème.

3.2.1.1 Histogramme de la projection horizontale

Chaque valeur de l'histogramme de la projection horizontale d'une image binaire comptabilise le nombre de pixels objets (pixels noirs) se trouvant sur une ligne de cette image. Si les lignes de portées sont horizontales, elles doivent coïncider avec les lignes de l'image et donc induire des pics marqués dans l'histogramme (Fig. 3.4).



(a)



(b)

Fig. 3.4. (a) Image originale biaisée et son histogramme. (b) Image après correction du biais et son histogramme.

Le principe de la méthode consiste alors à calculer cet histogramme pour plusieurs images obtenues par rotation de l'image originale. Les angles des rotations seront choisis dans un intervalle $[-\alpha_{max},+\alpha_{max}]$, où α_{max} est le biais maximum attendu, le centre de la rotation étant choisi arbitrairement (le centre de l'image par exemple). La rotation d'angle α_0 qui produit l'histogramme dont la variance est maximale sera choisie comme étant celle permettant de corriger le biais.

L'inconvénient majeure de cette méthode provient de ce que plusieurs rotations doivent être calculées (plus on veut de précision dans l'angle, plus on doit en calculer), rotations qui impliquent de coûteux calculs en nombres flottants. Même en ne travaillant que sur une partie de l'image (dont on est sûr qu'elle contient au moins une ligne de portée), les temps de calcul s'avèrent importants (voir table 3.1).

3.2.1.2 Transformée de Hough

La transformée de Hough a été largement utilisée en analyse d'image pour la détection de structures géométriques : droites, ellipses, ... [MAITRE 85], [ILLINGWORTH 88]. Le principe en est, informellement, le suivant : si l'on recherche dans une image une famille de formes géométriques caractérisées par **p** paramètres, la transformation de Hough est celle qui associe à chaque pixel objet P de l'image l'ensemble H(P) des points de $\mathbb{R}^p$ correspondant aux paramètres des structures auxquelles le point P peut appartenir : $H(P) = \{ Q \in \mathbb{R}^p$ / le point P appartient à la structure de paramètres Q $\}$. Si un nombre suffisant de pixels ont un même point Q pour image, alors on peut en déduire que la structure géométrique ayant Q pour paramètres est présente dans l'image. Pour en décider, après avoir discrétisé l'espace $\mathbb{R}^p$ des paramètres, on comptabilisera pour chaque point Q le nombre de pixels l'ayant pour image, ou autrement dit, le nombre de pixels ayant voté pour Q:

$$V(Q) = \left| \{ P \text{ pixel objet} / Q \in H(P) \} \right|$$

Dans notre cas, cette transformation peut nous aider à rechercher les lignes de portées, l'angle de celles-ci nous fournissant alors le biais recherché. Pour la recherche de droites, on utilise en général la représentation polaire [DUDA 72] où chaque droite est décrite par les paramètres ρ et θ (p=2). Pour un pixel P(x,y), H(P) est une sinusoïde dans $\mathbb{R}^2$:

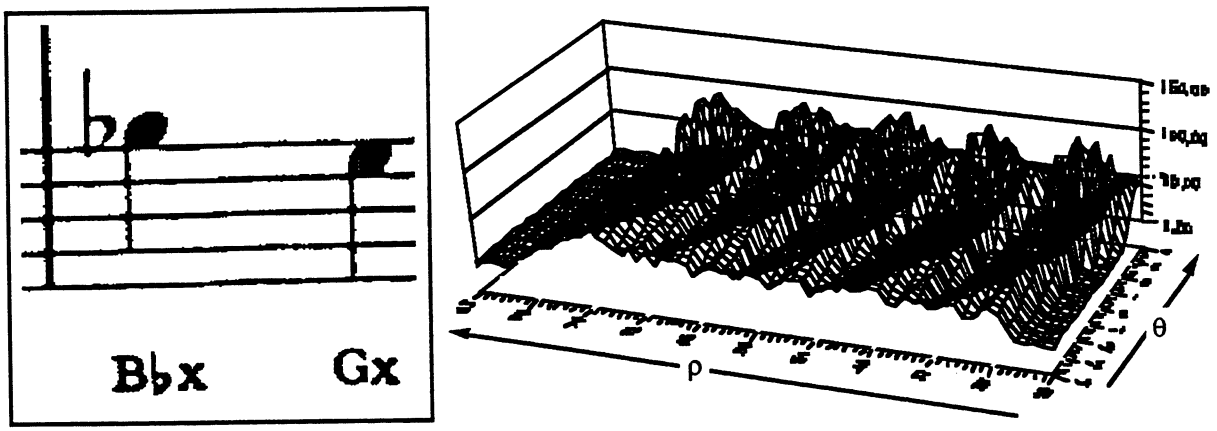
$$H(P) = \{ (\rho,\theta) \in \mathbb{R}^2 / \rho = x.\cos(\theta) + y.\sin(\theta) \}$$

En supposant que l'on recherche seulement les segments de droite dont la pente est comprise entre $-\alpha_{max}$ et $+\alpha_{max}$, le calcul de l'espace des votes pour une image de taille $h \times l$ se fait de la

manière suivante :

```
Pour i variant de 1 à h faire
  Pour j variant de 1 à l faire
    Si P(i,j) est un pixel objet alors faire
      Pour θ variant de -αmax à +αmax faire
        ρ = i.cos(θ) + j.sin(θ)
        V(ρ,θ) = V(ρ,θ) + 1
```

L'avantage principal de cette méthode réside dans sa résistance au bruit : les segments de droites peuvent être détectés même en étant déconnectés. Ses inconvénients sont les temps de calcul importants (voir table 3.1) et les difficultés que l'on rencontre d'une part à discrétiser convenablement l'espace des votes et d'autre part à analyser celui-ci. En effet, à la résolution à laquelle nous travaillons, une ligne de portée peut avoir de 2 à 5 pixels d'épaisseur. Si les dimensions ρ et θ sont discrétisées de manière trop fine, chaque ligne de portée va se traduire par la présence de plusieurs pics voisins dans l'espace des votes (Fig. 3.5). Comment dès lors déterminer exactement les paramètres des lignes de portée ?



(a) (b)

Fig. 3.5. (a) Une image 256x256 contenant cinq lignes de portées. (b) L'espace des votes calculé pour θ variant entre -4° et +4°, avec un pas de discrétisation de 0,25 selon θ et de 1 selon ρ.

Pour résoudre ce problème, [FLETCHER 88] propose, dans le cadre de la ROC, de calculer la résolution selon les ρ de manière à ce que les pixels compris dans l'épaisseur d'une ligne de caractères votent, à θ donné, pour le même point de l'espace des paramètres. Ceci

suppose de déterminer auparavant, dans son cas la taille de la police utilisée et, dans le nôtre, l'épaisseur des lignes de portée, ceci au risque de perdre dans cette étape préliminaire le temps que l'on espère gagner ensuite dans l'analyse des votes.

Cette méthode ne nous a pas donné satisfaction et s'avère être bien trop lourde par rapport à la qualité des images dont nous disposons : il n'y a ici nul besoin d'une technique résistant au bruit, les droites que nous recherchons étant toujours parfaitement connectées sur des partitions imprimées (la transformée de Hough pourra par exemple être utilisée pour de vieilles partitions détériorées ou des documents manuscrits, mais ceci est un autre problème débordant du cadre que nous nous sommes fixé...). Nous proposons donc une méthode simple adaptée à ces conditions favorables.

3.2.1.3 Correction du biais par les cordes

Cette méthode utilise la notion de *corde* discrète que l'on définit de la manière suivante :

> Soit P un pixel objet appartenant à une composante 8-connexe C. La corde d'orientation θ en P est le segment de droite discret de pente θ passant par P et inscrit dans la composante C. La longueur de cette corde est définie comme étant la distance entre les deux points P1 et P2 qui sont à l'intersection du segment avec la frontière de C (Fig. 3.6).

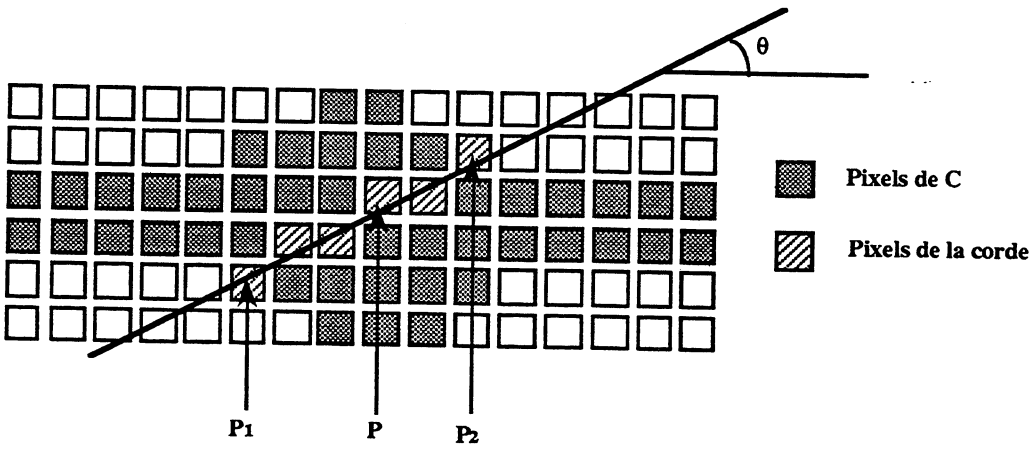


Fig. 3.6. Corde discrète d'orientation θ en P.

La corde peut être calculée de manière efficace si l'on utilise un algorithme de tracé de droites discrètes tel que celui de Bresenham [BRESENHAM 65] : en partant du pixel P on se

déplace de part et d'autre de celui-ci selon la droite discrète jusqu'à atteindre un pixel fond ou le bord de l'image. La longueur de la corde, $L(P,\theta)$, est également calculée de manière discrète : nous l'avons prise égale à la distance d_4 ou *city-block distance* :

$$L(P,\theta) = d_4(P_1(x_1,y_1) , P_2(x_2,y_2)) = | x_1 - x_2 | + | y_1 - y_2 |$$

La corde permet donc de détecter si un segment de droite connexe, d'orientation donnée, passe en un point. Si un point P appartient à une des lignes de portée, celles-ci étant toujours les plus grandes lignes présentes dans l'image, l'orientation de la plus longue corde passant par P fournira l'orientation des portées et donc le biais du document. La méthode de correction consiste donc à calculer ces cordes pour différentes orientations sur $[-\alpha_{max},+\alpha_{max}]$ en chaque point de la colonne centrale de l'image, celle-ci coupant quasi-infailliblement au moins une des lignes de portée.

Pour une image de taille $h \times l$ la correction du biais se fait de la manière suivante :

$L_0 = 0$
Pour i variant de 1 à h faire
  Si P(i,l /2) est un pixel objet alors
    Pour θ variant de $-\alpha_{max}$ à $+\alpha_{max}$ faire
      Calculer la corde d'orientation θ en P
      Si $L(P,\theta) > L_0$ alors
        $L_0 = L(P,\theta)$
        $P_0 = P$
        $\theta_0 = \theta$
Corriger le biais par une rotation de l'image de centre P_0 et d'angle $-\theta_0$

L'image de la Fig. 3.4 a été corrigée par cette méthode ; l'histogramme de la projection horizontale permet d'en observer l'efficacité. Elle s'est avérée robuste, fonctionnant correctement avec toutes les images traitées, et également plus rapide que les deux techniques présentées précédemment (Table 3.1).

Table 3.1. Temps CPU pour la correction du biais d'une image 512x512. Pour chaque méthode on a considéré 30 valeurs de biais possibles sur l'intervalle [-3.5,+3.5] (en degrés). Pour l'histogramme de la projection horizontale, le fragment d'image traité était de taille 31x512 (tous les temps indiqués ici, comme ceux qui suivent, ont été mesurés sur station Apollo DN3550, programmation en Pascal et en C).

Histogramme de la projection horizontale	Transformée de Hough	Recherche des cordes discrètes
11 s	1 mn 05 s	4 s

3.2.2 Segmentation : érosion des portées

La segmentation consiste à isoler les symboles des portées sur lesquelles ils sont disposés et qui les connectent. Pour cela, l'approche la plus répandue consiste à "effacer" de l'image les lignes de portées. On peut également envisager d'analyser la partition en conservant les portées [COUASNON 91], mais cela semble bien moins naturel.

L'option "effacement" étant choisie, une alternative s'offre à nous : soit l'on efface "brutalement" les lignes, déconnectant ainsi tous les symboles qui les chevauchent, soit l'on tente de préserver la connexité de ces symboles en n'effaçant que les fragments de ligne nécessaires. Dans le premier cas, la difficulté réside dans la reconstruction des symboles déconnectés, processus qui, de l'avis même de son auteur, s'avère assez périlleux [PRERAU 71,75]. C'est sans doute pourquoi tous les travaux postérieurs ont opté pour la seconde voie consistant à réaliser une érosion des portées suffisamment "intelligente" pour n'effacer que ce qui doit l'être [LUBAR 82], [TOJO 82], [ROACH 88]. Nous avons également choisi cette option parce qu'au-delà de ces considérations "historiques", il ne semble pas très logique de détériorer sans nécessité une information qui, à l'origine, est présente dans l'image (la connexité des symboles), tout en sachant qu'il faudra de toute façon la reconstituer plus tard.

3.2.2.1 Détection et suivi des portées

La première étape de la segmentation consiste à rechercher et suivre les lignes de portées. La détection est aisément réalisée sur l'histogramme de la projection horizontale de l'image. Le pré-traitement ayant fourni une image redressée, chaque portée est détectée par la présence, dans l'histogramme, d'un groupe de cinq pics régulièrement espacés (Fig. 3.4), pics que l'on aura isolés par un simple seuillage (seuil proportionnel au maximum de l'histogramme). Chaque pic nous donne ainsi la position d'une ligne de portée dans l'image, ainsi qu'une évaluation de son épaisseur en pixels (la largeur du pic). Le suivi consiste alors à balayer chaque ligne de gauche à droite pour identifier les pixels objets susceptibles d'appartenir à la portée. Le balayage d'une ligne est réalisé de la manière suivante (Fig. 3.7) :

Soit une ligne de portée d'épaisseur e détectée entre les lignes x et x+e-1.
i_init = x+e/2 /* on part du centre présumé de la ligne de portée */
Pour j variant de 1 à m faire
. Rechercher, en s'éloignant verticalement de part et d'autre de P(i_init,j) mais en restant dans une zone comprise entre les lignes x-∂e_1 et x+e-1+∂e_1, le premier pixel objet : P(i,j) /*∂e_1 représente une incertitude "large" sur la position ou l'épaisseur de la ligne de portée */
. Si aucun P(i,j) n'est trouvé, cela signifie que la ligne de portée est interrompue ou n'existe pas en colonne j

. Sinon rechercher dans la colonne j, toujours entre x-∂e_1 et x+e-1+∂e_1, l'ensemble connexe de s pixels objets contenant P(i,j). Cet ensemble peut être de la forme {(i,j), (i+1,j), ..., (i+s-1,j)} ou {(i,j), (i-1,j), ..., (i-s+1,j)}. Nous appelerons cette suite la **section verticale** de la ligne de portée en j.
i_init = centre de la section verticale en j /* pour la recherche de la section en j+1 on part du centre de la section en j */

Fin pour

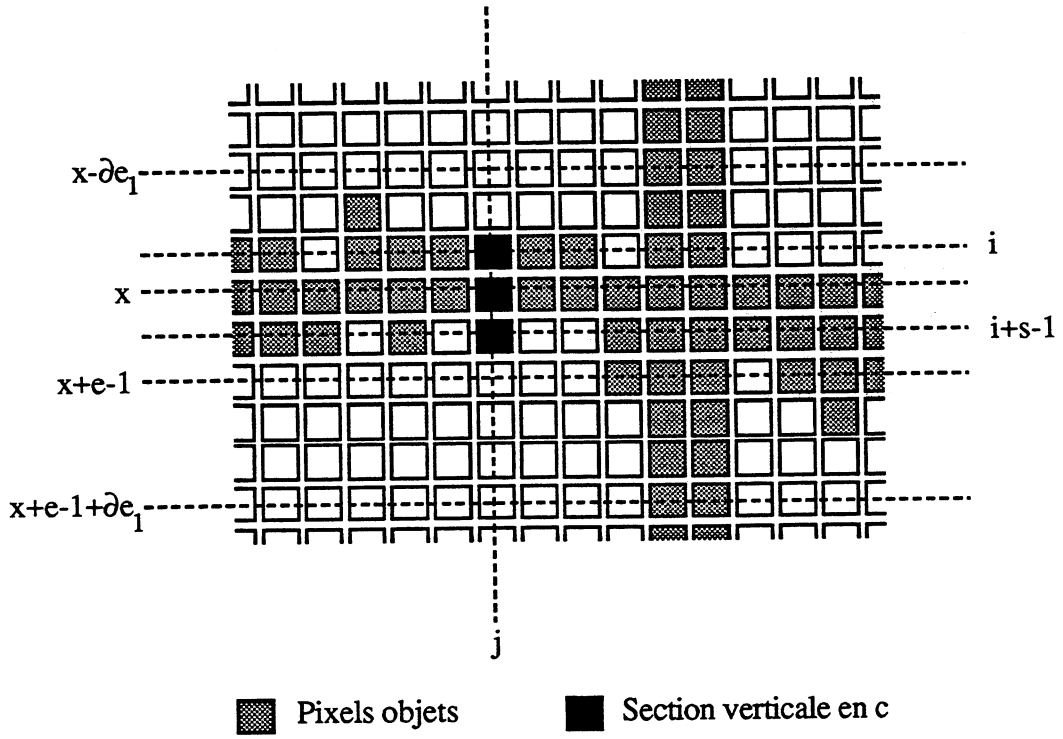


Fig. 3.7. Suivi d'une ligne de portée : recherche de la section verticale en colonne j.

Tout le problème de l'érosion d'une ligne de portée consiste alors à déterminer, pour chaque colonne de l'image, si les pixels de sa section verticale sont nécessaires ou non à la connexité d'un symbole. Une condition nécessaire pour ne pas effacer une section verticale est que celle-ci soit "trop grande". En effet, si la ligne de portée est localement épaisse ($s>e+\partial e_2$, ∂e_2 incertitude sur l'épaisseur, $\partial e_2<\partial e_1$), cela signifie qu'un symbole chevauche cette ligne à cet endroit et donc que l'effacement de la section le déconnecterait. Cependant ce critère d'épaisseur locale n'est pas suffisant puisqu'un symbole dont le trait est tangent à une ligne de portée, n'entraîne aucune variation de l'épaisseur de la ligne à son intersection avec cette dernière. Ce peut être par exemple le cas de la boucle d'un bémol entre deux lignes, ou d'une liaison presque horizontale croisant une portée. Pour pouvoir détecter ces situations, il est alors nécessaire de prendre en compte un contexte plus large que celui de la simple section verticale. Ceci sera à nouveau réalisé grâce à la notion de corde.

3.2.2.2 Erosion selon la distribution des cordes par réseau multi-couche

La méthode que nous proposons ici est inspirée d'un modèle complexe proposé par Chen et Hsu [CHEN 89a] pour réaliser la séparation d'objets dans une image de traits. Ce modèle est fondé sur la continuité de l'orientation de la "plus longue droite en vue", c'est-à-dire de la corde, en chaque point d'un objet mince. Ce principe peut s'appliquer de manière plus simple à notre problème : la notion de corde permet de caractériser facilement les pixels appartenant à l'intersection d'une ligne de portée et d'un symbole. En effet, un tel pixel possède idéalement deux "longues" cordes d'orientation distinctes : une corde horizontale correspondant à la ligne de portée et une autre selon la direction du trait par lequel le symbole croise la ligne (Fig. 3.8).

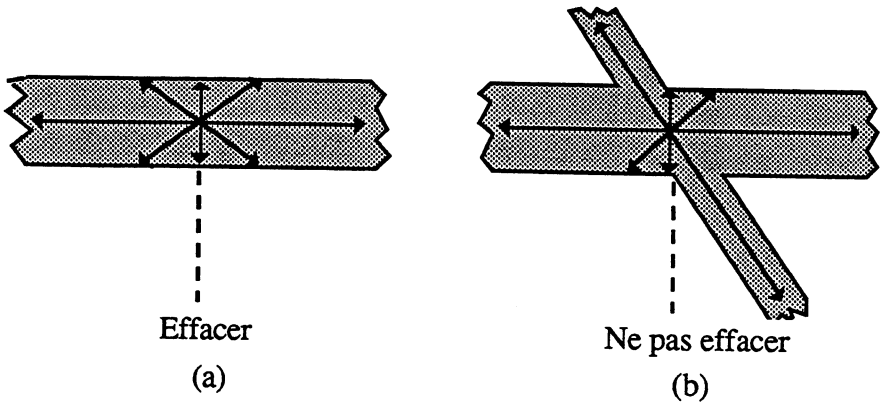


Fig. 3.8. (a) Un pixel que l'on peut effacer ne possède qu'une grande corde. (b) Un pixel que l'on doit conserver en possède au moins deux.

Ceci peut s'observer sur l'histogramme des longueurs de cordes, histogramme obtenu en recherchant les cordes en un point à différentes orientations sur l'intervalle $[-\pi/2,\pi/2[$. Un pixel à effacer possèdera un histogramme ne contenant qu'un pic marqué en 0 dû à la ligne de portée (Fig. 3.9c), alors que l'histogramme d'un pixel "intersection" en contiendra au moins deux : un en 0 et l'autre en θ, θ étant l'orientation du trait du symbole (Fig. 3.9a,b ; l'histogramme (b) étant cyclique, il ne comporte en fait que deux pics puisque les pics en $-\pi/2$ et $\pi/2$ proviennent du même trait vertical).

Une stratégie pourrait donc consister à rechercher le nombre de pics contenus dans l'histogramme ; on n'effacerait alors que les pixels dont l'histogramme ne contient qu'un seul pic autour de l'orientation 0. Cependant une simple analyse de l'histogramme par seuillage est insuffisante, ce dernier pouvant être fortement perturbé par les irrégularités de l'image (Fig. 3.9d) ou lorsqu'il est calculé aux abords d'un symbole. Plûtot que d'essayer de

reconnaître (et comprendre) toutes les situations particulières, nous avons tenté de résoudre ce problème grâce à un apprentissage par l'exemple à l'aide d'un réseau de neurones.

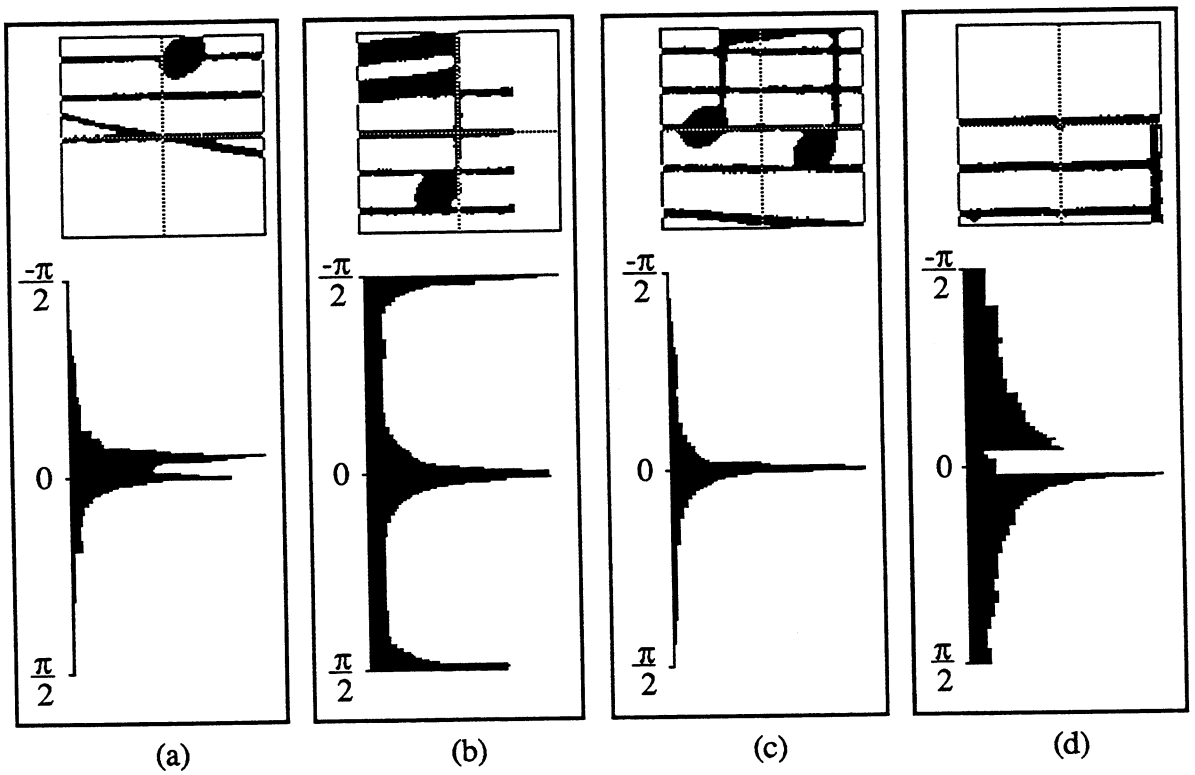


Fig. 3.9. Histogrammes des longueurs de corde. Les histogrammes sont calculés pour le pixel désigné par le réticule.

Classification des histogrammes par réseau multi-couche

Nous avons pour cela utilisé un réseau à 3 couches entraîné classiquement par rétro-propagation du gradient. Les formes soumises au réseau sont ici les valeurs normalisées d'un l'histogramme. La valeur de sortie de l'unique cellule de la dernière couche indique si le pixel dont l'histogramme a été soumis au réseau doit être effacé (valeur positive) ou non (valeur négative).

La nature des primitives utilisées permet aisément de contraindre l'architecture du réseau. Il existe en effet entre les différentes longueurs de corde une organisation topologique évidente : l'orientation. La contrainte consiste donc à construire (§ 2.1.2.3) des cellules cachées dédiées qui vont se focaliser sur les cellules d'entrée correspondant à des cordes d'orientations voisines. Chaque cellule va ainsi jouer un rôle de détecteur de pic pour une orientation particulière.

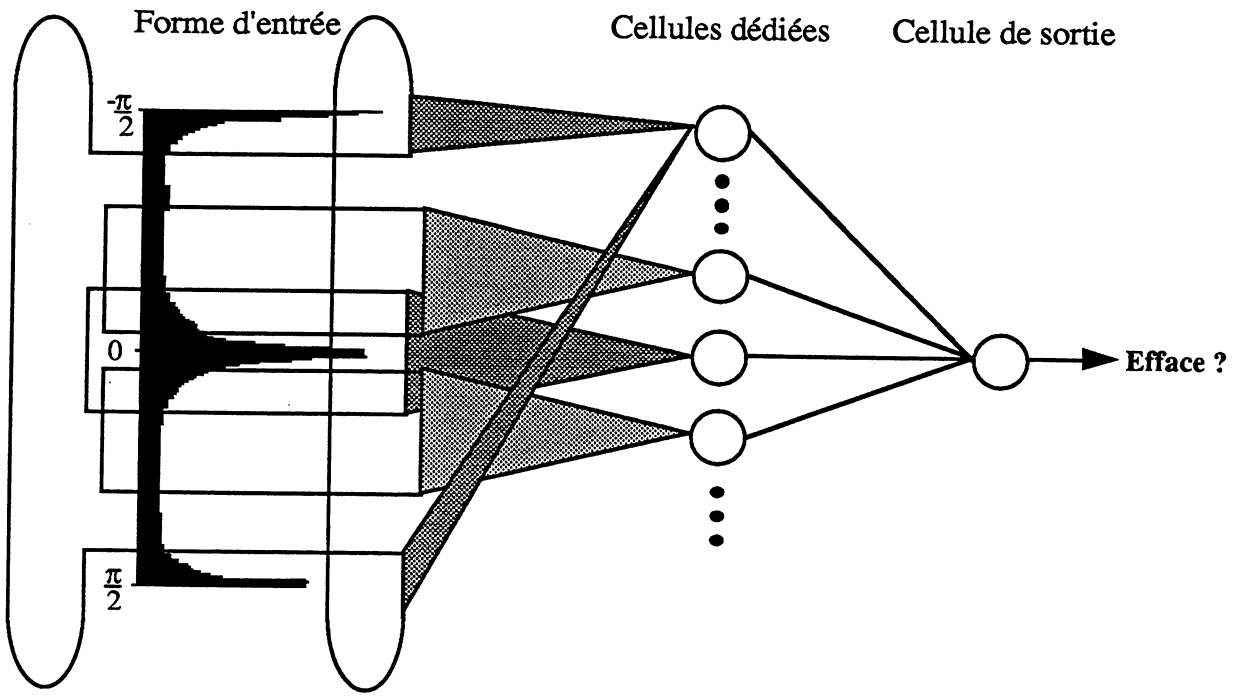


Fig. 3.10. Architecture du réseau pour la classification des histogrammes de longueur de corde.

Pratiquement, l'histogramme des cordes n'est évalué que pour un seul pixel d'une section verticale de portée, le pixel central. En fonction de la classification de ce pixel, toute la section sera effacée ou conservée. Afin de gagner du temps, et pour que le pic correspondant à la portée ne soit pas plus important que les éventuels autres, la recherche des extrémités d'une corde n'est réalisée qu'à l'intérieur d'une petite fenêtre centrée sur le pixel considéré : 51x51 pixels en l'occurrence.

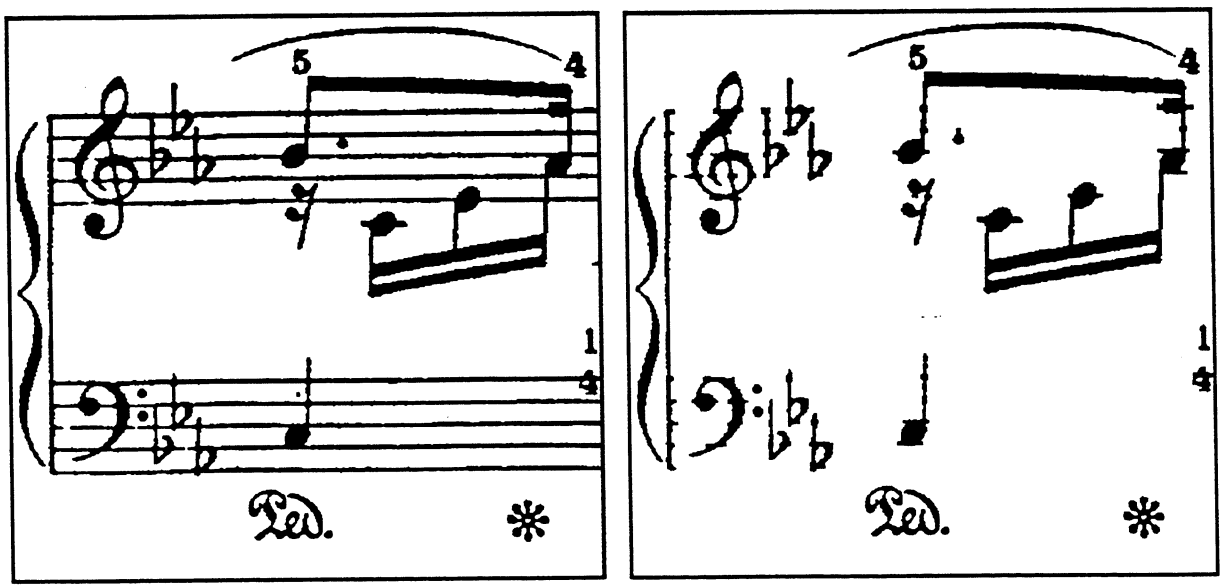
Les cordes sont calculées pour toutes les orientations correspondant aux droites joignant le pixel central à l'un des pixels du bord de la fenêtre, soit 100 orientations distinctes ; le réseau comporte donc 100 cellules d'entrée. Dans l'architecture retenue, la couche cachée contient 20 cellules, une cellule ayant un champ réceptif limité à 10 orientations contiguës ; le champ de chaque cellule recouvre la moitié du champ de la cellule qui la précède (Fig. 3.10).

Pour l'apprentissage, réalisé dans les conditions vues au § 2.1.1, nous avons utilisé un ensemble composé de 228 histogrammes "représentatifs", la classe de chaque histogramme (pixel effaçable ou non) ayant été déterminée "visuellement" en fonction de la segmentation espérée ; le taux d'apprentissage final est de 98%.

Une fois l'apprentisage réalisé, le traitement d'un segment vertical de portée se fait de la manière suivante :

Soit {(i,j), (i+1,j), ..., (i+s-1,j)} la section verticale en colonne j d'une ligne de portée d'épaisseur e
Si s > e + ∂e_2 alors conserver la section verticale
Sinon
  Calculer l'histogramme des cordes en P(i+s/2,j)
  Soumettre cet histogramme au réseau
  Si la sortie du réseau indique que P est effaçable alors
    Mettre à jour, en fonction de la section verticale, les statistiques concernant la ligne de portée courante
    Effacer la section verticale
  Sinon conserver la section

Les statistiques concernant la ligne de portée en cours de traitement sont d'une part l'épaisseur de la ligne (c'est-à-dire la moyenne de la longueur des sections verticales effacées, valeur plus précise que l'épaisseur e mesurée sur l'histogramme de la projection horizontale) et d'autre part les équations aux moindres carrés des bords haut et bas de la ligne (Fig. 3.12e). Ces deux équations doivent évidemment être conservées afin de pouvoir déterminer par la suite l'information "hauteur" attachée à certains symboles musicaux.



(a) (b)

Fig. 3.11. (a) Image 512x512. (b) Segmentation par réseau multi-couche.

On peut constater (Fig. 3.11a,b et 3.12a,d) que la segmentation ainsi obtenue préserve bien la connexité des symboles, mais en perturbe la forme, de petits morceaux de portée restant "collés" aux objets. Ces déformations pourront toutefois être éliminées lors de la phase d'extraction d'indices.

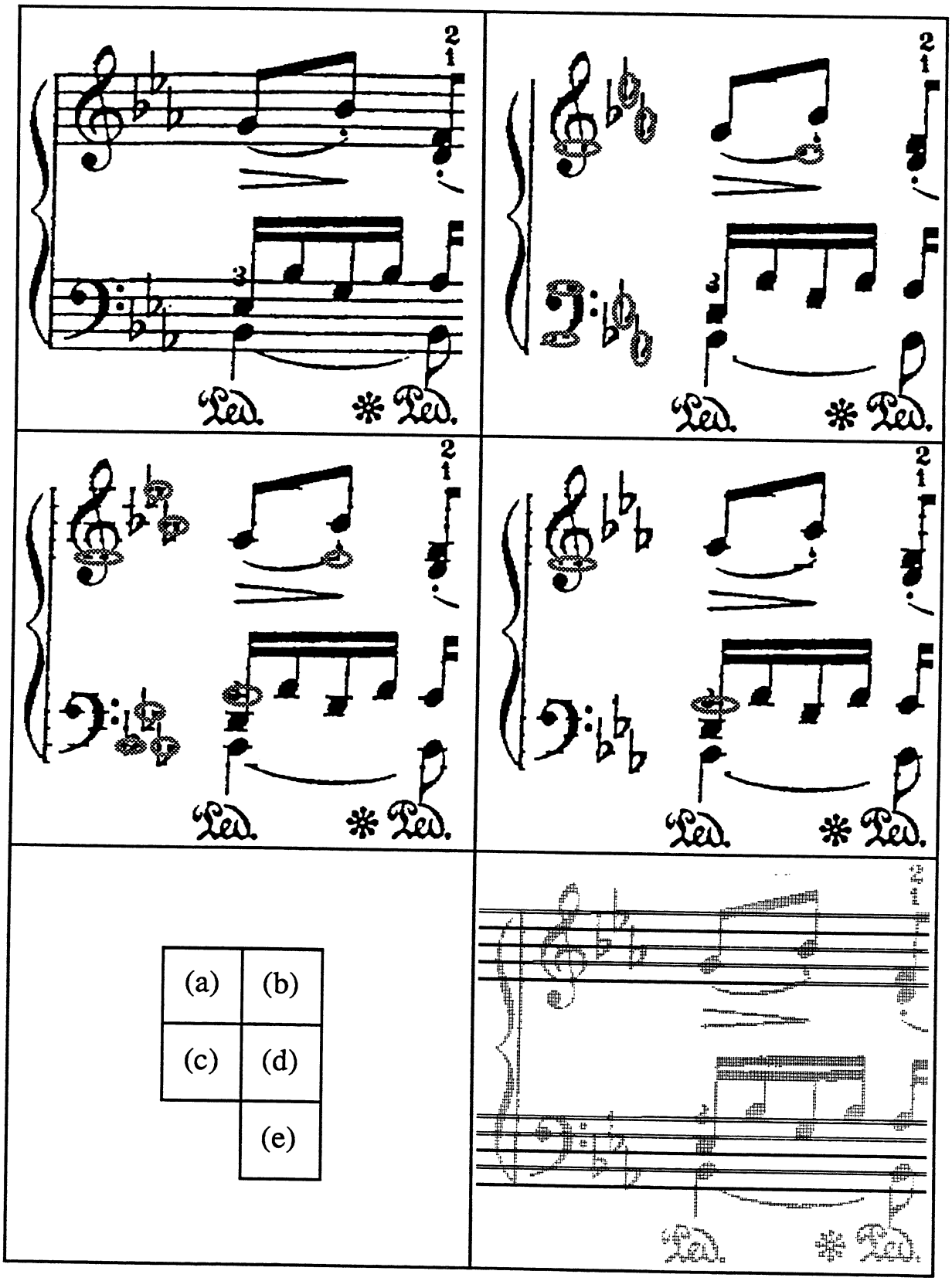


Fig. 3.12. Image 512x512 (a) et sa segmentation par : le critère "d'épaisseur locale" (b) l'analyse de l'histogramme (c) et le réseau (d). Les erreurs sont cerclées (grisé). (e) Droites des moindres carrés obtenues dans le cas de (d).

La segmentation par réseau multi-couche s'avère plus efficace (Fig. 3.12d) que la méthode reposant sur le seul critère d'épaisseur locale (Fig. 3.12b) ou que celle recherchant les plus longues cordes par seuillage de l'histogramme (Fig. 3.12c). Ce gain de qualité s'accompagne toutefois d'une augmentation du temps de calcul relativement importante (Table 3.2).

Table 3.2. Temps CPU pour la segmentation d'une image 512x512, l'image utilisée contenant 10 lignes de portée.

Critère de l'épaisseur locale	Seuillage de l'histogramme des cordes	Réseau multi-couche
1 s	2 mn 59 s	3 mn 30 s

Il faut également noter que nous n'avons résolu ici qu'une partie des problèmes de segmentation propres aux partitions. Sur les documents graphiquement complexes, les symboles peuvent être connectés non seulement par les portées, mais également par eux-mêmes. Ainsi, il arrive fréquemment qu'une altération soit connectée à la note dont elle modifie la hauteur, ou qu'une liaison connecte les notes qu'elle relie. En dehors de l'érosion des portées, il reste donc à développer des méthodes permettant des traiter ces cas parfois complexes.

3.3 RECONNAISSANCE DES SYMBOLES MUSICAUX

Les symboles musicaux ayant été isolés par la segmentation, leurs positions respectives dans l'image (rectangle d'encadrement) sont calculées par un algorithme d'étiquetage des composantes 8-connexes (on pourra trouver dans [MONTANVERT 87] la description d'un tel algorithme). On supprime au passage les objets dont la surface est trop faible pour en faire des symboles de la partition (la surface minimale est évaluée au carré de la moitié de l'intervalle inter-portées). La reconnaissance proprement dite peut alors s'opérer, en deux étapes : notes puis "autres symboles".

3.3.1 Reconnaissance des notes

Si l'on excepte les rondes (qui appartiennent en fait à la catégorie des signes symboliques du fait de l'invariance de leur forme), on peut discriminer les notes des autres symboles grâce à deux primitives caractéristiques : l'ellipse qui compose la tête de chaque note et le segment de

droite qui compose la queue de celle-ci. La recherche de ces deux primitives constitue donc un schéma simple mais robuste pour l'identification d'un tel signe (Fig. 3.13). Elle intervient durant ce que nous avons appelé la phase d'extraction d'indices structurels.

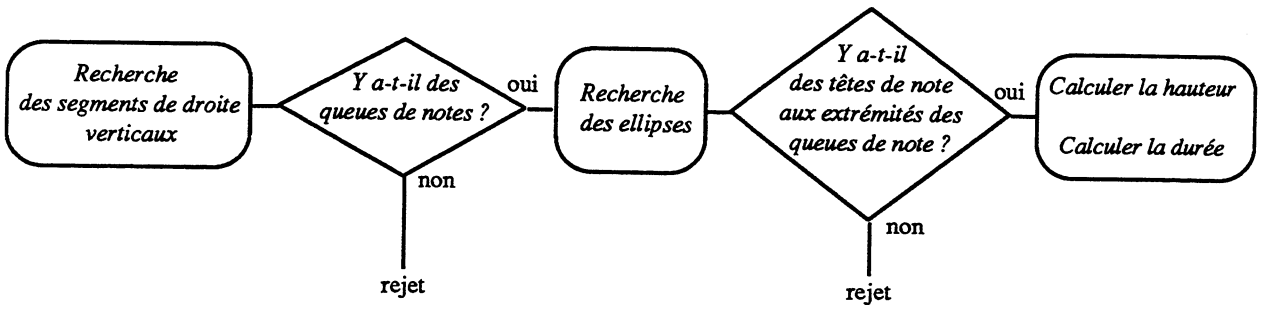


Fig. 3.13. Principe de la reconnaissance des notes.

3.3.1.1 Recherche des queues de note dans le graphe du squelette

Une queue de note est un segment de droite vertical dont la longueur minimale est de deux fois l'intervalle séparant deux lignes de portée [BEASSE 87]. L'étape précédente nous permettant d'évaluer cette longueur, nous disposons donc d'un critère pour la détection des queues.

La recherche d'un segment de droite dans un objet filiforme peut être réalisée aisément sur son squelette. Les objets sont donc tout d'abord amincis par l'un des nombreux algorithmes usuels de squelettisation (nous avons expérimenté [ZHANG 84] et [CHEN 89b] avec des résultats équivalents en termes de taux de classification *in fine*). Le squelette doit ensuite être décomposé en segments. Pour cela, la nature de chaque pixel doit être déterminée : *extrémité* d'un segment, *jonction* de plusieurs segments ou simple point *chemin* permettant de connecter les extrémités d'un segment.

Cette classification est en général réalisée au vu du nombre de pixels objets dans le 8-voisinage, les extrémités ne possédant qu'un seul 8-voisin, les pixels chemins en possédant 2 et les jonctions 3 ou plus. Ceci suppose cependant que le squelette traité soit parfaitement 8-connexe (c'est-à-dire qu'il ne contienne que des points indispensables à la 8-connexité), ce qui n'est souvent pas le cas. Une coûteuse étape supplémentaire de "nettoyage" du squelette [HOLT 87] ou de fusion des jonctions voisines est alors nécessaire.

Pour éviter cela, nous avons remarqué qu'il était possible de déterminer directement la nature d'un pixel non plus en fonction de son nombre de voisins, mais en fonction du nombre de composantes 4-connexes dans son 8-voisinage. En effet, dans ce voisinage, deux pixels qui

sont eux-mêmes 4-voisins appartiennent toujours à un même segment, hormis dans certains cas particuliers (cf. les ensembles irréductibles [ECKHARDT 88]) que l'on doit traiter spécifiquement. Un pixel est donc étiqueté *extrémité*, *chemin* ou *jonction* si le nombre de composantes 4-connexes dans son 8-voisinage est, respectivement, égal à un, égal à deux ou supérieur à deux (Fig. 3.14a). Dans notre implémentation, seuls deux cas particuliers, que l'on peut réellement rencontrer sur le squelette d'un objet, échappent à cette règle : ce sont les configurations de pixels en étoile à 4 ou 8 branches (Fig. 3.14b,c).

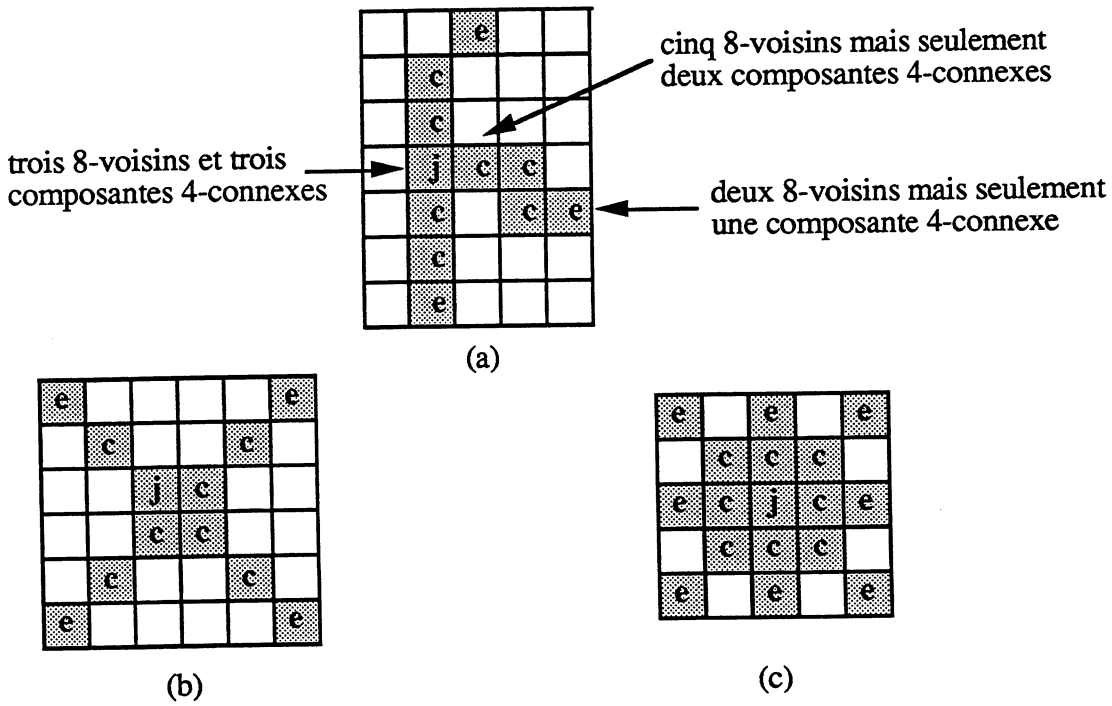


Fig. 3.14. (a) Exemple d'étiquetage des pixels du squelette : extrémité, jonction et chemin. Les jonctions des étoiles à 4 (b) et 8 (c) branches sont des cas particuliers.

Une fois les pixels du squelette étiquetés, les segments sont construits, en partant des extrémités et des jonctions, par un suivi de chemin pouvant être 4-connexe en certains endroits et 8 connexes en d'autres ; chaque fois qu'un pixel *chemin* est rencontré en parcourant un segment, il est marqué et ne peut dès lors plus être utilisé pour la construction d'un autre segment.

Un graphe planaire est ainsi construit dont les sommets sont les extrémités et les jonctions du squelette. Un arc entre deux sommets traduit la présence d'un segment reliant les deux pixels correspondants et est valué par la chaîne de Freeman décrivant ce segment.

Ce graphe est ensuite ébarbulé en supprimant les petits arcs (seuillage sur la longueur de

la chaîne de Freeman) joignant une jonction à une extrémité. On élimine ainsi les fragments de portée restés collés aux objets lors de la segmentation et qui sont à l'origine de la plupart des barbules supprimées.

Le squelette est alors découpé en segments qui ne sont pas linéaires. Pour obtenir de tels segments, une décomposition polygonale est appliquée. Nous avons utilisé, avec des résultats similaires en termes de reconnaissance, l'algorithme du découpage récursif [RAMER 72] et l'algorithme de Roberts [LUX 85]. L'approximation polygonale introduit de nouveaux sommets dans le graphe qui correspondent aux points de courbure importante.

Dans le graphe final, chaque arc décrit alors un segment quasi-linéaire (Fig. 3.15).

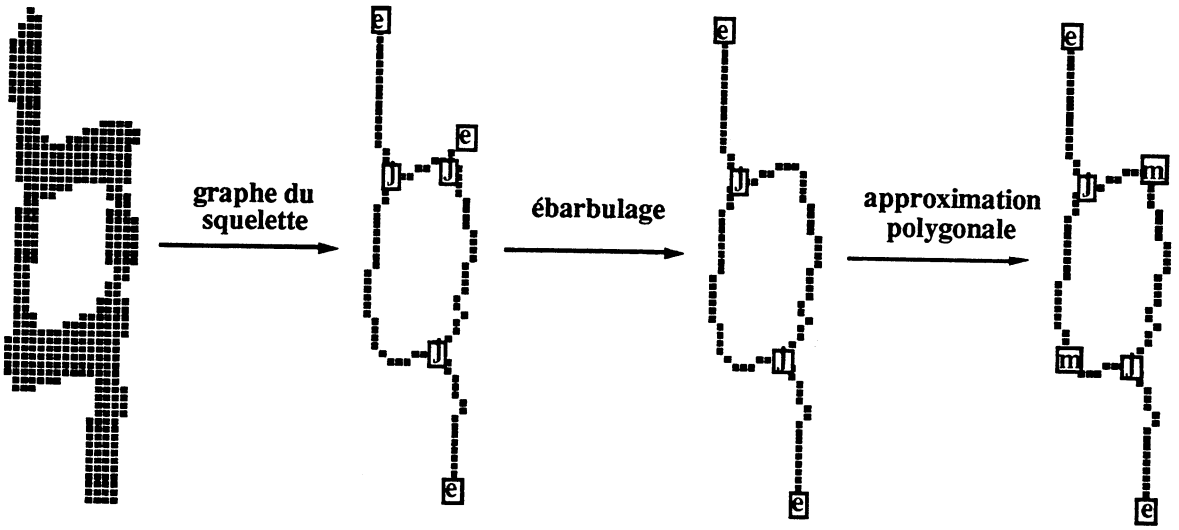


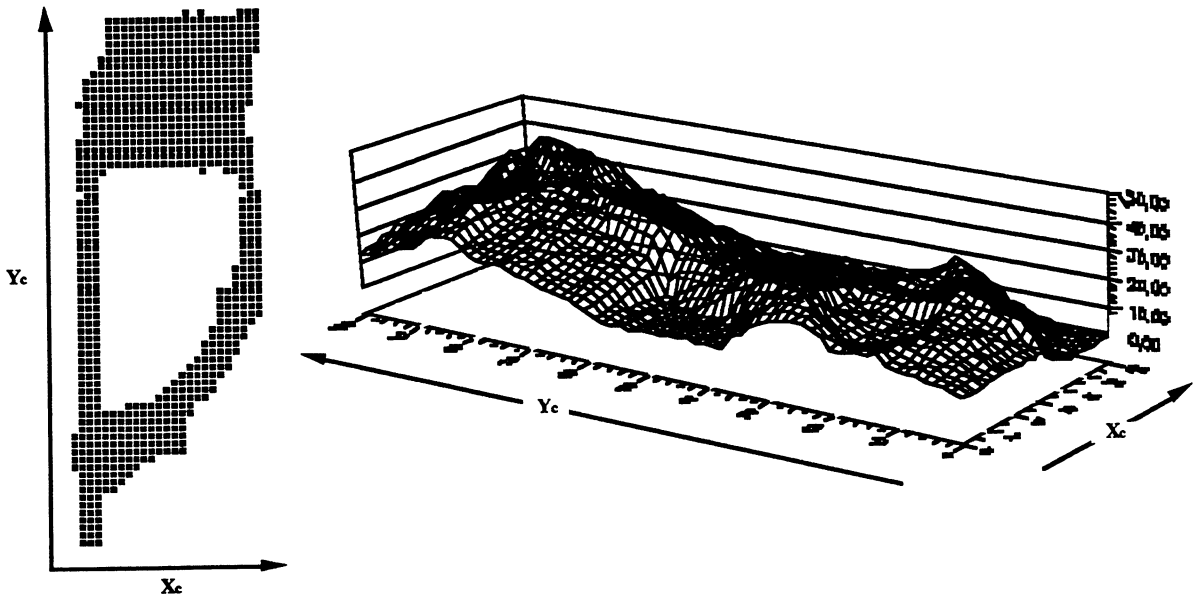
Fig. 3.15. Extraction des segments de droite d'un objet. Un sommet du graphe final représente une extrémité, une jonction ou un point maximum local de courbure.

Une queue de note est alors détectée pour toute paire de sommets adjacents du graphe telle que l'angle de la droite reliant ses sommets est proche de $\pi/2$ et la distance entre ses sommets est supérieure à deux fois l'intervalle moyen inter-portée. Si une telle paire n'est pas trouvée dans le graphe d'un objet, celui-ci est rejeté de l'étape "reconnaissance de note" ; dans le cas contraire, on tente alors de valider les queues détectées en recherchant les têtes de note auxquelles elles doivent être connectées.

3.3.1.2 Recherche des têtes de note par mise en correspondance d'ellipses.

Une tête de note est une ellipse, pleine ou creuse, attachée à la queue de note et dont la position sur la portée indique la hauteur de la note. Nous avons à nouveau tenté [MARTIN 89] de réaliser la détection d'ellipses à l'aide de la transformation de Hough, transformation qui a

souvent été employée à cet effet. Les ellipses que l'on doit détecter ici sont en effet bruitées par la segmentation (contour déformé voire même déconnecté pour les blanches) et l'emploi de la transformation de Hough trouve alors une justification dans sa résistance au bruit. On retrouve cependant les mêmes obstacles que dans le cas de la recherche des portées : temps de calculs importants et difficulté de l'analyse de l'espace des votes (Fig. 3.16).



(a) (b)

Fig. 3.16. (a) Un objet contenant une tête de note. (b) L'espace des votes qui est ici l'espace à deux dimensions des centres d'ellipse, espace homotope au rectangle exinscrit (les autres paramètres de l'ellipse sont figés).

Nous avons finalement opté pour la méthode plus simple et plus rapide de mise en correspondance (matching). Pour chaque queue de note trouvée, un masque (binaire) contenant une ellipse est positionné sur le symbole dans les deux zones où la tête de note est susceptible de se trouver : à gauche de l'extrémité inférieure de la queue ou à droite de l'extrémité supérieure (Fig. 3.17). Le score de la meilleure mise en correspondance (i.e. le nombre de pixels objets du symbole coïncidant avec les pixels objets du masque) indique alors, s'il est suffisant, la présence d'une tête de note.

Les paramètres de l'ellipse recherchée sont calculés en fonction de "l'échelle" de la partition, c'est-à-dire de l'intervalle moyen inter-portée (Fig. 3.17).

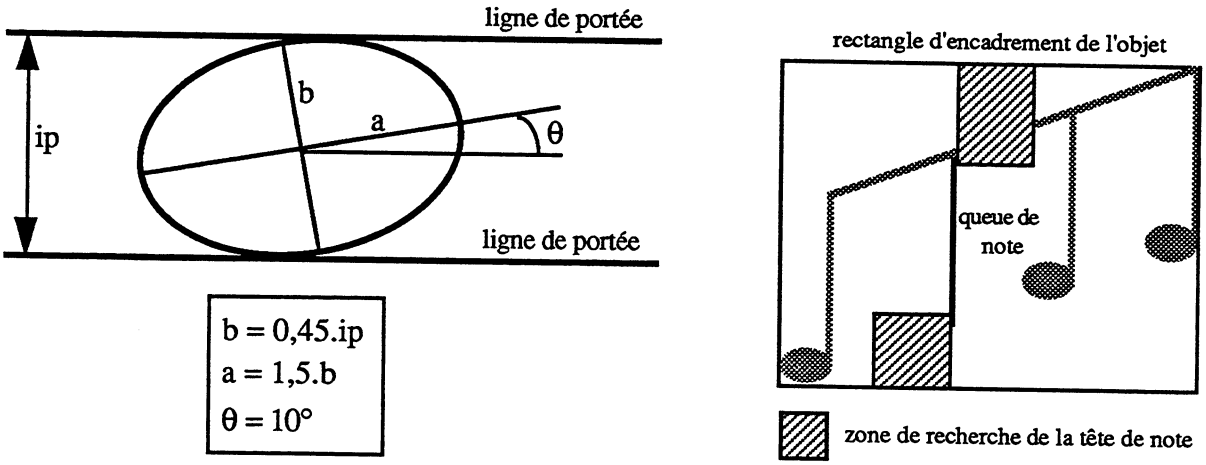


Fig. 3.17. Les paramètres de l'ellipse, comme la taille de la zone de recherche, sont définis proportionnellement à l'intervalle moyen inter-portée calculé lors de la segmentation. Les coefficients de proportionnalité, comme l'inclinaison de l'ellipse, ont été déterminés de manière empirique.

Si l'on n'arrive pas à déceler une ellipse pour chacune des queues détectée sur l'objet, celui-ci est rejeté : il n'est pas de type note. Dans le cas contraire, il faut alors déterminer les deux attributs attachés à chacune de ses notes : hauteur et durée.

3.3.1.3 Hauteur et durée

Hauteur

Le calcul de la hauteur est réalisé de manière très simple en déterminant la position du centre d'une ellipse par rapport aux lignes de portée, lignes dont les équations ont été mises à jour durant la segmentation. La dimension verticale est décomposée en zones de tailles égales, une zone correspondant soit au cas où la note est disposée entre deux lignes de portée, soit au cas où elle chevauche une de ces lignes (Fig. 3.18). En fonction de la zone dans laquelle "tombe" le centre de l'ellipse, la hauteur de la note est déterminée.

Pour les notes se trouvant au-dessus (resp. en-dessous) de la portée, on extrapole la position des lignes de portée "imaginaires" depuis la ligne supérieure (resp. inférieure) et d'après l'épaisseur et l'intervalle inter-portée moyens. Naturellement, la hauteur que l'on calcule ainsi n'est que relative. Sa valeur absolue, qui dépend de plusieurs autres paramètres (clé, armure, altérations), ne peut être obtenue qu'après une analyse du contexte qui ne relève pas de la reconnaissance "bas-niveau".

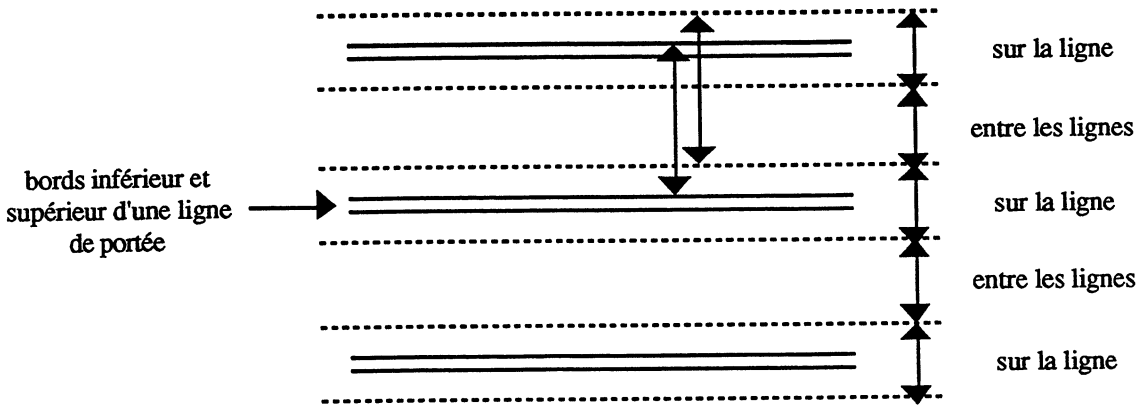


Fig. 3.18. Pour déterminer la hauteur d'une note, la portée est partitionnée en zones de tailles égales autour et entre les lignes.

Durée

La durée d'une note dépend de la nature de l'ellipse composant la tête de note (pleine ou creuse) et du nombre de crochets pour les notes simples, ou de barres pour les groupes de notes, attachés à la queue.

La note est reconnue comme étant une blanche si l'on détecte la présence de pixels fond autour du centre de son ellipse.

Lorsque l'ellipse est pleine, la durée de la note doit être calculée d'après le nombre de crochets (note simple) ou de barres (groupe) de cette note. Nous avons envisagé d'extraire cette information du graphe du squelette, mais la recherche des segments du graphe correspondant à ces primitives s'est avérée plus délicate que dans le cas des queues de note. Nous avons donc employé une technique plus simple, celle des *sondes* [PAKKER 85].

Une sonde est un procédé consistant à compter le nombre de transitions objet/fond rencontrées lorsque l'on chemine d'un point de l'image à un autre. Ceci permet de compter le nombre de traits traversés et donc ici, en utilisant les sondes adéquates, le nombre de crochets ou de barres. Plusieurs cas sont à distinguer (Fig. 3.19) :

- Si l'objet ne comporte qu'une note, on doit utiliser une sonde horizontale ; sa valeur sera celle du nombre de crochets croisés plus un (on traverse également la queue de la note).

- Si l'objet comporte plusieurs notes, on utilisera une seule sonde pour chacune des deux notes à l'extérieur du groupe, alors que deux sondes seront nécessaires pour chaque note à l'intérieur du groupe. Dans ce cas on doit en effet compter les barres arrivant et partant de la note, le maximum des deux permettant de déterminer la durée. Dans les deux cas, la sonde doit être

verticale ; on partira d'un point à droite ou à gauche du centre de la queue pour se diriger vers le bas si la tête de note est en haut de la queue, et vers le haut si la tête est en bas.

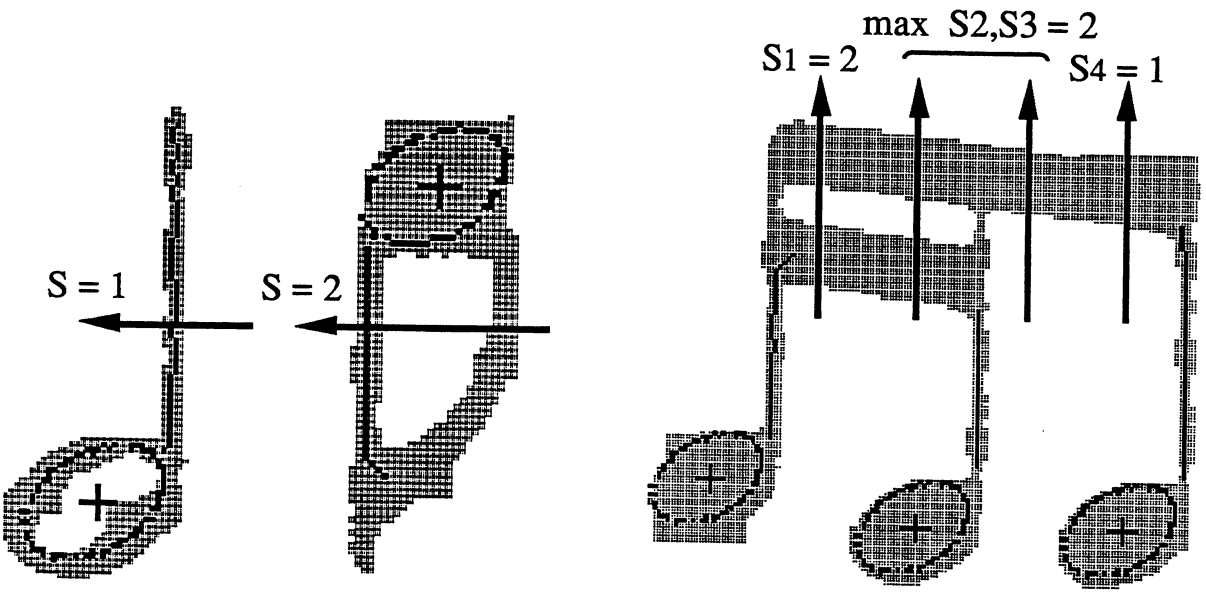


Fig. 3.19. Pour différents objets (en gris), les segments verticaux détectés sur le squelette et les ellipses associées (en noir), ainsi que les sondes utilisées pour compter le nombre de crochets ou de barres (les flèches).

3.3.1.4 Résultats

Nous avons testé cette première phase de classification sur quatre partitions de complexité graphique croissante (voir annexe). Malgré la simplicité des techniques et des contrôles mis en œuvre, sa robustesse s'avère assez satisfaisante (table 3.3) tout en étant relativement rapide (table 3.4). Pour les trois partitions les plus simples, les objets de type note sont pratiquement tous détectés, et leurs attributs de hauteur et durée sont presque tous correctement déterminés.

Dans le cas de la dernière partition, les performances chutent sensiblement. Ceci est plus dû à la segmentation qu'à la classification proprement dite. Les erreurs prroviennent en effet de connexions entre une note et son altération ou une liaison, ce qui se traduit par un graphe du squelette perturbé sur lequel la détection des queues de note ne peut plus être réalisée. Tout le processus de classification est alors mis en échec. Comme nous l'avons dit auparavant, c'est au niveau de la segmentation qu'il faudra chercher remède à ce problème.

Table 3.3. Performances de la classification primaire.

Partition	Notes détectées	Hauteur			Durée		Notes non détectées
		Juste	Fausse d'un Ton	Fausse	Juste	Fausse	
#1	203	203 (100%)	-	-	202 (99,5%)	1 (0,5%)	1 (0,5%)
#2	142	141 (99,3%)	1 (0,7%)	-	140 (98,6%)	2 (1,4%)	2 (1,4%)
#3	194	194 (100%)	-	-	191 (98,5%)	3 (1,5%)	7 (3,5%)
#4	218	196 (89,9%)	2 (0,9%)	20 (9,2%)	195 (89,5%)	23 (10,5%)	4 (1,8%)
total	757	734 (97%)	3 (0,4%)	20 (2,6%)	728 (96,2%)	29 (3,8%)	14 (1,8%)

Table 3.4. Temps CPU pour la recherche des composantes connexes, l'extraction d'indices et la classification primaire d'une image 512x512. La charge de calcul la plus importante provient de l'extraction d'indices et en particulier de la squelettisation.

Recherche des composantes connexes	Extraction d'indices (squelettisation + graphe du squelette)	Classification primaire
9 s	22 s	11 s

Les erreurs rapportées dans la dernière colonne de la table 3.3 concernent des objets de type "note" n'ayant pas été reconnus comme tels. Il faut souligner que le cas inverse ne s'est jamais produit : un signe symbolique de la partition ne s'est jamais vu assimilé à une note, ce qui confirme le fort pouvoir de discrimination des primitives "queue et tête". Les objets que l'on a ainsi rejetés peuvent alors être traités par le deuxième niveau de classification.

3.3.2 Reconnaissance des symboles

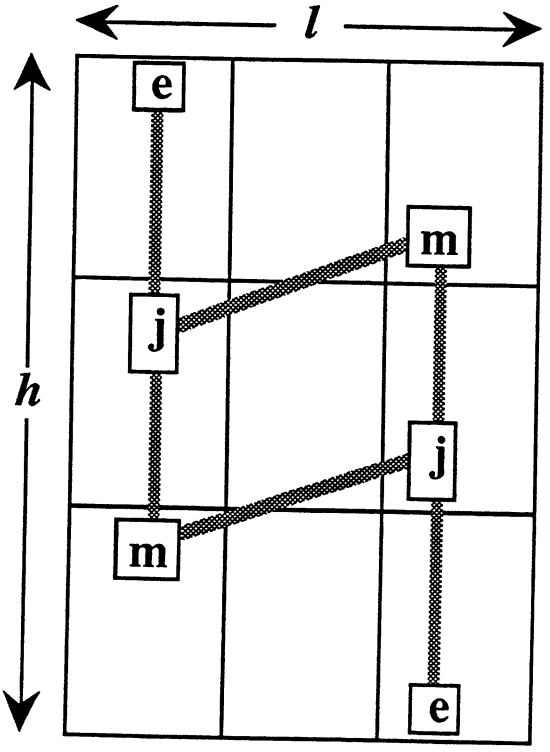
Pour la reconnaissance de ces symboles, nous avons développé deux approches. La première repose sur les indices structurels disponibles à l'issue de la classification primaire, la seconde utilisant un autre type de primitives, de plus bas-niveau : l'imagette normalisée. Toutes deux ont donné lieu à la comparaison de plusieurs méthodes de classification parmi celles évoquées lors des deux chapitres précédents.

3.3.2.1 Classification par codage du graphe du squelette

Codage du graphe

Pour que l'information structurelle symbolique contenue dans le graphe du squelette puisse être manipulée par un réseau de neurones, il est nécessaire tout d'abord de la coder de manière numérique. Le codage que nous avons employé est un codage binaire permettant de représenter la présence/absence de primitives dans le graphe.

Pour cela, le rectangle d'encadrement du squelette est partitionné de manière arbitraire en $h \times l$ fenêtres régulières. Une variable binaire est associée à chaque couple (NP,F) où NP est une nature de point du graphe (extrémité, jonction, ou maximum de courbure) et F une fenêtre. La valeur de cette variable indique la présence ou l'absence d'un pixel de type NP dans la fenêtre F. De la même manière, on associe à chaque couple (F_1,F_2) une variable détectant l'existence d'un segment dont l'une des extrémités se trouve dans F_1 et l'autre dans F_2. Le graphe du squelette est ainsi codé par un vecteur binaire (Fig. 3.20).



E(i,j) = 1 si au moins une extrémité dans (i,j)
-1 sinon.

J(i,j) = 1 si au moins une jonction dans (i,j)
-1 sinon.

M(i,j) = 1 si au moins une "coupure" dans (i,j)
-1 sinon

S((i,j),(p,q)) = 1 si au moins un segment entre (i,j) et (p,q)
-1 sinon

[E(1,1), ..., E(h,l), M(1,1), ..., M(h,l), J (1,1), ..., J (h,l),
S((1,1),(1,1)), S((1,1),(1,2)), ..., S((h,l), (h,l))]

vecteur de $\{-1,1\}^{\left(3\,h.l + \frac{h.l\,(h.l+1)}{2}\right)}$

Fig. 3.20. Codage binaire du graphe du squelette.

Ce codage, invariant aux changements d'échelle, s'est avéré relativement insensible aux petites variations de graphisme pouvant intervenir d'une partition à l'autre, mais néanmoins

suffisamment discriminant pour traiter le jeu de symboles que nous nous sommes pour l'instant fixé. Le meilleur découpage *a posteriori* est d'une taille de 3x3 fenêtres, c'est-à-dire que chaque forme est codée par un vecteur de 72 composantes binaires. On pourra dans un deuxième temps déterminer les fenêtres les plus pertinentes par une étude sur la distribution des primitives à l'intérieur du rectangle d'encadrement en fonction de chaque classe d'objet [KAHAN 87].

Résultats

Nous avons tenté de reconnaître 28 types de symboles (clés, altérations, silences, liaisons, signes de nuances, indications de mesure, chiffres de doigté, ...). La petite bibliothèque d'environ 700 symboles dont nous disposons, issue des quatre partitions citées précédemment, a été divisée en un ensemble d'apprentissage de seulement 200 éléments et un ensemble de test de 500 éléments.

Avec ce codage binaire, nous avons comparé quatre types de classification : une première aux trois plus proches voisins selon la distance de Hamming, une autre par réseaux multi-couches et une dernière par l'arbre de décision binaire décrit au § 2.1.2.5. Pour les réseaux multi-couches, deux architectures on été utilisées : la première à une couche cachée totalement connectée, la seconde déduite de l'arbre d'apprentissage selon la méthode proposée au § 2.1.2.5. La table 3.5 rend compte des résultats obtenus ; afin d'obtenir des résultats comparables, aucune procédure de rejet n'a été mise en œuvre.

Table 3.5. Performances de la classification de 500 symboles opérant sur le codage binaire du graphe du squelette.

Classifieur	Taux de reconnaissance	Temps CPU de reconnaissance de l'ensemble de test	Nombre de poids
3 plus proches voisins distance de Hamming	94,7 %	3 mn 09s	-
Réseau à une couche cachée totalement connectée	93,2 %	24 s	2755
Réseau à architecture déduite de l'arbre + 10 cellules tot. conn.	92,6 %	18 s	1817
Réseau à architecture déduite de l'arbre	85,2 %	14 s	807
Arbre binaire	73,6 %	0 s 07	-

Comme on peut le voir, les réseaux multi-couches n'arrivent pas à égaler le taux de reconnaissance d'une conventionnelle classification aux plus proches voisins. Ils s'en approchent toutefois tout en étant beaucoup plus rapides en phase de reconnaissance. Parmi les différentes architectures à une couche totalement connectée (TC), celle fournissant le meilleur

taux de classification, tout en étant la plus compacte, comporte 27 cellules cachées. L'architecture déduite de l'arbre de décision comporte pour sa part 90 cellules cachées (28 formes normales disjonctives comprenant 90 conjonctions, voir table 3.6) mais seulement 817 poids au lieu de 1817. Si son taux de classification s'avère meilleur que celui obtenu en utilisant directement l'arbre binaire, il reste tout de même éloigné de celui du réseau TC. Toutefois, en rajoutant 10 cellules cachées totalement connectées, on obtient un réseau aux performances proches de celui du réseau TC mais comportant encore 34 % de poids en moins et étant donc plus rapide que celui-ci : cette architecture semble réaliser un bon compromis.

Table 3.6. Les Formes Normales Disjonctives générées par l'arbre de décision binaire sur l'ensemble d'apprentissage en fonction des 72 primitives du codage du graphe du squelette.

```
Classe  1 = ¬F1.¬F2.¬F4.F15.¬F19.F42
          + ¬F1.¬F2.¬F4.F11.F15.F19.F42

Classe  2 = ¬F2.F4.F5.F8.F14.¬F24.¬F42
          + ¬F1.F2.¬F6.F8.F10.¬F37.¬F42

Classe  3 = F1.¬F2.¬F5.¬F7.F11.¬F14.¬F42
          + F1.¬F2.¬F5.¬F7.F11.F14.¬F18.¬F42
          + F2.¬F5.¬F8.F16.F28.¬F37.¬F42
          + F1.F2.¬F12.F28.F37.¬F42.F52
          + F1.F2.¬F7.F12.¬F14.F37.¬F42

Classe  4 = F1.¬F2.¬F5.F7.F18.¬F42
          + F1.F2.F7.F12.F37.¬F42
          + F1.¬F2.¬F4.F15.F19.F42
          + F1.¬F2.F4.F15.F19.F42

Classe  5 = F1.¬F2.¬F5.¬F7.F11.F14.F18.¬F42
          + F1.F2.¬F7.F12.F14.F37.¬F42
          + F1.¬F2.¬F4.¬F15.F42
          + F1.F2.¬F5.¬F7.¬F26.F42
          + F1.F2.¬F5.¬F16.F26.F42

Classe  6 = ¬F2.F4.¬F15.F17.F20.¬F26.F42
          + ¬F1.¬F2.F4.F5.F15.F19.F42
          + F1.F2.F5.¬F7.¬F8.F11.¬F26.F42
          + F1.F2.F5.¬F7.F8.¬F26.F42
          + F1.F2.F5.¬F8.F26.F42

Classe  7 = F1.¬F2.¬F4.F15.¬F19.F42
          + F2.F7.¬F17.¬F20.¬F26.F42
          + F1.F2.¬F5.F7.F17.¬F26.F42

Classe  8 = ¬F2.F4.F7.F15.¬F19.F42

Classe  9 = ¬F1.¬F2.¬F5.F7.F11.¬F42
          + ¬F1.F2.¬F4.¬F5.¬F8.¬F16.¬F37.¬F42

Classe 10 = F1.F2.F5.¬F7.¬F8.¬F11.¬F26.F42
          + F1.F2.F5.F7.F17.¬F26.F42

Classe 11 = ¬F2.¬F4.F5.¬F7.¬F8.¬F42
          + F2.¬F4.F5.¬F8.¬F12.¬F37.¬F42
          + ¬F1.F2.¬F6.F8.¬F10.¬F37.¬F42
          + ¬F1.F2.¬F4.¬F8.¬F15.F37.¬F42
          + ¬F1.F2.F8.¬F16.¬F18.F19.F37.¬F42
          + ¬F1.F2.F8.F16.F37.¬F42
          + ¬F1.¬F2.F4.¬F5.F15.F19.F42

Classe 12 = ¬F2.¬F5.¬F7.¬F8.¬F11.¬F20.¬F42

Classe 13 = ¬F2.¬F5.¬F7.F8.¬F11.¬F42
          + ¬F1.¬F2.¬F5.¬F7.F11.¬F42
          + ¬F2.¬F4.F5.¬F7.F8.F14.F15.¬F42
          + ¬F2.F4.F5.F8.¬F14.¬F42
          + F1.F2.F5.F8.¬F37.¬F42
          + ¬F2.F4.¬F15.F17.¬F20.F42
          + ¬F2.F4.¬F15.F17.F20.F26.F42

Classe 14 = F1.¬F2.¬F5.F7.¬F18.¬F42
          + F1.F2.¬F5.F7.¬F8.¬F16.¬F37.¬F42
          + F2.¬F5.¬F8.F16.¬F28.¬F37.¬F42
          + F1.F2.¬F5.F8.¬F37.¬F42
          + F1.F2.¬F12.¬F28.F37.¬F42
          + F1.F2.¬F5.F16.F26.F42

Classe 15 = F1.F2.¬F12.F28.F37.¬F42.¬F52

Classe 16 = ¬F1.F2.F6.F8.¬F37.¬F42

Classe 17 = ¬F2.F4.F5.¬F15.¬F17.F42
          + F2.F7.¬F17.F20.¬F26.F42

Classe 18 = ¬F1.F2.¬F4.¬F7.¬F26.F42

Classe 19 = ¬F1.F2.F4.F8.¬F12.F26.F42

Classe 20 = ¬F2.F4.¬F7.F15.¬F19.F42

Classe 21 = ¬F2.¬F5.¬F7.¬F8.¬F11.F20.¬F42
          + ¬F1.¬F2.¬F5.F7.¬F11.¬F42
          + ¬F2.¬F4.F5.¬F6.F7.¬F8.¬F42
          + ¬F2.F4.F5.¬F6.¬F8.F11.¬F42
          + ¬F1.F2.F4.¬F5.¬F8.¬F16.¬F37.¬F42
          + F1.F2.¬F5.¬F7.¬F8.¬F16.¬F37.¬F42
          + F2.F4.F5.¬F8.¬F37.¬F42
          + ¬F2.F4.¬F5.¬F15.¬F17.F42

Classe 22 = ¬F2.F4.F5.¬F6.¬F8.¬F11.¬F42
          + ¬F2.F4.F5.F8.F14.F24.¬F42
          + F2.¬F4.F5.¬F8.F12.¬F37.¬F42
          + ¬F1.F2.¬F4.¬F8.F15.F37.¬F42
          + ¬F1.¬F2.¬F4.¬F11.F15.F19.F42
          + ¬F1.F2.¬F3.¬F4.¬F7.F26.F42

Classe 23 = ¬F2.¬F4.F5.¬F7.F8.¬F14.¬F42
          + ¬F2.¬F4.F5.¬F7.F8.F14.¬F15.¬F42
          + ¬F1.F2.F8.¬F16.F18.F37.¬F42

Classe 24 = ¬F2.¬F4.F5.F6.F7.¬F8.¬F42
          + ¬F2.¬F4.F5.F7.F8.¬F42
          + ¬F2.F4.F5.F6.¬F8.¬F42
          + ¬F1.F2.F4.¬F8.F37.¬F42
          + ¬F1.F2.F8.¬F16.¬F18.¬F19.F37.¬F42

Classe 25 = ¬F1.¬F2.¬F4.F10.¬F15.F42

Classe 26 = ¬F1.F2.F4.¬F7.¬F26.F42
          + ¬F1.F2.F3.¬F4.F26.F42
          + F1.F2.F5.F8.F26.F42

Classe 27 = ¬F1.F2.F4.¬F8.F26.F42
          + ¬F1.F2.F4.F8.F12.F26.F42

Classe 28 = ¬F1.¬F2.¬F4.¬F10.¬F15.F42
          + ¬F1.F2.F7.F17.¬F26.F42
          + ¬F1.F2.¬F3.¬F4.F7.F26.F42
```

3.3.2.2 Classification selon l'imagette normalisée

L'approche employée ici consiste à travailler sur une représentation proche de l'originale, c'est-à-dire de l'image, afin de pouvoir utiliser un réseau à convolution (§ 2.1.2.4). Pour cela, on construit une imagette dont la taille est normalisée pour tous les objets. Le rectangle d'encadrement de l'objet (et non plus de son squelette) est découpé en $h \times l$ fenêtres de tailles égales et on calcule pour chaque fenêtre le pourcentage de sa surface qui est occupé par des pixels objets (on considère qu'un pixel à cheval sur plusieurs fenêtres est divisible). Le résultat est en quelque sorte un sous-échantillonnage (ou sur-échantillonnage si l'objet est de taille inférieure à $h \times l$)en niveaux de gris de l'objet binaire original (Fig. 3.21). Les composantes du vecteur résultant sont à valeur sur [0,1]. Ce codage, comme le précédent, est invariant aux changements d'échelle.

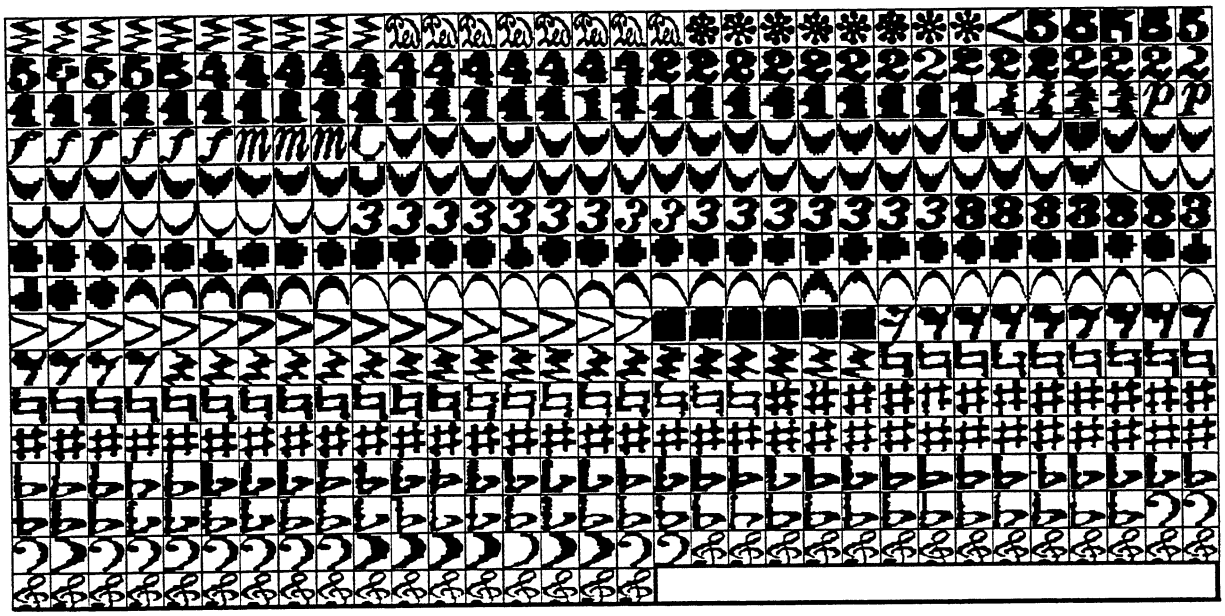


Fig. 3.21. Exemple de symboles représentés par leur imagette normalisée 50x50 que l'on a ici seuillée pour retrouver une image binaire.

Résultats

Trois types de classification utilisant ce codage ont été expérimentés (avec les mêmes ensembles d'apprentissage et de test que précédemment) : un réseau à convolution, une classification aux trois plus proches voisins selon la distance euclidienne et l'arbre de classification (ou réseau d'automates à seuil) que nous avons proposé au § 2.2.3.3. La taille

des imagettes normalisées a été fixée à 10x10, meilleure valeur *a posteriori*. La table 3.7 présente les résultats obtenus.

Table 3.7. Performances de la classification de 500 symboles opérant sur le codage de l'imagette normalisée.

Classifieur	Taux de reconnaissance	Temps CPU de reconnaissance de l'ensemble de test	Nombre de poids
3 plus proches voisins dist. euclidienne	98,8 %	4 mn	-
Réseau à convolution à une couche cachée (3 masques 6x6)	96,3 %	44 s	3839
Arbre de classification	90,9 %	0 s 83	2341

Le réseau à convolution implémente 3 masques à poids partagés de taille 6x6, soit un réseau comprenant 75 cellules cachées, 3839 poids mais seulement 1175 "paramètres libres". Le taux de reconnaissance réalisé s'avère cependant toujours inférieur à celui de la classification aux trois plus proches voisins. L'arbre de classification, correspondant à un réseau contenant 2 couches cachées de 53 et 54 cellules respectivement, est quant à lui assez en retrait. Ceci est sans doute dû, en partie, à la faible taille de l'ensemble d'apprentissage, paramètre plus handicapant pour ce type de méthode que pour les deux autres. Néanmoins sa supériorité en vitesse de reconnaissance la rend potentiellement très intéressante. Nous espérons à l'avenir, dans de meilleures conditions et en mettant en œuvre des processus de développement/élagage de l'arbre, approcher les taux de classification des réseaux à rétro-propagation du gradient.

3.4 CONCLUSION

Nous avons présenté au cours de ce chapitre les premiers éléments d'un système de reconnaissance "bas-niveau" de partitions. Celui-ci fait appel à des techniques "classiques" de reconnaissance des formes combinées à des algorithmes connexionnistes. Comme nous l'avons expliqué auparavant, ce caractère hybride est bien adapté à la nature des documents traités ici (signes iconiques vs signes symboliques).

La première conclusion que l'on peut tirer de cette expérience est qu'il est très facile d'intégrer des réseaux de neurones à un système d'analyse d'images. C'est sans doute là l'une des raisons majeures de leur succès dans ce domaine. On dispose, avec les réseaux multi-couches et la rétro-propagation du gradient en particulier, de "boîtes noires" susceptibles d'être

insérées partout où se pose un problème de transformation d'information que l'on ne sait modéliser et qui ne peut être clairement décrit autrement que par des exemples du comportement attendu. La tentation est grande alors de substituer ce mécanisme à toutes les phases de l'analyse d'images et de laisser ainsi à l'algorithme d'apprentissage le soin de découvrir seul comment passer de l'information bas-niveau issue du capteur à l'information symbolique recherchée. Il est maintenant admis que l'on ne saurait espérer, par une telle approche, résoudre un problème de reconnaissance un tant soit peu complexe [KOHONEN 90].

On se dirige donc ([HAMPSHIRE 89], [BOTTOU 91]) vers des systèmes utilisant plusieurs réseaux de manière ponctuelle afin que le "saut" réalisé par chaque réseau dans les niveaux d'abstraction ne soit pas trop important. L'idéal consiste alors à faire coopérer ces multiples réseaux au sein d'un "méta-réseau" afin que chacun d'entre eux contribue au mieux à la réalisation de l'objectif final, la reconnaissance, tout comme chaque cellule concourt à l'intérieur d'un réseau à l'optimisation d'un critère global .

Nous nous sommes pour l'instant contentés d'utiliser des réseaux, de manière indépendante, à deux étapes du système : pour la segmentation d'images binaires et pour l'identification des symboles. Dans les deux cas les primitives manipulées sont d'assez haut niveau (l'histogramme des longueurs de corde et le codage du graphe du squelette respectivement) et les architectures peuvent être contraintes.

En ce qui concerne la segmentation, les réseaux à rétro-propagation du gradient se sont avérés satisfaisants surtout par leur capacité à s'accommoder d'un ensemble d'apprentissage incertain. Les histogrammes des longueurs de corde ont en effet été étiquetés "manuellement" et rien ne garantit *a priori* que cet étiquetage soit parfaitement cohérent. Malgré cela, le réseau obtenu a fourni une bonne segmentation.

Cette impression favorable ne s'est pas confirmée dans le cas de l'identification des symboles, tâche pour laquelle les différents réseaux expérimentés n'ont pas montré de supériorité décisive sur des méthodes aussi classiques que les plus proches voisins (tout au moins en ce qui concerne l'un des critères les plus importants : la capacité à généraliser). Cette constatation, effectuée par d'autres auparavant [WEIDEMAN 89], demande dans le cas présent à être confirmée par des tests à plus grande échelle, la petite taille des échantillons dont nous disposions ne permettant pas de certifier définitivement les tendances observées. Celles-ci sont cependant intéressantes dans la mesure où les conditions d'expérimentation qui ont été les nôtres sont celles que l'on peut rencontrer dans n'importe quel problème pratique de reconnaissance de formes n'ayant (pratiquement) jamais été traité et pour lequel de grosses

bases de test manquent. Nous espérons avoir fourni de premières estimations réalistes sur la robustesse que l'on peut attendre d'un système de reconnaissance de partitions. Nous sommes conscients que de nombreuses questions restent ouvertes : comment réaliser la segmentation de partitions graphiquement très complexes ? Comment faire coopérer efficacement les niveaux "image" et "syntaxe" de la reconnaissance ? Autant d'interrogations auxquelles nous espérons pouvoir contribuer à répondre dans l'avenir.

CONCLUSION

Nous nous sommes intéressés aux réseaux de neurones artificiels dans l'optique de résoudre un problème pratique de reconnaissance de formes. Parmi les différents modèles de réseaux susceptibles de réaliser une tâche de classification, les réseaux multi-couches nous ont semblé les plus prometteurs ; nous avons tenté de justifier ce choix au cours du premier chapitre.

En essayant de classifier des formes binaires à l'aide de tels réseaux, nous nous sommes heurtés aux problèmes inhérents à ce type de modèle : difficulté de mise en œuvre de l'algorithme d'apprentisage par rétro-propagation du gradient et choix de l'architecture. Nous avons exposé au second chapitre quelques unes des solutions proposées dans la littérature pour résoudre ces problèmes. En ce qui concerne l'architecture, nous avons proposé une méthode adaptée à notre application : l'architecture y est synthétisée d'après un arbre binaire construit sur l'ensemble d'apprentissage.

Nos résultats expérimentaux n'ayant pas démontré la supériorité décisive des réseaux multi-couches sur des méthodes plus conventionnelles de classification, nous avons tenté de mieux cerner le fonctionnement de ces réseaux. Ceci a pu en partie être réalisé grâce aux réseaux multi-couches d'automates à seuil que l'on peut facilement caractériser à l'aide des régions de décision qu'ils définissent. Nous avons ainsi montré comment il était possible de construire des réseaux multi-couches d'automates à seuil réalisant une classification au plus proche voisin.

De la même manière, on peut synthétiser des réseaux implémentant la classification d'un arbre de décision quelconque. Nous avons proposé un algorithme permettant de construire un arbre de décision en utilisant un principe proche de celui de l'apprentissage par rétro-propagation du gradient : la minimisation, par descente en gradient, d'un critère non statistique. Bien sûr, l'intérêt de cette méthode réside dans le fait que l'architecture n'a plus ici a être choisie

a priori puisqu'elle est automatiquement définie au cours de l'apprentissage.

Au cours du troisième chapitre, nous avons exposé la manière dont ces différentes techniques ont été appliquées à la reconnaissance de partitions musicales. Naturellement, ces méthodes ne peuvent résoudre seules tous les problèmes qui se posent et nous avons été amenés à développer différentes solutions ad-hoc pour le pré-traitement, la segmentation et la classification. Ces solutions relèvent alors plus de l'analyse d'images que des réseaux de neurones.

Le système de reconnaissance bas-niveau qui en résulte permet la classification des symboles d'une partition de manière robuste, tout au moins dans le cas de partitions graphiquement peu complexes. L'accomplissement de ce système passe par son interfaçage avec un système d'analyse "haut-niveau" tel qu'il en existe déjà, afin de tirer parti de la forte syntaxe musicale. Il passe également par le développement de méthodes originales permettant d'aborder les problèmes de segmentation qui se posent dans le cas de partitions très complexes. Ces deux points constituent des directions de recherche privilégiées pour l'avenir.

En utilisant les réseaux de neurones artificiels au cours de cette étude et sachant que d'autres solutions, plus éprouvées, s'offraient à nous, nous avons toujours eu à l'esprit l'interrogation suivante : les réseaux apportent-ils réellement quelque chose de nouveau ? A l'issue de ce travail, nous ne sommes pas certains de pouvoir répondre positivement à cette question.

A propos du Perceptron, Minsky et Papert [MINSKY 69] évoquaient la théorie selon laquelle tout processus informatique qui marche peut être compris, et tous ceux qui ne peuvent être compris sont suspects. Les réseaux les plus susceptibles d'apporter des améliorations notables aux méthodes classiques de classification nous semblent être les réseaux multi-couches à unités sigmoïdales. Or leur fonctionnement n'est pas totalement maîtrisé et ils seraient donc, à ce titre, suspects. Si, pour s'affranchir de cette incertitude, on considère des réseaux composés d'unités plus simples, il apparaît que l'originalité connexionniste ne réside en fait que dans une formulation différente de techniques déjà connues.

Finalement, c'est peut être en cette capacité à décrire différentes approches à l'aide d'une présentation unifiée que nous verrons le principal attrait des réseaux de neurones. En utilisant une formulation connexionniste, des méthodes aussi diverses que les plus proches voisins, les arbres de classification, l'analyse discriminante, la classification bayesienne ou les chaînes de Markov ont été "redécouvertes". Les réseaux de neurones sont probablement, et avant toute

chose, un bon support pour la définition de méthodes de classification et, à ce titre, seront peut-être à l'origine d'approches réellement novatrices dans l'avenir.

En tout état de cause, ils ne sauraient seuls répondre aux différents problèmes soulevés par la reconnaissance de partitions musicales. Nous espérons modestement avoir contribué à résoudre quelques-uns de ces problèmes et surtout suscité chez le lecteur un intérêt pour ce sujet d'étude. Ses multiples facettes en font un exercice pluridisciplinaire par excellence, un terrain de rencontre privilégié pour l'analyse d'images, la reconnaissance des formes et l'intelligence artificielle...

REFERENCES BIBLIOGRAPHIQUES

[ALPAYDIN 90] ALPAYDIN E.
Grow-and-Learn : an Incremental Method for Category Learning.
Proc. of the INNC 90, vol II, pp 761-764, Paris 1990.

[ATIYA 90] ATIYA A.F.
An unsupervised learning technique for artificial neural networks.
Neural networks, vol. 3, pp 707-711, 1990.

[ATLAS 90] ATLAS L., COLE R., MUTHUSAMY Y., LIPPMAN A., CONNOR J., PARK D., EL-SHARKAWI M., MATKS II R.J.
A performance comparison of trained multilayer Perceptrons and trained classification trees.
Proc. of the IEEE, vol. 78, n° 10, pp 1614-1618, 1990.

[BAUM 89] BAUM E.
What size net gives valid generalization ?
Neural computation, vol. 1, n° 1, pp 151-160, 1989.

[BEASSE 87] BEASSE F, BLUTEAU J., CAHAREL M.-H., DELAUNAY Ch., DERRIEN D.
Traduction automatique d'une partition musicale en notation braille à partir d'un télécopieur.
Mémoire de maîtrise, Université de Rennes, 1987.

[BECKER 88] BECKER S., LE CUN Y.
Improving the convergence of back-propagation learning with second order methods.
Proc. of the 1988 Connectionist summer school, eds D.S Touretzki *et al.*, Morgan Kaufmann, pp 29-37, 1988.

[BLAYO 89] BLAYO F.
Tour d'horizon sur les implémentations de réseaux de neurones.
Cours de la 3ème école d'été du CIRILLE, Université de Lyon I, juillet 1989.

[BLOSTEIN 91] BLOSTEIN D., BAIRD H.S.
A critical survey of music image analysis.
Structured document image analysis, Baird, Bunke, Yamamoto eds, Springer-Verlag, 1991.

[BONNEFOY 87] BONNEFOY J.P., LORETTE G.
Une méthode adaptative de sélection de variables en reconnaissance de formes.
6ème congrès RFIA, vol. 1, pp 473-478, Dunod editor, 1987.

[BOTTOU 88] BOTTOU L.-Y.
Reconnaissance de la parole par réseaux multi-couches.
Actes des journées Neuro-Nîmes, pp 197-217, EC2 éditeur, 1988.

[BOTTOU 91] BOTTOU L., GALLINARI P.
A framework for the cooperation of learning algorithms.
Rapport de recherche n° 635, URA CNRS 410 / Université d'Orsay, 1991.

[BOWYER 80] BOWYER A.
Computing Dirichlet tessellations.
The computer journal, vol 24, n° 2, pp 162-166, 1980.

[BREIMAN 84] BREIMAN L., FRIEDMAN J.H., OLSHEN R.A., STONE C.J.
Classification and regression trees.
Wadsworth, 1984.

[BRESENHAM 65] BRESENHAM J.E.
Algorithm for digital control of a digital plotter.
IBM system journal, vol. 4, n° 1, pp 25-30, 1965.

[CARPENTER 87] CARPENTER G.A., GROSSBERG S.
A massively parallel architecture for self-organizing neural pattern recognition machine.
Computer vision, graphics and image processing, vol. 37, pp 54-115, 1987.

[CASH 87] CASH G.L., HATAMIAN M.
Optical character recognition by the method of moments.
Computer vision, graphics and image processing, n° 39, pp 291-310, 1987.

[CELEUX 91] CELEUX G., LECHEVALLIER Y.
Méthodes de segmentation.
Analyse discriminante, INRIA editeur, pp 127-147, 1991.

[CHAUVIN 90] CHAUVIN Y.
Generalized performance of overtrained back-propagation networks
Proc. of the Eurasip workshop on neural networks, eds L.B. Almeida *et al.*, Springer-Verlag, pp 46-55, 1990.

[CHEN 89a] CHEN Y.S., HSU W.H.
An interpretive model of line continuation in human visual perception.
Pattern Recognition, vol 22, n° 5, pp 619-639, 1989.

[CHEN 89b] CHEN Y.S., HSU W.H.
A systematic approach for designing 2-subcycle and pseudo 1-subcycle parallel thinning algorithms.
Pattern Recognition, vol 22, n° 3, pp 267-282, 1989.

[CHERIET 89] CHERIET M., CRETTEZ J.-P.
Mise en œuvre d'un processus de sélection de primitives robustes en vue de la reconnaissance des caractères multifontes.
Actes de Pixim 89, pp 357-370, Hermès editeur, Paris, 1989.

[CHOWNING 73] CHOWNING J.
The synthesis of complex audio spectra by means of frequency modulation.
Journal of audio engineers society, vol. 21, no. 7, pp 526-534, 1973.

[CIARDELLO 88] CIARDELLO G., DEGRANDI M.T., ROCCOTELLI M.P., SCAFURO G., SPADA M.R.
An experimental system for office document handling and text recognition.
Proc. of the 9th ICPR, vol II, pp 739-743, Rome, 1988.

[CIOS 91] CIOS K.J., LIU N.
A comparative study of machine learning algorithms for generation of a neural network architecture.
Artificial neural networks, eds T. Kohonen *et al*, North-Holland, pp. 189-194, 1991.

[COUASNON 91] COUASNON B.
Réseaux de neurones appliqués à la reconnaissance de partitions musicales.
Mémoire de D.E.A. informatique, INSA/IRISA, Rennes, 1991.

[COUEIGNOUX 81] COUEIGNOUX Ph.
La reconnaissance des caractères.
La recherche, n° 126, vol. 12, pp 1094-1103, 1981.

[DIDAY 80] DIDAY E., SIMON J.C.
Clustering analysis.
Digital Pattern Recognition, K.S. Fu editor, Springer-Verlag, Berlin, pp 47-94, 1980.

[DIDAY 82] DIDAY E., LEMAIRE J., POUGET J., TESTU F.
Eléments d'analyse de données.
Dunod éditeur, 1982.

[DUDA 72] DUDA R.O., HART P.E.
Use of the Hough transformation to detect lines and curves in pictures.
Communications of the ACM, vol. 15, n° 1, pp 11-15, 1972.

[DUDA 73] DUDA R.O., HART P.E.
Pattern classification and scene analysis.
John Wiley and sons, 1973.

[ECKHARDT 88] ECKHARDT U., MADERLECHNER G.
The structure of irreducible digital sets obtained by thinning algorithms.
Proc. of the 9th ICPR, vol II, pp 727-729, Rome, 1988.

[EDELSBRUNNER 87] EDELSBRUNNER H.
Algorithms in combinatorial geometry.
Springer-Verlag, Berlin, 1987.

[FAHLMAN 88] FAHLMAN S.E.
An empirical study of learning speed in back-propagation networks.
Technical report n° CMU-CS-88-162, Carhengie Mellon university, computer science department, juin 1988.

[FAHMY 91] FAHMY H., BLOSTEIN D.
A graph-grammar for high-level recognition of music notation.
Proc. of the 1st International Conference on Document Analysis and Recognition, AFCET éditeur, vol. I, pp 70-78, 1991.

[FLETCHER 88] FLETCHER L.A., KASTURI R.
A robust algorithm for text string separation from mixed text/graphics images.
IEEE trans. on PAMI, vol. 10, pp 910-918, 1988.

[FOGELMAN SOULIE 87] FOGELMAN SOULIE F., GALLINARI P., LE CUN Y., THIRIA S.
Automata networks and artificial intelligence.
Automata networks in computer science, theory and applications, chap. 7, Fogelman Soulié F., Robert Y. and Tchuente M. editors, Manchester University press, 1987.

[FOGELMAN SOULIE 89] FOGELMAN SOULIE F.
Applications des méthodes connexionnistes en intelligence artificielle.
Architecture avancées pour l'intelligence artificielle, actes des 11èmes journées francophones sur l'informatique, EC2 éditeur, pp 101-116, 1989.

[FOGELMAN SOULIE 91] FOGELMAN SOULIE F.
Neural network architectures and algorithms : a perspective.
Artificial neural networks, vol. I, eds T.Kohonen *et al.*, North-Holland, pp 605-615, 1991.

[FREAN 90] FREAN M.
The upstart algorithm : a method for constructing and training feedforward neural networks.
Neural computation, n° 2, pp 198-209, 1990.

[FU 80] FU K.S.
Introduction.
Digital Pattern Recognition, K.S. Fu editor, Springer-Verlag, Berlin, pp 1-14, 1980.

[GALLANT] GALLANT S.I.
Optimal linear discriminants
Proc. of the 8th ICPR, vol. 2, pp 849-852, Paris, 1986.

[GALLINARI 88] GALLINARI P., FOGELMAN SOULIE F.
Progressive design of MLP's architecture.
Actes Neuro'Nîmes 88, pp 171-182, EC2 éditeur, Nîmes, 1988.

[GELFAND 91] GELFAND S.B., RAVISHANKAR C.S., DELP E.J.
An iterative growing and pruning algorithm for classification tree design.
IEEE Trans. Pattern Anal. Mach. Intell., vol. 13, n° 2, pp 163-174, 1991.

[GEMAN 84] GEMAN S., GEMAN D.
Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images.
IEEE transactions on PAMI, vol 6, pp 721-741, 1984.

[GIBSON 90] GIBSON G.J., COWAN C.F.N.
On the decision regions of multilayer Perceptrons.
Proc. of the IEEE, vol. 78, n° 10, pp 1590-1594, 1990.

[GOLES 87] GOLES E.
Lyapunov functions.
Automata networks in computer science, theory and applications, chap. 4, Fogelman Soulié F., Robert Y. and Tchuente M. editors, Manchester University press, 1987.

[GUYON 89] GUYON I., POUJAUD I.,PERSONNAZ L., DREYFUS G.
Comparing different neural network architectures for classifying handwritten digits.
Proc. Int. Joint Conf. on Neural Networks, vol. II, Washington D.C., U.S.A, pp. 127-132, 1989.

[GUYON 91] GUYON I., ALBRECHT P., LE CUN Y., DENKER J.S., HUBBARD W.
Design of a neural network character recognizer for a touch terminal.
Pattern Recognition, vol. 24, n° 2, pp 105-119, 1991.

[HAMPSHIRE 89] HAMPSHIRE II J.B., WAIBEL A.H.
The Meta-Pi network : building distributed knowledge representations for robust pattern recognition.
Rapport de recherche CMU-CS-89-166, Carnegie Mellon university, 1989.

[HEBB 49] HEBB D.O.
The organization of behavior
Wiley, New-York, 1949.

[HECHT-NIELSEN 90] HECHT-NIELSEN R.
Neurocomputing.
Addison-Wesley editor, Reading, Massachusets, 1990.

[HINDS 90] HINDS S.C., FISHER J.L., D'AMATO D.P.
A document skew detection method using run-length encoding and the Hough transform.
Proc. of the 10th ICPR, vol I, pp 464-468, Rome, 1990.

[HINTON 83] HINTON G.E., SEJNOWSKI T.J.
Optimal perceptual inference.
Proc. of the IEEE computer society conference on computer vision and pattern recognition, pp 448-453, Washington D.C., 1983.

[HINTON 86] HINTON G.E., SEJNOWSKI T.J.
Learning and relearning in Boltzmann machines
Parallel distributed processings : explorations in the microstructure of cognition, vol 1, chap 7, MIT press, 1986.

[HOLT 87] HOLT C.M., STEWART A., CLINT M., PERROT R.H.
An improved parallel thinning algorithm.
Communications of the ACM, vol. 30, n° 2, pp 156-160, 1987.

[HOPFIELD 82] HOPFIELD J.J.
Neural networks and physical systems with emergent collective computational abilities
Proc. of the National Academy of Sciences, USA, vol. 79, pp. 2554-2558, avril 1982.

[HOPFIELD 84] HOPFIELD J.J.
Neurons with graded response have collective computational properties like those of two-states neurons
Proc. of the National Academy of Sciences, USA, vol. 81, pp. 3088-3092, mai 1984.

[HOPFIELD 85] HOPFIELD J.J., TANK D.W.
Neural computation of decisions in optimization problems.
Biological cybernetics, n° 52, 1985.

[ILLINGWORTH 88] ILLINGWORTH J., KITTLER J.A.
A survey of the Hough transform.
Computer Vision, Graphics, Image Processing, vol. 44, pp 87-116, 1988.

[JUTTEN 90] JUTTEN C., GUERIN A., HERAULT J.
Simulation machine and integrated implementation of neural networks : a review of methods, problems and realizations.
Proc. of the Eurasip workshop on neural networks, eds L.B. Almeida *et al.*, Springer-Verlag, pp 244-266, 1990.

[KAHAN 87] KAHAN S., PAVLIDIS T., BAIRD H.S.
On the recognition of printed characters of many font and any size.
IEEE transactions on PAMI, vol 9, n° 2, pp 274-288, 1987.

[KELLY 91] KELLY M.
Self-organizing map training using dynamic k-D trees.
Artificial neural networks, vol. II, eds T.Kohonen *et al.*, North-Holland, pp 1041-1044, 1991.

[KIRKPATRICK 83] KIRKPATRICK S., GELATT C.D., VECCHI M.P.
Optimization by simulated annealing.
Science, n° 220, pp 671-680, 1983.

[KLASSEN 89] KLASSEN M.
Incremental learning for recognizing handwritten characters using neural networks
Technical report TR 89-113, Case western reserve university, CAISR, 1989.

[KNERR 91] KNERR S., PERSONNAZ L., DREYFUS G.
From theory to silicon : an efficient procedure for the design of "neural" classifiers ans its application to the automatic recognition of handwritten digits.
Actes des journées Neuro-Nîmes, EC2 éditeur, 1991.

[KOHONEN 84] KOHONEN T.
Self-organization and associative mémories.
Springer-Verlag, Berlin, 1984.

[KOHONEN 90] KOHONEN T.
The self-organizing map.
Proceedings of the IEEE, vol. 78, n° 9, pp 1464-1480, 1990.

[KURKELA 88] KURKELA K.
Partition, vision, action.
La musique et les sciences cognitives, Mc Adams S. et Deliège I., Pierre Mardaga éditeur, Liège-Bruxelles, pp 587-612, 1988.

[LANG 88] LANG K., WITBROK M.J.
Learning to tell two spirals apart.
Proc. of the 1988 Connectionist summer school, eds D.S Touretzki *et al.*, Morgan Kaufmann, 1988.

[LE CUN 90] LE CUN Y., JACKEL L.D., GRAF H.P., BOSER B., DENKER J.S., GUYON I., HENDERSON D., HOWARD R.E, HUBBARD W., SOLLA S.A.
Optical character recognition and neural-net chips.
INNC 90, vol. 2, pp 651-655, Paris 1990.

[LEUNG 88] LEUNG C.H.
Structural matching using neural networks
Proc. of the 1st ICNN, p 31, Boston, 1988.

[LIPPMAN 87] LIPPMAN R.P.
An introduction to computing with neural nets.
IEEE ASSP Magazine, pp 4-22, avril 1987.

[LOY 85] LOY G.
Musicians make a standard : the MIDI phenomenon.
Computer music journal, vol. 9, n° 4, pp 8-26, 1985.

[LUBAR 82] LUBAR N.
Extraction et reconnaissance structurelle de graphismes
Application à la reconnaissance de partitions musicales.
Mémoire de diplôme d'ingénieur CNAM, Belfort-Montbéliard, 1982.

[LUX 85] LUX A.
Algorithmique et contrôle en vision par ordinateur.
Thèse de doctorat es sciences, ENSIMAG, 1985.

[MAGGIONI 91] MAGGIONI C., WIRTZ B.
A neural net approach to 3-D pose estimation.
Artificial neural networks, vol. I, eds T.Kohonen *et al.*, North-Holland, pp 75-80, 1991.

[MAITRE 85] MAITRE H.
Un panorama sur la transformation de Hough.
Traitement du signal, vol. 2, n° 4, pp 305-317, 1985.

[MAJANI 89] MAJANI E., ERLANSON R., ABU-MOSTAFA Y.
On the k-winners-take-all network.
Advances in neural information processing systems 1, eds D.S. Touretzky, Morgan Kaufmann, pp 634-642, 1989.

[MARTIN 89] MARTIN Ph.
Reconnaissance de partitions musicales et réseaux de neurones : une étude.
Actes du 7ème congrès RFIA, vol. I, pp 217-226, AFCET et INRIA éditeurs, Paris, 1989.

[MARTIN 90] MARTIN Ph., BELLISSANT C.
Segmentation et classification par réseau pour la reconnaissance de partitions musicales.
Colloque Reconnaissance Automatique de l'Ecrit, BIGRE n° 68, pp 102-11, IRISA éditeur - Rennes, mai 1990.

[MARTIN 91a] MARTIN Ph., BELLISSANT C.
Neural networks at different levels of a musical score image analysis system.
Proc. of the 7th Scandinavian Conference on Image Analysis, vol. II, pp 1102-1109, Aalborg, Danemark, 1991.

[MARTIN 91b] MARTIN Ph., BELLISSANT C.
Low-level analysis of music drawings.
Proc. of the 1st International Conference on Document Analysis and Recognition, AFCET éditeur, vol. I, pp 417-425, 1991.

[MARTIN 91c] MARTIN Ph., BELLISSANT C.
Geometrical learning in a network of automata
Artificial neural networks, eds T. Kohonen *et al*, North-Holland, pp. 1793-1796, 1991.

[MARTIN 91d] MARTIN Ph., BELLISSANT C.
Modèles connexionnistes pour la reconnaissance visuelle de partitions musicales
Ph. Martin et C. Bellissant.
Symposium sur l'Intelligence Artificielle Perceptive, Martigny, Suisse, 14-15 mars 1991.

[MARTIN 92] MARTIN Ph., BELLISSANT C.
Neural networks for the recognition of engraved musical scores.
Int. Journal of Pattern Recognition and Artificial Intelligence, vol. 6, n° 1, 1992.

[McCULLOCH 43] McCULLOCH W.S., PITTS W.
A logical calculus of the ideas immanent in nervous activity.
Bulletin of mathematical biophysics, vol. 5, pp 115-137, 1943.

[MINSKY 69] MINSKY M.L., PAPERT S.A.
Learning theory.
Perceptrons, expanded edition, chap. 10-13, pp 150-246, MIT Press, 1969 - 1988.

[MONTANVERT 87] MONTANVERT A.
Contribution au traitement de formes discrètes : squelette et codage par graphe de la ligne médiane.
Thèse de Doctorat de l'Université Joseph Fourier Grenoble I, spécialité Informatique, 1987.

[MOORE 90] MOORE R.
Elements of Computer Music.
Prentice Hall, 1990.

[NADAL 89] NADAL J.P.
Study of a growth algorithm for a feedforward network.
International journal of neural systems, vol. 1, n° 1, pp 55-59, 1989.

[PAKKER 85] PAKKER K., CHEHIKIAN A.
Reconnaissance de caractères alphanumériques multipolices par analyse structurelle hétérarchique.
Actes du 5ème congrès RFIA, vol. II, pp 817-827, Grenoble, 1985.

[PAO 89] PAO Y.-H.
Learning discriminants : the generalized perceptron
Adaptative Pattern Recognition and Neural Networks,
pp 113-139, Addison Wesley editor, 1989.

[PARK 89] PARK Y., SKLANSKY J.
Automated design of multiple-class piecewise linear classifiers.
Journal of classification, vol. 6, pp 195-222, 1989.

[PERETTO 90] PERETTO P.
Réseaux de neurones et optimisation combinatoire.
Neural networks : biological computers or electronic brains, actes des entretiens de Lyon, Springer-Verlag, pp. 127-134, 1990.

[PERSONNAZ 88] PERSONNAZ L., DREYFUS G., GUYON I.
Les machines neuronales.
La recherche, n° 204, pp 1362-1371, nov. 1988.

[POSTL 86] POSTL W.
Detection of linear oblique structures and skew scan in digitized documents.
Proc. of the 8th ICPR, vol II, pp 687-689, Paris, 1986.

[PREPARATA 85] PREPARATA F.P., SHAMOS M.I.
Computational geometry.
Springer-Verlag, Berlin, 1985.

[PRERAU 71] PRERAU D.S.
Computer pattern recognition of printed music.
Fall joint computer conference, pp 153-162, 1971.

[PRERAU 75] PRERAU D.S.
DO-RE-MI: a program that recognizes music notation.
Computer and the humanities, vol. 9, pp 25-29, 1975.

[RADOUI 90] RADOUI M., HATABIAN G.
Les réseaux formels de neurones : une généralisattion de l'analyse des données ?
Actes des XXIIèmes journées de statistique, pp 188-190, Tours, juin 1990.

[RAMER 72] RAMER
An iterative procedure for the polygonal approximation of plane curves.
Computer Graphics and Image Processing, vol. 1, pp 244-256, nov. 1972.

[REILLY 82] REILLY D.L., COOPER L.N., ELBAUM C.
A neural model for category learning.
Biological cybernetics, n° 45, pp 35-41, 1982.

[RISSET 69] RISSET J-C., MATHEWS M.V.
Analysis of musical instrument tones.
Physics Today, n° 22, pp 23-40, 1969.

[ROACH 88] ROACH J.W., TATEM J.E.
Using domain knowledge in low-level visual processing to interpret handwritten music : an experiment.
Pattern recognition, vol. 21, pp 33-44, 1988.

[ROADS 79] ROADS C.
Grammars as representations for music.
Computer music journal, vol. 3, n° 1, pp 48-55, 1979.

[ROBERT 86] ROBERT F.
Discrete iterations : a metric study.
Springer Verlag, Berlin, Heidelberg, New-York, 1986.

[RODET 84] RODET X., COINTE P.
FORMES : composition and scheduling of processes.
Computer music journal, vol. 8, n° 3, 1984.

[ROSENBLATT 57] ROSENBLATT F.
The Perceptron : a probabilistic model for information storage and organization in the brain
Psychological revue, n° 65, pp 386-408, 1958.

[RUMELHART 86] RUMELHART D.E, HINTON G.E., WILLIAMS R.J.
Learning internal representations by error propagation.
Parallel distributed processings : explorations in the microstructure of cognition, vol 1, chap 8, MIT press, 1986.

[SAMUELIDES 89] SAMUELIDES M.
La machine de Boltzmann : principes et applications.
Cours de la 3ème école d'été du CIRILLE, Université de Lyon I, juillet 1989.

[SATO 91] SATO A.
An analytical study of the momentum term in a backpropagation algorithm.
Artificial neural networks, vol. I, eds T.Kohonen *et al.*, North-Holland, pp 617-622, 1991.

[SCHRŒDER 80] SCHRŒDER M.R.
Towards better acoustics for concert Halls.
Physics Today, n° 33, pp 24-30, 1980.

[SERRA 82] SERRA J.
Image analysis and mathematical morphology.
Academic press, 1982.

[SETHI 91] SETHI I.K.
Entropy nets : from decision trees to neural networks.
Proc. of the IEEE, vol. 78, n° 10, pp 1605-1613, 1990.

[SOLLA 89] SOLLA S.A.
Learning and generalization in layered networks : the contiguity problem.
Neural Networks : from models to applications, pp 168-177, eds Personnaz *et al*, I.D.S.E.T, Paris, 1989.

[SOLLA 90] SOLLA S.A.
Supervised learning and generalization.
Neural networks : biological computersor electronic brains, actes des entretiens de Lyon, Springer-Verlag, pp. 21-28, 1990.

[SUDDARTH 90] SUDDARTH S.C., KERGOSIEN Y.L.
Rule-injection hints as a means of improving network performance and learning time.
Proc. of the Eurasip workshop on neural networks, eds L.B. Almeida *et al.*, Springer-Verlag, pp 120-129, 1990.

[THIRIA 89] THIRIA S.
L'apprentissage supervisé dans les modèles connexionnistes.
Thèse de doctorat de l'université René Descartes (Paris V), 1989.

[TODD 91] TODD P.M., LOY D.G.
Music and connectionism.
MIT Press, Cambridge, Massachusetts, 1991.

[TOJO 82] TOJO A., AOYAMA H.
Automatic recognition of music score.
Proc. of the 6th ICPR, p. 1223., Münich, 1982.

[TOMAN 91] TOMAN E., TOMANOVA J.
Some estimates on the complexity of disjunctive normal forms of a random boolean function.
Computers and artificial intelligence, vol. 10, n°4, pp 327-340, 1991.

[VASQUEZ 90] VASQUEZ M., DUBANT O.
Prototype de reconnaissance de texte par coopération de techniques : neuromimétique et automate fini.
Colloque Reconnaissance Automatique de l'Ecrit, BIGRE n° 68, pp 160-168, IRISA éditeur - Rennes, mai 1990.

[WAIBEL 89] WAIBEL A.
Consonant recognition by modular construction of large phonemic time-delay neural networks.
Advances in neural information processing systems 1, D.S. Touretzky éditeur, pp 215-223, 1989.

[WANG 89] WANG S.
Réseaux multi-couches de neurones artificiels.
Thèse de doctorat de l'Institut National Polytechnique de Grenoble, spécialité informatique, septembre 1989.

[WASSERMAN 89] WASSERMAN P.D.
Neural Computing.
Van Nostrand Reinhold editor, New York, 1989.

[WATROUS 87] WATROUS R.
Learning algorithms for connectionist networks : applied gradient methods of non-linear optimization.
Proc. of the Int. Conf. on Neural Networks, vol. 4, pp 619-627, San Diego, 1987.

[WATSON 80] WATSON D.F.
Computing the n-dimensional Delaunay tesselation with application to Voronoi polytopes.
The computer journal, vol 24, n° 2, pp 167-172, 1980.

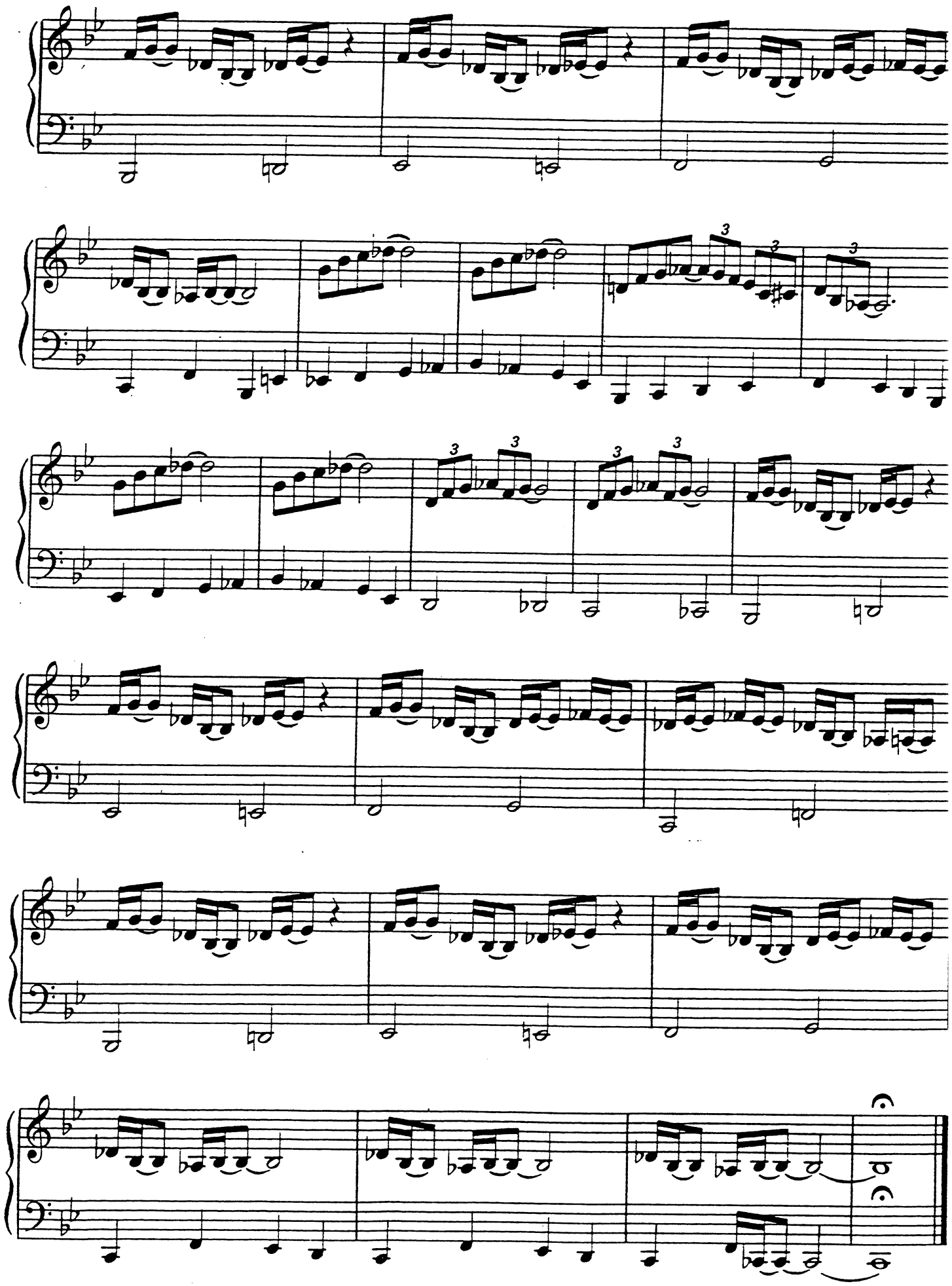
[WEIDEMAN 89] WEIDEMAN W.E., MANRY M.T., YAU H.C.
A comparison of nearest neighbor classifier and a neural network for numeric handprint recognition.
Proc. of the IJCNN, vol I, pp 117-120, Washington, 1989.

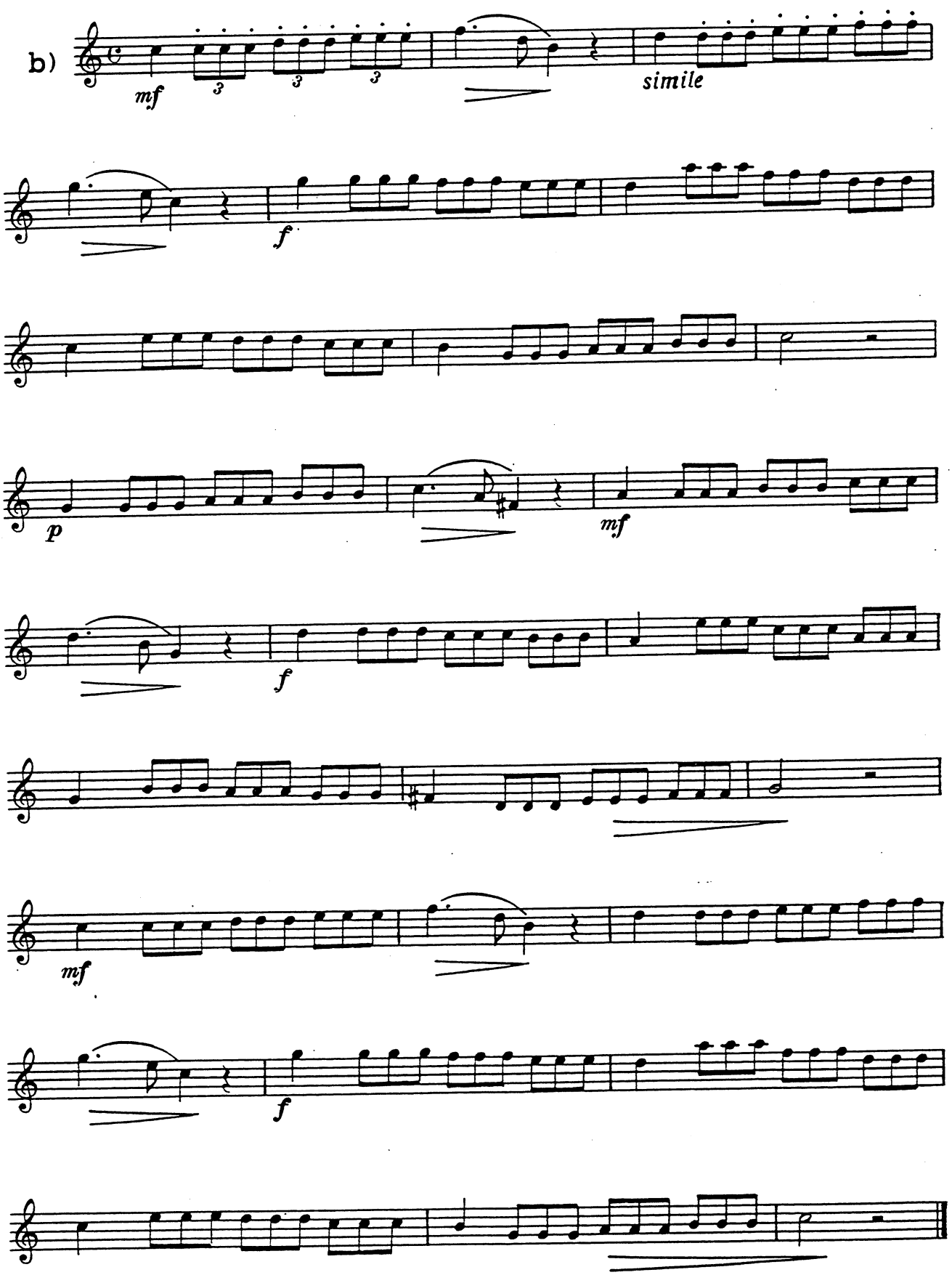
[WIDROW 60] WIDROW B., HOFF M.E.
Adaptative switching circuits.
IRE WESCON conv. record, part 4, pp 96-104, 1960.

[WILSON 72] WILSON D.
Asymptotic properties of nearest neighbor rules using edited data.
IEEE transactions on Systems, Man abd Cybernetics, vol. 2,
pp 408-421, 1972.

[ZHANG 84] ZHANG T.Y., SUEN C.Y.
A fast parallel algorithm for thinning digital patterns.
Comm. of the ACM n° 27, vol 3, pp 236-239, 1984.

ANNEXE

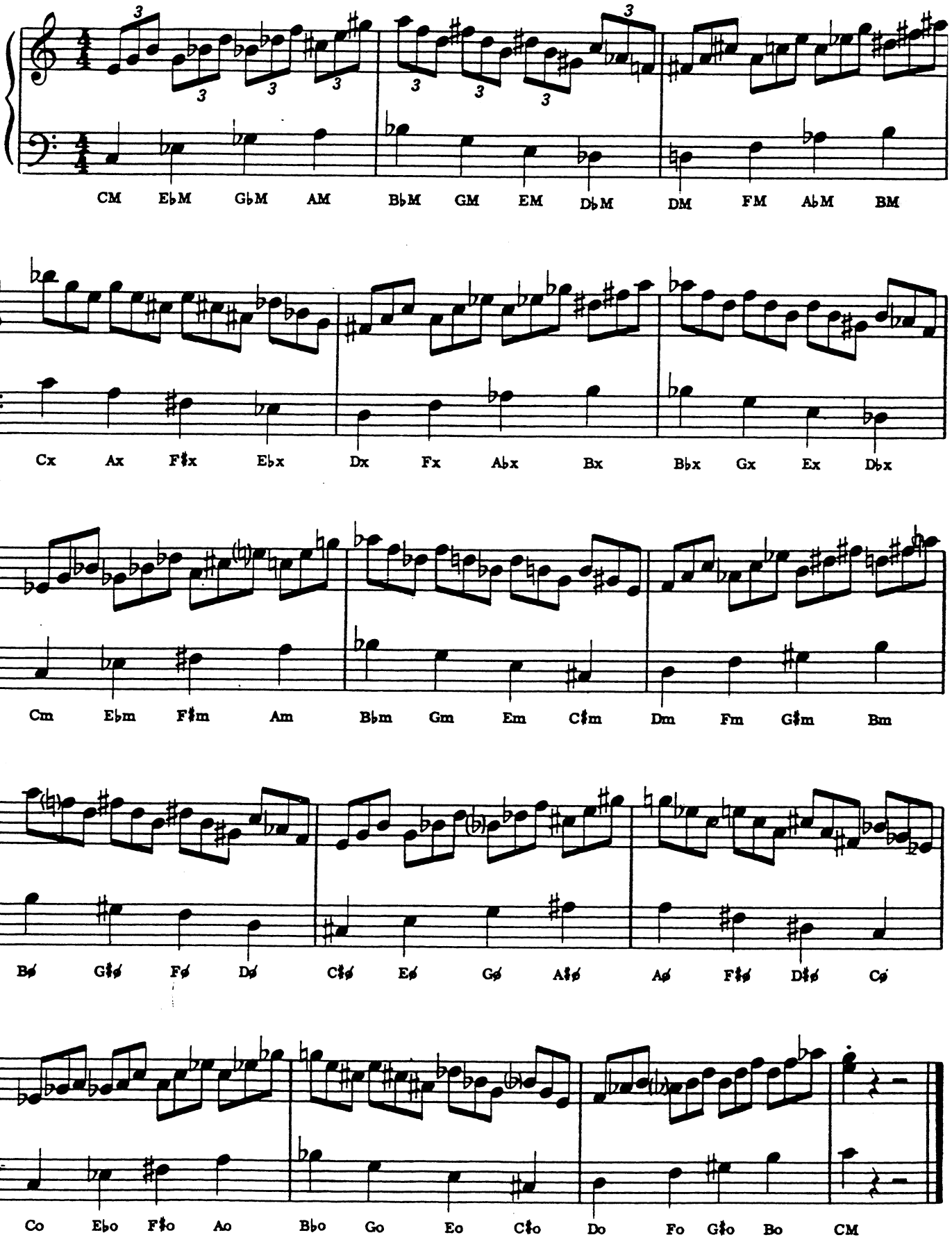



b)
mf
3
3
3
simile
f
p
mf
f
mf
f

10. CHROMATIC IMPROMPTU

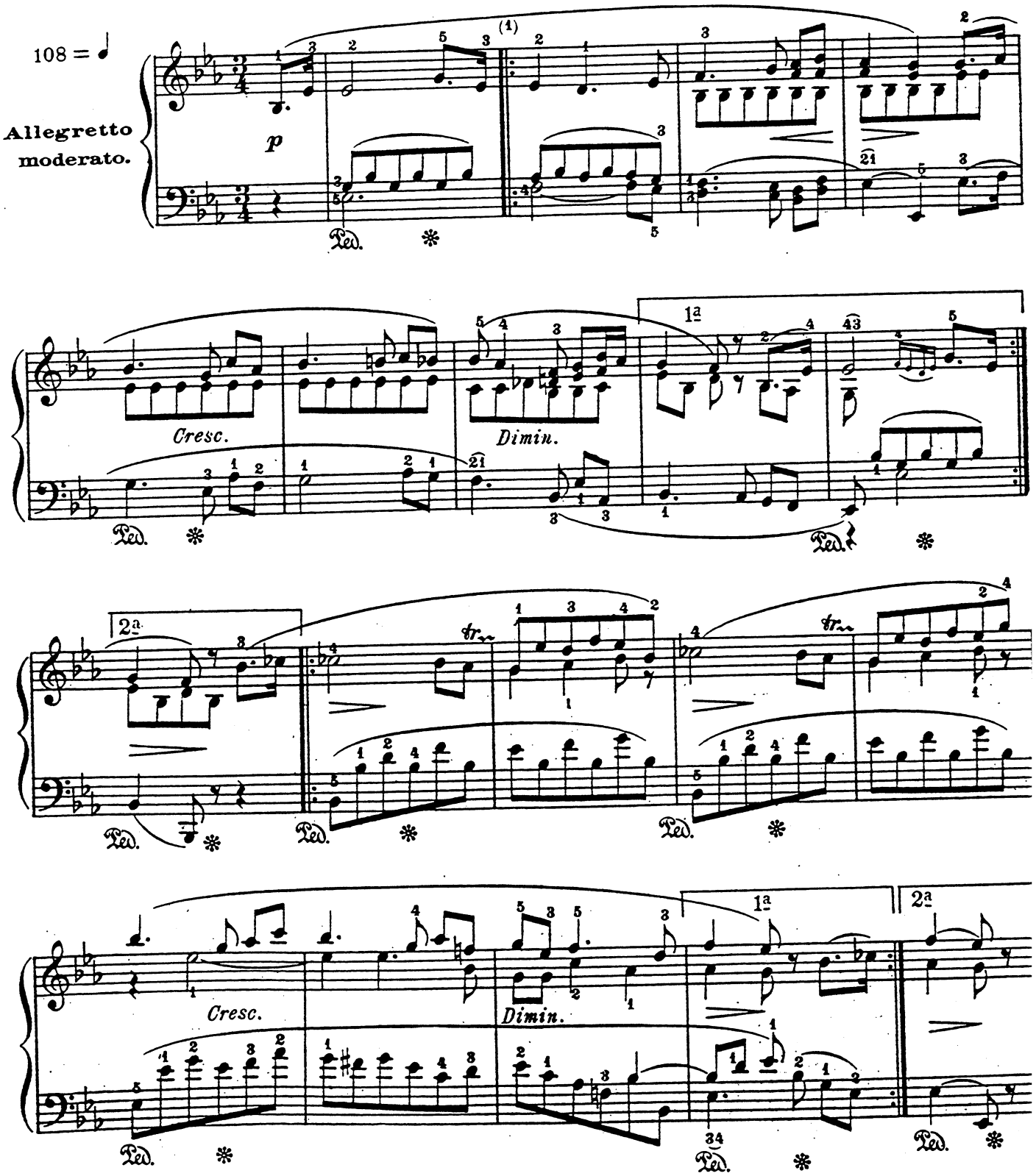
JOHN MEHEGAN

Bright



MINUETTO

BEETHOVEN.



Réseaux de Neurones Artificiels :
Application à la Reconnaissance Optique de Partitions Musicales

Résumé : Le travail présenté dans ce mémoire décrit le développement et l'utilisation de réseaux de neurones artificiels pour la résolution d'un problème de reconnaissance de formes particulières, les partitions musicales.

Le premier chapitre est consacré à une étude bibliographique des méthodes connexionnistes employées en reconnaissance de formes. Nous tentons de présenter les principaux modèles de réseaux de neurones de manière homogène et de justifier le choix du modèle auquel nous nous sommes particulièrement intéressés : celui des réseaux multi-couches.

Nous consacrons le deuxième chapitre à l'étude de ces derniers. Après une synthèse des différentes connaissances utiles au choix de l'architecture d'un réseau multi-couche et à la mise en œuvre de l'algorithme d'apprentissage par rétro-propagation du gradient, nous nous intéressons aux réseaux d'automates à seuil. Les propriétés de ces réseaux et leurs liens avec les méthodes usuelles de classification sont mis en évidence. Ceci nous amène à proposer un nouvel algorithme d'apprentissage hiérarchique.

Dans le dernier chapitre, nous décrivons un système de reconnaissance bas-niveau d'images de partitions musicales imprimées. Différentes solutions pour le pré-traitement, la segmentation et la classification sont proposées. Ces solutions font appel tant à l'analyse d'images conventionnelle qu'aux réseaux de neurones décrits dans les chapitres précédents ; elles sont illustrées par des résultats expérimentaux.

Mots clés : Réseaux de Neurones Artificiels, Réseaux Multi-couches, Automates à Seuil, Arbres de Classification, Reconnaissance de Formes, Analyse d'Images, Partitions Musicales.

Abstract : This thesis deals with the development and use of artificial neural networks to solve an unusual pattern recognition problem, those of optical musical score recognition.

The first chapter is devoted to a bibliographical study of the main connectionist models which are the most frequently used in the pattern recognition field. We try to present them in a homogeneous manner and to justify our interest in the multilayer models.

We focus our attention on these models in the second chapter. We first try to review the different techniques which are useful to design a multilayer architecture and train it with the well known gradient back-propagation learning algorithm. Next, we study the special case of threshold automata networks. Their properties and links with conventional classsification techniques are enhanced. This leads us to propose a new hierarchical learning scheme for these networks.

In the last chapter, we describe a low-level image analysis system for engraved musical scores. Several solutions for pre-processing, segmentation and classification are proposed. These solutions involve both classical image analysis techniques and connectionnist methods. Experimental results are presented.

Keywords : Artificial Neural Networks, Multilayer Networks, Threshold Automata, Classification Trees, Pattern Recognition, Image Analysis, Musical Scores.