THÈSE

Présentée par

Juan-Manuel TORRES-MORENO

Pour obtenir le titre de
DOCTEUR
de l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE
(Arrêté ministériel du 30 Mars 1992)

Spécialité : Sciences Cognitives

Apprentissage et généralisation par des réseaux de neurones : étude de nouveaux algorithmes constructifs

Soutenue le 22 Septembre 1997, devant le jury :

Président :	C. JUTTEN
Rapporteurs :	P. GALLINARI
	J. CABESTANY
Examinateurs :	H. PAUGAM-MOISY
	R. VELAZCO
Directrice :	M.B. GORDON

Thèse préparée au CEA/Grenoble. Département de Recherche Fondamentale sur la Matière Condensée/SPSMS/Groupe Théorie

Grenoble, FRANCE 1997

.

Pas pour tout le monde.*

No para cualquiera.*

*Hermann Hesse. El lobo estepario.

.

.

Remerciements

Mes remerciements vont particulièrement à *Mirta Gordon* pour m'avoir proposé ce sujet, pour son aide et le temps qu'elle m'a consacré, tout en me faisant profiter de sa culture et de sa rigueur scientifique.

Je suis très reconnaissant aussi à *M. Jutten* qui a bien voulu accepter la présidence du jury ; *Messieurs Gallinari* et *Cabestany* qui ont bien voulu être les rapporteurs ; *Mme. Paugam-Moisy* et *M. Velazco* qui ont accepté de prendre part au jury et pour l'intérêt qu'ils ont porté à mon travail.

Je remercie vivement le côté matériel [1] du même que le côté plus spirituel [2], le Commissariat à l'Energie Atomique, ainsi que tous ceux, collèges et amis (*Anna, Arnaud, Armilde Rivera, Catherine, Jérôme, José Antonio, J. Luis Hernandez, J. Luis Pinedo, Marisa, Pierre,* $\cdots$*, Rocío*), avec qui j'ai partagé tant de temps.

Je remercie également mes $(N-1)$ amis à Grenoble, le soutien de toute ma famille [3] et spécialement de *mi mamá*, car depuis longtemps elle voulait autant que moi la fin de ces études.

Finalement, sans oublier toutes ces années agréables passées ensemble, je dédie ce travail, à *Patricia* : parce que nous l'avons rêvé ensemble...

Cette thèse a été réalisée grâce au soutien du Consejo Nacional de Ciencia y Tecnología (CONACYT) et à l'Universidad Autónoma Metropolitana (UAM), México.

[1] Les machines *Lucifer, Sauron, Sun et la petite Karellen.*

[2] Spécialement le langage *C* et LaTeX, mais aussi d'autres logiciels.

[3] Tous ensemble, de A jusqu'à Z!

El Golem

Si, (como el griego afirma en el Cratilo) el nombre es arquetipo de la cosa, en las letras de *rosa* está la rosa y todo el Nilo en la palabra *Nilo*.

Y, hecho de consonantes y vocales, habrá un terrible *Nombre*, que la esencia cifre de Dios y que la Omnipotencia guarde en letras y sílabas cabales.

Adán y las estrellas lo supieron en el Jardín. La herrumbre del pecado (dicen los cabalistas) lo ha borrado y las generaciones lo perdieron.

Los artificios y el candor del hombre no tienen fin. Sabemos que hubo una día en que el pueblo de Dios buscaba el Nombre en las vigilias de la judería.

No a la manera de otras que una vaga sombra insinúan en la vaga historia, aún está verde y viva la memoria de Judá León, que era rabino en Praga.

Sediento de saber lo que Dios sabe, Judá León se dio a permutaciones de letras y complejas variaciones y al fin pronunció el Nombre que es *la Clave*,

la Puerta, el Eco, el Huésped y el Palacio, sobre un muñeco que con torpes manos labró, para enseñarle los arcanos del las Letras, del Tiempo y del Espacio.

El simulacro alzó los soñolientos párpados y vio formas y colores que no entendió, perdidos en rumores, y ensayó temerosos movimientos.

Gradualmente se vio (como nosotros) aprisionado en esta red sonora de *Antes, Después, Ayer, Mientras, Ahora, Derecha, Izquierda, Yo, Tú, Aquellos, Otros.*

El cabalista que ofició de numen a la vasta criatura apodó *Golem*. (Estas verdades las refiere Scholem en un docto lugar de su volumen.)

El rabí le explicaba el universo (Esto es mi pie; esto el tuyo; esto la soga) y logró, al cabo de años, que el perverso barriera bien o mal la sinagoga.

Tal vez hubo un error en la grafía o en la articulación del Sacro Nombre; a pesar de tan alta hechicería, no aprendió a hablar el aprendiz de hombre.

Sus ojos, menos de hombre que de perro y harto menos de perro que de cosa, seguían al rabí por la dudosa penumbra de las piezas del encierro.

Algo anormal y tosco hubo en el Golem, ya que a su paso el gato del rabino se escondía. (Ese gato no está en Scholem pero, a través del tiempo, lo adivino.)

Elevando a su Dios manos filiales, las devociones de su Dios copiaba o, estúpido y sonriente, se ahuecaba en cóncavas zalemas orientales.

El rabí lo miraba con ternura y con algún horror. *¿Comó (se dijo) pude engendrar este penoso hijo y la inacción dejé, que es la cordura?*

¿Por qué di en agregar a la infinita serie un símbolo más? Por qué a la vana madeja que en lo eterno se devana, di otra causa, otro efecto y otra cuita?

En la hora de angustia y del luz vaga, en su Golem los ojos detenía. ¿Quién nos dirá las cosas que sentía Dios, al mirar a su rabino en Praga?

Jorge Luis Borges, 1958.

Le Golem

Si (comme affirme un Grec dans le Cratyle) Le nom est archétype de la chose, Dans les lettres du mot *rose* est la rose Et le Nil tout entier dans le mot *Nil*.

Alors, fait de consonnes et de voyelles, Doit exister un Nom terrible qui condense L'Être de Dieu et sa Toute-Puissance En lettres et syllabes essentielles.

Il fut connu d'Adam et des étoiles, Dans le Jardin. La Cabbale prétend Que la rouille du péché le couvrit d'un voile Et qu'il fut perdu pour les générations.

L'industrie de l'homme n'a pas de limites, Ni sa candeur. Nous savons qu'arriva Un temps où le peuple de Dieu chercha Le Nom durant ses veilles israélites.

Plus sûre que celles par qui un vague Fantôme est glissé dans la vague histoire, Verte et vive demeure la mémoire de Juda Löw, qui fut rabbin à Prague

Assoiffé de savoir ce que Dieu sait, Ce Löw se voua aux substitutions De lettres, aux difficiles permutations, Si bien qu'un jour il prononça, Le Nom qui est

La Clef, la Porte et l'Hôte, la Demeure et la Grâce. Ce fut pour enseigner à un pantin Modelé par ses maladroites mains, Le secret de Lettres, du Temps et de l'Espace.

L'effigie, entrouvrant ses yeux somnolents, Perçut sans les comprendre formes et couleurs, Confusément comme autant de rumeurs, Puis se risqua aux premiers mouvements.

Graduellement (tout comme nous), elle se vit Prisonnière de ce réseau sonore : Avant, Après, Hier. Maintenant, Encore, Envers et Endroit, Moi et Toi, Toi et Lui.

(Le Cabbaliste qui faisait office de dieu A sa créature donna pour nom Golem : Ce sont là vérités que rapporte Sholem En certain chapitre de son ouvrage pieux)

Le Rabbin lui expliquait l'univers : Voici Une corde, ceci est mon pied, cela le tien. Après des années, le monstre réussit A balayer la synagogue, mal ou bien.

Peut-être Löw s'était-il un peu trompé Dans l'articulation ou bien dans la graphie Du Nom. Malgré si haute sorcellerie, L'Hommoncule-apprenti n'apprit pas à parler.

Ses yeux, qui semblaient moins d'homme que de chien, Et bien moins encore de chien que de chose, S'attachaient au Rabbin dans la pénombre close Et douteuse des pièces de sa maison.

Quelque chose ainsi clochait dans le Golem. De fait, le chat du voisin se cachait. A son passage. (Le chat n'est pas dans Sholem, Mais deviné par moi à travers la durée.)

Il élevait vers Dieu des mains filiales, Imitant les dévotions de son propre dieu. Souriant et stupide, il se creusait En concaves révérences orientales.

Le Rabbin le regardait avec tendresse Et non sans quelque horreur. Il se disait : *Comment Ai-je pu engendrer ce lamentable enfant, Et quitter le non-agir, qui est la sagesse ?*

Pourquoi me vint-il l'idée d'ajouter A la série sans fin un symbole inutile ? Et au vain écheveau que l'éternité file, D'autres maux, d'autres causes, d'autres effets ?

A l'heure de l'angoisse et de la clarté vague Sur son Golem, il laissait s'attarder ses yeux. Qui nous dira les sentiments qu'éprouvait Dieu. Contemplant Rabbi Löw, sa créature à Prague ?

Jorge Luis Borges, 1958.

L'auteur et autres textes. Paris, Gallimard 1965. Traduction par Roger Caillois. Je remercie M. Ivan Almeida [4], qui m'a cordialement transmis cette traduction.

[4] *J.L.Borges Center for Studies and Documentation*, Aarhus University

Table des Matières

1 Introduction **11**

2 Apprentissage et réseaux de neurones **13**
2.1 Introduction 13
2.2 Régression et classification 13
2.3 Neurones à fonction d'activation linéaire et sphérique 17
2.3.1 Neurones linéaires 18
2.3.2 Neurones sphériques 20
2.4 Les réseaux de neurones 22
2.4.1 Réseaux de neurones à unités linéaires 23
2.4.2 Réseaux de neurones à unités sphériques 23
2.5 Erreurs d'apprentissage et de généralisation 24
2.6 Capacité d'un réseau 25
2.6.1 Le théorème de Cover 26
2.6.2 L'approche de la mécanique statistique 28
2.6.3 La dimension de Vapnik-Chervonenkis 29
2.7 La généralisation 30
2.7.1 Ambiguïté et séparation linéaire 30
2.7.2 Nombre d'exemples et généralisation 31
2.7.3 Le sur-apprentissage 34
2.8 Conclusion 35

3 Algorithmes d'apprentissage **37**
3.1 Introduction 37
3.2 Méthodes à simplification 38
3.2.1 La Rétropropagation du Gradient 38
3.2.2 Méthodes d'élagage 40
3.3 Algorithmes constructifs 42
3.3.1 Heuristiques avec unités linéaires 42

3.3.2 Heuristiques avec unités sphériques 45
3.4 Conclusion 47

4 L'algorithme d'apprentissage Minimerror 49
4.1 Introduction 49
4.2 Minimerror-L 50
4.2.1 Introduction de deux températures 56
4.2.2 Le critère d'arrêt 57
4.2.3 La normalisation des entrées 57
4.2.4 Exemples 59
4.3 Minimerror-S 70
4.3.1 La stabilité sphérique λ 70
4.3.2 L'initialisation et l'arrêt 74
4.3.3 La normalisation des entrées 75
4.3.4 Exemples de séparations sphériques 76
4.4 Conclusion 80

5 Trois algorithmes constructifs 81
5.1 Introduction 81
5.2 Monoplan 82
5.2.1 Initialisation des unités 84
5.3 NetLines 87
5.3.1 Un exemple 87
5.3.2 Initialisation des unités 88
5.4 NetSphères 89
5.4.1 Initialisation des unités 89
5.4.2 Exemples 90
5.5 Dégénérescence des représentations internes 93
5.6 Généralisation à des problèmes de plus de 2 classes 94
5.7 Conclusions 96

6 Tests et applications 97
6.1 Introduction 97
6.2 Estimations empiriques de l'erreur de généralisation 97
6.3 Problèmes étalon 99
6.3.1 Un cas linéairement séparable : l'identification des échos du sonar 101
6.3.2 Entrées binaires 103
6.3.3 Entrées réelles 107
6.3.4 Problèmes à plus de deux classes 110
6.4 Conclusion 114

7 Conclusion et Perspectives **115**

A Algorithmes **117**

B La convergence de NetLines **123**

C Algorithmes incrémentaux **125**
C.1 L'algorithme *Tiling* 125
C.2 *Sequential Learning* 126
C.3 L'algorithme *Cascade Correlation* 126
C.4 L'algorithme *Offset* 127
C.5 L'algorithme *Upstart* 128
C.6 Arbre neuronal 128
C.7 *Restricted Coulomb Energy (RCE)* 130

D Décomposition en biais et variance **131**
D.1 Cas de la régression 131
D.2 Cas de la classification 133

Nomenclature

$\alpha = P/N$ Taille réduite de l'ensemble d'apprentissage.

$\beta = 1/T$ Inverse de la température.

$\mathcal{L}$ Ensemble d'apprentissage.

$\delta\beta$ Incrément de l'inverse de la température.

$\delta\epsilon$ Incrément du pas de l'algorithme.

η Bruit dans les exemples.

$\lambda^\mu = \tau^\mu \|\vec{w} - \vec{\xi}^\mu\|^2 - \rho^2$ Stabilité sphérique.

$\vec{\xi} = (\xi_1, \xi_2, \cdots, \xi_N)$ Vecteur d'entrées.

$\vec{W}$ Vecteur des poids synaptiques entre les neurones de la couche cachée et la sortie).

$\vec{w_i}$ Vecteur des poids synaptiques entre les entrées et le neurone i.

$\widetilde{\vec{\xi}}$ Vecteur des entrées standardisé.

$\mu = 1, \cdots, P$ Indice des P exemples $\in \mathcal{L}$.

π Polarité des hypersphères : $(\pi = \pm 1)$.

$\Psi = \beta_+/\beta_-$ Rapport des températures dans l'algorithme Minimerror.

ρ Rayon de l'hypersphère.

σ_k Sortie de l'unité cachée k.

τ^μ Classe de l'exemple μ.

ε_g Erreur de généralisation.

ε_t Erreur d'apprentissage.

ζ Sortie du neurone de sortie du réseau.

H Nombre d'unités cachées.

N Dimension de l'espace des entrées.

P Nombre d'exemples de l'ensemble d'apprentissage.

T Température dans l'algorithme Minimerror.

V Fonction de coût.

$\mathcal{P}_{LS}$ Probabilité de séparabilité linéaire d'un ensemble de points.

Chapitre 1

Introduction

La classification est l'attribution d'une classe spécifique à un objet donné. Cette attribution a besoin d'un certain degré d'abstraction pour pouvoir extraire des généralités à partir des exemples dont on dispose.

Pour une machine, la classification de visages, de données médicales, de formes, sont toutes des tâches assez difficiles. Par exemple, dans le cas de la reconnaissance de caractères manuscrits, il est difficile d'énoncer une description générale qui tienne compte de toutes les variations particulières de chaque caractère. Une autre approche qui peut être utilisée pour cette tâche est celle de l'apprentissage. Ainsi, le critère pour décider si une image correspond ou non à une lettre '$\mathcal{A}$' consiste à comparer si cette image est *suffisamment similaire* à des 'A' vus auparavant. De ce point de vue, on ne calcule pas la classification de caractères : elle doit être apprise à partir d'exemples.

Ces dernières années, de nouvelles techniques neuronales d'apprentissage ont été développées. Cet apprentissage avec des réseaux de neurones se fait actuellement en suivant deux approches : certains algorithmes comme la Rétropropagation du Gradient ont besoin d'introduire *a priori* le nombre et la connectivité des unités cachées et déterminer les poids des connexions par minimisation d'un coût. Le réseau ainsi obtenu est éventuellement élagué.

Avec une approche constructive on apprend en même temps le nombre d'unités et les poids, dans le cadre d'une architecture fixée, commençant généralement avec une seule unité.

Le but de cette thèse est de présenter de nouvelles heuristiques pour générer, d'une manière constructive, des réseaux de neurones pour la classification. Elles permettent de générer des réseaux à une seule couche cachée complètement connectée aux unités d'entrée, et un neurone de sortie connecté aux unités cachées. Les neurones cachés et de sortie sont des unités binaires, pouvant faire soit des séparations linéaires, soit des séparations sphériques. Ces heuristiques sont couplées avec

des algorithmes d'apprentissage pour le perceptron, Minimerror-L pour les séparations linéaires et Minimerror-S pour les séparations sphériques.

Ainsi, nous avons développé trois algorithmes constructifs, qui différent suivant le type de neurones cachés et aussi suivant la définition des cibles que ceux-ci doivent apprendre. Pendant le processus d'apprentissage, des neurones cachés entraînés pour apprendre ces cibles vont diminuer le nombre d'erreurs de classification du neurone de sortie. Les réseaux ainsi bâtis ont généralement moins de paramètres (poids) et généralisent mieux que les réseaux entraînés avec d'autres algorithmes.

La thèse est organisée de la manière suivante : dans le Chapitre 2 nous présentons les problèmes de la régression et de la classification, et nous introduisons la notation utilisée dans le reste du mémoire. Des questions sur l'apprentissage et la généralisation sont abordées du point de vue théorique.

La méthode de Rétropropagation du Gradient et ses variations, ainsi que l'état de l'art des heuristiques constructives sont présentés au Chapitre 3.

L'algorithme d'apprentissage Minimerror, pour le perceptron simple, dans ses versions à activation linéaire (Minimerror-L) et à activation sphérique (Minimerror-S) est décrit au Chapitre 4. Les performances sur quelques exemples académiques sont présentées.

Une description formelle des trois nouvelles heuristiques de croissance : Monoplan, NetLines et NetSphères est présentée et illustrée par des exemples simples, au Chapitre 5. Une généralisation qui permet l'application pratique des algorithmes à des problèmes à plus de deux classes y est proposée.

Des comparaisons avec des résultats obtenus par d'autres algorithmes d'apprentissage sur plusieurs problèmes étalon (*benchmarks*), sur la base de l'erreur de généralisation et du nombre de paramètres du réseau, sont présentées au Chapitre 6.

La conclusion est présentée au Chapitre 7, où nous suggérons aussi des développements futurs.

Quatre annexes complètent le manuscrit. Le premier décrit les algorithmes d'apprentissage et les heuristiques de croissance ; le deuxième présente un théorème de convergence ; le troisième présente le pseudo-code de plusieurs algorithmes constructifs et un dernier sur le thème du biais et de la variance.

Chapitre 2

Apprentissage et réseaux de neurones

2.1 Introduction

Dans ce chapitre nous introduisons les paradigmes de l'apprentissage supervisé de la classification de données et de la régression, ainsi qu'une brève introduction aux réseaux de neurones et à la notation utilisée dans ce mémoire.

En effet, les réseaux de neurones ont été appliqués avec succès à l'apprentissage de tâches de classification et d'approximation de fonctions. Pour situer les processus d'apprentissage et de généralisation dans un cadre théorique, deux grandes approches sont présentées : celle issue de la théorie de l'apprentissage de Vapnik (qui considère le cas le pire), et celle de la mécanique statistique, qui considère le cas typique au même sens que l'approche géométrique de Cover. Une discussion sur le surapprentissage et son impact en problèmes de classification et de régression est présentée à la fin du chapitre.

2.2 Régression et classification

- *Qu'est-ce qu'apprendre ?* L'apprentissage est le processus d'adaptation des paramètres d'un système pour donner une réponse désirée à une entrée ou stimulation quelconque.

- *Comment formalise-t-on théoriquement le problème de l'apprentissage ?* Il est habituel de le présenter en utilisant le paradigme du professeur et de l'élève. De façon conceptuelle, on admet qu'il existe un *professeur* qui connaît la relation exacte entre toutes les entrées et leurs sorties, mais le réseau *élève* ne connaît pas cette relation (figure 2.1). Si l'élève et le professeur sont exposés à une même entrée, le professeur est capable d'indiquer à l'élève la réponse désirée. Les paramètres (dans le cas des réseaux de neurones, le nombre de neurones et

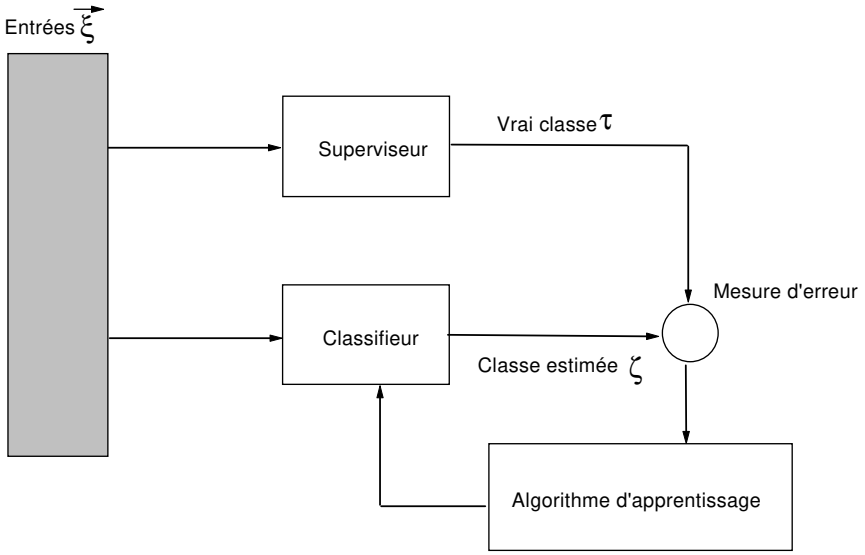

Figure 2.1: Le paradigme (en problèmes de classification) du professeur-élève de l'apprentissage supervisé.

les poids) de l'élève doivent être ajustés pour donner la même réponse que celle du professeur. Cet ajustement, l'apprentissage, est réalisé en général de façon itérative, en minimisant une mesure de l'erreur, jusqu'à ce que le réseau-élève puisse émuler aussi bien que possible le professeur.

- *Comment se présentent les exemples dans les problèmes réels ?* Nous n'avons que des données, dépendantes de chaque problème, et on suppose qu'il existe une fonction sous-jacente entre les entrées et les sorties. Ces données sont aussi connues comme *exemples*.

Nous représentons ces entrées par des vecteurs de N composantes $\vec{\xi} = \{\xi_1, \xi_2, \cdots, \xi_N\}$ qui appartiennent à un espace de dimension N, *l'espace des entrées*. Ces composantes peuvent prendre des valeurs binaires, entières (ordinales ou catégoriques sans un ordre particulier) ou réelles, suivant le problème. La sortie correspondante est représentée par un scalaire τ.

Dans les problèmes de régression, τ prend des valeurs continues. En classification, chaque vecteur $\vec{\xi}$ appartient à une classe $\tau = f(\vec{\xi})$ et alors τ est une variable discrète. En principe, il est possible de traiter des problèmes de classification à un nombre de classes quelconque, mais par souci de simplicité, dans ce qui suit, nous allons nous restreindre à des problèmes à deux classes, codées $+1$ et -1. Nous aborderons au Chapitre 5 les problèmes pratiques posés lorsque le nombre de classes est supérieur à deux.

L'ensemble d'exemples dont on dispose, appelé *l'ensemble d'apprentissage*, consiste en P couples d'entrée-sortie $(\vec{\xi}^{\mu}, \tau^{\mu})$; $\mu = 1, \cdots, P$ où les sorties τ^{μ} (les classes) sont connues. Dans les approches théoriques on dit, de façon imagée, qu'elles ont été fournies par le professeur. Nous dénoterons cet ensemble $\mathcal{L}^{\alpha} = \{(\vec{\xi}^{\mu}, \tau^{\mu}) \ ; \ \mu = 1, \cdots, P\}$, où :

$$\alpha = \frac{P}{N} \tag{2.1}$$

que nous appellerons *taille réduite* de l'ensemble d'apprentissage, représente la proportion d'exemples par rapport à la dimension N des entrées. Dans les approches théoriques décrites ci-après, on suppose généralement que les exemples sont des variables aléatoires indépendantes et identiquement distribuées (*i.i.d.*), dont la densité de probabilité est $\mathcal{P}(\vec{\xi}^{\mu}, \tau^{\mu})$.

La régression

Un problème de régression peut aussi être traité comme un problème d'apprentissage supervisé. Si on suppose qu'il y a une certaine relation entre les entrées $\vec{\xi}$ et la sortie τ, décrite par la fonction :

$$\tau^{\mu} = f(\vec{\xi}^{\mu}) + \eta^{\mu} \tag{2.2}$$

où η^{μ} représente le bruit (à cause des possibles erreurs commises dans les mesures) des échantillons, supposé de moyenne nulle et indépendant des $\vec{\xi}^{\mu}$. La fonction f est inconnue. Le problème consiste à approcher la fonction f en ne disposant que d'un ensemble d'apprentissage $\mathcal{L}^{\alpha}$. Au lieu de chercher une solution qui passe par tous les points ou *interpolation*, on va plutôt chercher des solutions approchées avec un certain degré de tolérance ; c'est-à-dire, une *régression*.

Un problème d'approximation est mal posé [16] s'il ne vérifie pas toutes les conditions suivantes :

- Il existe une solution au problème.
- La solution est unique.
- La solution est *stable* : de petites perturbations dans les entrées entraînent de petites perturbations de la solution.

Dans le cas de l'interpolation à partir d'un ensemble de données, il peut ne pas exister de solution ; par exemple si deux vecteurs d'entrée égaux ont des sorties différentes. En outre, l'information présente peut ne pas être suffisante et finalement la solution trouvée peut s'avérer instable, comme dans le cas de l'approximation polynomiale.

La classification

Nous supposons qu'il existe une certaine relation entre les entrées et la sortie qui est décrite par la fonction :

$$\tau^\mu = f(\vec{\xi}^\mu + \vec{\eta}^\mu) \tag{2.3}$$

Dans ce cas, la sortie τ ne prend que des valeurs discrètes, qui correspondent à des étiquettes ou classes. Ici, la définition du bruit η est prise en compte dans un autre sens : en effet, le bruit n'agit pas sur les sorties de la même manière que pour la régression (puisqu'il s'agit ici de classes), mais plutôt dans les entrées. L'hypothèse d'un petit bruit additif à la sortie comme en (2.2) n'est plus valable, car la somme $f(\vec{\xi}^\mu)+\vec{\eta}^\mu$ doit être $+1$ ou -1, ce qui suppose des contraintes très fortes sur $\vec{\eta}^\mu$. Nous allons considérer que les classes des exemples de $\mathcal{L}^\alpha$ ont été correctement attribués.

Nous allons illustrer cette formulation par un exemple issu du domaine médical, celui du diagnostic du cancer du sein, qui est discuté en détail au Chapitre 6. Depuis 1988, l'Université du Wisconsin alimente une base avec des données médicales concernant des prélèvements cytologiques [106] (épaisseur, uniformité de la taille et de la forme des cellules, etc.) avec leurs diagnostics : bénin ou malin. Chaque exemple est décrit par $N = 9$ attributs codés avec des entiers qui prennent des valeurs entre 1 et 10. La base contient 699 exemples, dont 16 sont incomplets, car il leur manque un attribut.

Attributs	Signification
ξ_1	Épaisseur de l'échantillon
ξ_2	Uniformité de la taille
ξ_3	Uniformité de la forme
ξ_4	Adhésion marginale
ξ_5	Taille cellule épithéliale
ξ_6	Noyaux
ξ_7	Chromatine terne
ξ_8	Nucleoli normal
ξ_9	Mitose

Tables 2.1: Les 9 attributs des données pour le diagnostic du cancer du sein (données de l'Université du Wisconsin).

Ce problème peut être formalisé comme un problème de classification par apprentissage supervisé, avec $P = 699$ exemples $\vec{\xi}^\mu = \{\xi_1^\mu, \xi_2^\mu, \cdots, \xi_9^\mu\}$; $\mu = 1, \cdots, P$,

dans un espace de dimension 9, dont la classe est $\tau = -1$ pour les cas bénins, $\tau = +1$ pour les cas malins (l'attribution de ± 1 pour chaque classe est arbitraire). Dans le tableau 2.1 nous présentons les 9 attributs mesurés. Cet exemple sera d'ailleurs utilisé comme problème étalon au Chapitre 6, où nous considérons le problème des attributs manquants.

2.3 Neurones à fonction d'activation linéaire et sphérique

Le neurone formel (ou perceptron) de McCulloch et Pitts [69] utilisé actuellement dans la plupart des modèles connexionnistes, est un automate à deux états : *actif* ou *inactif*, qu'on peut modéliser par une variable binaire codée $\zeta = \{-1, +1\}$ ou par $s = \{0, 1\}$. Evidemment les deux conventions sont équivalentes, le passage de l'une à l'autre pouvant se faire par le changement de variables :

$$\zeta = 2s - 1 \tag{2.4}$$

Dans ce qui suit, nous adopterons la première convention, et nous appellerons tout simplement ζ l'état de sortie du neurone. Cet état dépend de l'ensemble des N signaux d'entrée $\vec{\xi}$, comme on le montre dans la figure 2.2. Chaque lien entre un signal d'entrée i et le neurone représente un *poids* ou *efficacité synaptique*, qui est caractérisé par un nombre réel w_i, appelé parfois *couplage*. Ces poids peuvent prendre des valeurs positives ou négatives.

Le neurone formel est donc constitué des éléments suivants (figure 2.2) :

- 1. Les entrées $(\xi_i \ ; \ i = 1, \cdots, N)$ provenant de N sources externes.
- 2. Les connexions ou poids $w_i, i = 1, \cdots, N$, et le seuil θ
- 3. La sortie ζ, qui est fonction des entrées et des poids.

Suivant la façon de calculer la sortie, nous allons considérer deux sortes de neurones : linéaires et sphériques.

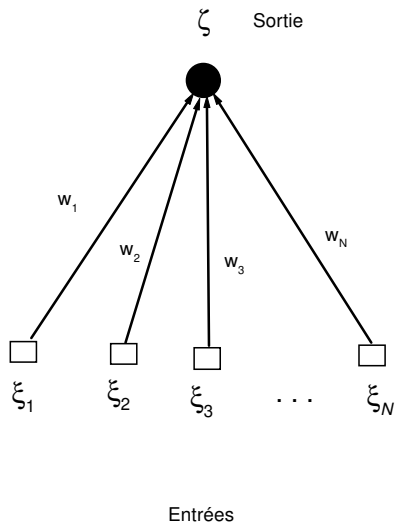

Figure 2.2: Le perceptron

2.3.1 Neurones linéaires

Le neurone linéaire calcule un *champ* ou potentiel défini comme la somme des signaux d'entrée $\vec{\xi}$ pondérés par les poids correspondants :

$$h = \sum_{i=1}^{N} w_i \xi_i \tag{2.5}$$

Si ce champ est plus grand qu'un certain seuil θ, le neurone est actif. La sortie ζ du neurone en fonction de son potentiel est :

$$\zeta = f\left(\sum_{i=1}^{N} w_i \xi_i - \theta\right) = f\left(\vec{w} \cdot \vec{\xi}\right) \tag{2.6}$$

f est appelé fonction d'activation ou fonction de transition du neurone. Pour simplifier la notation, la quantité $-\theta$ peut être introduite dans la somme en la considérant comme un poids supplémentaire $w_0 = -\theta$ provenant d'une entrée constante, $\xi_0 \equiv 1$, et s'appelle *biais* :

$$\zeta = f\left(\sum_{i=0}^{N} w_i \xi_i\right) \tag{2.7}$$

En ce qui suit, nous allons garder la notation $\vec{w} \cdot \vec{\xi}$ pour le produit scalaire quand on considère le poids w_0.

Différents choix de fonctions d'activation sont possibles. Les neurones binaires ont des fonctions d'activation discontinues. Suivant que les états sont codés $\{0, 1\}$ ou $\{-1, +1\}$, on a la fonction échelon, appelée aussi fonction de Heaviside $\Theta(x)$:

$$f(x) = \Theta(x) \equiv \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{autrement} \end{cases} \tag{2.8}$$

ou la fonction signe :

$$f(x) = \text{signe}(x) \equiv \begin{cases} +1 & \text{si } x \geq 0 \\ -1 & \text{autrement} \end{cases} \tag{2.9}$$

Par contre, dans les cas où les états des neurones sont continus, on utilise des fonctions sigmoïdes. On appelle sigmoïde toute fonction qui est croissante, continue et bornée. Suivant le codage, on utilise soit la fonction logistique :

$$f(x) \equiv \frac{1}{1 + \exp(-\beta x)} \tag{2.10}$$

pour des états dans l'intervalle $[0, 1]$, soit la tangente hyperbolique :

$$f(x) = \tanh \beta x \equiv \frac{\exp(\beta x) - \exp(-\beta x)}{\exp(\beta x) + \exp(-\beta x)} \tag{2.11}$$

pour les neurones dont les états se trouvent dans $[-1, +1]$. Le paramètre β représente un gain. En faisant varier β, on peut obtenir différentes formes de sigmoïdes. A la limite, si on fait $\beta \to \infty$, les différentes sigmoïdes (2.10) et (2.11) tendront vers les fonctions (2.8) et (2.9) respectivement.

Il est possible d'interpréter l'équation de sortie du perceptron (2.7) en considérant, dans un espace de dimension $N + 1$, le vecteur $\vec{\xi} = \{\xi_0, \xi_1, \xi_2, \cdots, \xi_N\}$ et le vecteur $\vec{w} = \{w_0, w_1, w_2, \cdots, w_N\}$. Si leur produit scalaire est positif, alors le point $\vec{\xi}$ appartient à la classe $+1$, sinon il appartient à la classe -1. Alors on dit que l'hyperplan défini par sa normale $\{w_1, w_2, \cdots, w_N\}$ et sa distance à l'origine $\theta = -w_0$ fait une séparation linéaire des points $\vec{\xi}$ en deux classes, car un hyperplan est une fonction linéaire dans l'espace des entrées.

Si tous les exemples d'un ensemble d'apprentissage peuvent être séparés par un hyperplan normal à $\vec{w}$, alors on dit que le problème est *linéairement séparable* (LS).

Si on utilise comme activation la fonction de Heaviside (2.8) ou la fonction signe (2.9), le neurone est binaire car il n'a que deux sorties possibles. Intuitivement,

l'utilisation de ce type de neurone est naturelle dans les problèmes de classification. La sortie du perceptron à une entrée $\vec{\xi}^{\mu}$ de l'ensemble d'apprentissage est correcte si le produit $\tau^{\mu}(\vec{w}\cdot\vec{\xi}^{\mu}) > 0$. La quantité $\tau^{\mu}(\vec{w}\cdot\vec{\xi}^{\mu})$ peut être aussi grande que l'on veut par simple multiplication de $\vec{w}$ par une constante, ce qui ne modifie pas la qualité de la solution trouvée. C'est pourquoi on est amené à définir la *stabilité* [56], aussi appelée *marge* [102], de l'exemple μ comme :

$$\gamma^{\mu}(\vec{w}) = \tau^{\mu}\sum_{i=0}^{N}\frac{w_i}{\|\vec{w}\|}\xi_i^{\mu} \equiv \tau^{\mu}\frac{\vec{w}\cdot\vec{\xi}^{\mu}}{\|\vec{w}\|} \tag{2.12}$$

La valeur absolue de la stabilité $|\gamma^{\mu}|$, mesure la distance de l'exemple μ à l'hyperplan séparateur, qui est normale à $\vec{w}$. Le signe de γ^{μ} est positif si l'exemple est bien classé, négatif autrement.

Une grande stabilité positive assure une certaine robustesse [1] de la réponse du perceptron. La division par la norme de $\vec{w}$ évite que de grandes stabilités soient produites par un simple changement d'échelle des poids. Dans la suite nous adopterons toujours la normalisation :

$$\|\vec{w}\| = \sqrt{\sum_{i=0}^{N} w_i^2} = \sqrt{N+1} \tag{2.13}$$

2.3.2 Neurones sphériques

Les neurones décrits précédemment effectuent une transformation par une fonction f du champ (2.5) qui est le produit scalaire entre leur vecteur de poids synaptiques $\vec{w}$ et le vecteur d'entrée : $f(\vec{\xi},\vec{w})$.

Il existe un autre type de neurones dont la fonction d'activation s'applique à la distance entre ces deux vecteurs. Ces neurones sont généralement appelés neurones *fonction de base radiale*. Leurs fonctions de transition, appelées "noyaux", sont des fonctions continues de la forme $f(\|\vec{\xi}-\vec{w}\|)$ où $\vec{w}$ est le centre de la fonction de base, $\vec{\xi}$ le vecteur d'entrée et :

$$\|\vec{x}\|^2 = \sum_{i=1} N x_i^2 \tag{2.14}$$

[1] Un apprentissage robuste permet de donner une sortie correcte même s'il y a des altérations des poids mémorisés. La robustesse de la réponse du perceptron fera l'objet d'une de nos expériences (Chapitre 4).

La fonction noyau la plus couramment utilisée est le noyau gaussien :

$$f(\|\vec{\xi} - \vec{w}\|) = \exp\left(-\frac{1}{2}\|\vec{\xi} - \vec{w}\|^2\right) \tag{2.15}$$

Mais on peut aussi bien utiliser une fonction échelon de la distance :

$$f(\|\vec{\xi} - \vec{w}\|) = \begin{cases} +1 & \text{si } \|\vec{\xi} - \vec{w}\|^2 \geq \theta^2 \\ -1 & \text{autrement} \end{cases} \tag{2.16}$$

Certains auteurs ont proposé l'utilisation de neurones dont la fonction d'activation est une fonction sigmoïde (continue) de la distance $\|\vec{\xi} - \vec{w}\|$. Dans ce travail, nous allons considérer des neurones binaires, dont la fonction d'activation est la même que pour les neurones linéaires, mais appliquée au carré de la distance [2]. Ceci revient à remplacer dans (2.5) le champ par le *champ radial* :

$$h^r = \sum_{i=1}^{N} (\xi_i - w_i)^2 = \|\vec{\xi} - \vec{w}\|^2 \tag{2.17}$$

Dans le cas de la fonction d'activation (2.16), l'état du neurone sera $+1$ si l'entrée $\vec{\xi}$ se trouve à l'extérieur de l'hypersphère de rayon θ centrée en $\vec{w}$, et -1 si $\vec{\xi}$ est à l'intérieur de l'hypersphère. Donc, ce type de neurones fait des séparations sphériques. Il est important de noter que le nombre de paramètres d'un neurone sphérique est le même que celui d'un neurone linéaire : le centre de la sphère est défini par les poids $\vec{w}$, son rayon par le seuil θ. D'autre part,

$$\begin{aligned} h^r &= \|\vec{\xi} - \vec{w}\|^2 = \|\vec{\xi}\|^2 + \|\vec{w}\|^2 - 2\vec{\xi} \cdot \vec{w} \\ &= \vec{\xi} \cdot (-2\vec{w}) - \left[-\|\vec{\xi}\|^2 - \|\vec{w}\|^2\right] \end{aligned} \tag{2.18}$$

Dans les cas où les entrées sont binaires, $\|\vec{\xi}\|^2 = N$ est constant. Dans ce cas, les neurones sphériques sont équivalents à des neurones linéaires avec des poids $-2\vec{w}$ et seuil $\theta^2 - N - \|\vec{w}\|^2$. Si les entrées sont à composantes réelles, l'équivalence n'est plus vérifiée, et les neurones sphériques permettent d'élargir le domaine de fonctions réalisables. L'intérêt de leur introduction est qu'ils peuvent être entraînés avec les mêmes algorithmes que les neurones linéaires, et être utilisés dans les heuristiques incrémentales, comme nous le montrerons au Chapitre 5.

[2] D'autres définitions sont possibles, comme nous le verrons au Chapitre 4.

2.4 Les réseaux de neurones

Les perceptrons linéaires peuvent apprendre correctement seulement les problèmes dont les exemples d'entrée sont linéairement séparables. Les neurones sphériques peuvent apprendre des séparations dont les exemples sont séparables par des hypersphères. Pour des tâches de classification plus complexes, il faut utiliser des réseaux comportant plusieurs unités interconnectées. Un réseau est défini par son *architecture* : le nombre d'unités, le nombre de poids et la disposition des entrées et sorties. L'architecture peut être assez variée : les unités peuvent être ou non complètement connectées, disposées en couches et d'autres possibles combinaisons.

Il existe une architecture particulièrement utilisée : celle des réseaux sans rétroaction [3]. Dans cette architecture, les unités sont arrangées en couches successives avec des connexions allant d'une couche à la suivante, comme dans la figure 2.3. Les données sont traitées successivement par les couches cachées. La dernière couche fournit la réponse et est appelée couche de sortie. Bien qu'une seule couche cachée suffise pour réaliser n'importe quelle fonction des entrées [51, 21], le nombre optimal d'unités de la couche cachée reste inconnu.

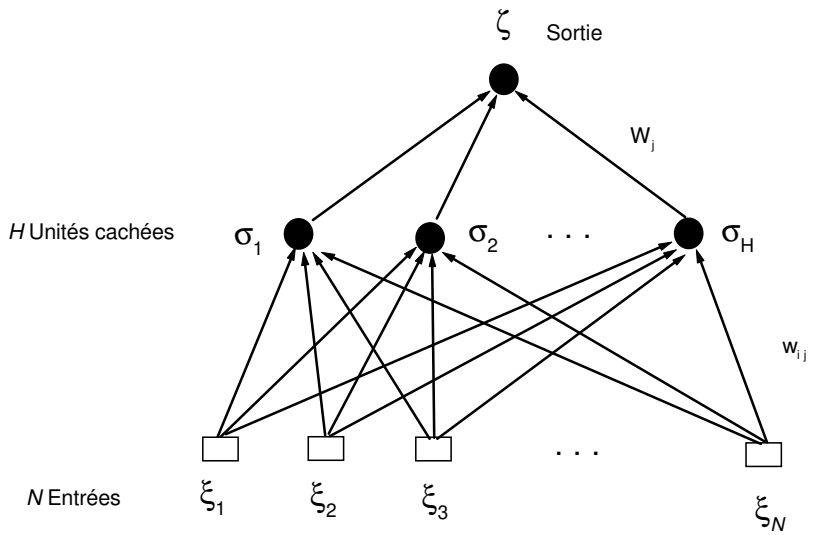

Figure 2.3: Un perceptron multicouche à une seule couche cachée. Il est composé de N entrées ξ_i ; $i = 1, \cdots, N$, de H unités cachées σ_j ; $j = 1, \cdots, H$, des poids w_{ij} entre les entrées et les unités cachées, et des poids W_j entre les unités cachées et la sortie finale ζ. Les biais ne sont pas montrés.

Les réseaux de neurones ont été appliqués avec succès à l'apprentissage de tâches de classification et à l'approximation de fonctions à partir d'exemples. Cet appren-

[3] Nous traduisons ainsi le terme anglais *feedforward*.

tissage se fait suivant deux approches : soit on fixe l'architecture et on apprend les poids, comme le fait la méthode très populaire de *Rétropropagation du Gradient* [4] ; soit on apprend de manière constructive le nombre d'unités et les poids. Dans ce mémoire nous utilisons l'approche constructive pour générer des réseaux de neurones à une seule couche cachée.

Pour être plus précis, dans le paragraphe suivant nous introduisons la notation utilisée par la suite.

2.4.1 Réseaux de neurones à unités linéaires

Soit un réseau à H neurones cachés connectés aux $N+1$ entrées ξ_i ; $i = 0, \cdots, N$ (N entrées plus le biais $\xi_0 \equiv 1$), et une unité de sortie. L'unité de sortie est connectée aux unités cachées avec des poids $\vec{W} = \{W_0, W_1, \cdots, W_H\}$ où W_0 est le biais et H le nombre d'unités cachées. Chaque unité cachée $1 \leq k \leq H$ est à son tour connectée aux entrées avec des poids synaptiques $\vec{w}_k = \{w_{k0}, w_{k1}, \cdots, w_{kN}\}$, w_{k0} étant le biais.

Si on utilise des unités cachées binaires, l'état $\vec{\sigma} = \{\sigma_0, \sigma_1, \cdots, \sigma_H\}$ des neurones cachés avec $\sigma_0 \equiv 1$, est considéré comme la *Représentation Interne* (RI) de dimension H associée par le réseau à l'exemple d'entrée $\vec{\xi}$. Etant donné une entrée $\vec{\xi}$ que le réseau doit classer, l'état σ_k du neurone caché k ($1 \leq k \leq h$) est donné par :

$$\sigma_k = f\left(\sum_{i=0}^{N} w_{ki}\xi_i\right) \equiv f\left(\vec{w}_k \cdot \vec{\xi}\right) \tag{2.19}$$

La réponse du réseau ζ, est donnée par le perceptron de sortie :

$$\zeta = f\left(\sum_{k=0}^{H} W_k\sigma_k\right) \equiv f\left(\vec{W} \cdot \vec{\sigma}\right) \tag{2.20}$$

où f est la fonction d'activation du neurone.

2.4.2 Réseaux de neurones à unités sphériques

L'architecture d'un réseau de neurones avec des unités d'activation sphérique est appelée réseau à fonctions de base radiale ou réseau *RBF*. Un réseau sans rétroaction de ce type comporte généralement une seule couche d'unités cachées dont la sortie dépend, comme nous l'avons déjà mentionné, de la distance de l'entrée à un centre. L'unité de sortie est un perceptron linéaire binaire connecté aux unités cachées.

[4] Cet algorithme sera présenté au Chapitre 3.

Pour une entrée $\vec{\xi}$, l'état σ_k du neurone caché k $(1 \leq k \leq h)$ est donné par :

$$\sigma_k = f(\|\vec{\xi} - \vec{w}_k\|) \tag{2.21}$$

où $\vec{w}_k$ est le centre de la fonction *RBF* correspondant au neurone k et f est la fonction d'activation (2.15) ou (2.16). Dans ce mémoire nous utilisons la deuxième. La réponse du réseau ζ, est donnée par (2.20).

2.5 Erreurs d'apprentissage et de généralisation

Comme nous l'avons vu, la caractéristique principale de l'apprentissage supervisé est que l'on dispose d'un ensemble d'exemples des données, les entrées avec leurs classes ou sorties désirées. Ces exemples constituent l'ensemble d'apprentissage $\mathcal{L}^\alpha$ qui sert à adapter l'architecture et les poids du réseau pour réaliser une fonction des entrées. La valeur de cette fonction, la réponse ou sortie du réseau, est la classe que le réseau attribue à l'entrée correspondante. En général, la réponse à certains exemples ainsi qu'à des nouvelles données, peut être incorrecte, ce qui pose le problème de la généralisation.

L'*erreur d'apprentissage* ε_t est la fraction d'exemples de l'ensemble d'apprentissage que l'élève classe mal. Puisque les exemples sont tirés au hasard, suivant une densité de probabilité $\mathcal{P}(\vec{\xi}, \tau)$, ε_t est une variable aléatoire, et nous nous intéressons à son espérance $\langle \varepsilon_t \rangle$. Formellement :

$$\langle \varepsilon_t \rangle = \sum_{\mu=1}^{P} \int \mathcal{P}(\vec{\xi}^\mu, \tau^\mu)\ e(\tau^\mu, \zeta^\mu)\ d\vec{\xi}^\mu d\tau^\mu \tag{2.22}$$

où $e(\tau^\mu, \zeta^\mu)$ est la mesure d'erreur sur l'exemple μ. En régression on pose généralement :

$$e(\tau^\mu, \zeta^\mu) \propto (\tau^\mu - \zeta^\mu)^2 \tag{2.23}$$

Cette mesure est aussi correcte dans les problèmes de classification, mais dans ce dernier cas nous préférons la notation alternative :

$$e(\tau^\mu, \zeta^\mu) = \Theta(-\tau^\mu \zeta^\mu) \tag{2.24}$$

bien adaptée à notre codage binaire des classes en ± 1, où Θ est donné par (2.8). En effet, si la sortie du réseau est correcte, $\zeta^\mu \tau^\mu = +1$, et $e = 0$, tandis que si $\zeta^\mu \neq \tau^\mu$

alors $\zeta^\mu \tau^\mu = -1$ et $e = 1$.

La quantité $\langle \varepsilon_t^\alpha \rangle$ est indépendante de la réalisation particulière des exemples : elle ne dépend que de la taille réduite α, et permet de caractériser le réseau et l'algorithme d'apprentissage en fonction uniquement de α, connaissant la distribution de probabilité $\mathcal{P}(\vec{\xi}, \tau)$.

Après l'apprentissage, il est important de mesurer la capacité du classifieur à effectuer une classification correcte sur de nouvelles données d'entrée. L'espérance de l'erreur sur de nouvelles données est appelée *erreur de généralisation*. Elle est définie par :

$$\varepsilon_g = \int \mathcal{P}(\vec{\xi}, \tau)\ e(\tau, \zeta)\ d\vec{\xi} d\tau \tag{2.25}$$

où $e(\tau, \zeta)$ est la mesure de l'erreur définie par (2.23) pour les problèmes de régression ou par (2.24) en cas de classification.

Le but de l'apprentissage consiste à minimiser l'erreur de généralisation (2.25), mais la distribution de probabilité $\mathcal{P}(\vec{\xi}, \tau)$ est inconnue, et on n'a que l'information contenue dans l'ensemble d'apprentissage $\mathcal{L}^\alpha$.

2.6 Capacité d'un réseau

Le paradigme professeur-élève de l'apprentissage supervisé permet d'étudier un grand nombre de questions. Dans le cas général, le professeur fournit à l'élève un nombre limité d'exemples. Si ce nombre est suffisamment petit, ou si la complexité de l'élève est suffisamment grande, il sera capable de les apprendre sans pour autant saisir la *structure* du professeur. Chaque élève a une *capacité* d'apprentissage *par cœur*, qu'il convient de connaître. Pour étudier cette capacité, on considère que le professeur attribue à chaque exemple $\vec{\xi}^\mu$ une classe τ^μ choisie au hasard. Si l'architecture de l'élève diffère de celle du professeur, il est probable que si le nombre d'exemples est suffisamment grand (c'est-à-dire supérieur à la capacité), l'élève sera incapable d'apprendre tous les exemples.

Plus précisément, la capacité d'un réseau de neurones est la plus grande quantité d'exemples $P_{max} = \alpha_c N$, que le réseau peut apprendre, pour laquelle $\langle \varepsilon_t^\alpha \rangle = 0$. Or, $\langle \varepsilon_t^\alpha \rangle$ dépend de l'architecture du réseau et de la distribution des exemples $\mathcal{P}(\vec{\xi}, \tau)$. Etant donné une architecture et une distribution, il n'est pas certain que l'algorithme d'apprentissage utilisé soit capable d'atteindre cette capacité. Nous distinguerons la capacité du réseau $\alpha_c^R N$ de la capacité de l'algorithme d'apprentissage $\alpha_c^A N$;

clairement, $\alpha_c^A \leq \alpha_c^R$

Deux situations peuvent se poser : si l'on veut mémoriser les αN exemples (les apprendre *par cœur*) il faut un réseau avec $\alpha_c^R > \alpha$. Par contre, si l'on veut extraire l'information des données pour ensuite généraliser, il faut avoir $\alpha > \alpha_c^R$. D'où l'importance de connaître la capacité $\alpha_c^R N$ des réseaux de neurones.

2.6.1 Le théorème de Cover

La capacité d'un réseau peut être calculée si l'on sait répondre à la question suivante :

Etant donné P points $\vec{\xi}^{\mu}\,(\mu = 1, \cdots, P)$ de R^N, combien de dichotomies [5] de ces P points sont réalisables par le réseau ?

La réponse exacte à cette question est connue seulement pour le réseau le plus simple possible, le perceptron linéaire. Cover [19] a calculé la capacité du perceptron, en se servant d'arguments géométriques. Il a montré que le nombre de dichotomies produites par des séparations linéaires de P points en position générale [6] dans un espace de dimension N est :

$$C(P,N) = \begin{cases} 2\sum_{i=1}^{N} \binom{P-1}{i} & \text{si } P > N \\ 2^P & \text{si } P \leq N \end{cases} \tag{2.26}$$

Puisque le nombre total de dichotomies possibles de P points est 2^P, la probabilité de séparabilité linéaire $\mathcal{P}_{LS}$ qu'un perceptron à N entrées sépare P points en position générale est :

$$\mathcal{P}_{LS}(P,N) = \frac{C(P,N)}{2^P} = \begin{cases} (\frac{1}{2})^{P-1} \sum_{i=1}^{N} \binom{P-1}{i} & \text{si } P > N \\ 1 & \text{si } P \leq N \end{cases} \tag{2.27}$$

Dans la figure 2.4 nous avons représenté la probabilité $\mathcal{P}_{LS}(P,N)$ en fonction de $\alpha = P/N$. A partir de cette figure, on peut voir que :

- Pour N fini :

[5] Une dichotomie est un étiquetage binaire particulier des P points.

[6] P points sont en position générale en R^N si :
(a) $P > N$: si et seulement si aucun sous-ensemble de N points ne se trouve sur un hyperplan.
(b) Si : $P \leq N$ si aucun hyperplan ne contient les P points.
Si les exemples ont une distribution uniforme, on a une forte probabilité qu'ils soient en position générale.

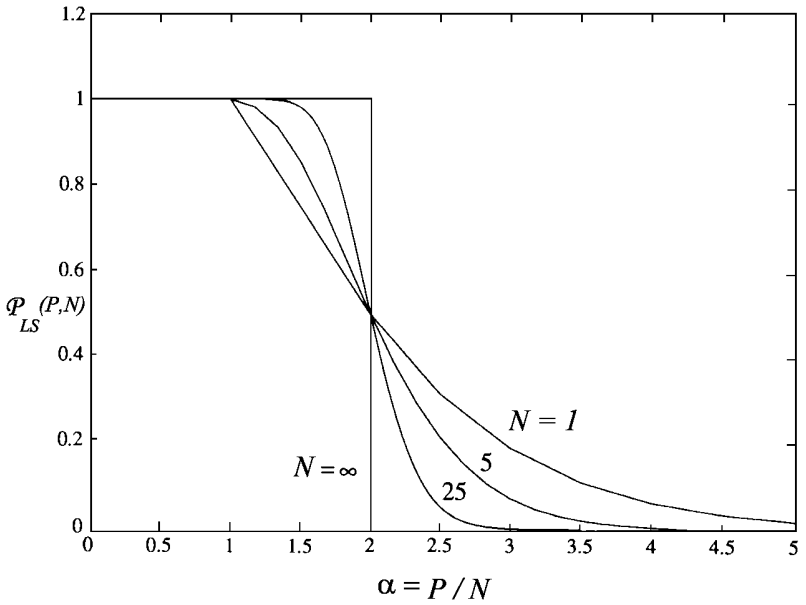

Figure 2.4: Capacité d'un perceptron linéaire (Cover).

- Si $\alpha < 1$ $(P < N)$ alors $\mathcal{P}_{LS}(P,N) = 1$: un perceptron peut toujours séparer les P points.
- Si $\alpha > 1$ $(P > N)$ alors la $\mathcal{P}_{LS}(P,N)$ décroît avec α.

La valeur de $P = \alpha N$ au delà de laquelle $\mathcal{P}_{LS}(P,N)$ devient inférieure à 1 est la dimension de Vapnik-Chervonenkis d_{vc} , comme on le verra, dans la section (2.6.3).

- Dans la limite où $N \to \infty$ et $P \to \infty$ avec $\alpha = P/N$ fini :
  - Si $\alpha < 2$ $(P < 2N)$, alors : $\lim_{N\to\infty} \mathcal{P}_{LS}(P,N) = 1$
  - Si $\alpha > 2$ $(P > 2N)$, alors : $\lim_{N\to\infty} \mathcal{P}_{LS}(P,N) = 0$

Dans la limite $N \to \infty$, la probabilité (2.27) présente une discontinuité à $\alpha_c = 2$. La probabilité qu'un ensemble de moins de $2N$ points en position générale soit linéairement séparable est 1, tandis que pour $P > 2N$ elle s'annule. Ces prédictions avec probabilité 1 décrivent ce que l'on appelle le cas typique. Bien que dans la limite où $N \to \infty$ on a une transition nette, la figure 2.4 montre que pour N grand mais fini, $\mathcal{P}_{LS}(P,N) \approx 1$ bien au delà de $\alpha = 1$.

Malgré sa simplicité, l'approche de Cover est difficilement généralisable au calcul de la capacité de réseaux plus complexes. Mitchison et Durbin [63], ont obtenu une capacité de l'ordre de :

$$\alpha_c \simeq O(H \log H) \tag{2.28}$$

pour un réseaux à N entrées, avec une couche de H unités cachées.

2.6.2 L'approche de la mécanique statistique

L'approche théorique de l'apprentissage par la mécanique statistique a été developpée par Gardner [37] et Gardner et Derrida [41]. En particulier, ces auteurs ont calculé la capacité typique d'un perceptron. Cette méthode est plus compliquée que celle de Cover, mais elle peut être généralisée à l'étude de réseaux plus complexes et à celle des algorithmes d'apprentissage.

Dans cette approche, on considère un réseau de neurones avec une architecture donnée, caractérisé par ses poids, que nous noterons génériquement comme un vecteur $\vec{w}$. Une réalisation particulière des poids $\vec{w}$ peut être vue comme un point dans l'espace de tous les poids possibles. Chaque vecteur $\vec{w}$ représente un réseau qui attribue une sortie ζ à chaque exemple $\vec{\xi}$. L'idée de base pour déterminer la capacité d'un réseau, consiste à calculer la fraction du volume dans l'espace des poids, qui réalise la fonction désirée des entrées-sorties pour l'ensemble d'apprentissage, moyennée sur la distribution des exemples. Intuitivement on s'attend à ce que cette fraction de volume soit une fonction décroissante du nombre réduit d'exemples α : si l'on a peu d'exemples, il y a beaucoup de vecteurs poids différents qui donnent la sortie correcte aux exemples. Lorsque α augmente, cette fraction de volume décroît et tend vers zéro. La valeur de α pour laquelle elle s'annule correspond à la capacité typique du réseau : pour des α plus grands, seulement des points isolés (de volume nul) dans l'espace des poids correspondent à des réseaux capables d'apprendre l'ensemble d'apprentissage. Ce raisonnement est très proche de celui de Cover, mais le calcul est très différent, car il utilise des outils de la mécanique statistique des systèmes désordonnés. Nous décrivons qualitativement la formulation dans le cas d'un perceptron simple. Etant donné un ensemble d'apprentissage $\mathcal{L}^\alpha$, on veut calculer la fraction du volume de l'espace des poids qui satisfait les P inégalités suivantes :

$$\tau^\mu \frac{\vec{w} \cdot \vec{\xi}^\mu}{\|\vec{w}\|} > \kappa \geq 0 \ ; \ \ \mu = 1, \cdots, P \tag{2.29}$$

Puisque ces inégalités ne dépendent pas de la norme des poids, on peut se limiter à considérer des poids de norme donnée. Ceci correspond à limiter l'espace des poids à une hypersphère de rayon donné. Il est habituel de prendre $\|\vec{w}\| = \sqrt{N}$. Le fait d'imposer une marge $\kappa > 0$ donne une certaine robustesse de la sortie du réseau par rapport à de petites perturbations des entrées. Pour $\kappa = 0$ on retrouve les inégalités

habituelles que doivent satisfaire les poids.

La quantité fondamentale du calcul de Gardner et Derrida, la fraction du volume dans laquelle (2.29) est vérifiée, s'écrit :

$$V = \frac{\int d\vec{w} \left(\prod_\mu \Theta(\zeta^\mu N^{-1/2} \sum_j w_j \xi_j^\mu - \kappa) \right) \delta(\sum_j w_j^2 - N)}{\int d\vec{w} \delta(\sum_j w_j^2 - N)} \tag{2.30}$$

Ce volume d'espace dépend de l'ensemble d'apprentissage. Sa valeur *typique* s'obtient en moyennant son logarithme sur la distribution des exemples, $\mathcal{P}(\vec{\xi}^\mu, \tau^\mu)$, puis prenant l'exponentielle :

$$V_{typique} = \mathrm{e}^{\langle \log V \rangle} \tag{2.31}$$

dans la limite $N \to \infty, P \to \infty, \alpha = P/N$ avec α fini, qui s'appelle la limite thermodynamique. C'est la même limite que celle considérée par Cover. En [41] et [49] le lecteur intéressé peut trouver tous les détails du calcul. Un des résultats les plus importants est que dans la limite thermodynamique $V_{typique} = 0$ pour $\alpha > \alpha_c(\kappa)$, avec :

$$\alpha_c(\kappa) = \left[\int_{-\kappa}^{\infty} \frac{dt}{\sqrt{2\pi}} \mathrm{e}^{-t^2/2} (t+\kappa)^2 \right]^{-1} \tag{2.32}$$

L'équation (2.32) donne la capacité pour κ fixé. Si $\kappa = 0$, on retrouve le résultat de Cover, $\alpha_c(0) = 2$.

Pour des réseaux plus complexes, on connaît la capacité de réseaux à N entrées avec une couche cachée à H unités, pour des cas particuliers [59]. Si le neurone de sortie met en œuvre la parité de la représentation interne, on a $\alpha_c^{\mathrm{par}} \simeq H \log H$ comme le résultat de Mitchison et Durbin (eq. 2.28). Si le neurone de sortie implémente le vote des neurones cachés (*comité de machines*), la capacité est inférieure :

$$\alpha_c^{\mathrm{com}} \simeq \frac{8\sqrt{2}}{\pi H} \sqrt{\log H} \tag{2.33}$$

2.6.3 La dimension de Vapnik-Chervonenkis

La dimension de Vapnik-Chervonenkis d_{vc} d'un classifieur [103, 100] est le plus grand nombre d'exemples P tel que toutes les 2^P dichotomies possibles sont réalisables par le classifieur. La dimension de Vapnik-Chervonenkis des réseaux complexes est en

général inconnue. Le calcul de Cover (2.27) montre que pour un perceptron à N entrées, $d_{vc} = N$, ce qui correspond à $\alpha_{vc} = 1$. Si l'on introduit un biais, la dimension de l'espace d'entrées est $N + 1$ et alors $d_{vc} = N + 1$.

Pour un réseau de N entrées, à une seule couche cachée avec H unités et une unité de sortie (voir figure 2.3), avec un nombre de poids N_w (biais inclus) égal à :

$$N_w = (N + 1)H + (H + 1) \tag{2.34}$$

Baum et Haussler [7] montrent que :

$$2\lfloor \frac{H}{2} \rfloor N \ \leq \ d_{vc} \ \leq \ 2N_w \log_2(\mathrm{e}H) \tag{2.35}$$

où e est la base des logarithmes naturels, et $\lfloor x \rfloor$ est le plus grand entier inférieur à x.

La différence entre la d_{vc} et la capacité statistique $P_c = \alpha_c N$ étudiée par Cover ou par l'approche de la mécanique statistique, est que la d_{vc} n'est pas un concept probabiliste. Si $P < d_{vc}$, on a la certitude que le réseau peut faire n'importe quelle dichotomie des entrées. Si :

$$d_{vc} < P < P_c \tag{2.36}$$

la probabilité qu'une dichotomie soit réalisable tend vers 1 pour $N \to \infty$. Dans le cas du perceptron, cette différence subtile se manifeste par le résultat étonnant que $d_{vc} = N$, mais $P_c = 2N$.

2.7 La généralisation

2.7.1 Ambiguïté et séparation linéaire

La propriété de généralisation est en réalité la question la plus importante en ce qui concerne l'apprentissage des machines. Elle n'est bien comprise que dans le cas du perceptron simple.

Pour Cover [19], la classification d'un nouvel exemple $\vec{\xi} \notin \mathcal{L}^\alpha$ est ambiguë, s'il existe deux hyperplans classant correctement l'ensemble $\mathcal{L}^\alpha$, mais qui attribuent une classe différente à l'exemple $\vec{\xi}$. Cover montre que la probabilité $\mathcal{A}(P, N)$ que la classification de l'exemple $\vec{\xi}$ soit ambiguë par rapport à une dichotomie donnée est :

$$\mathcal{A}(P, N) = \frac{C(P, N-2)}{C(P, N-1)} \tag{2.37}$$

Dans la limite $N \to \infty$ la fonction (2.37) a un comportement (voir figure 2.5) donné par :

$$\mathcal{A}(P,N) = \lim_{N\to\infty, P=\alpha N} \mathcal{A}(P,N) = \begin{cases} 1 & \text{si } 0 < \alpha < 2 \\ \frac{1}{\alpha - 1} & \text{si } \alpha > 2 \end{cases} \tag{2.38}$$

L'ambiguïté décroît de façon inversement proportionnelle au nombre d'exemples, et ceci dès que le nombre d'exemples est supérieur à la capacité critique α_c.

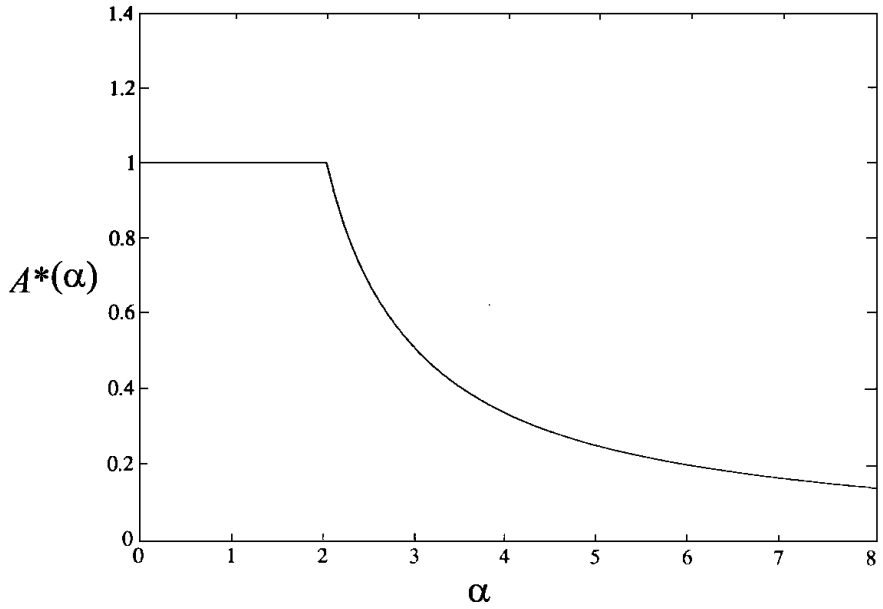

Figure 2.5: Ambiguïté et séparation linéaire (Cover).

2.7.2 Nombre d'exemples et généralisation

Le théorème de *convergence uniforme* de Vapnik et Chervonenkis [103, 100] :

$$p\left\{\sup_{\vec{w}} |\varepsilon_g(\vec{w}) - \varepsilon_t(\vec{w})| > \varepsilon\right\} \to 0 \text{ quand } P \to \infty \tag{2.39}$$

permet de trouver une borne supérieure au nombre d'exemples nécessaires P_{min} pour avoir une bonne généralisation [7].

Si l'erreur d'apprentissage $\varepsilon_t(\vec{w})$ sur un ensemble d'apprentissage $\mathcal{L}^\alpha$ est petit, on espère que l'erreur de généralisation $\varepsilon_g(\vec{w})$ soit (avec une précision arbitraire $\varepsilon > 0$) aussi petit que l'on veut.

[7] sup dénote le *supremum* : le poids $\vec{w}$ sur l'espace des poids qui maximise l'écart entre ε_g et ε_t.

En utilisant la définition de la dimension d_{vc} et pour des valeurs de P finis, l'inégalité suivante est vraie :

$$p\left\{\sup_{\vec{w}}|\varepsilon_g(\vec{w})-\varepsilon_t(\vec{w})|>\varepsilon\right\}<\left(\frac{2\mathrm{e}P}{d_{vc}}\right)^{d_{vc}}\exp(-\varepsilon^2 P) \tag{2.40}$$

où e est la base des logarithmes naturels.

Avec une probabilité $(1-\eta)$, où :

$$\eta=\left(\frac{2\mathrm{e}P}{d_{vc}}\right)^{d_{vc}}\exp(-\varepsilon^2 P) \tag{2.41}$$

l'inégalité $\varepsilon_g(\vec{w})<\varepsilon_t(\vec{w})+\varepsilon$ est vraie pour tous les vecteurs $\vec{w}$ de l'espace des poids. Vapnik [101] a calculé la borne :

$$\varepsilon_0(P,d_{vc},\eta)=\sqrt{\frac{d_{vc}}{P}\left(\log\frac{2P}{d_{vc}}+1\right)-\frac{1}{P}\log\eta} \tag{2.42}$$

où ε_0 est une intervalle de confiance pour le cas le pire où $\varepsilon_g=\frac{1}{2}$, et $\varepsilon=\varepsilon_0$; mais elle est inutilisable pour ε_g petits. Pour le cas intéressant où ε_g est petit, une borne plus utile dérivée de (2.40) a été calculée [101] :

$$p\left\{\sup_{\vec{w}}\left|\frac{\varepsilon_g(\vec{w})-\varepsilon_t(\vec{w})}{\sqrt{\varepsilon_g(\vec{w})}}\right|>\varepsilon\right\}<\left(\frac{2\mathrm{e}P}{d_{vc}}\right)^{d_{vc}}\exp\left(-\frac{\varepsilon^2 P}{4}\right) \tag{2.43}$$

d'où :

$$\varepsilon_g(\vec{w})<\varepsilon_t(\vec{w})+\varepsilon_1 \tag{2.44}$$

où ε_1 est un nouvel intervalle de confiance tel que :

$$\varepsilon_1=2\varepsilon_0^2\left(1+\sqrt{1+\frac{\varepsilon_t(\vec{w})}{\varepsilon_0^2}}\right) \tag{2.45}$$

qui dépend de l'erreur d'apprentissage $\varepsilon_t(\vec{w})$. Si $\varepsilon_t=0$, on a $\varepsilon_1=4\varepsilon_0^2$.

Pour résumer, on aura donc, les bornes suivantes :

- Pour des $\varepsilon_g\geq 1/2$: $\varepsilon_g(\vec{w})\leq\varepsilon_t(\vec{w})+\varepsilon_0$
- Pour des ε_g petits : $\varepsilon_g(\vec{w})\leq\begin{cases}\varepsilon_t(\vec{w})+4\varepsilon_0^2 & \text{si } \varepsilon_t(\vec{w})\to 0\\ \varepsilon_t(\vec{w})+\varepsilon_1 & \text{en général.}\end{cases}$

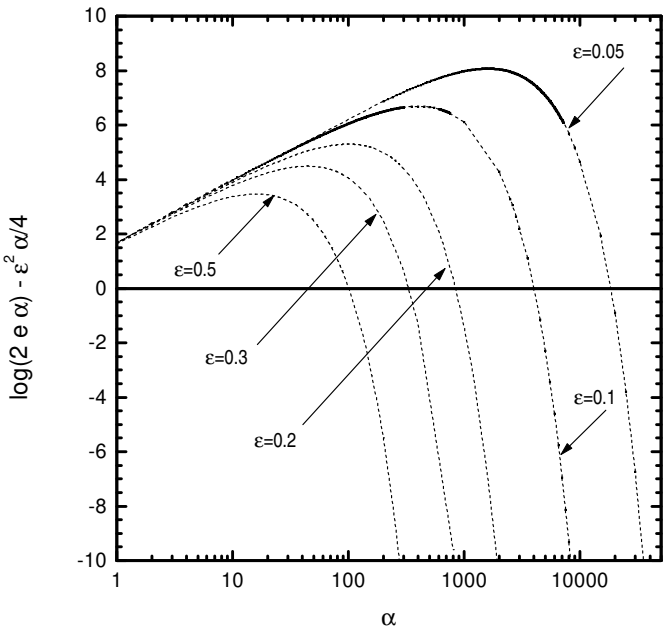

Figure 2.6: Exposant de l'équation (2.46) pour les valeurs de $\varepsilon = (0.5, 0.3, 0.2, 0.1, 0.05)$ en fonction de α.

En utilisant les bornes (2.43), il est possible de calculer le nombre d'exemples nécessaires dans le cas le pire P_{min} pour obtenir une erreur de généralisation donnée, avec un perceptron simple. Si $d_{vc} = N$ et ε est fixé, alors par (2.43) :

$$\begin{aligned} p\left\{\sup_{\vec{w}}\left|\frac{\varepsilon_g(\vec{w})-\varepsilon_t(\vec{w})}{\sqrt{\varepsilon_g(\vec{w})}}\right| > \varepsilon\right\} &< (2e\alpha)^N \exp\left(-\frac{\varepsilon^2\alpha N}{4}\right) \\ &= \exp\left[\log(2e\alpha) - \frac{\varepsilon^2\alpha}{4}\right] N \end{aligned} \tag{2.46}$$

Cette probabilité devient petite seulement si l'exposant $\log(2e\alpha) - \frac{\varepsilon^2\alpha}{4}$ est négatif, c'est-à-dire, pour des valeurs de α suffisamment grandes. Dans la figure 2.6 on voit que même pour des valeurs modérées de ε, le nombre d'exemples nécessaires ($P = \alpha N$) devient trop grand.

En particulier, dans le cas d'un perceptron, on montre que si P et N sont grands, et $\varepsilon_t = 0$, alors $P_{min} > 8N \log N/\varepsilon^2$ (Hertz *et al.* [49]). Si ε est petit, le nombre d'exemples nécessaires pour avoir $\varepsilon_g < \varepsilon$ peut être très grand.

Dans le cas de réseaux à une couche cachée avec N_w poids, si ε est défini par $\varepsilon_t = \varepsilon/2$, Baum et Haussler [7] montrent que le nombre minimum d'exemples P_{min} nécessaires pour obtenir une erreur de généralisation $\varepsilon_g < \varepsilon$, est de l'ordre de :

$$P_{min} \approx \frac{N_w}{\varepsilon} \log \frac{H}{\varepsilon} \tag{2.47}$$

Dans la figure 2.7 nous montrons une comparaison des estimations de P_{min}, ainsi que les bornes à d_{vc} fournies par (2.35) : il est facile de voir que le nombre d'exemples nécessaire pour que $\varepsilon_g < \varepsilon_t + \varepsilon$ est très supérieur à d_{vc} , même pour des valeurs de ε grandes. Cependant en pratique, il arrive souvent qu'on obtienne une faible erreur de généralisation bien que le nombre d'exemples disponibles soit inférieur à la borne (2.47).

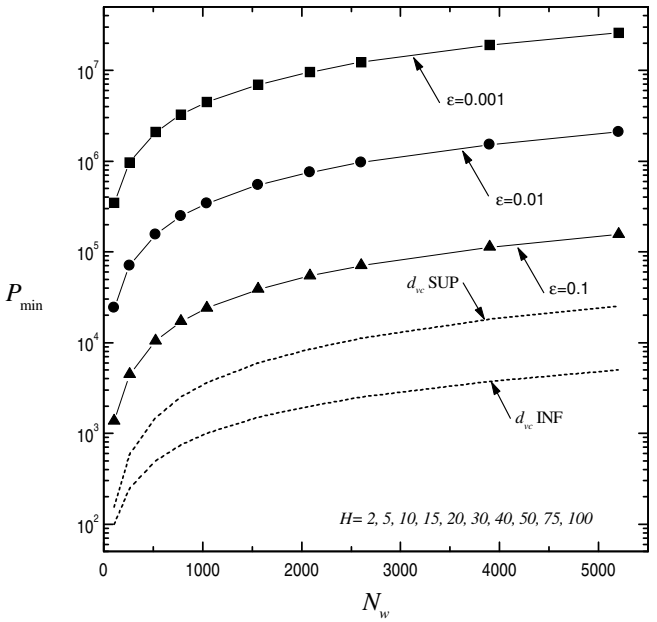

Figure 2.7: Nombre d'exemples nécessaires P_{min} (2.47) pour que $\varepsilon_g < \varepsilon_t + \varepsilon$, pour un réseau avec $N = 50$ entrées à une seule couche cachée de H unités et une unité de sortie. Nombre de poids N_w donné par (2.34), en fonction de H. En traits pointillés, bornes de d_{vc} définies par (2.35).

2.7.3 Le sur-apprentissage

Dans les problèmes de régression, on fait souvent l'analogie avec la régression polynomiale : un réseau de neurones qui dispose de très peu de paramètres (qui est peu complexe) n'a pas assez de flexibilité pour apprendre les données correctement. C'est une situation qu'on appelle le *sous-apprentissage*. Par contre, un réseau avec trop de paramètres dispose de trop de flexibilité (trop degrés de liberté) et les utilise pour apprendre toutes les particularités des données, c'est ce qu'on appelle le sur-apprentissage, apprentissage par cœur ou *overfitting*. Empiriquement, cette situation pourrait être évitée en cherchant le minimum de l'erreur de généralisation sur un

ensemble de validation (indépendant de l'ensemble d'apprentissage) et en arrêtant l'apprentissage avant la convergence du réseau (*early stopping*). Ainsi, on espère que les poids qui minimisent l'erreur sur l'ensemble de validation, généralisent mieux.

Le sur-apprentissage est en rapport avec le dilemme du biais/variance [40]. Mais ce dilemme doit être considéré différemment en problèmes de régression et en problèmes de classification : en effet, une décomposition additive de l'erreur n'est possible que pour la régression [34] ; car en problèmes de classification la situation n'est pas la même parce que la mesure d'erreur n'est plus une fonction continue mais discrète puisqu'il s'agit de classes.

Dans l'Annexe D nous présentons une décomposition en biais et variance pour des problèmes de régression et une autre, proposée par Friedman [34] pour la classification. Dans ce dernier cas, le sur-apprentissage peut être un problème pour des réseaux qui considèrent la classification comme un problème d'approximation de fonctions, notamment dans des réseaux qui minimisent un coût (voir Chapitre 3).

Dans les problèmes de classification, nous avons trouvé expérimentalement qu'un réseau qui apprend tout l'ensemble d'apprentissage jusqu'à obtenir zéro fautes peut obtenir de faibles erreurs de généralisation, sans besoin de validation croisée (voir Chapitre 6).

2.8 Conclusion

Dans ce chapitre nous avons présenté le paradigme du professeur-élève de l'apprentissage supervisé, qui est utile pour étudier des problèmes de régression et de classification. Nous avons introduit les perceptrons linéaires et ceux sphériques, ainsi que la notation adoptée pour les réseaux de neurones.

Des questions sur l'apprentissage et la généralisation ont été abordées du point de vue théorique, en utilisant les approches probabilistes de Cover et de la mécanique statistique, et celle de Vapnik-Chervonenkis. Ainsi, nous avons montré qui'il existe un important décalage entre les prédictions théoriques et les performances des algorithmes. Par exemple, bien qu'un réseau de neurones avec une seule couche cachée puisse approcher toute fonction des entrées, le nombre d'unités cachées est inconnu : les bornes fournies par la dimension de Vapnik-Chervonenkis sont excessives, donc inutilisables pour les applications. Hormis le cas du perceptron, la compréhension du problème de la généralisation est encore insuffisante. Nous sommes donc réduits à comparer les algorithmes d'apprentissage sur des problèmes étalon.

Chapitre 3

Algorithmes d'apprentissage

3.1 Introduction

L'apprentissage avec des réseaux de neurones multicouches se fait principalement suivant deux approches. La *Rétropropagation du Gradient* (RPG) détermine les poids par minimisation d'un coût. Cet algorithme nécessite l'introduction *a priori* de l'architecture du réseau, qui peut être élaguée après ou pendant l'apprentissage.

Avec une approche constructive on apprend *en même temps* le nombre d'unités et les poids, commençant généralement avec une seule unité. L'introduction successive de nouvelles unités produit une application des entrées sur des représentations internes qui doit être fidèle : des exemples de classes différentes doivent avoir des représentations internes différentes.

Bien que la classification de données soit intrinsèquement une tâche discrète, elle peut être envisagée comme un problème d'*approximation de fonctions*, en attribuant des valeurs réelles aux classes à apprendre. Cette approche est utilisée par la Rétropropagation du Gradient, qui minimise l'erreur quadratique d'apprentissage à la sortie. La fonction d'approximation doit être hautement non-linéaire, car elle doit avoir une valeur constante dans le domaine de chaque classe, et présenter une grande variation aux frontières entre classes. Par exemple, dans la tâche de classification binaire du diagnostic du cancer du sein (déjà présentée), la fonction d'approximation doit être constante et positive dans les régions ou domaines de l'espace des entrées correspondant à la classe 1, constante et négative pour ceux de la classe -1. Avec l'algorithme de Rétropropagation du Gradient, on cherche des poids qui approchent cette fonction partout, et en particulier à l'intérieur des domaines, au lieu de se concentrer sur le problème pertinent de déterminer les frontières entre eux.

Avec la RPG, le nombre des paramètres requis pour apprendre la tâche n'est pas connu *a priori*. L'apprentissage avec un grand nombre de poids permet, au moins en principe, d'approximer une grande variété de fonctions parmi lesquelles on espère

se trouve la *vraie* solution. Mais il est difficile de minimiser une fonction de coût dans un espace de grande dimension, car le risque de tomber dans des minimums locaux, augmente avec la dimension. Dans la pratique, l'architecture est déterminée par essai et erreur, en faisant varier le nombre de neurones et de poids.

Une approche alternative est fournie par les algorithmes *à croissance* ou *constructifs*, dans lesquels les unités cachées sont successivement ajoutées au réseau. Outre le fait que cela permet la détermination automatique de l'architecture du classifieur, bâtir un réseau de neurones par un algorithme de croissance permet l'utilisation de neurones *binaires*. Ce fait a deux avantages principales : d'une partie, les unités cachées binaires déterminent les limites ou frontières (ou morceaux de frontières) des régions dans l'espace des entrées qui contiennent des exemples de la même classe; et d'un autre côté, ce type d'unités permet l'extraction de règles. En plus de cela, les unités binaires sont bien adaptés pour leur implantation matérielle à l'aide des circuits intégrés digitaux.

Dans ce chapitre nous présentons d'abord l'algorithme de la Rétropropagation du Gradient et quelques méthodes utilisées pour réduire la complexité du réseau. Ensuite nous passons en revue plusieurs algorithmes constructifs.

3.2 Méthodes à simplification

3.2.1 La Rétropropagation du Gradient

L'algorithme de Rétropropagation du Gradient a été développé par plusieurs auteurs [105, 61, 83]. C'est une technique très populaire dans les applications des réseaux de neurones. Etant donnée une mesure d'erreur aussi appelée *coût* $E(\vec{w})$, l'algorithme donne une manière de trouver un ensemble des poids $\vec{w}$ d'un réseau sans rétroaction par une descente de gradient dans l'espace des poids sur la surface définie par la fonction de coût.

Soient un réseau de neurones (figure 2.3) avec N entrées, une couche cachée avec H unités et un ensemble $\mathcal{L}^\alpha$ de P exemples. Etant donné un exemple μ, le champ sur l'unité cachée j est (toutes les sommes vont sur $i = 0, \cdots, N$, $j = 0, \cdots, H$, $\mu = 1, \cdots, P$.) :

$$h_j^\mu = \sum_i w_{ji} \xi_i^\mu \tag{3.1}$$

La sortie σ_j^μ de cette unité est :

$$\sigma_j^\mu = f(h_j^\mu) = f\left(\sum_i w_{ji}\xi_i^\mu\right) \tag{3.2}$$

Le champ sur l'unité de sortie ζ est :

$$h^\mu = \sum_j W_j\sigma_j^\mu = \sum_j W_j f\left(\sum_i w_{ji}\xi_i^\mu\right) \tag{3.3}$$

et la sortie ζ^μ du réseau est :

$$\zeta^\mu = f\left(\sum_j W_j\sigma_j^\mu\right) = f\left(\sum_j W_j\ f\left(\sum_i w_{ij}\xi_i^\mu\right)\right) \tag{3.4}$$

Soit $\vec{w}$ l'ensemble de tous les poids du réseau. La fonction de coût :

$$E(\vec{w}) = \frac{1}{2}\sum_\mu (\tau^\mu - \zeta^\mu)^2 \tag{3.5}$$

s'écrit :

$$E(\vec{w}) = \frac{1}{2}\sum_\mu \left[\tau^\mu - f\left(\sum_j W_j f\left(\sum_i w_{ij}\xi_i^\mu\right)\right)\right]^2 \tag{3.6}$$

Si f est dérivable, l'équation (3.6) est aussi dérivable par rapport aux poids. Une descente en gradient donne, pour les poids $\vec{W}$:

$$\begin{aligned}\delta W_j &= -\epsilon\frac{\partial E}{\partial W_j} = \epsilon\sum_\mu(\tau^\mu - \zeta^\mu)f'(h^\mu)\sigma_j^\mu \\ &= \epsilon\frac{\partial E}{\partial W_j} = \epsilon\sum_\mu \delta^\mu\sigma_j^\mu\end{aligned} \tag{3.7}$$

où $\delta^\mu \equiv f'(h^\mu)(\tau^\mu - \zeta^\mu)$.

Pour les connexions w_{ji} de l'entrée vers la couche cachée, on a :

$$\begin{aligned}\delta w_{ji} &= -\epsilon\frac{\partial E}{\partial w_{ji}} \\ &= \epsilon\sum_\mu(\tau^\mu - \zeta^\mu)f'(h^\mu)W_j f'(h_j^\mu)\xi_i^\mu \\ &= \epsilon\sum_\mu \delta^\mu W_j f'(h_j^\mu)\xi_i^\mu \\ &= \epsilon\sum_\mu \delta_j^\mu\xi_i^\mu\end{aligned} \tag{3.8}$$

où $\delta_j^\mu \equiv f'(h_j^\mu) W_j \delta^\mu$.

L'équation (3.8) détermine les δw_{ji} pour l'unité cachée σ_j en termes de l'erreur à la sortie ζ : la modification des poids des neurones cachées nécessite l'information d'erreurs rétropropagées, d'où le nom de la méthode. La généralisation à des réseaux à plusieurs couches n'est guère plus compliquée, et le lecteur intéressé peut la trouver en [49].

Bien qu'on ait écrit les équations des modifications des poids (3.7) et (3.8) comme des sommes sur les P exemples (méthode connue sous le nom de *gradient total*), on pourrait faire la modification des poids après la présentation de chaque exemple : soit de manière séquentielle, soit aléatoire (gradient stochastique). Le choix des méthodes (gradient total, séquentiel ou stochastique) dépend de chaque problème particulier, toutefois la dernière approche est utile dans des cas où on a des exemples redondants [49].

Malgré ses succès [89, 60, 77], la méthode de rétropropagation du gradient décrite présente l'inconvénient majeur du choix de l'architecture : il n'existe pas de critère pour commencer avec une architecture plutôt qu'avec une autre.

3.2.2 Méthodes d'élagage

L'apprentissage par la méthode RPG donne lieu à des réseaux qui éventuellement peuvent être élagués, soit pendant l'apprentissage (méthodes de régularisation) soit après l'apprentissage. Ces méthodes permettent de réduire la complexité du réseau, avec l'espoir d'améliorer la capacité de généralisation.

Elagage pendant l'apprentissage

Les techniques de *régularisation* [76, 8] permettent d'obtenir des réseaux moins complexes en ajoutant un terme de pénalité à la fonction de coût. L'idée consiste à équilibrer la fonction coût (3.5) avec un autre terme $C(\vec{w})$ qui représente une mesure de la complexité du réseau. Le nouveau coût est :

$$\tilde{E}(\vec{w}) = E(\vec{w}) + \lambda C(\vec{w}) \tag{3.9}$$

où λ est un coefficient de compromis entre l'importance attribuée au terme de complexité C par rapport à la mesure de l'erreur E. E et C sont donc des termes antagonistes pendant le processus d'apprentissage. Plusieurs termes de complexité ont été proposés [76, 52]. Parmi eux on trouve *Weigth decay*, qui est l'une des méthodes les plus simples de régularisation, qui introduit des pénalités de la forme $C(\vec{w}) = \frac{1}{2}\sum_i |w_i|$ ou $C(\vec{w}) = \frac{1}{2}\sum_i w_i^2$ (l'indice i parcourt *tous* les poids du réseau, biais inclus). L'un des problèmes avec cette méthode, est qu'elle favorise le développement de plusieurs petits poids.

Dans la méthode d'élimination des poids (*weigth elimination*) [107] le terme de complexité est de la forme $C(\vec{w}) = \sum_i \|w_i\|^2/(\hat{w}^2 + \|w_i\|^2)$, où $\hat{w}$ est un facteur de normalisation. En pratique, les poids peuvent ne pas être nuls, et on les élimine en choisissant ceux au-dessus d'un petit seuil. Le paramètre $\hat{v}$ doit être choisi empiriquement.

D'autres termes ont été proposés [52, 8], mais dans tous les cas le paramètre λ, auquel la méthode de RPG semble être assez sensible, est difficile à régler.

Elagage post-apprentissage

D'autres méthodes simplifient les réseaux suivant la sensibilité de l'erreur à la suppression de poids ou neurones. Ils réalisent ainsi un élagage post-apprentissage. On commence, donc, avec un réseau surdimensionné et on le simplifie en suivant des critères de sélection des poids à éliminer. Ces critères deviennent plus simples à élaborer car l'élagage intervient lorsque l'algorithme d'apprentissage a convergé selon un critère choisi, et on utilise cet état comme une référence.

- Suppression des poids. L'un des critères les plus intuitifs pour mesurer l'importance d'un poids est sa valeur absolue : les poids proches de zéro peuvent généralement être supprimés sans que ceci ait une influence sur le comportement du réseau. En pratique on arrive à obtenir ainsi des résultats comparables à ceux d'autres méthodes d'élagage post-apprentissage [23].

- *Optimal Brain Damage (OBD).* Proposée par Le Cun *et al.* [62] cette méthode est fondée sur le calcul d'un terme de sensibilité correspondant à la variation moyenne de la fonction de coût entraînée par la suppression de chaque poids. On peut supprimer les poids pour lesquels la sensibilité est inférieure à un seuil. Après l'élagage, le réseau n'est plus optimal, même s'il reste très proche d'un optimum, et il faut refaire quelques itérations d'apprentissage.

- *Optimal Brain Surgeon (OBS).* Proposée par Hassibi et Stork [50], ceci est une extension améliorée de la méthode antérieure. L'algorithme calcule par une minimisation sous contrainte, les poids à supprimer et la mise à jour des paramètres restants. Elle est fondée sur le développement de Taylor mais ne suppose pas que la matrice Hessienne soit diagonale.

- *Statistical Stepwise Method (SSM).* Cet algorithme (Cottrell *et al.* [17]) permet d'éliminer les poids statistiquement non significatifs, seulement si le résultat du réseau est meilleur qu'un précèdent, selon un critère empirique de qualité

BIC [1]. On mesure le *BIC* à chaque pas, et on arrête l'élagage quand ce critère est minimum (qui peut être un minimum local).

Une difficulté des méthodes *OBD* et *OBS* est la détermination du seuil de sensibilité. De plus, les deux méthodes sont fondées sur un développement de Taylor au second ordre de la fonction de coût, qui n'est valide que pour des variations faibles des poids, c'est-à-dire, pour les suppressions de poids *proches* de zéro. Jutten et Fambon [52] et Hérault et Jutten [48] montrent que si l'on est dans un minimum local, les méthodes *OBD, OBS* et *SSM* peuvent sélectionner les mêmes poids.

3.3 Algorithmes constructifs

Les heuristiques constructives pour la classification peuvent être groupées en deux classes, suivant qu'elles utilisent des unités cachées faisant des séparations linéaires, désormais appelées unités linéaires, ou des unités cachées à fonction de base radiale *RBF* (unités sphériques, décrites au Chapitre 2). Dans les deux cas la fonction d'activation est une fonction sigmoïde (voir Chapitre 2). Des algorithmes constructifs ont été proposés pour traiter des problèmes d'approximation de fonctions [18].

Nous décrivons dans la suite quelques algorithmes dont les résultats seront comparés aux nôtres au Chapitre 6.

3.3.1 Heuristiques avec unités linéaires

Algorithme *Tiling*

Cet algorithme, proposé par Mézard et Nadal [68], permet de construire des réseaux couche par couche. On commence avec un perceptron simple, qui apprend l'ensemble $\mathcal{L}^\alpha$ en utilisant l'algorithme *Pocket* [2], par exemple. Si une solution sans erreurs est trouvée l'algorithme s'arrête. Dans le cas contraire, on gèle les poids du neurone, qui devient l'unité *maîtresse*, laquelle produit un certain nombre d'erreurs d'apprentissage e_1 sur $\mathcal{L}^\alpha$. On rajoute des unités *auxiliaires* qui, avec l'unité maîtresse constituent la première couche cachée. Ces unités doivent apprendre des cibles pour remplir la condition de fidélité suivante : deux exemples de classes différentes doivent avoir des représentations internes différentes.

[1] Le critère d'information B d'Akaike *BIC* peut s'écrire :

$$BIC = \log\left(\frac{v_r^2}{P}\right) + N_w \frac{\log P}{P} \tag{3.10}$$

où v_r^2 est la variance de l'erreur résiduelle, P le nombre d'exemples et N_w le nombre de poids. Le premier terme mesure la qualité de l'approximation des données, le second est un terme qui met en évidence le lien entre le nombre de poids et le nombre d'exemples [53].

[2] L'algorithme du perceptron et l'algorithme *Pocket* seront présentés au Chapitre 4.

La couche suivante est bâtie en utilisant la même stratégie : maintenant la première couche cachée joue le rôle de couche d'entrée. On construit l'unité maîtresse de la deuxième couche, laquelle produit un nombre d'erreurs e_2, puis on complète la couche avec des unités auxiliaires, et ainsi de suite. Le réseau va croître jusqu'à ce que, pour l'unité maîtresse d'une couche cachée L, le nombre d'erreurs s'annule, $e_L = 0$, et cette unité sera donc, l'unité de sortie finale.

La convergence de la procédure est prouvée en [68], car on montre que :

- Une couche cachée k ayant été construite, on peut toujours trouver une unité maîtresse pour la couche $k+1$ qui fait un nombre d'erreurs strictement plus petit que l'unité maîtresse de la couche précédente : $e_{k+1} < e_k$.

- Il est toujours possible de rajouter suffisamment de neurones pour obtenir des représentations internes fidèles.

Plusieurs résultats en [68, 31] montrent que l'algorithme est très "gourmand" en ressources (nombre d'unités et de couches cachées).

Algorithme *Sequential Learning*

Dans l'algorithme *Sequential Learning*, de Marchand *et al.* [65], la première unité est entraînée pour séparer l'ensemble d'apprentissage en gardant un sous-espace "pur", *i.e.* un sous-espace qui contient des exemples d'une seule classe. Les exemples mal classés, s'il y en a, doivent se trouver dans l'autre sous-espace.

Chaque neurone rajouté est entraîné pour séparer les exemples qui sont mal classés, toujours avec cette contrainte, *i.e.* en gardant toujours un sous-espace "pur", libre d'erreurs. L'algorithme est difficile à mettre en œuvre pratiquement, car il n'est pas facile d'imposer la contrainte de pureté durant l'apprentissage.

Algorithme *Cascade-Correlation*

Un des algorithmes constructifs les plus connus est *Cascade-Correlation*, créé par Fahlman et Lebière [29]. Dans cette méthode, chaque unité cachée ajoutée est choisie parmi une collection de plusieurs unités à activation continue, entraînées pour apprendre la corrélation entre les sorties et les erreurs d'apprentissage. L'unité qui maximise cette corrélation est alors connectée aux entrées et à toutes les autres unités cachées déjà connectées au réseau. Falhman *et al.* utilisent l'algorithme *quickprop* [28] pour entraîner chaque neurone.

On peut remarquer que l'architecture du réseau est en cascade : chaque unité est connectée aux entrées, mais aussi aux unités cachées déjà existantes.

Algorithme *Upstart*

Cet algorithme a été développé par Frean [31]. On considère un perceptron qui a été entraîné sur un ensemble d'apprentissage. Ce perceptron peut commettre des erreurs du type:

- *WON* (*wrongly-on*) : L'unité répond $\zeta = +1$ alors que la classe est $\tau = -1$.
- *WOFF* (*wrongly-off*) : L'unité répond $\zeta = -1$ alors que la classe est $\tau = +1$.

Des unités *filles* X et Y sont rajoutées afin de corriger respectivement ce type d'erreurs. Les ensembles d'apprentissage de ces unités sont déterminés de façon à ce que l'unité X (respectivement Y) soit active pour les exemples correspondants à une erreur du type *WON* (respectivement *WOFF*) et pas pour les autres. Les unités filles sont ensuite connectées à l'unité mère. Si les unités filles n'arrivent pas à corriger toutes les erreurs, alors elles peuvent à leur tour engendrer d'autres unités filles en suivant le même procédé.

Algorithme *Offset*

Cet algorithme, créé par Martinez et Estève [64] réalise l'apprentissage d'une machine à parité c'est-à-dire, telle que l'unité de sortie implémente la parité de ses entrées [9]. Chaque unité cachée ajoutée est entraînée à corriger les erreurs de la dernière unité cachée, une procédure qui engendre une *machine à parité* : la classe de l'exemple d'entrée est la parité des représentations internes apprises.

Une deuxième couche cachée (qui peut être élaguée) est bâtie sans besoin d'apprentissage, en n'utilisant que des considérations géométriques, pour implanter la parité. La convergence de l'algorithme a été demontrée dans les cas d'entrées *binaires*[64] et réelles [44].

Autres algorithmes

Une stratégie alternative consiste à construire une architectures en *arbre*. Les *Arbres de décision*, introduits par Breiman *et al.* [6] sont connus sous le nom de méthode CART (*Classification And Regression Trees*). Ils partitionent hiérarchiquement l'espace des entrées par des dichotomies successives. Des critères de découpage (sous la forme de questions binaires sur l'ensemble de valeurs possibles des entrées) permettent de faire cette partition. Plusieurs critères de "pureté" des régions peuvent être choisis pour arrêter la croissance. La discrétisation des entrées peut s'avérer indispensable pour mener à bien le découpage.

Des versions neuronales de ce type de classifieur ont été proposées par plusieurs équipes (M. Golea et M. Marchand [43], par Frean [31], et par Mézard et Nadal [87])

sous des formes légèrement différentes. Le réseau consiste en un ensemble de perceptrons fonctionellement organisés comme un arbre binaire. Dans cette architecture il faut distinguer entre l'*organisation structurelle* et le fonctionnement : chaque neurone de l'arbre n'est connecté qu'à la couche d'entrée. Sa sortie n'est pas connectée vers d'autres neurones, mais elle est utilisée pour décider comment parcourir l'arbre. Chaque neurone de l'arbre introduit une dichotomie de l'espace des entrées. Chacun des sous espaces est traité séparément par des nœuds fils, qui éventuellement produisent d'autres partitions. En plus des poids, les réseaux doivent stocker le parcours de décision.

Les heuristiques proposées pour engendrer des arbres neuronaux diffèrent par l'algorithme utilisé pour entraîner chaque nœud, et/ou dans le critère d'arrêt.

En particulier, *Neural-Trees* [87] peut être vue comme une généralisation de *CART* [6], dans lequel les hyperplans ne sont pas limités à être perpendiculaires aux axes de coordonnées. L'heuristique du *MNTM* (Modified Neural Tree Network) [30] similaire à *Neural-Trees*, ajoute un critère d'arrêt (*early stopping*) basé sur une mesure de confiance de la partition. La méthode *Stepwise* de Knerr *et al.* [57] permet de bâtir un réseau de neurones à deux couches cachées : la première sert à faire une séparation en chaque classe en isolant les exemples mal classés. Ceci permet de traiter des sous-ensembles d'exemples de plus en plus petits pour l'apprentissage des neurones successifs. Quand on a réussi à bien séparer tous les exemples, on ajoute une deuxième couche qui réalise une fonction booléenne (neurones de sortie type ET) avec des poids binaires ($\vec{W} = \pm 1$), sans besoin d'apprentissage.

3.3.2 Heuristiques avec unités sphériques

Les classifieurs à neurones binaires sphériques ont été introduits par Cooper [80]. Ils sont basés sur le stockage d'exemples représentatifs ou prototypes auxquels on associe un rayon *d'influence* du point. L'intérieur de l'hypersphère représente un domaine de décision associé à la classe du prototype qui est au centre.

Algorithme *Restricted Coulomb Energy (RCE)*

Développé par Reilly *et al.* [80], cet algorithme propose l'heuristique suivante : si un exemple est à l'extérieur de toutes les hypersphères existantes, il ne peut pas être classé. On rajoute alors un neurone dont il est le centre (prototype), de rayon ρ. Par contre, si l'exemple se trouve dans les domaines de décision d'unités déjà existantes, on réduit le rayon des unités qui codent pour des classes différentes de celle de l'exemple. Éventuellement une nouvelle unité doit être rajoutée. L'apprentissage se fait en présentant l'ensemble $\mathcal{L}^\alpha$ itérativement jusqu'à ce que la procédure n'entraîne aucune modification: ni création d'unités ni réduction des rayons d'influence. Cer-

tains problèmes qui peuvent se présenter sont l'apprentissage par cœur et le bon choix du rayon initial, souvent choisi empiriquement.

Algorithme *Grow and Learn*

Bien que l'algorithme *Grow and Learn (GAL)* [1] n'utilise pas des perceptrons sphériques, il est assez proche de *RCE*, tant par l'architecture que par la procédure d'apprentissage.

La différence avec *RCE*, réside dans une structure *Winner-Take-All (WTA) ou le gagnant-prend-tout*, qui remplace les rayons d'influence. Le *WTA* reçoit les sorties des unités cachées et calcule un vecteur de sortie qui vaut 1 pour la composante i telle que σ_i est minimale, et -1 pour toutes les autres. Si on utilise comme critère d'activation la distance euclidienne entre le poids et l'exemple, l'ensemble de ces régions réalise une partition de l'espace selon un pavage de Voronoï [49].

Avec les algorithmes *RCE* et *GAL*, le nombre de prototypes créés c'est-à-dire la complexité du réseau *dépend de l'ordre de présentation* des exemples à apprendre. Pour remédier à cela, Alpaydin [1] propose une amélioration appelée phase de *sommeil*, qui n'est autre chose qu'une phase d'élagage : on cherche à garder seulement les unités dont le domaine de décision contient des exemples proches des frontières entre classes, en éliminant les neurones qui sont à l'intérieur d'une région d'influence d'une autre unité de la même classe. Les phases de *sommeil* et d'*éveil* (apprentissage) rendent le nombre final d'unités aussi indépendant que possible de l'ordre d'exemples présentés.

Autres algorithmes

Les algorithmes *Glocal* [23] et *Growing cells* [33] proposent de couvrir l'espace des entrées avec des hypersphères de taille variable contenant des exemples de la même classe. Glocal a l'idée intéressante de faire une première séparation avec un perceptron linéaire, et ensuite de corriger les exemples mal classés avec des hypersphères. Il en résulte un réseau hybride à unités linéaires et sphériques. Ces approches terminent souvent avec un grand nombre d'unités cachées.

La méthode *Covering Regions by the Linear Programmation Method* [71] est une procédure d'essai et erreur dans le but de choisir le type de neurone (appelé *masque* par les auteurs) le plus efficace parmi hyperplans, hypersphères et hyperellipsoïdes. Les paramètres des neurones sont déterminés par programmation linéaire.

3.4 Conclusion

La méthode de Rétropropagation du Gradient traite un problème de classification comme un problème d'approximation de fonctions, en minimisant un coût. Cependant, il est difficile de minimiser une fonction de coût dans un espace de grande dimension sans tomber dans des minimums locaux. En outre, l'architecture est déterminée par essai et erreur, ce qui est dépendant de l'expertise de l'utilisateur. Les variations de la RPG par élagage pendant l'apprentissage, permettent de finir avec un réseau moins complexe, mais elles sont dépendantes de paramètres de régularisation difficiles à trouver. D'un autre côté, les critères des méthodes de simplification après l'apprentissage sont difficiles à implanter.

Nous avons montré l'état de l'art des heuristiques constructives pour la classification. Elles réduisent l'apprentissage à celui des unités individuelles. Ces heuristiques différent dans les cibles à apprendre mais aussi dans les types de neurones utilisés (linéaires, sphériques, continus, binaires). La complexité de la machine est déterminée par l'algorithme d'apprentissage. Le point commun essentiel de ces algorithmes est l'apprentissage individuel des unités.

Chapitre 4

L'algorithme d'apprentissage Minimerror

4.1 Introduction

DAns ce chapitre nous allons présenter l'algorithme Minimerror [45, 39] qui sert à trouver des séparations linéaires, auquel nous avons ajouté un critère d'arrêt efficace et que nous appellerons désormais *Minimerror-L*, et l'algorithme *Minimerror-S*, que nous avons développé pour la classification par des séparations hypersphériques. Tous les deux sont des algorithmes d'apprentissage pour des perceptrons binaires. Nous avons trouvé une *standardisation* adéquate des entrées (transparente à l'utilisateur) qui évite d'adapter les paramètres de Minimerror-L/S pour chaque problème particulier. Quelques simulations numériques sur des problèmes à entrées binaires et réelles illustrent leurs performances.

Ces deux algorithmes sont utilisés pour l'apprentissage des perceptrons individuels dans les algorithmes constructifs décrits au Chapitre 5.

L'un des avantages des heuristiques constructives consiste à ramener le problème de l'apprentissage à celui de l'apprentissage par des perceptrons : chaque unité rajoutée est traitée comme un perceptron simple, ce qui permet de faire un apprentissage plus rapide que dans les réseaux à architecture fixe.

Le premier algorithme d'apprentissage pour le perceptron linéaire a été proposé par Rosenblatt [85]. La recherche des poids se fait comme suit : on initialise les poids $\vec{w}$ avec des nombres aléatoires. Ensuite, pour tous les exemples $\mu = 1, \cdots, P$ dans un ordre quelconque, si $\tau^\mu \vec{w} \cdot \vec{\xi}^\mu \leq 0$ alors on modifie les poids suivant $w_i = w_i + \tau^\mu \xi_i^\mu$. L'algorithme converge en un nombre fini de pas si l'ensemble $\mathcal{L}^\alpha$ est linéairement séparable (voir par exemple [85, 70, 24, 49]). Par contre, si l'ensemble ne l'est pas, il ne s'arrête jamais.

Une amélioration de l'algorithme du perceptron a été proposée par Gallant [35] avec le *Pocket Algorithm*, qui permet d'obtenir une solution avec un nombre d'erreurs aussi petit que possible. L'idée consiste à garder dans la *poche* la meilleure solution

trouvée, $\vec{w}^*$ qui fait un nombre d'erreurs e^*. Pour cela on applique l'algorithme standard du perceptron. A chaque instant t, les poids $\vec{w}(t)$ produisent un certain nombre d'erreurs $e(t)$. Si $e(t) < e^*$, alors on remplace les poids de la poche, $\vec{w}^* \leftarrow \vec{w}(t)$.

Au bout d'un certain temps t_{max} on arrête l'algorithme en prenant comme solution non pas $\vec{w}(t)$ mais les poids $\vec{w}^*$ qui sont dans la *poche*. Si on considère que le nombre d'exemples P est fini, alors en utilisant cet algorithme pendant un temps *suffisamment long*, on peut espérer trouver les poids qui minimisent le nombre d'erreurs avec une grande probabilité. La solution trouvée n'est pas garantie d'être optimale en termes de la distance des exemples à l'hyperplan séparateur, c'est-à-dire, au sens de la stabilité (2.12). En raison de sa simplicité, cet algorithme est souvent utilisé dans les méthodes constructives présentées au Chapitre 3.

4.2 Minimerror-L

Le but essentiel de l'apprentissage est de trouver des poids qui minimisent l'erreur de généralisation ε_g. Cependant, en ne disposant que d'un ensemble d'apprentissage $\mathcal{L}^\alpha$ fini, on minimise le nombre d'erreurs ε_t sur les exemples de cet ensemble, en espérant que les poids trouvés minimisent aussi ε_g. Ceci peut se poser formellement comme la minimisation de la fonction de coût suivante :

$$E(\vec{w}) = \sum_{\mu=1}^{P} \Theta\left(-\gamma^\mu(\vec{w})\right) \tag{4.1}$$

où P est le nombre d'exemples, γ^μ est la stabilité définie par (2.12) de l'exemple μ, et Θ est la fonction de Heaviside (équation 2.8). Cette fonction compte simplement le nombre d'erreurs (ou sorties incorrectes) produits par les poids $\vec{w}$. La difficulté pour minimiser cette fonction est qu'elle n'est pas dérivable.

L'algorithme du perceptron [85] est capable de trouver une solution qui minimise (4.1) seulement si l'ensemble d'apprentissage est linéairement séparable, *i.e.* si une solution existe. Il y a d'autres algorithmes qui permettent de trouver des solutions optimales en termes de la stabilité [55, 86], et de la généralisation [14, 15] si l'ensemble est linéairement séparable ; mais si l'ensemble d'apprentissage ne l'est pas, ils ne s'arrêtent pas. Certains algorithmes [32, 36] détectent l'absence d'une solution sans erreurs d'apprentissage, et permettent de trouver des poids *raisonnables*, mais ils ne peuvent pas assurer que les poids minimisent le nombre d'erreurs.

Le but fondamental d'un bon algorithme d'apprentissage consiste à trouver le vecteur des poids $\vec{w}$ qui minimise l'erreur d'apprentissage, et qui maximise les stabilités, pour avoir un apprentissage robuste. Pour cela il n'est pas suffisant de minimiser le nombre d'erreurs (le nombre d'exemples dont les stabilités $\gamma \leq 0$). Le

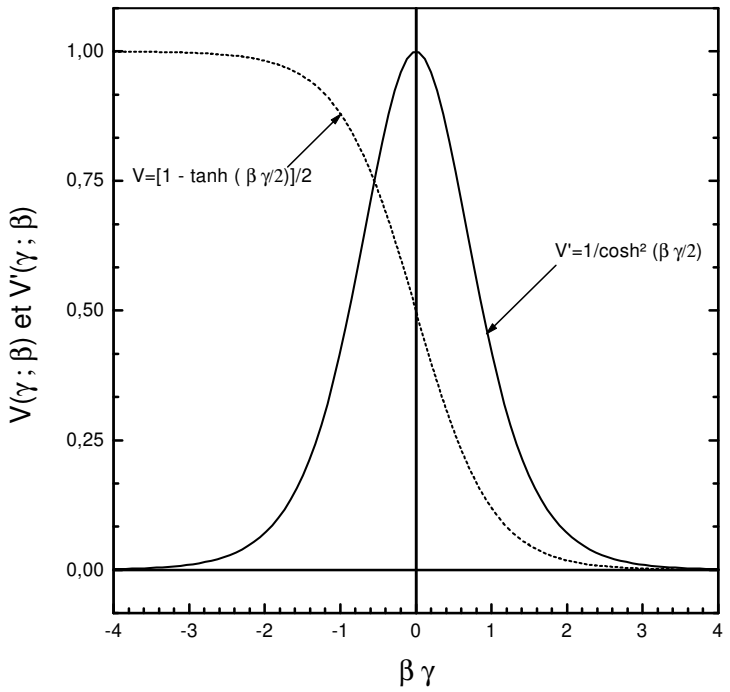

Figure 4.1: La fonction de coût $V(\gamma\ ;\ \beta) = \frac{1}{2}[1 - \tanh(\beta\gamma/2)]$ et sa dérivée.

coût (4.1) compte de la même façon tous les exemples non appris, quelles que soient leurs stabilités. Pour pouvoir extraire l'information des exemples pertinents, qui constituent les frontières entre classes, il faut modifier la fonction de coût.

L'algorithme Minimerror-L [45, 39] minimise la fonction de coût suivante :

$$E(\vec{w}\ ;\ \beta) = \sum_{\mu=1}^{P} V(\gamma^{\mu}\ ;\ \beta) \tag{4.2}$$

où :

$$V(\gamma\ ;\ \beta) = \frac{1}{2}(1 - \tanh\frac{\beta\gamma}{2}) \tag{4.3}$$

représente la contribution d'un exemple avec stabilité γ à la fonction de coût. V dépend d'un paramètre β (pour des raisons qui apparaîtront plus loin, on introduit $T = 1/\beta$, appelé *température*).

Dans la limite où $T \to 0$ ($\beta \to \infty$), on a :

$$V(\gamma\ ;\ \infty) = \begin{cases} 0 & \text{si } \gamma > 0 \\ 1 & \text{si } \gamma < 0 \end{cases} \tag{4.4}$$

Dans cette limite, la fonction $E(\vec{w}\ ;\ \infty)$ (4.2) est égale au coût (4.1) : elle compte *strictement* le nombre d'erreurs commises sur l'ensemble d'apprentissage. Dans la limite contraire, quand $T \to \infty$ ($\beta \to 0$) chaque exemple contribue au coût proportionnellement à sa stabilité.

A température T finie, les exemples avec une grande stabilité positive $\gamma \gg 0$ ont $V \approx 0$ et ceux avec $\gamma \ll 0$ ont $V \approx 1$. Donc, pour les exemples qui se trouvent loin de l'hyperplan séparateur, $E(\vec{w}\ ;\ \beta)$ compte le nombre de fautes ; mais les exemples dont les stabilités se trouvent *dans* une fenêtre de largeur $\approx 2T$ des deux cotés de l'hyperplan séparateur ($-2/\beta < \gamma < 2/\beta$), contribuent à la fonction de coût proportionnellement à $1 - \beta\gamma/2$: même les exemples bien appris contribuent au coût (4.2). Gordon et Grempel [42] ont montré que si l'ensemble $\mathcal{L}^\alpha$ est non linéairement séparable, si β est suffisamment grand (strictement dans la limite $\beta \to \infty$) les poids minimisent la fonction de coût (4.2) minimisant le nombre d'erreurs d'apprentissage. Par contre, si l'ensemble d'apprentissage est linéairement séparable, il existe une valeur optimale de β, telle que les poids trouvés avec Minimerror-L généralisent avec un erreur ε_g numériquement indiscernable de la valeur minimale, qui correspond au perceptron Bayesien [81].

A température T finie, (4.2) est dérivable et l'on peut chercher son minimum par une descente en gradient, ce que l'on ne peut pas faire avec le coût (4.1). Les poids sont modifiés itérativement :

$$\vec{w}(t+1) \leftarrow \vec{w}(t) + \delta\vec{w}(t) \tag{4.5}$$

$$\delta\vec{w}(t) = -\varepsilon \frac{\partial E(\vec{w}\ ;\ \beta)}{\partial \vec{w}} \tag{4.6}$$

L'équation (4.6) peut s'écrire comme d'autres algorithmes d'apprentissage du type itératif Hebbien [81] :

$$\delta\vec{w}(t) \propto \sum_{\mu=1}^{P} c^\mu(t)\tau^\mu \vec{\xi}^\mu \tag{4.7}$$

où le coefficient c^μ dépend de l'algorithme [1]. Minimerror-L a comme coefficient :

$$c^\mu \propto \frac{1}{\cosh^2(\beta\gamma^\mu/2)} \tag{4.8}$$

[1] Par exemple, le *Perceptron de Stabilité Maximale* (MSP) [41] a $c^\mu = \Theta(\kappa - \gamma^\mu)$, où κ est la stabilité imposée à l'exemple le moins stable. Pour la règle de Widrow-Hoff, $c^\mu = 1 - \gamma^\mu$.

qui a son maximum à $\gamma = 0$ et décroît exponentiellement pour des $|\gamma| \gg 2T$. Ceci signifie que la contribution la plus importante à la modification des poids provient des exemples situés dans une fenêtre de largeur $\approx 2T$ des deux cotés de l'hyperplan séparateur, avec des stabilités positives ou négatives. Les exemples se trouvant loin en-dehors de cette fenêtre auront des coefficients exponentiellement petits.

La descente peut être faite par la méthode de gradient simple (4.5) ou de gradient conjugué [75]). Raffin et Gordon [81] ont utilisé cette dernière méthode pour trouver le minimum de (4.2) sur des ensembles linéairement séparables à entrées binaires. Nous avons fait plusieurs tests sur des ensembles non linéairement séparables en utilisant les deux méthodes et nous avons trouvé que la méthode du gradient simple proposée par Gordon et Berchier [39] a des performances supérieures à celle du gradient conjugué, comme on le montre plus loin sur des exemples.

Dans les applications, la valeur de β qui donne les meilleurs résultats est inconnue, car elle dépend de l'ensemble d'apprentissage. L'idée intuitive la plus importante de l'algorithme consiste à l'ajuster en cours d'apprentissage, en combinant un *recuit déterministe* avec la descente en gradient [39] : à chaque itération [2] on décroît T (d'où le nom de recuit). La variation de T implique une modification du coût, ce qui amène à considérer à chaque pas un coût *différent*, contrôlé par la température. Cette procédure permet d'utiliser l'information des exemples qui sont de plus en plus *proches* de l'hyperplan pour le positionner.

Afin de trouver le minimum du coût, on cherche d'abord la direction de la descente en gradient à haute température T_0 [39]. Puisque tous les exemples contribuent à l'apprentissage avec la même "force", ceci revient à utiliser la règle de Hebb. Au fur et à mesure des itérations, on décroît T : la fenêtre devient de plus en plus étroite. Quand la température est suffisamment basse, le nombre d'exemples dans la fenêtre de largeur $2T$ est négligeable et l'apprentissage s'arrête.

A partir de (4.2) et (4.3) on aura :

$$\frac{\partial E(\vec{w}\ ;\ \beta)}{\partial w_i} = \sum_{\mu} \frac{\partial V(\gamma^{\mu}\ ;\ \beta)}{\partial w_i} \tag{4.9}$$

Où chaque terme de la somme est :

$$\frac{\partial V(\gamma^{\mu}\ ;\ \beta)}{\partial w_i} = -\frac{\beta}{4\|\vec{w}\|} \frac{\xi_i^{\mu}\tau^{\mu}}{\cosh^2(\frac{\beta\gamma^{\mu}}{2})} \left\{ \frac{\xi^{\mu}\tau^{\mu}}{\|\vec{w}\|} - \gamma^{\mu}\frac{w_i}{\|\vec{w}\|^2} \right\} \tag{4.10}$$

Mais l'implantation de Minimerror (Gordon et Berchier [39]) utilise seulement :

[2] Une itération dans notre contexte représente le passage de tout l'ensemble d'apprentissage qui fournit l'information pour trouver la bonne position de l'hyperplan séparateur

$$\frac{\partial V(\gamma^\mu \ ; \ \beta)}{\partial w_i} = -\frac{\beta}{4} \frac{\xi_i^\mu \tau^\mu}{\cosh^2(\frac{\beta\gamma^\mu}{2})} \tag{4.11}$$

où le préfacteur $\varepsilon\beta/4$ est remplacé par un seul facteur ϵ :

$$\delta\vec{w}(t) = -\epsilon \frac{\xi_i^\mu \tau^\mu}{\cosh^2(\frac{\beta\gamma^\mu}{2})} \tag{4.12}$$

et le dernier terme de (4.10) est remplacé par une normalisation des poids :

$$\vec{w}(t+1) \leftarrow \frac{\vec{w}(t+1)}{\|\vec{w}(t+1)\|} \tag{4.13}$$

Le choix de la température a une influence directe sur l'aspect du "paysage" défini par la fonction $E(\vec{w} \ ; \ \beta)$: à haute température T on aura un paysage de "collines" douces avec des variations d'amplitude peu importantes entre un sommet et une vallée. Par contre, à basses températures, le paysage aura des formes de paliers avec des variations d'amplitude très fortes entre configurations voisines.

Quand le critère d'arrêt a été satisfait (voir 4.2.2), on fait une dernière minimisation par gradient conjugué, ce qui permet de trouver le minimum du coût. Ce dernier pas doit être fait avec une fonction de coût légèrement différente [81] :

$$E^*(\vec{w} \ ; \ \beta) = \sum_{\mu=1}^{P} V(\gamma^\mu \ ; \ \beta) + (\sqrt{N+1} \ - \ \|\vec{w}\|)^2 \tag{4.14}$$

car, n'ayant pas le droit de normaliser les poids $\vec{w}$ dans la méthode du gradient conjugué, le dernier terme de (4.14) contraint la recherche des poids $\vec{w}$ dans l'hypersphère de rayon $\sqrt{N+1}$.

L'algorithme modifie itérativement les poids suivant (4.5) et (4.6) en utilisant pour chaque pas une température différente. En effet, après avoir modifié tous les poids, la valeur de β est augmentée suivant :

$$\beta(t+1) = \beta(t) + \delta\beta(t) \tag{4.15}$$

où $\delta\beta(t)$ est petit [3] pour que la fonction de coût soit peu modifiée.

Puisque $T = 1/\beta$, la nouvelle température est :

$$T(t+1) = \frac{T(t)}{1 + T(t) \cdot \delta\beta(t)} \simeq T(t) - T^2(t)\delta\beta(t) \tag{4.16}$$

[3] En pratique, $\delta\beta(t)$ est de l'ordre de 10^{-3}.

La température décroît de plus en plus lentement lors des itérations successives. Ce procédé de recherche du minimum à des températures décroissantes est appelé *recuit déterministe*.

Empiriquement, Gordon et Berchier [39] ont trouvé que l'on peut accélérer la convergence si $\delta\beta$ est adapté dynamiquement, de manière à introduire des pas plus grands lorsque le nombre d'erreurs d'apprentissage ne varie pas. Pour cela, on compte le nombre d'itérations t_s successives pendant lesquelles le nombre d'exemples mal classés n'est pas modifié, et $\delta\beta(t)$ est donné par :

$$\delta\beta(t+1) = \delta\beta_0 \log(1 + t_s) \tag{4.17}$$

où $\delta\beta_0$ est une constante. Si à l'itération t le nombre d'erreurs d'apprentissage a diminué, on remet le compteur à zéro $t_s(t) = 0$ et $\delta\beta(t+1) = 0$: l'itération suivante se fait à la même température. Par contre, si $t_s(t) \neq 0$, la température est diminuée avec des pas qui augmentent logarithmiquement avec le temps écoulé depuis la dernière modification du nombre d'erreurs.

Le pas de l'algorithme ϵ, souvent appelé *taux d'apprentissage*, est aussi adaptable. Il est fonction de l'itération t et de β. On commence avec un $\epsilon = \epsilon_0$ petit, de l'ordre de 10^{-2}. On vérifie la valeur de ϵ toutes les t_0 itérations : si le nombre d'erreurs à l'instant t n'a pas changé, on ne modifie pas ϵ. Par contre, si ce nombre a changé, on réduit ϵ en le multipliant par un facteur q entre 0 et 1. Empiriquement, nous avons choisi comme facteur $q = 0.8$, de sorte que :

$$\epsilon(t+1) = q\epsilon(t) \tag{4.18}$$

permet de chercher de plus en plus finement le minimum.

4.2.1 Introduction de deux températures

Gordon et Berchier [39] et Raffin et Gordon [81] ont étudié la performance de l'algorithme Minimerror-L avec l'utilisation de deux températures : une température T_- pour les exemples mal classés, ($\gamma \leq 0$) et une autre, T_+ pour ceux avec stabilité positive ($\gamma > 0$), avec $T_- > T_+$. Les exemples mal classés, sont donc considérés à une température plus haute que les autres. Les deux températures peuvent être interprétées comme deux fenêtres différentes : l'une étroite pour les exemples bien appris (avec des stabilités positives) et une autre plus large pour les exemples à stabilités négatives. Ainsi, puisque $T_- > T_+$, les modifications des poids sont plus sensibles aux exemples non appris. Le rapport des températures est indiqué par :

$$\Psi = \frac{\beta_+}{\beta_-} \tag{4.19}$$

La fonction de coût avec deux températures s'écrit :

$$E(\vec{w}\ ;\ \beta_-, \beta_+) = \sum_{\gamma^\mu \leq 0} V(\gamma^\mu\ ;\ \beta_-) + \sum_{\gamma^\mu > 0} V(\gamma^\mu\ ;\ \beta_+) \tag{4.20}$$

La descente en gradient est donnée par :

$$\vec{w}(t+1) \leftarrow \vec{w}(t) + \delta\vec{w}(t) \tag{4.21}$$

$$\delta\vec{w}(t) = \epsilon \sum_\mu \delta\vec{w}^\mu \tag{4.22}$$

$$\begin{aligned} \delta w_i^\mu &= -\frac{\xi_i^\mu \tau^\mu}{\cosh^2 \frac{\beta_- \gamma^\mu}{2}} \quad \text{si} \quad \gamma^\mu \leq 0 \\ \delta w_i^\mu &= -\frac{\xi_i^\mu \tau^\mu}{\cosh^2 \frac{\beta_+ \gamma^\mu}{2}} \quad \text{si} \quad \gamma^\mu > 0 \end{aligned} \tag{4.23}$$

où nous avons défini $\epsilon = \varepsilon\beta/4$. C'est cette quantité ϵ qui sera considérée désormais le pas d'apprentissage.

On procède de la même façon qu'avec une seule température, en gardant le rapport $\Psi = \beta_+/\beta_-$ constant. Quand le critère de convergence est satisfait, on fait la dernière minimisation (avec la méthode du gradient conjugué) à une seule température, que l'on prend égale à $1/\beta_+$ pour tous les exemples, de manière à minimiser la fonction de coût originale (4.2) à température $T = 1/\beta_+$, qui est la plus petite des deux températures T_+, T_-.

4.2.2 Le critère d'arrêt

Nous avons trouvé un critère d'arrêt adéquat pour la minimisation de la fonction de coût qui permet de bien placer l'hyperplan séparateur (avec une précision donnée), sans besoin de faire les dernières itérations, souvent inutiles.

Pour cela, nous arrêtons les itérations dès que tous les exemples satisfont :

$$|\beta\gamma^{\mu}| > d \tag{4.24}$$

avec $\beta = \beta_+$ si $\gamma^{\mu} > 0$, $\beta = \beta_-$ si $\gamma^{\mu} < 0$. La quantité $d > 0$ est une distance à l'hyperplan séparateur. Avec ce critère, l'algorithme s'arrête dès qu'aucun exemple ne se trouve dans une fenêtre de largeur d/β, des deux côtés de l'hyperplan séparateur, même si le gradient n'est pas nul. Les termes négligés sont tous plus petits que $\cosh^{-2}(d/2)$, et si l'on choisit d suffisamment grand, le fait de négliger ces termes n'aura pas de conséquences. Nous avons trouvé empiriquement que $d = 2.5$ est un bon compromis entre la précision du placement de l'hyperplan et la vitesse de convergence de l'algorithme, indépendamment de l'application.

4.2.3 La normalisation des entrées

L'algorithme Minimerror-L a trois paramètres : le rapport $\Psi = \beta_+/\beta_-$, le pas de l'algorithme ϵ et le choix initial du pas de refroidissement $\delta\beta$. Afin de rendre ces paramètres les plus indépendants possible de la nature du problème, nous avons choisi de faire une normalisation des entrées $\vec{\xi^{\mu}}$. En appliquant une transformation linéaire adéquate, on peut ramener toutes les variables à avoir des ordres de grandeur semblables [8]. Ainsi, nous avons pu déterminer des valeurs *standard* pour les paramètres de l'algorithme. Ceci permet d'obtenir des résultats satisfaisants sans avoir à les régler pour chaque nouveau problème.

Nous appliquons la transformation linéaire standard de chaque composante :

$$\widetilde{\xi}_i^{\mu} = \frac{\xi_i^{\mu} - \langle \xi_i \rangle}{\triangle_i} \tag{4.25}$$

où $1 \leq i \leq N$. La moyenne $\langle \xi_i \rangle$ et la variance $\triangle_i$ de chaque composante étant définies comme d'habitude :

$$\langle \xi_i \rangle = \frac{1}{P} \sum_{\mu=1}^{P} \xi_i^{\mu} \tag{4.26}$$

$$\triangle_i{}^2 = \frac{1}{P}\sum_{\mu=1}^{P}\left(\xi_i^\mu - \langle\xi_i\rangle\right)^2 = \frac{1}{P}\sum_{\mu=1}^{P}(\xi_i^\mu)^2 - \langle\xi_i\rangle^2 \tag{4.27}$$

leur calcul ne nécessite qu'une seule lecture de l'ensemble des P exemples.

La recherche des poids se fait dans un nouvel espace d'entrées standardisées $\widetilde{\vec{\xi}^\mu}$, où on va noter les poids $\vec{j}$, et la stabilité $\widetilde{\gamma}^\mu$; on écrit :

$$\begin{aligned}
\widetilde{\gamma}^\mu &= \frac{\tau^\mu}{\|\vec{j}\|}\left\{\vec{j}\cdot\widetilde{\vec{\xi}^\mu}\right\} && (4.28)\\
&= \frac{\tau^\mu}{\|\vec{j}\|}\left\{j_0 + \sum_{i=1}^{N}\left(\frac{\xi_i^\mu - \langle\xi_i\rangle}{\triangle_i}\right)j_i\right\} && (4.29)\\
&= \frac{\tau^\mu}{\|\vec{j}\|}\left\{\sum_{i=1}^{N}\frac{j_i}{\triangle_i}\xi_i^\mu \;+\; j_0 - \sum_{i=1}^{N}\frac{j_i}{\triangle_i}\langle\xi_i\rangle\right\} && (4.30)
\end{aligned}$$

Après l'apprentissage, la transformation inverse est appliquée aux poids :

$$\begin{aligned}
w_0 &= \|\vec{j}\|\frac{j_0 - \sum_{i=1}^{N} j_i\langle\xi_i\rangle/\triangle_i}{\sqrt{\left[j_0 - \sum_{i=1}^{N} j_i\langle\xi_i\rangle/\triangle_i\right]^2 + \left[\sum_{i=1}^{N} j_i/\triangle_i\right]^2}}\\
w_i &= \|\vec{j}\|\frac{j_i/\triangle_i}{\sqrt{\left[j_0 - \sum_{i=1}^{N} j_i\langle\xi_i\rangle/\triangle_i\right]^2 + \left[\sum_{i=1}^{N} j_i/\triangle_i\right]^2}} \qquad (4.31)
\end{aligned}$$

La standardisation (4.26) et (4.27) est complètement transparente pour l'utilisateur : avec les poids renormalisés (4.31), le classifieur s'applique aux nouveaux exemples, définis dans les unités de l'utilisateur, sans besoin de les standardiser.

Ce procédé correspond à ramener tous les exemples de l'ensemble d'apprentissage au voisinage d'un hypercube de coté 2, centré sur la moyenne. Le nouvel ensemble ainsi défini, $\widetilde{\vec{\xi}^\mu}$ a la propriété d'avoir des composantes de moyenne zéro et de déviation standard unitaire. Ils ont donc une distribution dont les deux premiers moments sont identiques à ceux des exemples binaires utilisés pour mettre au point les valeurs numériques des paramètres ϵ, $\Psi = \beta^+/\beta^-$ et $\delta\beta$ de l'algorithme. Les poids après l'apprentissage sont renormalisés pour être appliqués dans l'espace initial des entrées. On n'a pas besoin de mémoriser $\langle\xi_i\rangle$ ni $\triangle_i^2$. D'autres algorithmes ont besoin de garder $\langle\xi_i\rangle$ et $\triangle_i^2$, ce qui en principe équivaut à augmenter la complexité de la machine.

La minimisation de la fonction $E(\vec{w}\ ;\ \beta_-, \beta_+)$ est réalisée avec les valeurs suivantes :

$$\begin{aligned} \frac{\beta_+}{\beta_-} &= 6 \\ \epsilon &= 0.02 \\ \delta\beta_0 &= 0.001 \end{aligned}$$

Les valeurs de ces paramètres ont été déterminées pour des exemples à entrées binaires situés sur les sommets d'un hypercube à dimension N (comme par exemple les problème du XOR ou de la N-Parité [4]). Ces valeurs donnent de bons résultats dans la plupart des cas.

4.2.4 Exemples

Dans cette section nous allons présenter les performances de Minimerror-L sur quelques problèmes, ainsi que des tests sur la robustesse des solutions trouvées.

Dépendance suivant la méthode de minimisation

Cette expérience nous a permis de choisir la méthode de minimisation du coût (4.3). Pour cela, nous avons préparé des ensembles d'apprentissage $\mathcal{L}^\alpha$ avec des exemples à entrées binaires $\vec{\xi}$ choisies au hasard avec des probabilités $p(\xi_i^\mu = +1) = p(\xi_i^\mu = -1) = \frac{1}{2}$ et sorties $\tau = \pm 1$ avec des probabilités $p(\tau = +1) = p(\tau = -1) = \frac{1}{2}$. Ceci permet d'obtenir des ensembles d'apprentissage avec une forte probabilité de n'être pas linéairement séparables, même pour des valeurs de N petites. Nous avons créé des ensembles $\mathcal{L}^\alpha$ pour des valeurs $\alpha = 2, 3, \cdots, 6$, avec $N = 20$ et $P = \alpha N$. Le nombre de simulations a été de 30.

[4] La $N-$Parité est la fonction booléenne qui a comme sortie $\tau^\mu = +1$ si le nombre de bits dans l'état $+1$ d'une entrée binaire $\vec{\xi}^\mu$ est pair, $\tau^\mu = -1$ autrement. La $N-$Parité à deux entrées est le complément du célèbre problème du XOR, qui a démontré la faiblesse du perceptron [70, 49] dans des tâches non LS et qui sert (encore) pour tester des algorithmes.

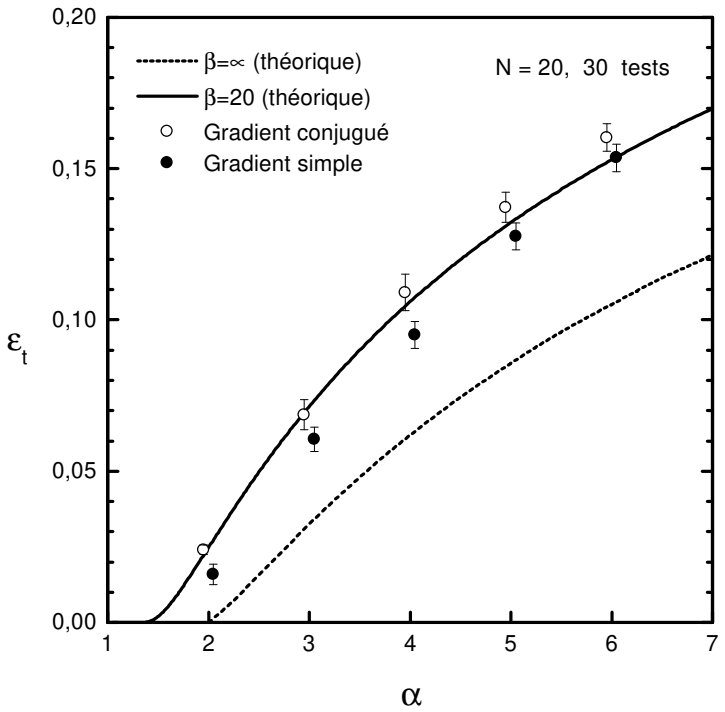

Figure 4.2: Erreur d'apprentissage ε_t obtenue avec gradients simple et conjugué en fonction de α. $N = 20$ entrées binaires et sortie $\tau = \pm 1$ aléatoire. Moyennes sur 30 ensembles d'apprentissage.

Les méthodes de gradient simple (4.5) et de gradient conjugué ont été comparées sur les mêmes ensembles d'apprentissage : l'apprentissage à été fait avec le recuit déterministe avec deux températures et la minimisation finale à β avec un seul pas de gradient conjugué, comme on l'avait déjà indiqué.

Nous présentons dans la figure 4.2 la fraction d'erreurs d'apprentissage ε_t comme fonction de α ayant appris soit avec le gradient simple soit avec le gradient conjugué. Nous montrons aussi les courbes théoriques correspondants à $\beta = 20$ et $\beta = \infty$. On voit que le gradient simple permet de bien classer une quantité plus grande d'exemples que le gradient conjugué. Bien que le gradient conjugué ait bien des avantages sur le gradient simple (rapidité de convergence grâce à la moindre sensibilité de l'augmentation du $\beta_{\pm}$) dans le cas d'exemples qui sont linéairement séparables [81], ce n'est pas le même comportement si les exemples ne le sont pas. Du fait que le gradient conjugué cherche toujours le minimum *minimorum* de la fonction de coût (avec une seule itération), il peut rester piégé dans un minimum local. Par contre, avec le gradient simple nous nous approchons du minimum mais petit à petit, et la descente contrôlée de la température permet, en principe d'échapper des minimums locaux.

Dépendance suivant l'initialisation des poids

Pour vérifier expérimentalement l'importance du point de départ des poids pour la minimisation, nous avons préparé des ensembles d'apprentissage $\mathcal{L}^\alpha$ non linéairement séparables avec des entrées $\vec{\xi}$ choisies au hasard et des sorties $\tau = \pm 1$, de la même manière que pour le test précédent. Nous avons créé 30 ensembles $\mathcal{L}^\alpha$ pour des valeurs $\alpha = 2, 3, \cdots, 6$, en choisissant la méthode du gradient simple pour la minimisation, compte tenu des résultats obtenus dans l'expérience antérieure. L'initialisation des poids $\vec{w}$ a été :

- (a) Au hasard : $w_i =$ aléatoire $[-1, +1]$; $i = 0, \cdots, N$
- (b) Avec la règle de Hebb : $w_i = \sum_\mu \xi_i^\mu \tau^\mu$; $i = 0, \cdots, N$

Dans les deux cas, on normalise les poids : $\vec{w} \leftarrow \vec{w}/\|\vec{w}\|$.

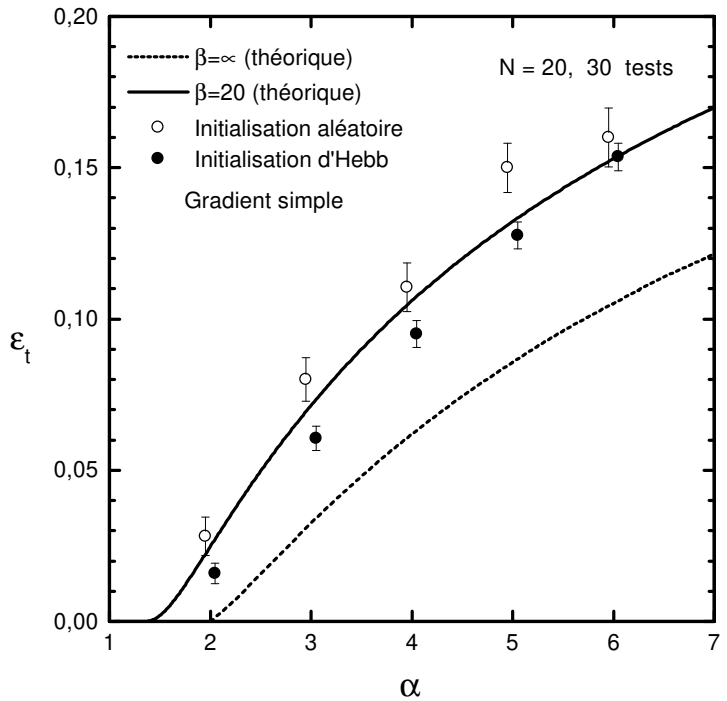

Figure 4.3: Dépendance à l'initialisation Hebb/Aléatoire. Erreur d'apprentissage ε_t obtenue avec le gradient simple en fonction de α, pour $P = \alpha N$ exemples. $N = 20$ entrées binaires et sortie $\tau = \pm 1$ aléatoire. Moyennes sur 30 ensembles d'apprentissage.

A partir des courbes de la figure 4.3, on peut conclure qu'il est préférable d'initialiser les poids avec la règle de Hebb, en bon accord avec le résultat théorique : le minimum du coût à haute température correspond à la règle de Hebb.

Comparaison du nombre d'époques

Nous avons comparé le nombre d'itérations (ou époques) nécessaires pour résoudre une tâche linéairement séparable avec Minimerror-L par rapport à l'algorithme du perceptron. Pour cela, nous avons créé des ensembles d'apprentissage linéairement séparables engendrés par un perceptron professeur $\vec{v}$, les entrées binaires $\vec{\xi}$ ont été choisies au hasard avec des probabilités $p(\xi_i^\mu = +1) = p(\xi_i^\mu = -1) = \frac{1}{2}$ et les sorties suivant $\tau(\vec{\xi}) = \text{signe}(\vec{v} \cdot \vec{\xi})$.

Des ensembles d'apprentissage pour $N = 20, 50, 100$ avec $P = \alpha N$ exemples ($\alpha = 0.5, 1, 1.5, \cdots, 10$) ont été utilisés. Dans les figures 4.4 ($N = 20, N = 50$ et $N = 100$) on montre que Minimerror-L converge en un nombre d'époques plus ou moins indépendant de la taille de l'ensemble d'apprentissage. Par contre, l'algorithme du perceptron nécessite un nombre d'époques croissant avec α *et avec* N pour pouvoir séparer les mêmes exemples. Mais bien entendu, une époque de l'algorithme du perceptron est beaucoup plus simple que celle de Minimerror-L.

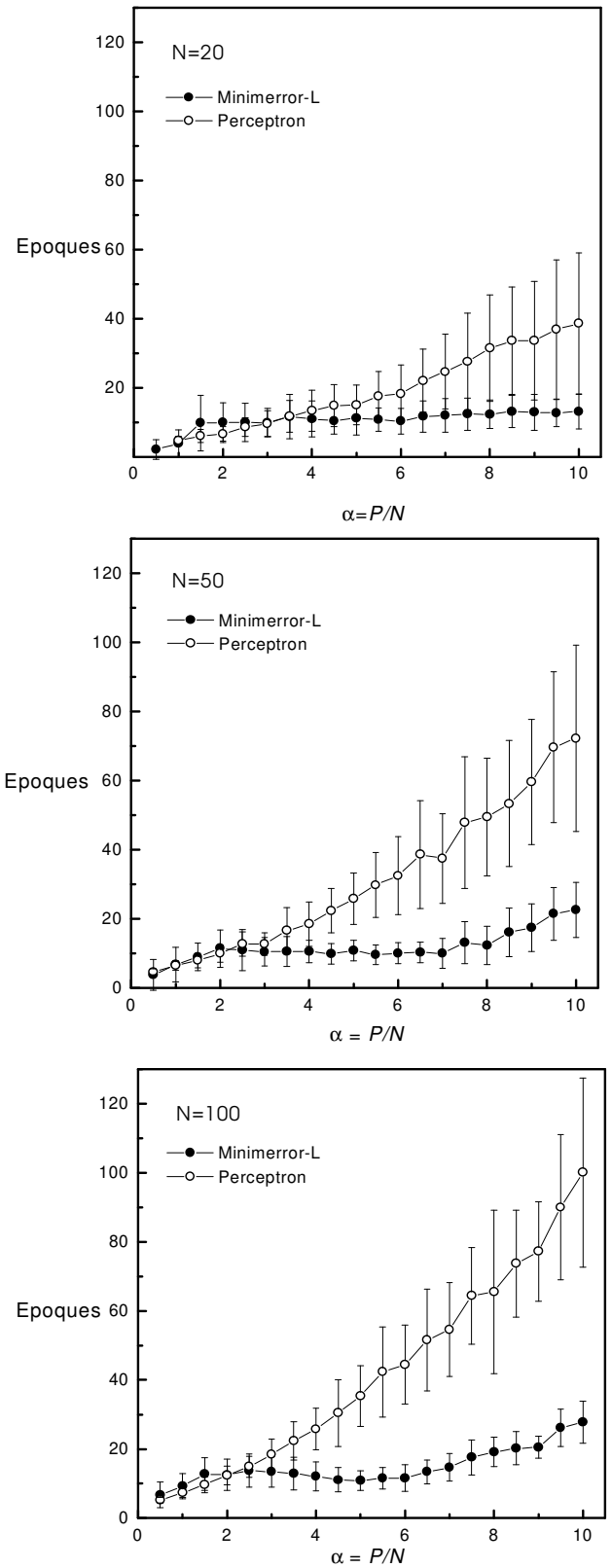

Figure 4.4: Ensembles linéairement séparables : comparaison des performances de Minimerror-L avec l'algorithme du perceptron. Moyennes sur 30 ensembles d'apprentissage.

Illustrations à deux dimensions

- 1) Nous montrons d'abord un problème à deux dimensions $N = 2$ avec un ensemble $\mathcal{L}^\alpha$ de $P = 200$ exemples générés au hasard avec des probabilités $p(\xi_i^\mu = +1) = p(\xi_i^\mu = -1) = \frac{1}{2}$. Les classes τ sont données par un perceptron professeur de poids $\vec{v}$, suivant $\tau = \text{signe}(\vec{v} \cdot \vec{\xi})$ qui a assuré la séparabilité linéaire. Nous avons mesuré l'erreur de généralisation par $\varepsilon_g = \frac{1}{\pi}\arccos(R)$ où :

$$R = \frac{\vec{v} \cdot \vec{w}}{\|\vec{v}\|\|\vec{w}\|} \tag{4.32}$$

  Nous montrons dans la figure 4.5 le professeur $\vec{v}$ et la solution $\vec{w}$ trouvée par Minimerror-L.

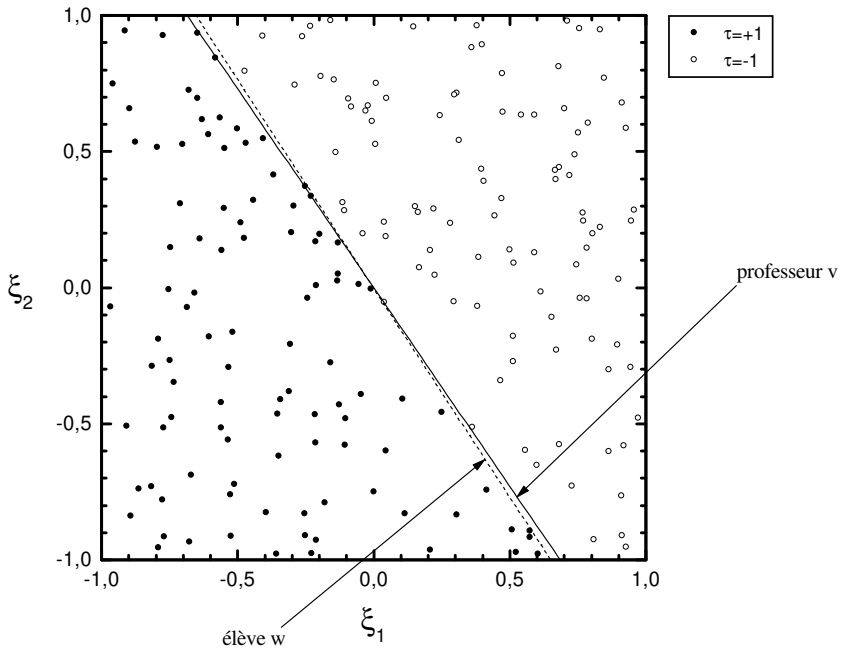

Figure 4.5: Solution d'un ensemble linéairement séparable engendré par un perceptron professeur $\vec{v}$, avec $N = 2, P = 200$. Le perceptron élève est $\vec{w}$, et $R = 0.999831$. $\varepsilon_g = \frac{1}{\pi}\arccos(R) = 0.018$.

- 2) Nous montrons ensuite un problème non linéairement séparable à deux dimensions $N = 2$. Nous avons préparé un ensemble d'apprentissage $\mathcal{L}^\alpha$ de $P = 200$ exemples à entrées binaires $\vec{\xi}$ choisies au hasard avec des probabilités $p(\xi_i^\mu = +1) = p(\xi_i^\mu = -1) = \frac{1}{2}$. Un perceptron professeur de poids $\vec{v}$, donne les sorties $\tau = \text{signe}(\vec{v} \cdot \vec{\xi})$, que nous avons bruitées en changeant au hasard le signe de la classe des exemples proches de l'hyperplan défini par le professeur. La classe des exemples dont la stabilité est inférieure à 0.1, c'est-à-dire, les

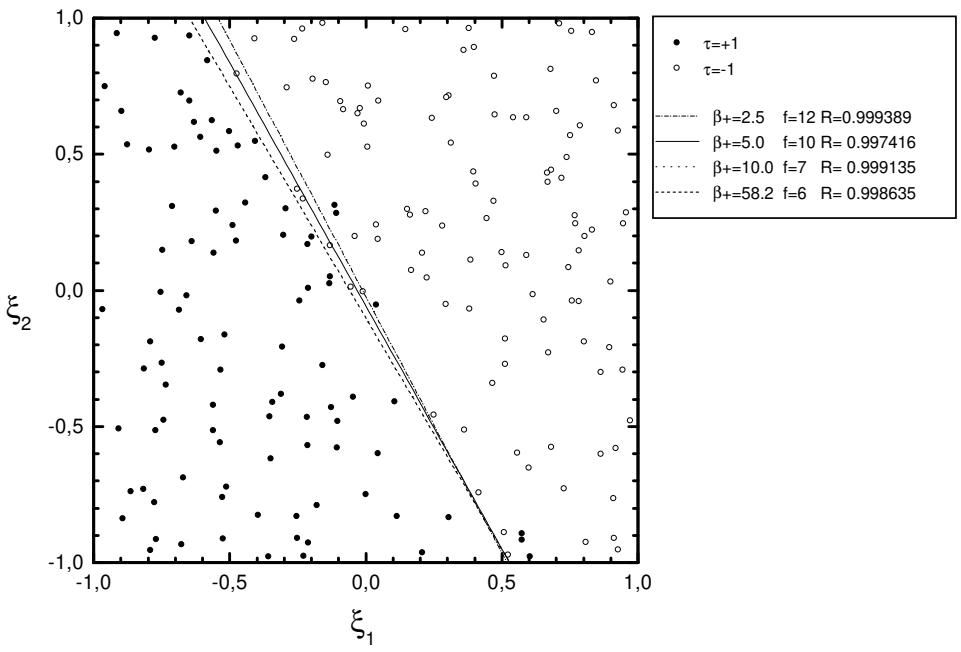

Figure 4.6: Solutions trouvées avec un ensemble non linéairement séparable. $N = 2$, $P = 200$. L'apprentissage a été arrêté à des valeurs de $\beta_+ = 2.5, 5, 10$ et $\beta_+ = 58.2$, donné par le critère d'arrêt, qui permet de trouver le nombre minimum de fautes $f = 6$.

exemples se trouvant à une distance de moins de 0.1 de l'hyperplan professeur, est mis à $-\tau$. L'ensemble ainsi obtenu est non linéairement séparable. Dans la figure 4.6 nous montrons plusieurs solutions $\vec{w}^k$ en fonction de la valeur du paramètre β où nous avons arrêté la minimisation.

Robustesse des solutions

La robustesse des réseaux de neurones implantés à l'aide de circuits électroniques digitaux peut s'avérer une propriété importante dans certains milieux hostiles (bruit, rayonnement) [13, 90, 79]. L'effet de ces milieux peut se traduire soit par des altérations des poids mémorisés, soit par des altérations des entrées présentées [5]. C'est pourquoi nous nous sommes intéressés aux effets des altérations des bits sur des perceptrons entraînés avec Minimerror-L, modifiant tant les poids $\vec{w}$ que les entrées $\vec{\xi}$.

[5] En particulier, l'un des effets considérés critiques pour des applications embarquées (satellites, sondes,...) est l'inversion du contenu des cellules de mémorisation comme résultat de l'impact d'une particule chargée (effet dit *Single Event Upset.*)

Nous avons étudié des perceptrons de taille $N = 50$. Les ensembles d'apprentissage $\mathcal{L}^\alpha$ ont été engendrés par un perceptron professeur, ce qui assure la séparabilité linéaire. L'influence de la taille des ensembles d'apprentissage (contrôlée par α), sur la probabilité de généralisation a été l'un des paramètres considérés. Nous avons fait trois types de tests [3]:

- 1. Altération des entrées.

  Nous avons altéré b bits au hasard pour chaque exemple μ de l'ensemble d'apprentissage. Si la réponse du perceptron face à l'exemple corrompu est la même que celle face à l'exemple original, on le considère comme étant bien reconnu. Des moyennes sur 200 ensembles d'apprentissage différents de la fraction d'exemples bien reconnus ρ (pourcentage de réponses correctes) pour $\alpha = 1, 2, \cdots, 6$ en fonction du nombre de bits altérés sont montrées dans la figure 4.7. La dégradation est approximativement linéaire par rapport à b, à α constante :

$$\rho(\%) \approx \frac{100(N-b)}{N}$$

- 2. Altération des poids codés comme des nombres réels.

  Nous avons déterminé la quantité ρ par rapport à l'altération des poids. On a stocké les poids comme des valeurs réelles d'après le standard IEEE 754 [6]. Pour chaque perceptron entraîné, la sortie ζ a été évaluée après avoir modifié chacun des bits de chaque poids. La moyenne de ρ sur $N = 50$ poids de chaque perceptron sur $l = 50$ ensembles d'apprentissage est présentée dans la figure 4.8 comme fonction de la position du bit altéré et dans la figure 4.9 comme fonction de α. Comme on l'attendait, la modification du bit de signe perturbe la reconnaissance de l'exemple, l'altération de l'exposant est catastrophique mais seulement 4 ou 5 bits de la mantisse semblent être sensibles. En effet, la mantisse est codée sur 23 bits et les perturbations causent des erreurs seulement pour 40% des bits de poids fort.

[6] Dans la norme IEEE 754, le bit 0 correspond au signe, les bits 1 à 8 à l'exposant et les bits 9 à 31 à la mantisse.

- 3. Altération des poids comme des nombres entiers.

  Le même test qu'en 2 a été refait, mais en stockant les poids comme des entiers codés sur 16 bits, le bit 0 étant le bit de signe. Comme on voit dans les figures 4.10 en fonction du numéro du bit et 4.11 en fonction de α, la reconnaissance globale est moins dramatiquement affectée pour des poids entiers que pour des poids réels. En effet, les bits 8 à 15 n'ont guère d'influence sur la sortie.

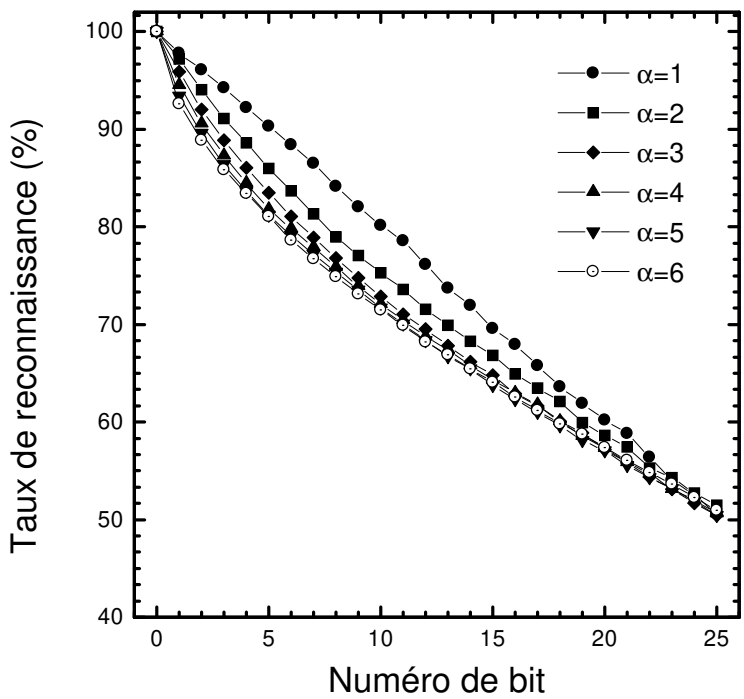

Figure 4.7: Pourcentage de dégradation vs. le nombre de bits altérés, pour $N = 50$. Les écart-types (de l'ordre de 10%) ne sont pas montrés pour des raisons de clarté du dessin.

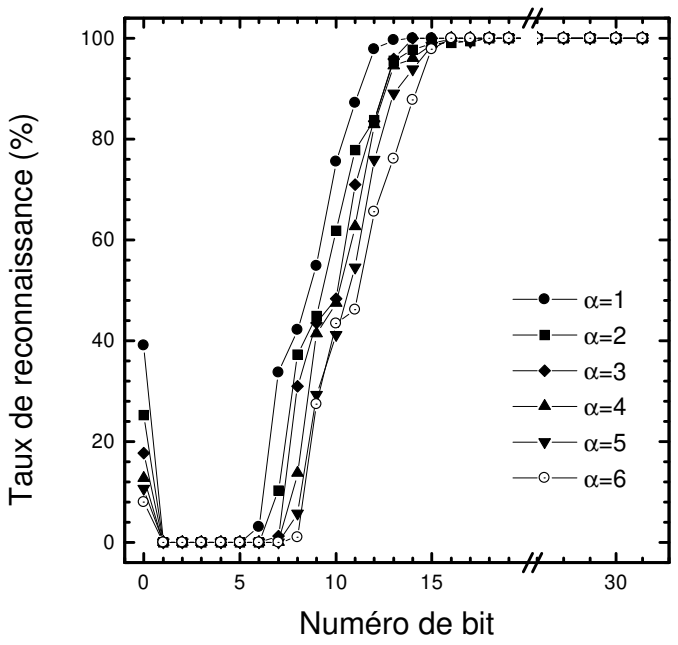

Figure 4.8: Dégradation moyenne de la reconnaissance en fonction du numéro de bit $i = 0, \cdots, 31$. $N = 50$. Poids stockés comme réels de 32 bits. Par la clarté du dessin, les écart-types (de l'ordre de 10%) ne sont pas montrés.

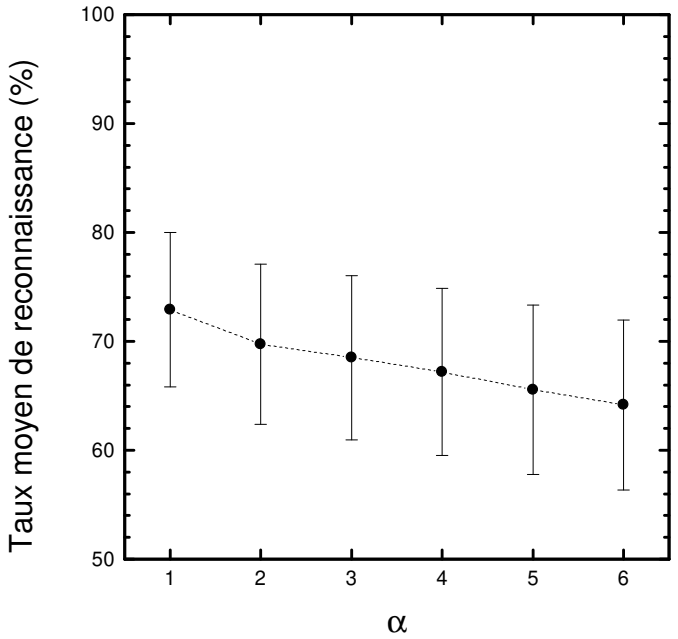

Figure 4.9: Dégradation moyenne de la reconnaissance en fonction de la taille de l'ensemble d'apprentissage, α. Poids stockés comme réels de 32 bits.

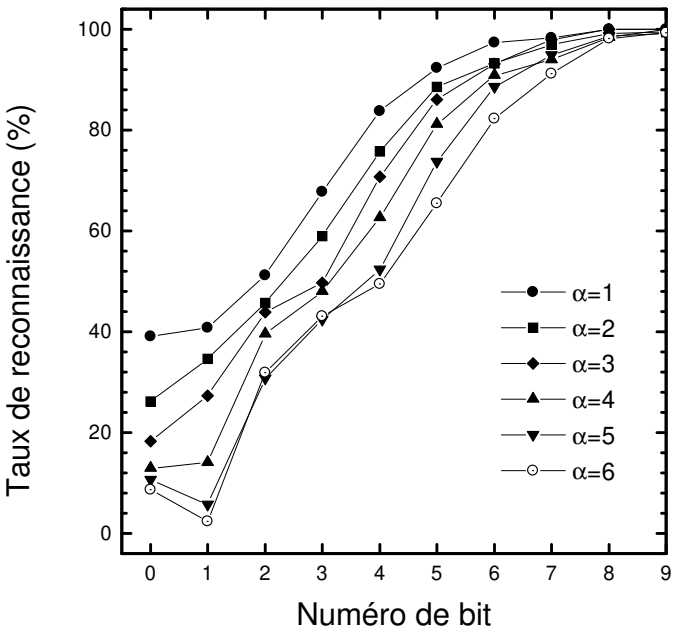

Figure 4.10: Dégradation de la reconnaissance en fonction du numéro de bit $i = 0, \cdots, 15$. $N = 50$. Poids stockés comme entiers de 16 bits. Les écart-types (de l'ordre de 10%) ne sont pas montrés.

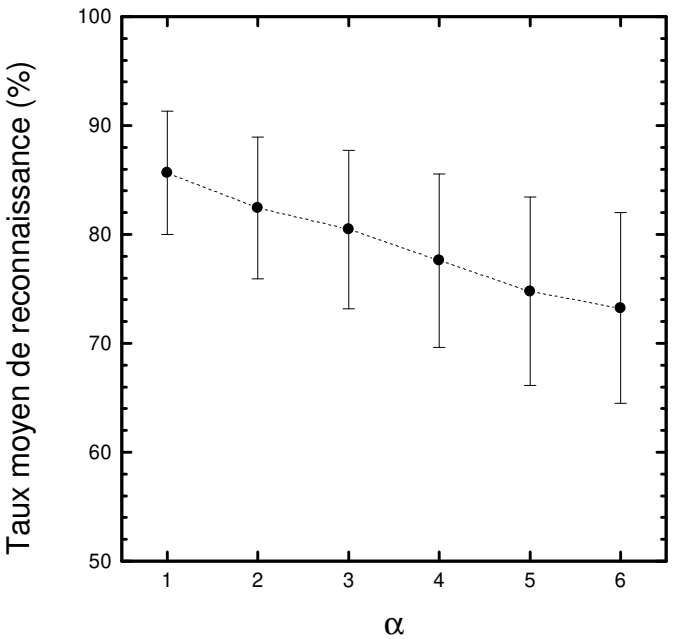

Figure 4.11: Dégradation moyenne de la reconnaissance en fonction de la taille de l'ensemble d'apprentissage, α. Poids stockés comme entiers de 16 bits.

4.3 Minimerror-S

Dans la figure 4.12 est donné un exemple de points qui ne sont pas linéairement séparables par un hyperplan, mais qu'on peut séparer par une hypersphère. On dit alors, que ils sont *sphériquement séparables* (SS).

Ceci a été l'idée de base qui nous a inspiré pour construire une variante de Minimerror, mais en utilisant des hypersphères pour faire des dichotomies. Une dichotomie sphérique est définie par la surface de l'hypersphère de rayon ρ et de centre $\vec{w}$, de manière à attribuer une classe aux vecteurs $\vec{\xi}$ se trouvant à l'intérieur de l'hypersphère et la classe opposée à ceux se trouvant à l'extérieur.

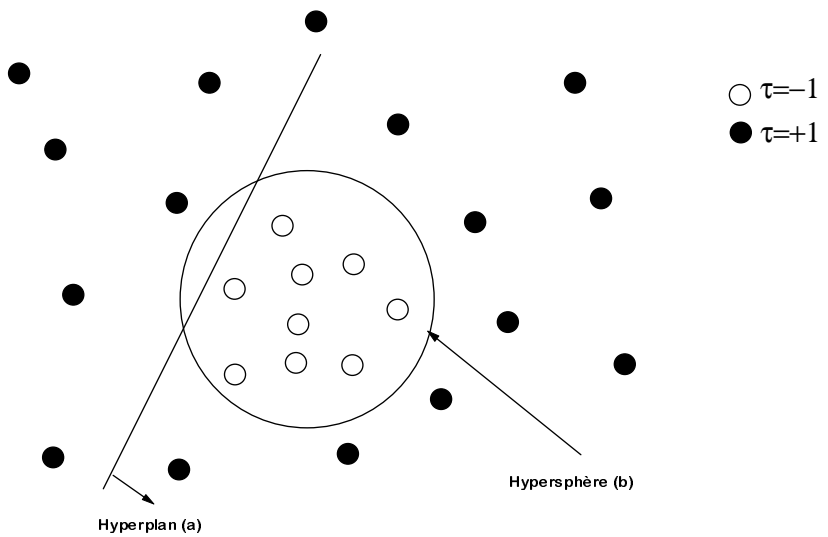

Figure 4.12: Exemple schématique de données en deux dimensions à deux classes avec polarité négative (exemples à sortie $\tau = -1$ entourés des exemples de la classe opposée). (a) Solution avec un hyperplan trouvé par Minimerror-L. (b) Une solution alternative sans erreurs, qui peut être trouvée avec Minimerror-S.

4.3.1 La stabilité sphérique λ

En supposant que les exemples à sortie $\tau = +1$ entourent les exemples à sortie $\tau = -1$ (fait que nous appellerons *polarité positive*), nous avons défini une nouvelle stabilité, que nous appellerons désormais stabilité sphérique λ, en utilisant le champ radial (2.17) de la façon suivante :

$$\lambda^{\mu}(\vec{w}) = \tau^{\mu}\left[\sum_{i=1}^{N}(w_i - \xi_i^{\mu})^2 - \theta^2\right] \equiv \tau^{\mu}[\|\vec{w} - \vec{\xi}^{\mu}\|^2 - \rho^2] \tag{4.33}$$

où nous avons noté le biais θ comme un rayon ρ. La valeur $\sqrt{|\lambda^\mu|}$ est une mesure de la distance de l'exemple μ à la frontière de l'hypersphère centrée sur $\vec{w}$, et de rayon ρ, comme cela est montré dans la figure 4.13. Le signe de λ^μ est positif si l'exemple $\vec{\xi}^\mu$ se trouve à l'intérieur de la sphère et sa classe est -1, ou s'il se trouve à l'extérieur et sa classe est $+1$. Donc, λ^μ a des propriétés semblables à celles de la stabilité dans le cas des séparations linéaires. En particulier une valeur positive de λ^μ assure une classification correcte, et plus λ^μ est grand, plus robuste sera la solution trouvée.

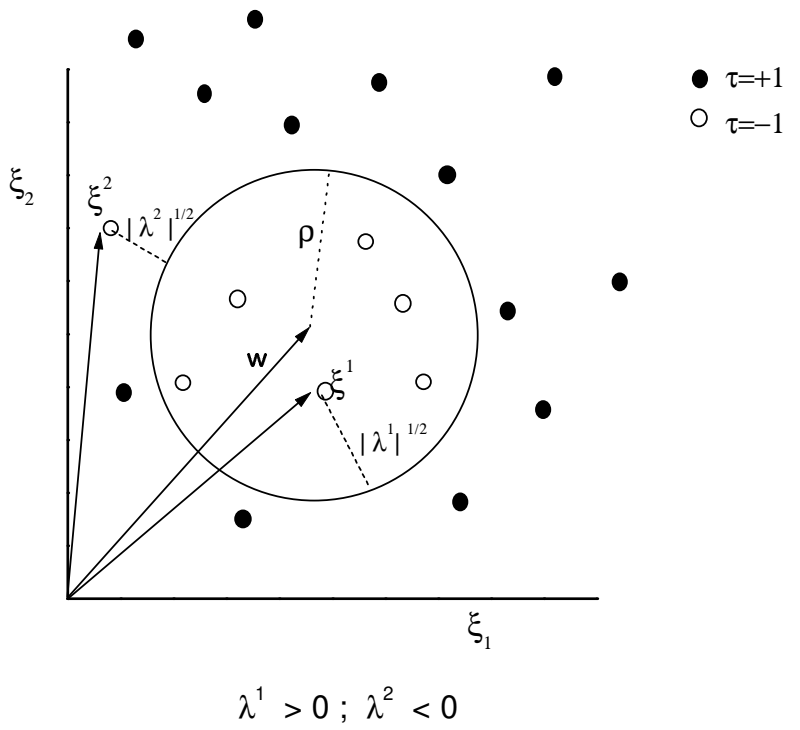

Figure 4.13: La stabilité sphérique λ^μ de l'exemple μ par rapport à l'hypersphère centrée sur $\vec{w}$, de rayon ρ.

Polarité négative ou positive?

La supposition de polarité négative dans l'équation (4.33) pourrait ne pas être adéquate : il n'existe aucune raison pour que les exemples soient distribués de cette manière. Là encore, deux situations doivent être considérées : l'une où les exemples dans l'espace des entrées n'ont pas une polarité positive ni négative, ou bien l'autre qui considère une polarité positive. La première situation indéfinie peut être traitée aussi bien avec la même stabilité à polarité négative (4.33) qu'auparavant. Si l'on suppose une polarité positive, les définitions de stabilité auraient le signe opposé.

Le problème suivant se pose : quelle hypothèse de polarité doit être faite ? Nous avons résolu le problème de façon pragmatique : en utilisant une *polarité* π, qui peut être mise à $\pi = \pm 1$. Ainsi nous apprenons des poids avec $\pi = +1$ et avec $\pi = -1$. Après l'apprentissage, on compte le nombre de fautes commises par chacune des solutions, avec $\pi = +1$ et $\pi = -1$ et on garde celle qui fait le moindre nombre d'erreurs. C'est-à-dire, on utilise la définition de stabilité suivante :

$$\lambda^{\mu}(\vec{w}) = \pi\tau^{\mu}\left[\sum_{i=1}^{N} \|w_i - \xi_i^{\mu}\|^2 - \rho^2\right] \equiv \pi\tau^{\mu}\left(\|\vec{w} - \vec{\xi}^{\mu}\|^2 - \rho^2\right) \tag{4.34}$$

Après l'apprentissage, lorsque les poids $\vec{w}, \rho$ sont appris, la sortie donnée par le perceptron sphérique à une entrée $\vec{\xi}$ quelconque est :

$$\sigma = \text{signe}\left\{\pi\left[\|\vec{w} - \vec{\xi}\|^2 - \rho^2\right]\right\} \tag{4.35}$$

Ainsi, avec l'hypothèse de polarité positive ($\pi = +1$), les exemples à l'extérieur de l'hypersphère auront une sortie positive tandis que si la polarité est négative, c'est les exemples à l'intérieur de l'hypershpère qui auront une sortie positive.

Avec la définition de stabilité (4.34), il peut se présenter la difficulté suivante : les exemples qui sont à l'intérieur de l'hypersphère contribueront à l'apprentissage avec une *intensité* maximale limitée par la valeur du rayon ρ, car si l'exemple est situé au centre de la sphère, alors $|\lambda^{max}| = \rho^2$. Par contre, les exemples situés loin à l'extérieur de la sphère peuvent avoir des stabilités bien supérieures en valeur absolue. Bien que ceci ne soit peut-être pas trop important, car le recuit déterministe de Minimerror restreint l'apprentissage aux exemples qui sont de plus en plus proches du bord de l'hypersphère, nous avons considéré une définition alternative pour la stabilité. En effet, le champ radial (2.17) peut être défini comme :

$$h_{\log}^{r} = \log\left[\sum_{i=1}^{N} \|w_i - \xi_i^{\mu}\|^2\right] \tag{4.36}$$

et la stabilité correspondante par :

$$\lambda_{\log}^{\mu}(\vec{w}) = \pi\tau^{\mu}\log\left[\sum_{i=1}^{N} \frac{\|w_i - \xi_i^{\mu}\|^2}{\rho^2}\right] \equiv \pi\tau^{\mu}\log\left[\frac{\|\vec{w} - \vec{\xi}^{\mu}\|^2}{\rho^2}\right] \tag{4.37}$$

qui peut avoir des valeurs de $|\lambda^{\mu}| \to +\infty$ aussi bien si $\vec{\xi}$ est à l'infini, à l'extérieur de l'hypersphère, qu'au centre.

Les problèmes de stabilité numérique, quand l'exemple est très proche du centre, sont facilement contournables en limitant par une borne inférieure les valeurs possibles de l'argument (*cut-off*), pour calculer le logarithme.

Il est important de remarquer que, quelle que soit la définition utilisée pour la stabilité, la sortie est donnée par (4.35), car le signe de $\log \|\vec{w} - \vec{\xi}\|^2 - \log \rho^2$ est le même que celui de $\|\vec{w} - \vec{\xi}\|^2 - \rho^2$.

Nous nous sommes posés la question de quelle est la définition de la stabilité qui donne les meilleurs résultats. Nous avons utilisé les deux expressions pour la stabilité sphérique pour faire plusieurs tests sur des ensembles séparables (LS ou SS) ou non sphériquement séparables. Nous avons corroboré numériquement que les deux approches sont équivalentes, différant seulement par le nombre d'époques nécessaires pour converger : la méthode qui utilise l'expression logarithmique (4.37) nécessite beaucoup plus de temps que celle qui utilise l'expression euclidienne, car il faut calculer la valeur des logarithmes et comparer avec la borne inférieure ou *cut-off* à chaque itération.

Il est important de dire qu'ici on ne fait pas une normalisation des poids $\vec{w}$, comme pour la définition de la stabilité linéaire. Ceci nous permet d'avoir des hypersphères de tous les rayons ρ possibles et centrées dans des points $\vec{w}$ quelconques. Ainsi, si l'on prend un centre suffisamment éloigné des exemples et un rayon suffisamment grand les perceptrons sphériques sont équivalents à des perceptrons linéaires.

L'autre propriété intéressante qu'on avait déjà signalée est que les perceptrons sphériques et linéaires ont la même complexité, car le rayon de l'hypersphère prend la place du biais : $\rho \sim w_0$. Donc, la complexité d'un réseau de neurones à perceptrons sphériques reste la même que celle d'un réseau construit avec des perceptrons linéaires.

Introduisant les définitions (4.34) ou (4.37) dans la fonction de coût (4.20), de Minimerror avec deux températures (T_- et T_+, avec $T_- > T+$, et un rapport $\Psi = \beta_+/\beta_-$),

$$E_\lambda(\vec{w}\ ;\ \beta_-, \beta_+) = \sum_{\lambda^\mu \leq 0} V(\lambda^\mu\ ;\ \beta_-) + \sum_{\lambda^\mu > 0} V(\lambda^\mu\ ;\ \beta_+) \tag{4.38}$$

avec $V(\lambda, \beta_\pm) = \frac{1}{2}(1 - \tanh(\frac{\beta_\pm \lambda}{2}))$, on peut trouver par minimisation les paramètres de l'hypersphère ($\vec{w}$ et ρ) que nous continuerons à appeler poids.

L'algorithme de minimisation, que nous appellerons Minimerror-S, fait une descente en gradient qui s'écrit :

$$\begin{aligned} \delta\vec{w}(t) &= -\varepsilon\frac{\partial E_\lambda(\vec{w}\ ;\ \beta_-,\beta_+)}{\partial\vec{w}} = -\varepsilon\sum_\mu \frac{\partial V(\lambda^\mu\ ;\ \beta_-,\beta_+)}{\partial w_i} \\ \delta\rho(t) &= -\varepsilon\frac{\partial E_\lambda(\vec{w}\ ;\ \beta_-,\beta_+)}{\partial\rho} = -\varepsilon\sum_\mu \frac{\partial V(\lambda^\mu\ ;\ \beta_-,\beta_+)}{\partial\rho} \end{aligned} \tag{4.39}$$

Avec la définition euclidienne (4.33) de la stabilité, on a, pour le centre :

$$\frac{\partial V(\lambda\ ;\ \beta_\pm)}{\partial w_i} = \frac{\beta_\pm}{2}\frac{(w_i-\xi_i^\mu)}{\cosh^2(\frac{\beta_\pm\lambda^\mu}{2})}\tau^\mu\pi \tag{4.40}$$

et pour le rayon :

$$\frac{\partial V(\lambda\ ;\ \beta_\pm)}{\partial\rho} = -\frac{\beta_\pm}{2}\frac{\rho\tau^\mu\pi}{\cosh^2(\frac{\beta_\pm\lambda^\mu}{2})} \tag{4.41}$$

Si l'on utilise la définition logarithmique (4.37) de la stabilité, on a :

$$\frac{\partial V(\lambda_{\log}\ ;\ \beta_\pm)}{\partial w_i} = -\frac{\beta_\pm}{4\|\vec{w}-\vec{\xi}\|^2}\frac{(w_i-\xi_i^\mu)\tau^\mu\pi}{\cosh^2(\frac{\beta_\pm\lambda_{\log}^\mu}{2})} \tag{4.42}$$

et pour le rayon :

$$\frac{\partial V(\lambda_{\log}\ ;\ \beta_\pm)}{\partial\rho} = \frac{\beta_\pm}{4\rho}\frac{\tau^\mu\pi}{\cosh^2(\frac{\beta_\pm\lambda_{\log}^\mu}{2})} \tag{4.43}$$

Dans les deux cas, on utilise β_+ si $\gamma^\mu > 0$ et β_- si $\gamma^\mu < 0$.

4.3.2 L'initialisation et l'arrêt

Le centre $\vec{w}(t=0)$ est initialement placé sur le barycentre (ou centre de gravité) des exemples à sortie $\tau = -\pi$, donné par :

$$\vec{w}(0) = \frac{1}{P_-}\sum_{\mu|\tau^\mu=-\pi}\vec{\xi}^\mu \tag{4.44}$$

où P_- est le nombre d'exemples de l'ensemble $\mathcal{L}^\alpha$ tels que $\tau^\mu = -\pi$.

Le rayon initial $\rho(t=0)$ est :

$$\rho(0) = \min_{\{\nu|\tau^\nu=+\pi\}} \left\{ \sqrt{\|\vec{w} - \vec{\xi}^\nu\|^2} \right\} \tag{4.45}$$

qui est égal à la distance euclidienne du barycentre $\vec{w}(0)$ à l'exemple ν le plus proche, qui appartient à la classe opposée ($\tau = \pi$).

Le critère d'arrêt de la minimisation de (4.38) est le même que celui que nous avons mis au point pour les perceptrons linéaires, contrôlé par la fenêtre définie par l'inégalité (4.2.2).

4.3.3 La normalisation des entrées

En ce qui concerne la normalisation des entrées pour les perceptrons sphériques, nous utilisons la transformation linéaire suivante :

$$\widetilde{\xi}_i^\mu \leftarrow \frac{\xi_i^\mu - \langle \xi_i \rangle}{\langle \triangle \rangle} \tag{4.46}$$

où $\langle \xi_i \rangle$ est définie par (4.26), mais où $\langle \triangle \rangle$ est :

$$\begin{aligned} \triangle_i &= \sqrt{\frac{1}{P}\sum_{\mu=1}^{P} (\xi_i^\mu - \langle \xi_i \rangle)^2} \\ \langle \triangle \rangle &= \frac{1}{N}\sum_{i=1}^{N} \triangle_i \end{aligned} \tag{4.47}$$

La recherche des poids se fait dans l'espace d'entrées standardisées $\widetilde{\vec{\xi}^\mu}$, où on va noter les poids $\vec{j}$, et la stabilité $\widetilde{\lambda}^\mu$; on écrit :

$$\begin{aligned} \widetilde{\lambda}^\mu &= \pi\tau^\mu \left\{ (\widetilde{\vec{\xi}^\mu} - \vec{j})^2 - j_0^2 \right\} \\ &= \pi\tau^\mu \left\{ \sum_{i=1}^{N} \left[\frac{\xi_i^\mu}{\langle \triangle \rangle} - \left(\frac{\langle \xi_i \rangle}{\langle \triangle \rangle} + j_i \right) \right]^2 - j_0^2 \right\} \\ &= \pi\tau^\mu \left\{ \sum_{i=1}^{N} \frac{[(\xi_i^\mu)^2 - (\langle \xi_i \rangle + \langle \triangle \rangle j_i)]^2 - \langle \triangle \rangle^2 j_0^2}{\langle \triangle \rangle^2} \right\} \end{aligned} \tag{4.48}$$

et finalement :

$$\widetilde{\lambda}^\mu = \frac{\pi\tau^\mu}{\langle \triangle \rangle^2} \left\{ \sum_{i=1}^{N} \left[(\xi_i^\mu)^2 - (\langle \xi_i \rangle + \langle \triangle \rangle j_i) \right]^2 - (j_0 \langle \triangle \rangle)^2 \right\} \tag{4.49}$$

Cette procédure permet de revenir à l'espace des entrées original de manière simple, en utilisant les poids rénormalisés suivants :

$$\begin{aligned} \rho^2 &= (\langle\triangle\rangle j_0)^2 \\ w_i &= \langle\triangle\rangle j_i + \langle\xi_i\rangle \end{aligned} \tag{4.50}$$

Puisque $\langle\triangle\rangle > 0$ par définition, (voir 4.47), on a :

$$\text{signe}\left[\left(\vec{w} - \frac{\vec{\xi} - \langle\vec{\xi}\rangle}{\langle\triangle\rangle}\right)^2 - \rho^2\right] = \text{signe}\left\{\left[\left(\vec{w}\langle\triangle\rangle + \langle\vec{\xi}\rangle\right) - \vec{\xi}\,\right]^2 - \langle\triangle\rangle^2\rho^2\right\} \tag{4.51}$$

Donc, la sortie (4.35) donnée par les poids $(\vec{w}, \rho)$ dans l'espace des entrées original est la même que la sortie donnée par les poids renormalisés (4.50).

4.3.4 Exemples de séparations sphériques

Les perceptrons sphériques peuvent-ils résoudre des problèmes linéairement séparables ? La réponse est affirmative dans la mesure où l'on permet que le centre $\vec{w}$ puisse s'éloigner à l'infini en même temps que le rayon diverge. Cover montre que séparabilité linéaire implique séparabilité sphérique, qui à son tour implique séparabilité quadratique, mais le cas contraire n'est pas toujours vrai [19].

Nous allons présenter quelques exemples en deux dimensions, afin de rendre plus intuitives les explications.

- Dans la figure 4.14, nous montrons un ensemble d'apprentissage $\mathcal{L}^\alpha$ à $N = 2$ entrées et $P = 200$ exemples, qui est linéairement séparable, donc soluble par un perceptron linéaire. Minimerror-L trouve une solution qui est représentée par la ligne droite. Pour le même ensemble, Minimerror-S trouve comme solution un cercle centré sur $\vec{w} = \{10.58, 6.85\}$ avec un rayon $\rho = 12.62$.

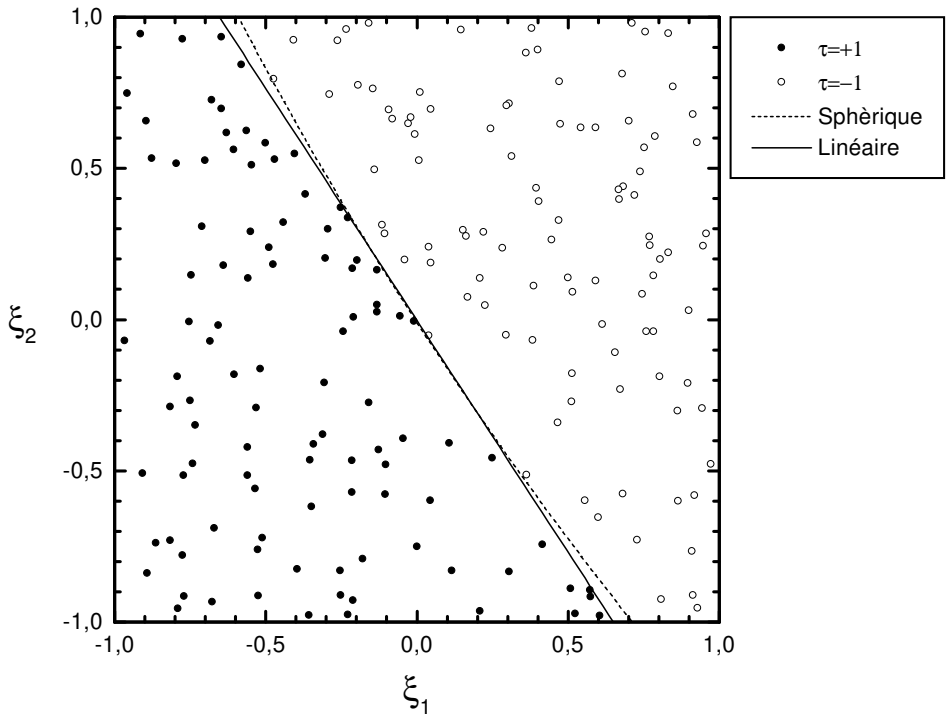

Figure 4.14: Exemple linéairement/sphériquement séparable résolu par un perceptron linéaire et un autre sphérique.

- Dans la figure 4.15 on montre un problème trivial avec $P = 10$ exemples disposés de façon non linéairement mais sphériquement séparable. Nous montrons le centre initial de la minimisation, ou centre de gravité des exemples, $\vec{w}(0) = \{0.355, -0.137\}$ et le rayon initial $\rho(0) = 0.499$ trouvés avec les équations (4.44) et (4.45). La solution trouvée avec Minimerror-S de centre $\vec{w} = \{0.91, -0.07\}$ et rayon $\rho = 0.86$, est aussi représentée.

- Dans la figure 4.16 on montre un problème plus difficile, de $P = 200$ exemples en dimension $N = 2$. La classe $\tau = -1$ à été donnée par deux professeurs de polarité $\pi = +1$ avec centres : $\vec{w}_1 = \{-0.30, 0.00\}$ et $\vec{w}_2 = \{+0.30, 0.00\}$ respectivement, et de rayons $\rho_1 = \rho_2 = 0.5$. Dans l'ensemble d'apprentissage il y a $P_1 = 38$ exemples de classe -1 donnés par le premier professeur et $P_2 = 29$ exemples de classe -1 donnés par le deuxième professeur. Parmi eux, beaucoup d'exemples avec $\tau = -1$ forment une région à frontière complexe près du centre de l'espace des entrées. On montre la solution finale qui atteint le nombre minimum de fautes avec un seul perceptron sphérique. Ce problème sera repris au Chapitre 5, pour illustrer la solution complète produite par notre algorithme constructif.

- Finalement, dans la figure 4.17 on montre un problème avec $P = 200$ exemples disposés de façon non séparable par une seule sphère. En effet, les exemples ont été choisis aléatoirement dans deux régions non connexes, de centres $\vec{w}_g = \{-0.5, 0.0\}, \vec{w}_d = \{0.5, 0.0\}$ et de rayons $\rho_g = \rho_d = 0.25$. La région gauche a $P_g = 7$ exemples à sortie $\tau = -1$, l'autre en a $P_d = 11$. L'initialisation sur le centre de gravité est $\vec{w}(0) = \{0.02, 0.03\}$, $\rho(0) = 0.41$. A partir d'elle, le centre évolue jusqu'à se situer sur $\vec{w} = \{0.49, 0.00\}$, avec un rayon final de $\rho = 0.21$.

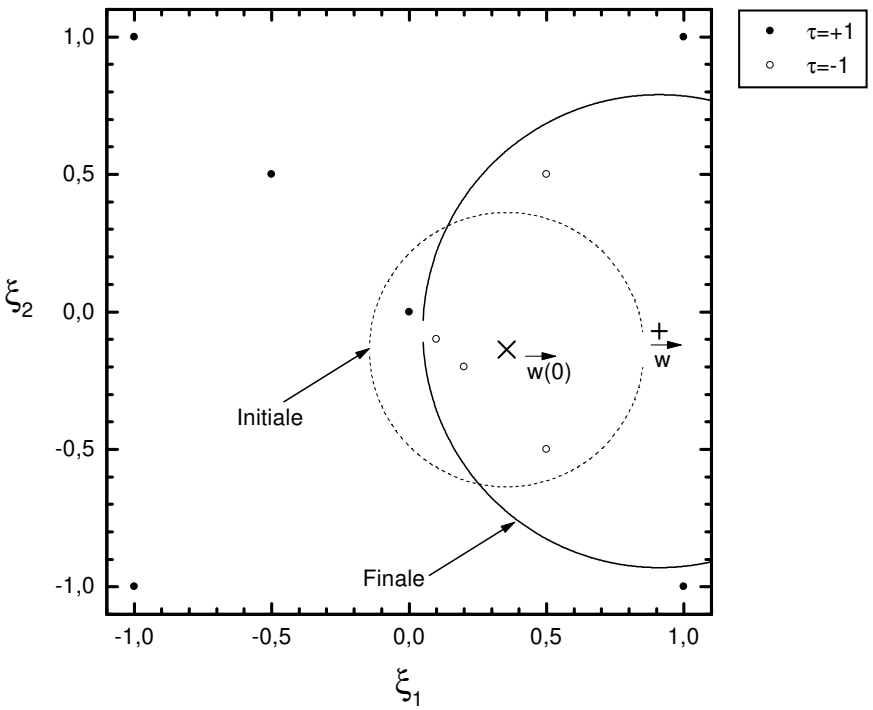

Figure 4.15: Exemple sphériquement séparable trivial, résolu par un perceptron sphérique. On montre l'initialisation des poids et la solution finale trouvée par Minimerror-S.

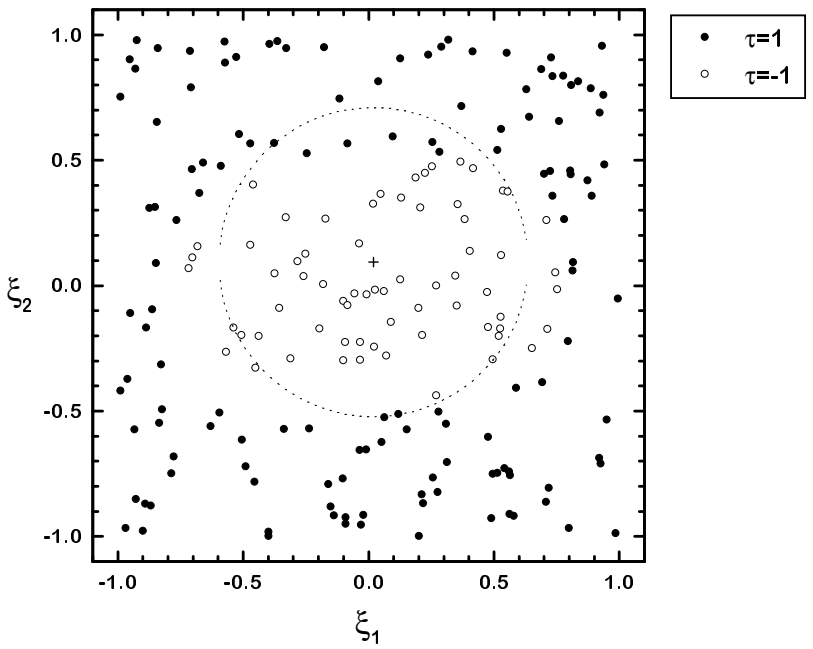

Figure 4.16: Exemple non séparable. On montre la solution trouvée par Minimerror-S $\vec{w} = \{0.019, 0.094\}, \rho = 0.616$. qui minimise le nombre d'erreurs.

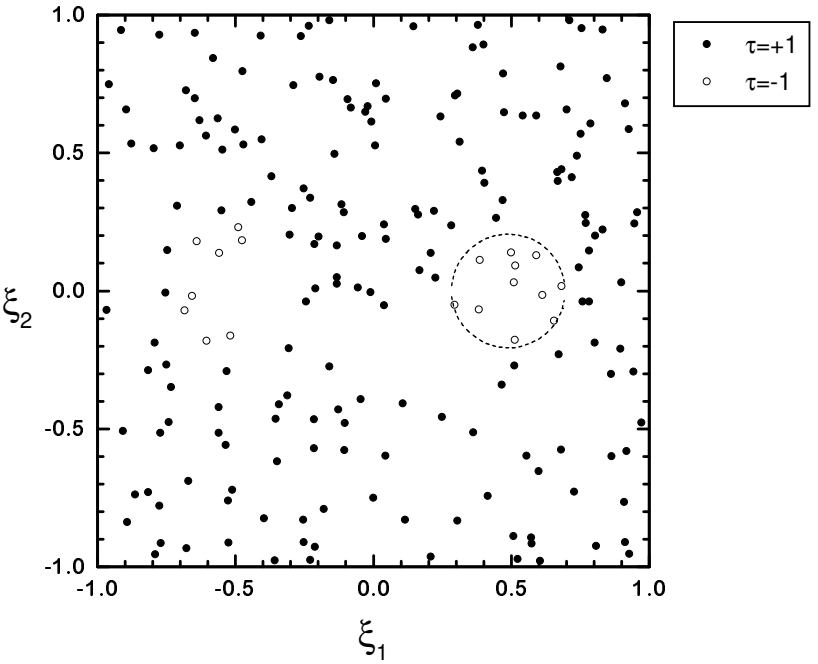

Figure 4.17: Exemple non séparable par une seule sphère. Le perceptron sphérique entraîné par Minimerror-S $\vec{w} = \{0.489, 0.004\}, \rho = 0.206$, sépare la région (droite) avec un nombre plus important d'exemples à sortie $\tau = -1$.

4.4 Conclusion

L'algorithme d'apprentissage Minimerror-L, pour le perceptron simple, a été décrit. Minimerror trouve un ensemble de poids $\vec{w}$ qui minimise une fonction de coût, par une descente en gradient combinée avec un recuit déterministe. Si une solution existe, l'algorithme minimise l'erreur de généralisation ; sinon, il trouve les poids qui minimisent le nombre d'erreurs commises sur l'ensemble d'apprentissage.

A partir de l'implémentation existante de Minimerror, nous avons trouvé un critère d'arrêt satisfaisant (basé sur les propriétés du recuit déterministe), qui permet d'éviter des itérations souvent inutiles. Des tests sur des ensembles d'apprentissage à entrées binaires nous ont permis de régler les trois paramètres libres de Minimerror-L (le rapport de températures, la température initiale et le choix de croissance de cette température). Ainsi, une standardisation adéquate des entrées à été trouvée, ce qui permet d'obtenir des résultats satisfaisants sans besoin de régler les paramètres pour chaque nouveau problème, ainsi que de revenir facilement à l'espace des entrées original.

Nous avons aussi proposé deux définitions alternatives d'une nouvelle stabilité λ, basée sur des propriétés d'un champ radial (proportionnel à la distance des exemples à la surface d'une hypersphère). Cette stabilité nous a permis de généraliser à des perceptrons sphériques l'algorithme d'apprentissage Minimerror. Plusieurs tests sur des exemples à deux dimensions ont confirmé les performances de l'algorithme, que nous avons appelé Minimerror-S.

D'autres tests sur le nombre d'époques nécessaires à la convergence, sur le choix de la méthode de minimisation et sur la robustesse ont aussi été décrits.

Chapitre 5

Trois algorithmes constructifs

5.1 Introduction

Dans ce chapitre, nous présentons trois heuristiques de croissance que nous avons développées pour engendrer des classifieurs binaires. Les algorithmes Monoplan et NetLines font des séparations par des hyperplans en utilisant Minimerror-L pour l'entraînement individuel de chaque perceptron et l'algorithme NetSphères utilise Minimerror-S pour faire des séparations hypersphériques. Les trois algorithmes permettent d'engendrer des réseaux à unités cachées binaires, adaptés à des tâches de classification, et qui présentent des bonnes performances en généralisation.

Nous avons implémenté une procédure qui élimine les représentations internes répétées, non nécessaires pour l'apprentissage du perceptron de sortie, valable dans les trois méthodes.

Nous proposons également une stratégie utile pour traiter des problèmes à plusieurs classes qui engendre des arbres de réseaux. Bien que développée en rapport avec nos algorithmes constructifs, cette stratégie peut aussi bien utiliser d'autres classifieurs binaires.

En général, les problèmes de classification peuvent se diviser en ceux à entrées binaires et ceux à entrées réelles. Dans les problèmes à entrées binaires il est toujours possible d'isoler des exemples par des hyperplans, car ils se trouvent sur les sommets d'un hypercube à dimension N. Une solution du type *grand-mère* [64] peut rendre les représentations internes séparables. Dans les problèmes à entrées réelles la situation n'est pas la même, car les exemples peuvent se trouver à l'intérieur de l'hypercube, et la possibilité de les isoler n'est pas évidente. C'est cette remarque qui est à l'origine du développement succesif des trois algorithmes présentés dans ce chapitre. Nous allons les décrire dans l'ordre où ils ont été développés et nous laissons pour le Chapitre 6 la présentation des applications.

5.2 Monoplan

Dans l'algorithme Monoplan [95], chaque unité cachée rajoutée sert à corriger les erreurs d'apprentissage commises par l'unité précédente. La construction du réseau se fait en deux phases :

- Couche cachée.
  D'abord un perceptron simple apprend l'ensemble d'apprentissage $\mathcal{L}^\alpha$ avec l'algorithme Minimerror-L. Si le nombre d'erreurs d'apprentissage est nul, $\varepsilon_t = 0$, alors $\mathcal{L}^\alpha$ est linéairement séparable et l'algorithme s'arrête. Le réseau ainsi engendré est un perceptron simple.

  Si $\varepsilon_t > 0$, ce perceptron, avec ses poids appris, devient la première unité cachée, $h = 1$. On ajoute une unité cachée $h+1$, et on modifie les cibles à apprendre. Les classes des exemples sont remplacées par les cibles suivantes : $\tau_{h+1} = +1$ pour tous les exemples bien classés par l'unité précédente, et $\tau_{h+1} = -1$ pour ceux mal appris. On peut résumer ceci par : $\tau^\mu_{h+1} = \sigma^\mu_h \ \tau^\mu_h$. Il a été démontré [64, 44] que chaque unité est capable d'apprendre au moins un exemple de plus que l'unité précédente, ce qui assure la convergence de l'algorithme.

  Une fois l'apprentissage d'une unité terminé, ses poids ne sont plus modifiés. La couche cachée se développe ainsi, jusqu'à ce qu'une unité ait appris toutes les sorties correctement.

  Il a aussi été démontré que la parité de la représentation interne associée à chaque exemple de l'ensemble d'apprentissage est la classe de l'exemple. Cette première phase de construction de Monoplan est identique à celle de la première couche de l'algorithme *Offset* [64]. La différence réside dans le fait que ce dernier algorithme implante la parité avec une seconde couche cachée, qui doit être élaguée, alors que Monoplan apprend la sortie avec une seule unité, en rajoutant des unités cachées si nécessaire, comme suit :

- Perceptron de sortie.
  Dans la deuxième phase de l'algorithme, l'unité de sortie est connectée aux unités de la couche cachée. Cette unité apprend les sorties désirées τ^μ. Si les représentations internes sont linéairement séparables, l'unité de sortie pourra les apprendre et l'algorithme s'arrête. Si l'unité de sortie ζ ne trouve pas de solution sans erreur, ce qui se produit si les représentations internes ne sont pas linéairement séparables, on retourne à la première phase d'ajout d'unités cachées, mais cette fois les cibles à apprendre par l'unité cachée $h+1$ sont : $\tau^\mu_{h+1} = \tau^\mu \ \zeta^\mu$. Il est possible que plusieurs exemples soient associés à la même représentation interne. C'est-à-dire que les représentations internes sont dégénérées. Dans 5.5 nous discutons ce point plus en détail, et nous décrirons

comment nous avons simplifié l'apprentissage de la sortie en tenant compte de cette dégénérescence.

Ce va-et-vient entre ces deux phases converge, comme il a été démontré en [99] (voir annexe B).

Pour résumer, Monoplan commence par engendrer une machine à parité : les sorties voulues sont la parité des représentations internes, comme il est montré en [64] et en [9]. Cependant, contrairement à l'algorithme *Offset*, qui utilise une deuxième couche cachée pour calculer la parité, si le neurone de sortie détecte que les représentations internes ne sont pas linéairement séparables, Monoplan augmente la dimension de la couche cachée jusqu'à ce que les représentations internes deviennent linéairement séparables.

Pour illustrer le fonctionnement de Monoplan, considérons le problème à deux dimensions de la 2-Parité (figure 5.1) : (a) la première unité cachée trouve un hyperplan $\vec{w}_1$ qui permet de bien classer les deux exemples à sortie -1 et l'exemple à sortie $+1$ du côté positif de $\vec{w}_1$. L'exemple positif restant est finalement isolé par un deuxième hyperplan $\vec{w}_2$. Les représentations internes correspondantes sont montrées dans la figure 5.2 (Il faut noter qu'il y a seulement 3 représentations internes différentes).

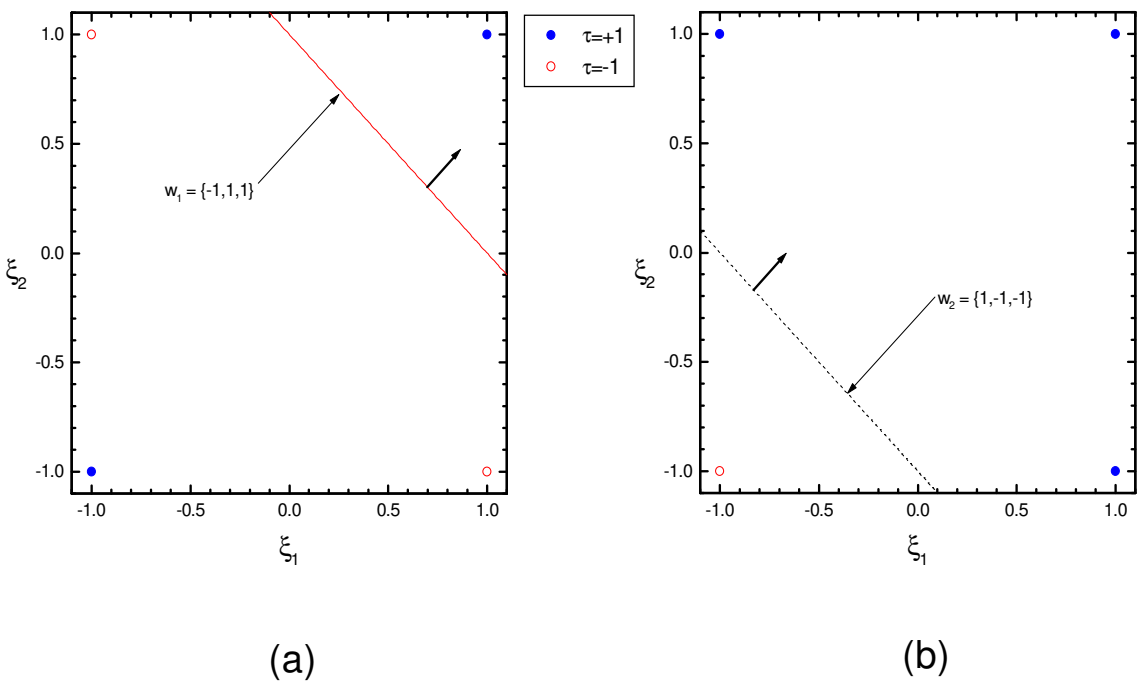

Figure 5.1: La parité à 2 entrées. (a) unité cachée 1. (b) Unité cachée 2.

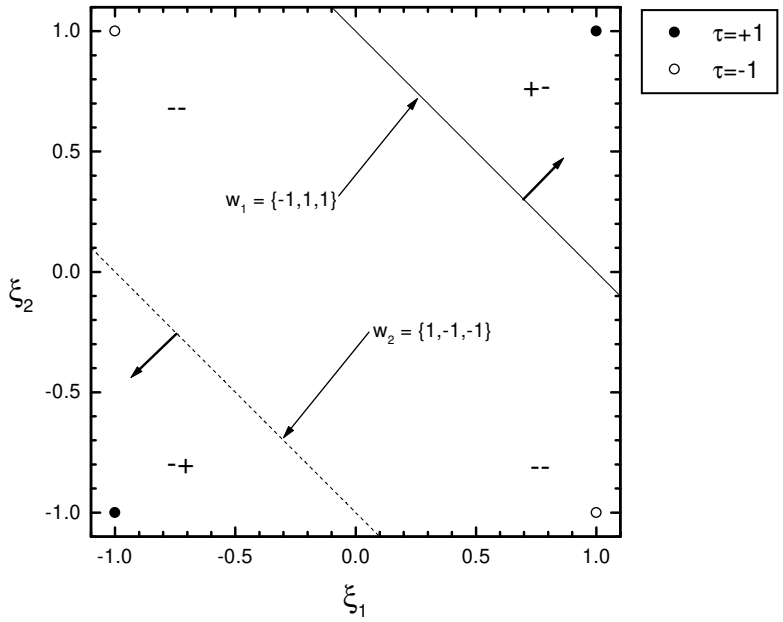

Figure 5.2: La parité à 2 entrées. Représentations internes.

5.2.1 Initialisation des unités

- Initialisation de la première unité.
  Compte tenu de la théorie et des résultats expérimentaux présentés au Chapitre 4, la première unité est initialisée avec la règle de Hebb.

  Dans certains problèmes d'apprentissage de fonctions à entrées binaires, comme la $N-$parité (voir Chapitre 6), ce mode d'initialisation des poids ne fonctionne pas, car en raison de la disposition symétrique des exemples de classes opposées dans l'espace des entrées, on obtient des valeurs $\vec{w} \equiv \{0, 0, \cdots, 0\}$. Dans cette situation il faut initialiser les poids avec une valeur aléatoire.

- Initialisation des autres unités.
  Il existe plusieurs façons d'initialiser les autres unités cachées. Nous avons étudié les suivantes :

  - 1. Aléatoire. Puisque les nouvelles unités ajoutées doivent corriger les erreurs de l'unité précédente, cette initialisation ne semble pas être très astucieuse, car elle place l'hyperplan avec une orientation arbitraire.
  - 2. Poids de l'unité précédente. Les poids de l'unité h peuvent être initialisés en utilisant les poids trouvés pour l'unité $h-1$: $\vec{w}_h \leftarrow \vec{w}_{h-1}$. Cependant, des tests sur la $N-$Parité n'ont pas donné de résultats satisfaisants.
  - 3. Hebb. Rien n'empêche d'utiliser cette initialisation, déjà utilisée pour

la première unité, pour les unités suivantes. En effet, nous avons vérifié qu'elle donne de bons résultats dans la plupart des cas étudiés.

– 4. Initialisation *proximale*. Les poids sont initialisés pour assurer l'apprentissage d'au moins un exemple ν (voir le théorème de convergence dans l'Annexe B) qui a été mal classé par l'unité précédente :

$$\vec{w}_{h+1} = \tau_h^\nu \vec{w}_h - (1-\epsilon_h)\tau_h^\nu(\vec{w}_h \cdot \vec{\xi}^\nu)\hat{e}_0 \tag{5.1}$$

où $\hat{e}_0 \equiv \{1, 0, \cdots, 0\}$ et ϵ_h est une petite tolérance : si $\epsilon_h = 0$, alors l'hyperplan sera situé exactement *sur* l'exemple ν. Nous utilisons cette initialisation si le nombre d'erreurs obtenues en initialisant avec la règle de Hebb n'est pas inférieur à celui de l'unité précédente.

Malgré son élégante simplicité, l'algorithme Monoplan peut avoir des problèmes quand les entrées sont réelles. L'exemple de la figure 5.3 permet de comprendre où se trouve la difficulté. En effet, Monoplan ne peut pas trouver la solution *simple* (a), avec trois unités cachées, car dans cet exemple chaque unité *désapprend* des exemples bien appris par l'unité précédente. La solution trouvée par Monoplan, si l'on utilise l'initialisation proximale, sera comme celle représentée en (b), qui n'est pas optimale, ni du point du vue de l'apprentissage car elle nécessite un grand nombre de neurones, ni du point du vue de la généralisation.

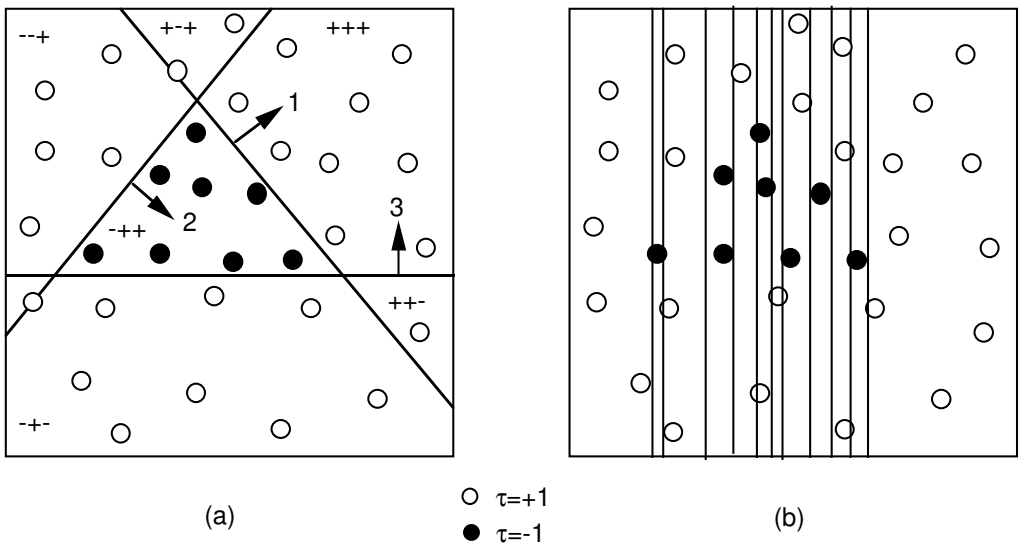

Figure 5.3: Exemple 1: (a) Solution *simple* (Les représentations internes de chaque domaine sont indiquées) et (b) Solution compatible avec Monoplan qui essaie d'isoler les exemples mal classés.

Cette difficulté de l'algorithme face aux exemples à entrées réelles est due au fait que chaque unité cachée doit apprendre des cibles qui sont fonctions des erreurs de l'unité précédente, et non pas des erreurs à la sortie. C'est pourquoi nous avons modifié l'heuristique de Monoplan afin de corriger les erreurs du point de vue de la sortie, en espérant ainsi améliorer la capacité de généralisation.

5.3 NetLines

Comme nous l'avons vu, il n'est pas nécessaire de corriger toutes les erreurs commises par les unités cachées, car il est suffisant que les représentations internes soient fidèles et linéairement séparables, pour que l'unité de sortie puisse apprendre les classes des exemples. Si l'unité de sortie est entraînée immédiatement après avoir ajouté chaque unité cachée, le réseau peut découvrir que les représentations internes sont fidèles et séparables et s'arrêter tout de suite.

Ceci nous a suggéré de suivre une stratégie légèrement différente pour la croissance de la couche cachée. Nous avons appelé cette heuristique *NetLines* : *Neural Encoder Trough pattern LINEar Separations*, ou codeur neuronal d'exemples par des séparations linéaires.

5.3.1 Un exemple

Nous allons décrire la stratégie générale de cet algorithme par un exemple en deux dimensions (voir figure 5.4). Les exemples dans la région grisée appartiennent à la classe $\tau = +1$, les autres à la classe $\tau = -1$. L'algorithme procède comme suit : une première unité cachée est entraînée pour séparer le mieux possible les exemples, et trouve une solution, disons $\vec{w}_1$, représentée dans la figure 5.4 par la ligne étiquetée 1, avec la pointe de la flèche vers le sous-espace positif. Comme il y a encore des erreurs d'apprentissage, une deuxième unité cachée est introduite. Celle-ci va être entraînée pour apprendre $\tau_2 = +1$ pour les exemples qui ont été bien classés par le premier neurone, et $\tau_2 = -1$ pour tous les autres (la convention opposée pourrait être adoptée, car toutes les deux sont strictement équivalentes). Supposons que la solution $\vec{w}_2$ est trouvée. Alors, l'unité de sortie est connectée aux deux unités cachées et est entraînée pour apprendre les sorties originales. Clairement elle ne pourra pas séparer correctement tous les exemples parce que la représentation interne $\vec{\sigma} = (-1, 1)$ n'est pas fidèle : il existe des exemples des deux classes qui ont la même représentation interne. Le neurone de sortie est alors annihilé, et une troisième unité cachée est ajoutée et entraînée pour apprendre les sorties $\tau_3 = +1$ pour tous les exemples qui ont été correctement classés par le neurone de sortie et $\tau_3 = -1$ pour tous les autres.

La solution $\vec{w}_3$ est trouvée, et il est facile de voir que maintenant les représentations internes sont fidèles, *i.e.* les exemples qui appartiennent à des classes différentes ont des représentations internes différentes. Donc, l'algorithme trouve 3 frontières qui définissent 6 régions ou domaines dans l'espace des entrées. Il est facile de vérifier que les représentations internes attribuées à chaque domaine, et qui sont indiquées sur la figure 5.4, sont linéairement séparables. Alors, l'unité de sortie pourra enfin trouver la solution correcte à ce problème. Dans le cas où les représentations internes

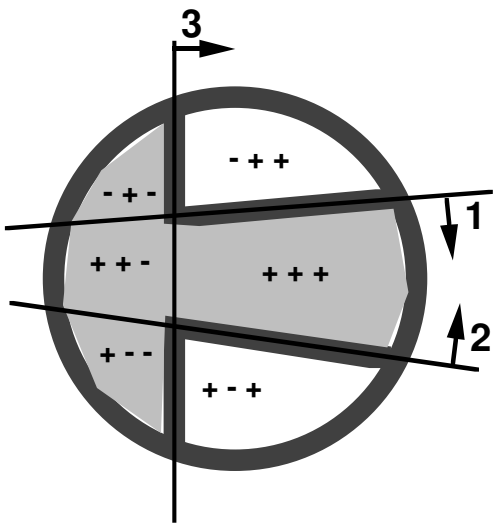

Figure 5.4: Exemple : les exemples dans la région grisée appartiennent à une classe, ceux de la région blanche, à l'autre. Les lignes (étiquetées 1, 2 et 3) représentent les hyperplans trouvés avec la stratégie de NetLines. Les flèches indiquent les sous-espaces positifs correspondants. Les représentations internes de chaque domaine sont aussi indiquées.

fidèles ne sont pas linéairement séparables, le neurone de sortie ne peut pas trouver une solution sans erreurs d'apprentissage, et on doit ajouter des unités cachées pour apprendre les sorties $\tau^\mu = +1$ pour les exemples bien classés et $\tau^\mu = -1$ pour les autres.

Le réseau engendré aura $H = h$ unités cachées. En Annexe B est présentée une solution [44] à la stratégie d'apprentissage qui établit une borne supérieure de $P - 1$ au nombre d'unités cachées, (on rappelle que P est le nombre d'exemples à classer). En pratique l'algorithme finit avec un nombre plus petit que P, comme nous le montrons au Chapitre 6.

5.3.2 Initialisation des unités

De la même manière que pour Monoplan, l'initialisation des unités est importante. Nous avons choisi d'initialiser la première unité avec la règle de Hebb (plus un nombre aléatoire compris entre $[-1, +1]$ si la règle de Hebb trouve des poids nuls) et d'utiliser l'initialisation *proximale* (5.1) pour les suivants : ceci assure l'apprentissage d'au moins un exemple ν mal classé par la sortie.

5.4 NetSphères

Du fait qu'en pratique on a des ensembles d'apprentissage ayant un nombre fini d'exemples, le problème de la figure 5.4 pourrait être résolu par seulement deux unités sphériques qui, en isolant les exemples dans les régions blanches, rendraient les représentations internes séparables par un perceptron à la sortie. C'est pourquoi nous avons pensé à combiner l'heuristique de NetLines avec les unités sphériques présentées au Chapitre 4.

L'algorithme NetSphères construit un réseau de neurones à une couche cachée d'unités sphériques, et une unité de sortie binaire. Cet algorithme suit la même heuristique de croissance que NetLines, mais pour l'entraînement des perceptrons de la couche cachée il utilise l'algorithme Minimerror-S avec la nouvelle stabilité sphérique λ définie par (4.33) (respectivement $\lambda_{\log}$ définie par (4.37)). Ceci permet d'engendrer des perceptrons à séparation hypersphérique qui font des séparations entre classes en utilisant des hypersphères et non pas des hyperplans. Cependant, la sortie reste un perceptron linéaire tout à fait comme avec NetLines ou Monoplan.

Les critères de convergence de NetLines restent valables pour NetSphères, car il est évident que le nombre d'erreurs d'apprentissage diminue si l'on rajoute des unités dédiées à isoler les exemples mal classés dans une mer d'exemples bien classés.

Les cibles à apprendre par les unités cachées sont définies de la même manière qu'avec NetLines, mais il se peut que ceci ne soit pas l'unique choix : le dernier exemple à la fin de ce chapitre montre qu'on pourrait avoir besoin de beaucoup d'unités cachées pour résoudre certaines tâches. C'est pourquoi nous pensons que NetSphères reste un algorithme expérimental, car il faudrait réaliser plus de tests pour le mettre au point. La combinaison d'hyperplans et d'hypersphères qui engendrerait une heuristique hybride peut être une autre voie à explorer.

5.4.1 Initialisation des unités

- Première unité.
  On initialise l'unité $h = 1$ avec un centre $\vec{w}_h$ placé sur le barycentre des exemples, calculé par l'équation (4.44) et un rayon initial ρ_0 défini par l'équation (4.45).

- Initialisation des autres unités.

  – La deuxième unité est placée sur un exemple $\vec{\xi}^{\nu}$ choisi au hasard, parmi ceux qui n'ont pas été bien classés par la première unité, $(\sigma_1^{\nu} \neq \tau^{\nu})$, le rayon ρ_0 est initialisé à un petit nombre réel.

– Les centres des unités suivantes ($h > 2$) sont placés sur un exemple $\vec{\xi}^{\nu}$ choisi au hasard, parmi ceux qui n'ont pas été bien classés par l'unité de sortie ($\zeta^{\nu} \neq \tau^{\nu}$).

5.4.2 Exemples

- Exemple 1. Dans la figure 5.5 nous montrons le problème classique de la parité (équivalent au XOR) à 2 entrées.

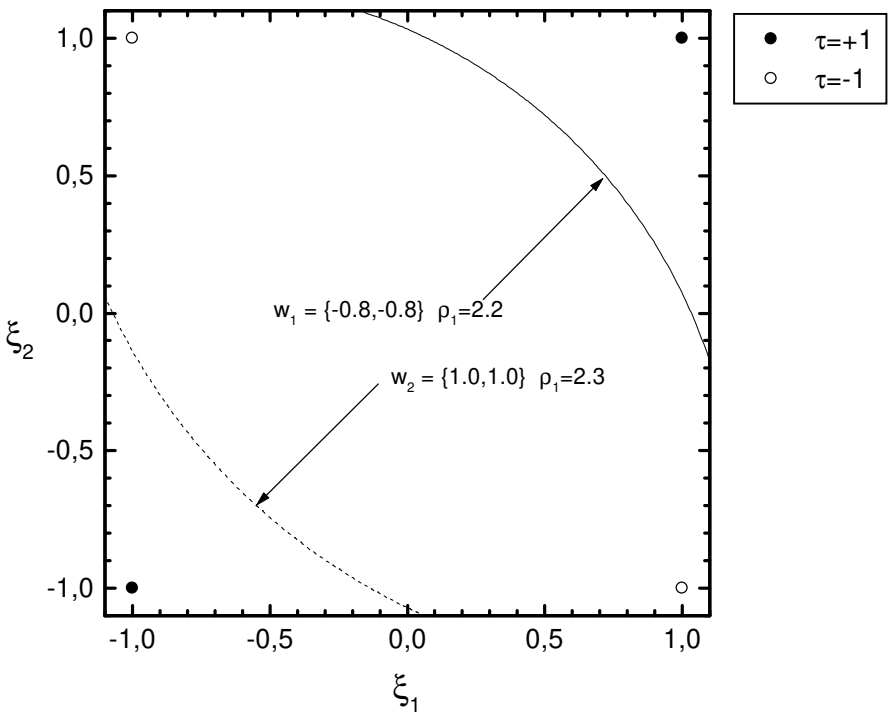

Figure 5.5: La parité à 2 entrées. Un réseau à deux perceptrons sphériques $\vec{w}_1$ et $\vec{w}_2$ permet de trouver la solution. Les représentations internes sont séparables et l'unité linéaire ζ avec des poids $W = \{-1.0, 1.0, 1.0\}$, donne la sortie finale.

- Exemple 2. Dans la figure 5.6 nous montrons un problème avec $P = 200$ exemples disposés de façon non séparable. La région de gauche a $P_1 = 8$ exemples à sortie $\tau = -1$, l'autre en a $P_2 = 9$. La distribution des $P = 200$ exemples a été tirée au hasard, et la classe a été donnée par un réseau professeur avec les paramètres suivants : centre gauche $w_1^* = \{-0.50, 0.00\}$ et droit $w_2^* = \{+0.50, 0.00\}$; rayons $\rho_1 = \rho_2 = 0.50$.

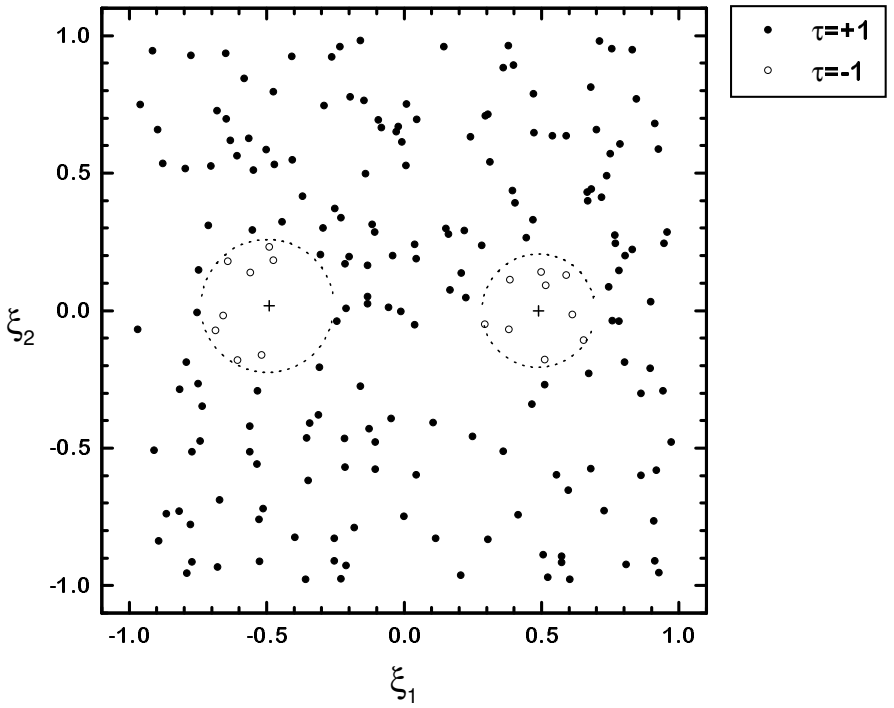

Figure 5.6: Exemple bidimensionnel (montré dans la figure 4.17) engendré par un réseau professeur. Un réseau à perceptrons sphériques trouve la solution commençant par la région de droite (qui a un nombre plus important d'exemples). Une deuxième unité permet d'isoler la région de gauche. Un perceptron linéaire donne la sortie finale.

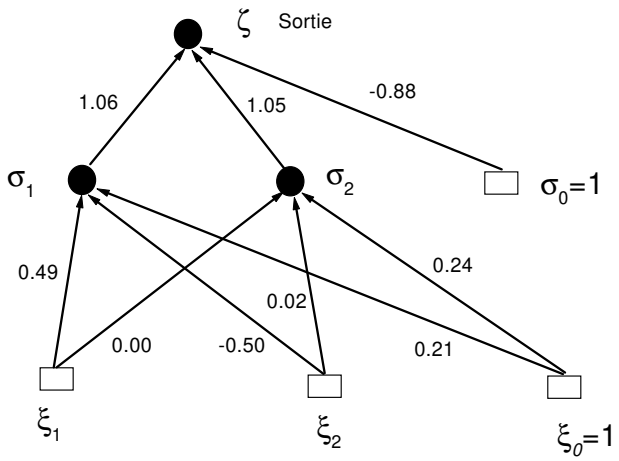

Figure 5.7: Réseau engendré par NetSphères pour le problème de la figure 5.6. On montre les poids $\vec{w}_1 = \{0.49, 0.00\}, \vec{w}_2 = \{-0.50, 0.02\}$ et les rayons (biais) correspondants $\rho_1 = 0.20$, $\rho_2 = 0.24$ pour les unités cachées, et pour la sortie ζ, $\vec{W} = \{-0.88, 1.06, 1.05\}$.

L'initialisation sur le centre de gravité donne $\vec{w}(0) = \{0.02, 0.03\}$. A partir d'ici, le premier centre va évoluer jusqu'à se situer sur $\vec{w}_1 = \{0.49, 0.00\}$, avec un rayon final de $\rho_1 = 0.21$. La deuxième unité va se situer sur $\vec{w}_2 = \{-0.50, 0.00\}$, et un rayon de $\rho_2 = 0.24$; donc toutes les deux assez proches du réseau professeur utilisé pour générer les exemples. La sortie finale est donnée par un perceptron simple avec des poids $W = \{-0.88, 1.06, 1.05\}$. Dans la figure 5.7 on a représenté le réseau final.

- Exemple 3. Dans la figure 5.8 nous montrons le problème d'isolation de coins (emprunté à [67]), qui a $P = 9$ exemples disposés de façon non linéairement séparable. L'initialisation sur le centre de gravité donne $\vec{w}(0) = \{0.0, 0.0\}$. A partir d'ici, la position du centre va stagner mais le rayon va grandir jusqu'à ce qu'il soit $\rho(t) = 1.22$. Par comparaison, nous montrons aussi une solution typique avec NetLines : quatre hyperplans H_1, H_2, H_3 et H_4 sont nécessaires pour séparer les classes.

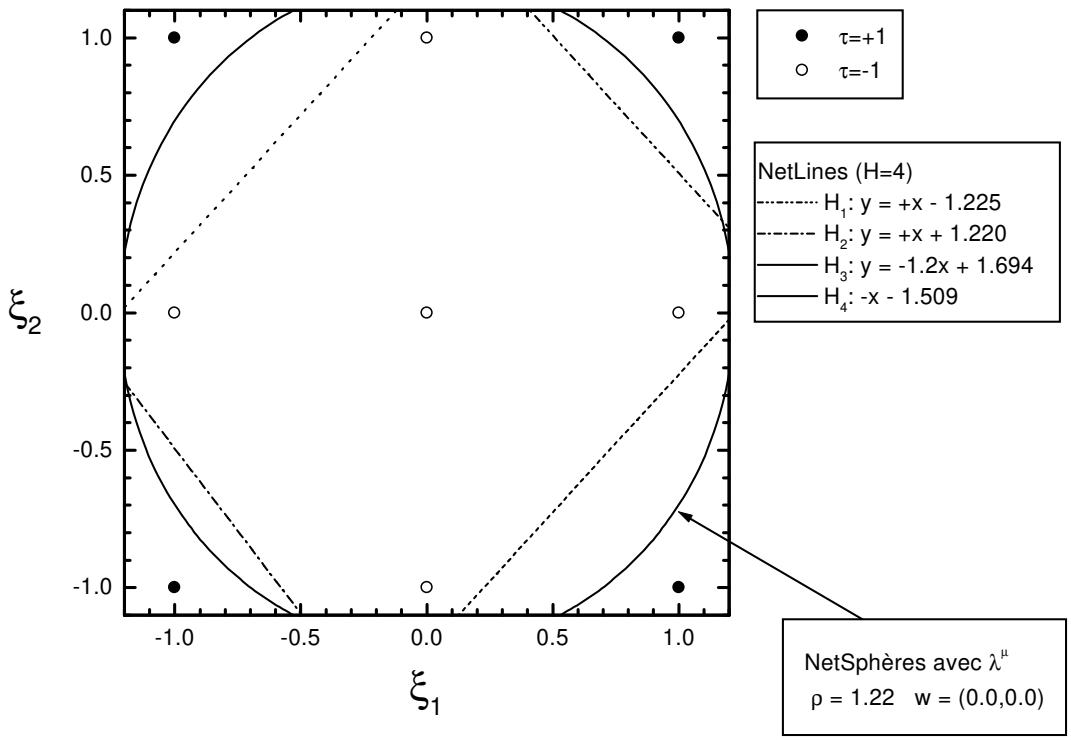

Figure 5.8: Exemple bidimensionnel d'isolation de coins [67]. Tandis qu'une seule unité sphérique est suffisante pour isoler les coins, on a besoin des quatre hyperplans trouvés par NetLines, pour faire la même séparation.

- Exemple 4. Dans la figure 5.9 nous montrons le problème non linéairement séparable de la figure 4.16, Chapitre 4. Un réseau avec $h = 8$ unités cachées a été créé par NetSphères. Bien que la solution puisse apprendre tout l'ensemble d'apprentissage, elle n'est pas optimale : il aurait suffit de deux sphères pour apprendre la structure du professeur. Ce problème n'est pas dû aux initialisations des unités, mais surtout à la manière dont on a choisi les cibles à apprendre par les unités cachées.

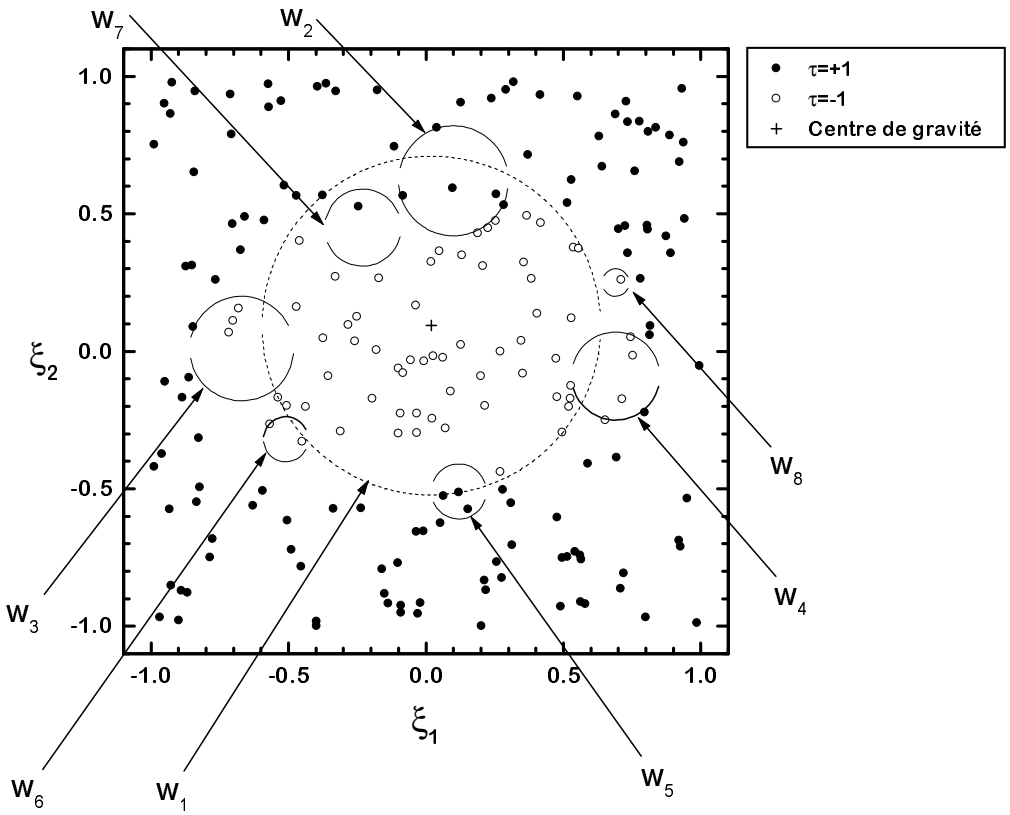

Figure 5.9: Même exemple bidimensionnel que dans la figure 4.16. On a besoin de huit sphères pour rendre les représentations internes séparables.

5.5 Dégénérescence des représentations internes

Dans les trois algorithmes décrits, on associe une représentation interne $\vec{\sigma}^{\mu}$ à chaque exemple $\mu = 1, \cdots, P$. Ainsi, il se peut que plusieurs exemples soient associés à un même état dans la couche cachée. Par exemple, dans le problème du XOR les quatre exemples peuvent être associés à seulement trois états σ différents (figure 5.2) [1]. Ceci est un phénomène désirable qu'on appelle *contraction de l'espace d'entrées*. En effet, pour P exemples appartenant à l'ensemble d'apprentissage $\mathcal{L}^{\alpha}$, on n'aura que $P_{\ell} \leq P$ représentations internes $\vec{\sigma}^{\nu}$; $\nu = 1, \cdots, P_{\ell}$.

[1] En général, pour la N-Parité on aura $N + 1$ représentations différentes.

Du point de vue du perceptron de sortie il est suffisant d'apprendre les P_ℓ représentations internes différentes, en négligeant celles qui sont répétées, soit dégénérées. En pratique, nous avons trouvé que un grand nombre de représentations internes répétées peut compliquer (voire empêcher) le positionnement correcte de l'hyperplan séparateur au niveau du neurone de sortie avec Minimerror.

En effet, si une représentation interne est très dégénérée, elle contribue avec un coefficient c^μ (voir l'éq. (4.8) au Chapitre 4) multiplié par sa dégénérescence (quantité de répétitions). Par exemple, dans le cas extrême où on n'aurait que deux représentations internes différentes : σ^1 et σ^2, avec un seul exemple associé à σ^1 et $P-1$ exemples associés à σ^2, σ^2 est très dégénérée. Si P est très grand, la contribution de σ^1 à l'apprentissage sera très faible. Dans ce cas, Minimerror placera l'hyperplan très près de σ^2, et aura besoin de beaucoup d'itérations. Puisque si deux représentations internes identiques sont fidèles, elles ne peuvent pas donner des sorties différentes. Pour l'apprentissage de la sortie, il est donc suffisant de garder seulement les représentations internes qui sont différentes. C'est pourquoi nous gardons une liste avec les représentations internes et leurs sorties, dont le code binaire (représenté comme un entier en puissances de 2) est différent. Cet ensemble constitue l'ensemble d'apprentissage non dégénéré $\{\vec{\sigma}^\nu \ , \ \tau^\nu\} \ ; \ \nu = 1, \cdots, P_\ell$ que nous utilisons pour l'apprentissage du perceptron de sortie. Cette procédure présente l'avantage d'un apprentissage plus. La solution obtenue est plus robuste au sens de la Sous-section 4.2.4.

5.6 Généralisation à des problèmes de plus de 2 classes

Dans cette section nous présentons une mise en œuvre qui permet de traiter des problèmes ayant un nombre de classes supérieur à deux.

La façon usuelle de résoudre des problèmes ayant plus de deux classes, consiste à générer autant de réseaux que de classes. Chaque réseau est entraîné pour séparer les exemples d'une des classes de tous les autres. Une stratégie de compétition du type "le gagnant prend tout" *Winner-Takes-All (WTA)*, basée sur la valeur de la somme pondérée des sorties en l'équation (2.20), est utilisée pour décider la classe si plus d'un réseau reconnaît l'exemple présenté. Comme nous utilisons des poids qui ont été normalisés, dans notre cas la somme pondérée à la sortie n'est autre chose que la distance à l'hyperplan séparateur de la représentation interne de l'entrée. Tous les exemples avec la même représentation interne donneront donc, la même valeur de la somme pondérée, de façon indépendante de la position relative de l'exemple dans l'espace des entrées. Une forte somme pondérée du *neurone de sortie* n'est pas inconsistante avec de petites sommes pondérées au niveau des neurones cachés.

Donc, une décision *naïve* du type *WTA* pourrait ne pas donner de bons résultats, comme montre l'exemple décrit dans 6.3.4.

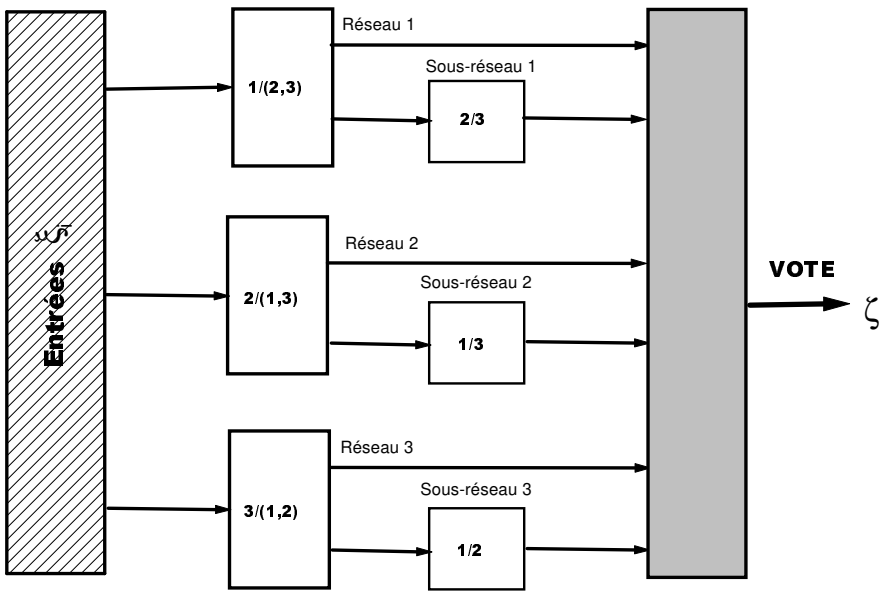

Figure 5.10: Un arbre de réseaux à trois classes : le réseau 1 sépare la classe 1 ($\tau = +1$) des deux autres 2,3 ($\tau = -1$) ; le sous-réseau 1 sépare la classe 2 ($\tau = +1$) de celle 3 ($\tau = -1$). La procédure pour les autres réseaux (et sous-réseaux) est similaire.

Nous proposons une mise en œuvre pour des problèmes à plusieurs classes, qui résulte en une architecture d'arbre de réseaux. Elle est plus complexe que le *WTA*, et peut être appliquée à n'importe quel classifieur binaire. Supposons que nous avons un problème à $C > 2$ classes. On doit d'abord choisir dans quel ordre les classes seront apprises, soit $(c_1, c_2, \cdots, c_C)$. Cet ordre constitue une *séquence particulière d'apprentissage*. Etant donné une séquence particulière d'apprentissage, un premier réseau apprend à séparer les exemples de la classe c_1, auxquels nous attribuons arbitrairement la cible $\tau_1 = +1$, de tous les autres (qui auront des sorties $\tau_1 = -1$). La convention opposée est tout à fait équivalente, et pourrait être utilisée. Après l'apprentissage, tous les exemples de la classe c_1 sont éliminés de l'ensemble d'apprentissage. Un deuxième réseau est entraîné à séparer les exemples de la classe c_2 des autres. Ce procédé, réitéré avec des ensembles d'apprentissage de taille de plus en plus petite, engendre $C-1$ réseaux hiérarchiquement organisés : les sorties sont des séquences ordonnées $\vec{\zeta} = \{\zeta_1, \zeta_2, \cdots, \zeta_{C-1}\}$. La classe attribuée à un exemple est c_i, où i est le premier réseau de la séquence ayant une sortie $+1$ ($\zeta_i = +1$ et $\zeta_j = -1$ pour $j < i$). On néglige alors les sorties des autres réseaux ($j > i$). Dans la figure 5.10 est représenté un arbre de réseaux pour un problème à trois classes.

La performance de ces arbres peut dépendre de la séquence d'apprentissage choisie. C'est pourquoi, il est convenable d'attribuer la sortie par le vote d'un nombre impair de réseaux entraînés avec des séquences d'apprentissage différentes. Nous avons vérifié empiriquement, comme il est montré dans la section 6.3.4, que ce vote améliore les résultats obtenus avec des arbres de réseaux isolés.

5.7 Conclusions

Nous avons présenté une description formelle de trois nouvelles heuristiques de croissance pour des problèmes de classification.

L'algorithme Monoplan a été conçu initialement pour des problèmes à entrées binaires. Chaque unité cachée linéaire sert à corriger les erreurs commises par l'unité précédente. La croissance du réseau est limitée quand l'ensemble d'apprentissage a été bien appris. Cependant, pour des problèmes à entrées réelles, nous avons montré que cette stratégie peut engendrer des réseaux qui ne sont pas optimaux.

C'est pourquoi nous avons développé une stratégie légèrement différente pour la croissance de la couche cachée. L'algorithme NetLines corrige les erreurs d'apprentissage qui sont détectées *à la sortie* du réseau, ce qui nous a permis, en principe de résoudre des problèmes à entrées réelles, plus difficiles avec moins d'unités.

L'algorithme NetSphères suit la même stratégie de croissance de NetLines, mais en utilisant des perceptrons sphériques et non pas des perceptrons linéaires. Ainsi, nous avons montré que certains problèmes à entrées réelles peuvent être résolus plus facilement avec cette stratégie.

Dans les trois cas, l'élimination des représentations internes dégénérées permet de trouver la solution la plus stable à la sortie.

Une généralisation à des problèmes de classification à plus de deux classes en utilisant des réseaux en arbre et un vote a été aussi proposée. Cette méthode a l'avantage de pouvoir être utilisée par n'importe quel algorithme de classification binaire.

Chapitre 6

Tests et applications

6.1 Introduction

Toute nouvelle approche doit être évaluée et testée sur des applications. Les algorithmes d'apprentissage Minimerror-(L/S), et les trois nouvelles méthodes incrémentales présentées, ont été testés avec des problèmes étalon plus ou moins *académiques* : les problèmes de Monks, les trois formes d'ondes de Breiman, et le problème de deux domaines ou plus ; et des problèmes bien connus plus réalistes : la classification des échos de sonar, celle des iris, le diagnostic du cancer du sein à partir de prélèvements et l'identification de cas de diabète.

Dans toutes les applications, nous analysons la taille du réseau engendré, qui est une mesure de sa complexité, ainsi que l'erreur de généralisation. L'erreur de généralisation peut être estimée par plusieurs méthodes empiriques. Nous commençons par un rappel de certaines de ces méthodes avant de décrire les résultats des tests.

6.2 Estimations empiriques de l'erreur de généralisation

En pratique, le nombre de données disponibles est limité. Il faut utiliser des mesures empiriques pour estimer les performances des réseaux. Une question se pose maintenant : comment mesurer réellement l'erreur d'apprentissage et la capacité de généralisation ?

On cherche à mesurer l'erreur de prédiction du réseau, mais ne disposant que d'un ensemble fini de données, il n'est pas possible de calculer exactement la valeur des intégrales (2.22) et (2.25), il faut donc l'estimer. Pour cela il existe des mesures empiriques qu'on verra dans la présente section.

Soit $D = P + G$ le nombre total d'exemples disponibles. On peut diviser ces données en deux sous ensembles : $\mathcal{L}^\alpha$ avec P exemples et un ensemble de G exemples $\mathcal{G} = \{\vec{\xi}^\mu, \tau^\mu\}$; $\mu = 1, \cdots, G$.

On fait l'apprentissage sur l'ensemble $\mathcal{L}^\alpha$ et on mesure la capacité de généralisation sur $\mathcal{G}$

Cette méthode utilise en effet l'information contenue dans deux ensembles d'apprentissage : $\mathcal{L}^\alpha$ et celui de validation. En problèmes de classification nous n'avons pas constaté ce phénomène de sur-apprentissage, alors nous utilisons que l'information disponible dans l'ensemble $\mathcal{L}^\alpha$ et arrêtons l'apprentissage quand $\varepsilon_t = 0$. L'erreur d'apprentissage ε_t est estimée par $\hat{\varepsilon}_t$, la fraction d'exemples qui ont été mal appris ou non appris par le réseau :

$$\hat{\varepsilon}_t = \frac{1}{P} \sum_{\mu=1}^{P} \Theta(-\tau^\mu \zeta^\mu) \tag{6.1}$$

L'erreur de généralisation ε_g peut être estimée par $\hat{\varepsilon}_g$, la fraction d'exemples de $\mathcal{G}$ que le réseau classe mal :

$$\hat{\varepsilon}_g = \frac{1}{G} \sum_{\mu=1}^{G} \Theta(-\tau^\mu \zeta^\mu) \tag{6.2}$$

Pour obtenir des valeurs significatives, il faut effectuer la moyenne des performances mesurés sur un nombre $l = 1, \cdots, L$ statistiquement pertinent d'expériences, chacune étant associée à des ensembles $\mathcal{L}_l^\alpha$ et $\mathcal{G}_l^\alpha$ différents choisis au hasard [48]. Deux méthodes que nous utiliserons par la suite sont :

- *1. Hold-out.* Pour obtenir des résultats non biaisés, il faudrait faire l'apprentissage et le test sur toutes les $\binom{D}{P}$ combinaisons possibles, mais en pratique on ne fait qu'un nombre suffisant V d'essais. L'erreur de généralisation est alors estimée par :

$$\hat{\varepsilon}_g = \frac{1}{V} \sum_{j=1}^{V} \frac{1}{G} \sum_{\mu=1}^{G} \Theta(-\tau^\mu \zeta_j^\mu) \tag{6.3}$$

- *Leave-one-out.* Dans le cas d'un ensemble de données assez réduit, on utilise une partition particulière : on prend D-1 exemples pour apprendre et 1 seul pour le test, et on calcule ensuite les performances moyennes sur toutes les $\binom{D}{1}$ combinaisons possibles, dans ce cas sont réduites à D, et l'erreur de généralisation est estimée par :

$$\hat{\varepsilon}_g = \frac{1}{D}\sum_{\mu=1}^{D}\Theta(-\tau^\mu\zeta^\mu) \tag{6.4}$$

6.3 Problèmes étalon

Les problèmes étalon permettent de comparer un algorithme avec d'autres quand ils sont confrontés à la même base d'apprentissage et dans les mêmes conditions de test, étant donné qu'on connaît la solution. Il est possible ainsi d'étudier les performances d'apprentissage et de généralisation, mais aussi de mesurer la complexité (ressources utilisées pour une tâche, comme le nombre d'unités cachées ou le nombre de poids) du classifieur.

Nous avons évalué les algorithmes présentés au chapitre précédent sur des problèmes étalon [1] à entrées binaires : la N-Parité, les trois problèmes de *Monk's* et le problème de deux domaines ou plus. Parmi les problèmes à entrées réelles, nous avons étudié : la classification des échos sonar, des iris, les trois formes d'ondes de Breiman et les diagnostics de cancer du sein et de diabète.

La N-Parité

L'une des premières tâches que nous avons considérée est l'apprentissage de la N-Parité avec les algorithmes Monoplan et NetLines. Bien qu'on sache que le nombre optimum d'unités cachées qui permet de résoudre ce problème avec un réseau à une couche cachée sans rétroactions est $H = N$, plusieurs tentatives [73] avec la RPG et d'autres approches non constructives n'ont pas réussi à trouver cette solution. Dans la N-Parité, le problème est assez difficile pour la première unité : si l'hyperplan correspondant est bien situé, il permettra de trouver le *nombre minimum de fautes* (erreurs), et étant donné la géométrie symétrique du problème, le reste est moins difficile à résoudre.

Mais, quel est le *nombre minimum de fautes* pour la N-Parité ? pour trouver ce nombre, considerons d'abord la figure 6.1a, pour la 2-Parité, le vecteur $\vec{w}_1$ sépare l'espace d'entrée, où on voit que les exemples $\mu = 2, 3$ et 4 sont bien classés, tandis que l'exemple à sortie négative $\mu = 1$ est mal classé : $\vec{w}_1$ fait donc (et on ne peut pas faire mieux), une erreur. Pour la 3-Parité il faut considérer un espace à trois dimensions. Dans la figure 6.1b, le vecteur $\vec{w}_1$ classe bien les exemples $\mu = 1, 2, 3, 6, 7, 8$ et ceux $\mu = 4, 5$ sont mal classés ; alors, il fait deux fautes.

[1] Toutes les bases de données des problèmes de cette Section ont été prises sur le site UCI à l'adresse Internet http://www.ics.uci.edu/ mlearn/MLRepository.html, sauf pour les problèmes de la N-Parité et celui de 2 domaines ou plus.

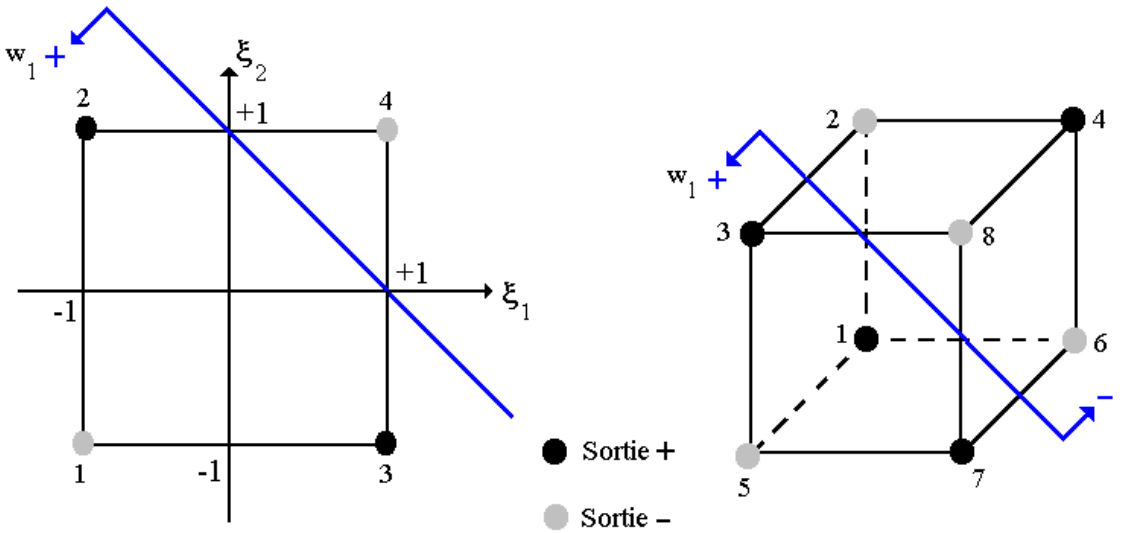

Figure 6.1: La N-Parité : (a) N=2 et (b) N=3 entrées.

ν_k	ν_0	ν_1	ν_2	ν_3	ν_4	ν_5	ν_6	ν_7	ν_8	f^*
N	-	+	-	+	-	+	-	+	-	Erreurs
2	1	2	1							1
3	1	3	3	1						2
4	1	4	6	4	1					5
5	1	5	10	10	5	1				10
6	1	6	15	20	15	6	1			22
7	1	7	21	35	35	21	7	1		44
8	1	8	28	56	70	56	28	8	1	93

Tables 6.1: Nombre minimum de fautes pour la N-Parité

En généralisant ce résultat, et puisque les sorties sont disposées de façon symétrique, on peut construire le tableau 6.1, qui représente le triangle de Pascal, où on a défini la distribution des sommets ν_k par les coefficients du binôme $\nu_k = \binom{N}{k}$; $k = 0, 1, \cdots, N$. Cette distribution alternée des sommets à sortie -1 ou $+1$ est séparée par des hyperplans successifs. Si l'on prend $N = 3$, on a un exemple à sortie -1 (ν_0), trois à sortie $+1$ (ν_1), trois à sortie -1 (ν_2) et un à sortie $+1$ (ν_3). L'hyperplan séparateur qui minimise le nombre d'erreurs *doit* se situer entre ν_1 et ν_2, ce qui donne deux fautes. La dernière colonne représente le nombre minimum de fautes pour la N-Parité commises par un perceptron à N entrées. D'où, une analyse a montré que ce nombre, qu'on désignera f^* est :

$$f^* = \begin{cases} f^*(N=2p) = & \sum_{i=1}^{p} \binom{2p}{2p+i-1} \\ f^*(N=2p+1) = & 2f^*(2p) \end{cases} p = 1, 2, 3, \cdots \tag{6.5}$$

Nous l'avons corroboré expérimentalement en résolvant le problème de la N-Parité pour $2 \leq N \leq 11$, toujours avec $H = N$ unités cachées [94].

Ce problème nous a permis entre autres, de régler les trois paramètres libres de Minimerror-L ($\epsilon, \delta\beta, \Psi = \beta_+/\beta_-$) et de vérifier que le nombre d'erreurs $f(N)$ trouvé par la première unité cachée $h = 1$ de Monoplan (ou NetLines) correspondait effectivement au *nombre minimum de fautes*.

6.3.1 Un cas linéairement séparable : l'identification des échos du sonar

La base des échos sonar [46] a été largement utilisée pour tester plusieurs algorithmes d'apprentissage [5, 4, 12, 20, 54, 82, 84, 91, 104]. Dans ce problème le classifieur doit discriminer si un écho sonar était produit par un cylindre de métal ou par un rocher trouvé dans le même environnement. La base contient 208 spectres de sonar pré-traités définis par $N = 60$ valeurs réelles dans la gamme $[0, 1]$, et leur classe correspondante. Parmi ceux-ci, les premiers $P = 104$ exemples sont habituellement employés comme l'ensemble d'apprentissage qui détermine les paramètres du classifieur. La fraction d'exemples mal classés parmi les restants $G = 104$ spectres, l'ensemble de test, est employé pour estimer l'erreur de généralisation produit par l'algorithme d'apprentissage. On a défini arbitrairement $\tau = +1$ pour les mines et $\tau = -1$ pour les rochers.

En étudiant cette base avec Minimerror-L [95, 10, 98] nous avons trouvé que non seulement l'ensemble d'apprentissage (*i.e.* les premiers $P = 104$ exemples appelés ci-après l'ensemble standard d'apprentissage) et l'ensemble de test (*i.e.* les derniers $G = 104$ exemples) de la base sont tous les deux linéairement séparables, fait déjà rapporté dans [47, 95], mais aussi que l'ensemble complet de $P + G = 208$ exemples est linéairement séparable. L'erreur de généralisation des poids $\vec{w}_P$ qui séparent l'ensemble standard d'apprentissage est $\varepsilon_g \cong 0.22$, correspondant à 23 erreurs de classification sur l'ensemble de test. Une erreur de généralisation plus faible peut être obtenue en arrêtant l'algorithme avant la convergence. Notre meilleure performance de généralisation, $\varepsilon_g \cong 0.15$ (16 erreurs), a été obtenue en arrêtant l'apprentissage avec 8 erreurs (nous désignons $\vec{w}_{Pe}$ les poids correspondants). Cependant, la performance générale (erreurs d'apprentissage et test ensemble) est pire que celle obtenue avec les poids $\vec{w}_P$. En entraînant avec les exemples de test, habituellement utilisés comme ensemble de test, nous déterminons les poids $\vec{w}_G$ qui séparent linéairement

l'ensemble de test. L'erreur correspondante de généralisation estimée en utilisant les P premiers exemples comme ensemble de test est $\varepsilon_g \cong 0.23$ (24 erreurs). Enfin, en entraînant avec l'ensemble complet de $P + G = 208$ exemples, les poids $\vec{w}_{P+G}$ séparent *tous* les exemples, montrant que cette base est linéairement séparable.

Les poids $\vec{w}$ obtenus en entraînant avec différents ensembles sont normaux à l'hyperplan séparateur correspondant. Les projections des exemples $d^\mu \equiv \vec{w}\cdot\vec{\xi}^\mu/\sqrt{N}$, sont proportionnelles à la somme pondérée $|d^\mu|$, qui est la distance de l'exemple μ à l'hyperplan, tandis que signe(d^μ) est la sortie du réseau à l'exemple μ. Nous représentons dans la figure 6.2 les valeurs de d^μ correspondant à $\vec{w}_P$ et à $\vec{w}_{P+G}$, comme une fonction du numéro de l'exemple dans la base. On voit que les poids $\vec{w}_P$ correspondent à une solution robuste : il y a une marge de largeur $\kappa = 0.1226$ libre d'exemples, sur les deux côtés de l'hyperplan. Cet écart est beaucoup plus réduit ($\kappa = 0.00284$) –à peine visible sur la figure– pour la solution séparant l'ensemble complet de données, montrant que c'est un problème beaucoup plus difficile.

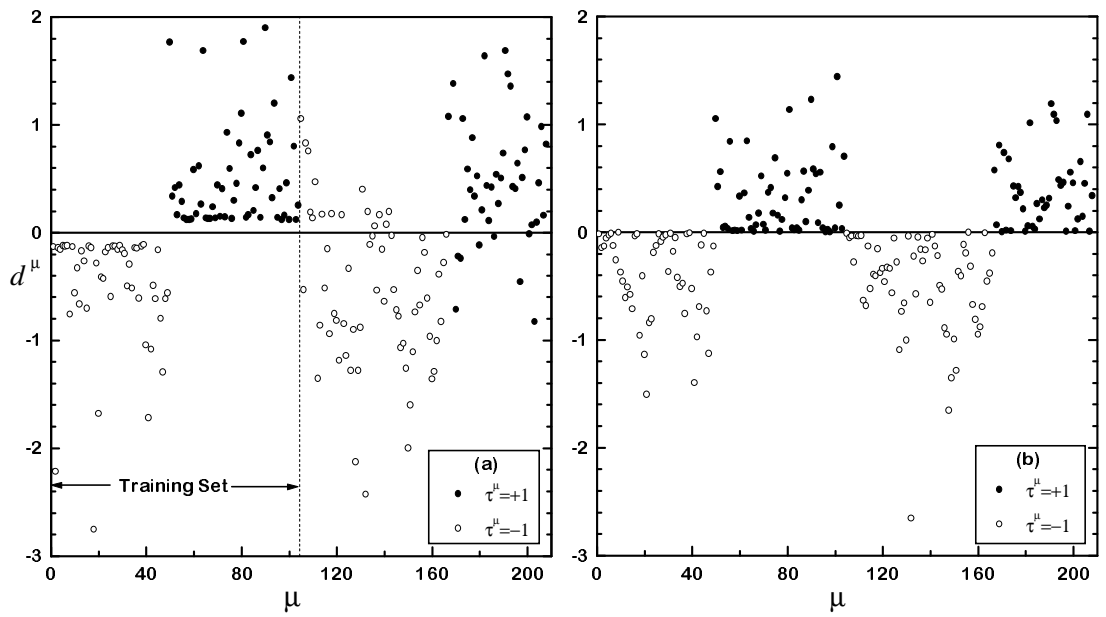

Figure 6.2: Distance des exemples à l'hyperplan séparateur, avec un signe correspondant à la sortie actuelle du perceptron. La classe correcte est τ^μ ($\tau^\mu = +1$ pour les mines, $\tau^\mu = -1$ pour les rochers). (a) Hyperplan déterminé avec l'ensemble d'apprentissage standard (qui contient les premiers $P = 104$ exemples de l'ensemble total), montrant les 23 erreurs sur l'ensemble de test. (b) Hyperplan déterminé avec l'ensemble d'apprentissage total de $P + G = 208$ exemples.

6.3.2 Entrées binaires

Les trois problèmes de Monk

L'intérêt de ces problèmes est qu'ils ont déjà servi de test à plusieurs autres méthodes d'apprentissage, symboliques et/ou numériques [92]. Ces sont des problèmes de logique, qui peuvent être énoncés à partir de six attributs sensés caractériser des robots, comme on le montre dans le tableau 6.2.

variable		valeurs
x_1	Forme de la tête	cercle, carré, octogone
x_2	Forme du corps	cercle, carré, octogone
x_3	Sourie ?	oui, non
x_4	Que porte-t-il ?	épée, ballon, drapeau
x_5	Couleur de la veste	rouge, jaune, vert, bleu
x_6	A-t-il une cravate ?	oui, non

Tables 6.2: Attributs pour les trois problèmes de Monk's.

Chaque problème est une proposition logique, que le réseau doit découvrir, en attribuant une sortie $\tau = +1$ aux configurations d'entrée pour lesquelles la proposition est vraie, ou $\tau = -1$ si elle est fausse. Le réseau de neurones est construit par l'apprentissage de P exemples parmi les $D = 432$ exemples possibles. Les trois propositions, et le nombre d'exemples dont on dispose, sont montrés dans le tableau 6.3.

Problème	Description	Nombre d'exemples P
M_1	$(x_1 = x_2)$ OU (x_5=rouge)	124
M_2	Exactement deux attributs parmi les six prennent leur première valeur	169
M_3	(x_5=vert ET x_4=épée) OU ($x_5 \neq$bleu ET x_2=octogone)	122 (bruités)

Tables 6.3: Les trois propositions logiques des problèmes de Monk's.

Nous avons codé les attributs x_j ; $j = 1, \cdots, 6$ avec $N = 17$ variables binaires ξ_i ; $i = 1, \cdots, N$, de la même façon que les méthodes neuronales (RPG et *Cascade Correlation*), pour travailler dans les mêmes conditions [92].

Une difficulté supplémentaire a été introduite dans M_3 : l'ensemble d'apprentissage contient 5% d'exemples incorrectement classés. Nous présentons dans les

figures 6.3-6.5, les performances en généralisation de NetLines et celle de plusieurs autres méthodes sur les trois problèmes. Cette performance est mesurée par le pourcentage d'exemples de test (les $G = 432 - P$ exemples n'appartenant pas à l'ensemble d'apprentissage) qui ne sont pas bien classés par le réseau.

Dans les problèmes sans bruit (M_1 et M_2), NetLines atteint $\varepsilon_t = 0$ en apprentissage et $\varepsilon_g = 0$ en généralisation. Par contre, lorsqu'il y a du bruit dans la base d'exemples (comme c'est le cas dans M_3), ni NetLines ni les autres méthodes neuronales n'arrivent à extraire la proposition sous-jacente. NetLines extrait une proposition compatible avec tous les exemples, même ceux qui sont erronés, ce qui explique une moins bonne performance en généralisation. Dans le cas de NetSphères, il atteint une performance inférieure avec 5 unités cachées, montrant que la solution sphérique n'est pas optimale dans ce problème.

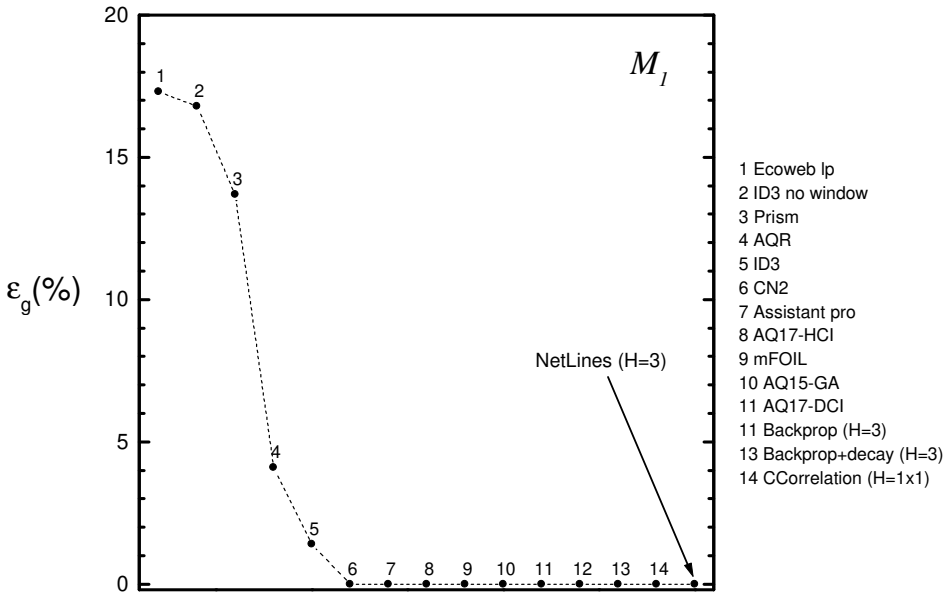

Figure 6.3: Problème M_1

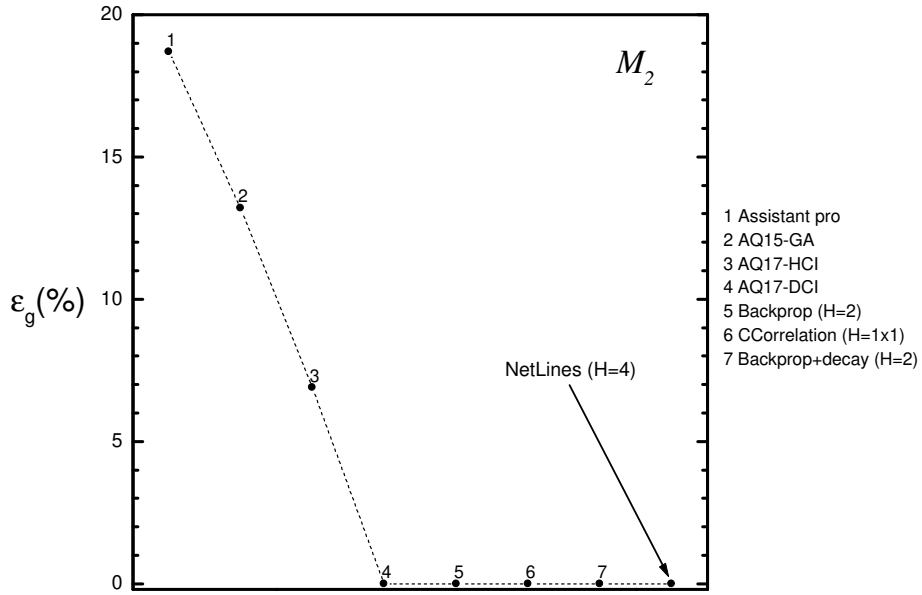

Figure 6.4: Problème M_2

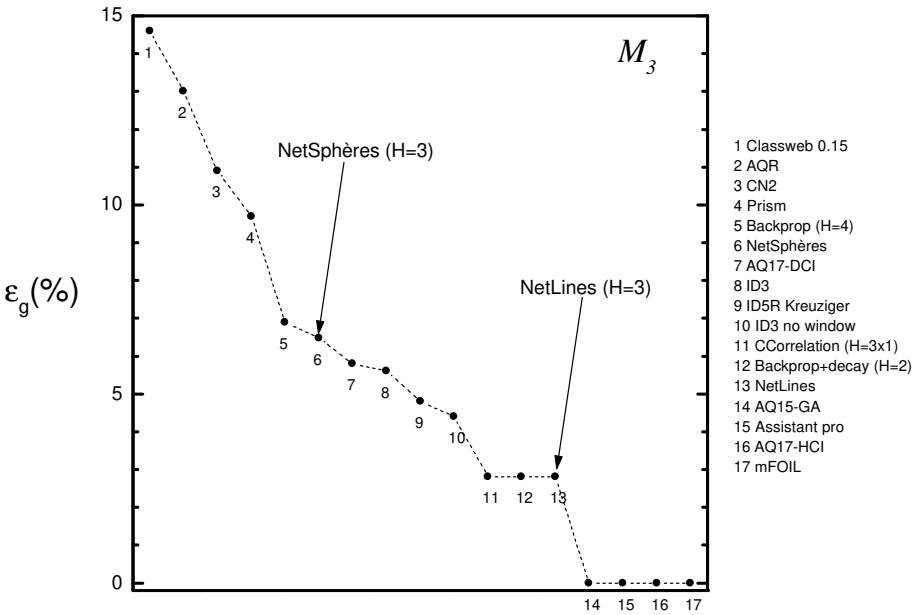

Figure 6.5: Problème M_3

Le problème de 2 domaines ou plus

Dans ce problème [27] le réseau doit discriminer si le nombre de domaines dans un anneau de N bits est strictement plus petit que 2 ou non. Un domaine est une séquence de bits identiques entourée par des bits de l'autre type, sans une limite précise. Les exemples sont engendrés par une méthode de MonteCarlo dans laquelle le nombre moyen de domaines est contrôlé par un paramètre k [68]. Nous avons généré des ensembles d'apprentissage de P exemples avec $k = 3$, correspondant à un nombre moyen de domaines de ≈ 1.5, pour des anneaux de $N = 10$ et $N = 25$ bits.

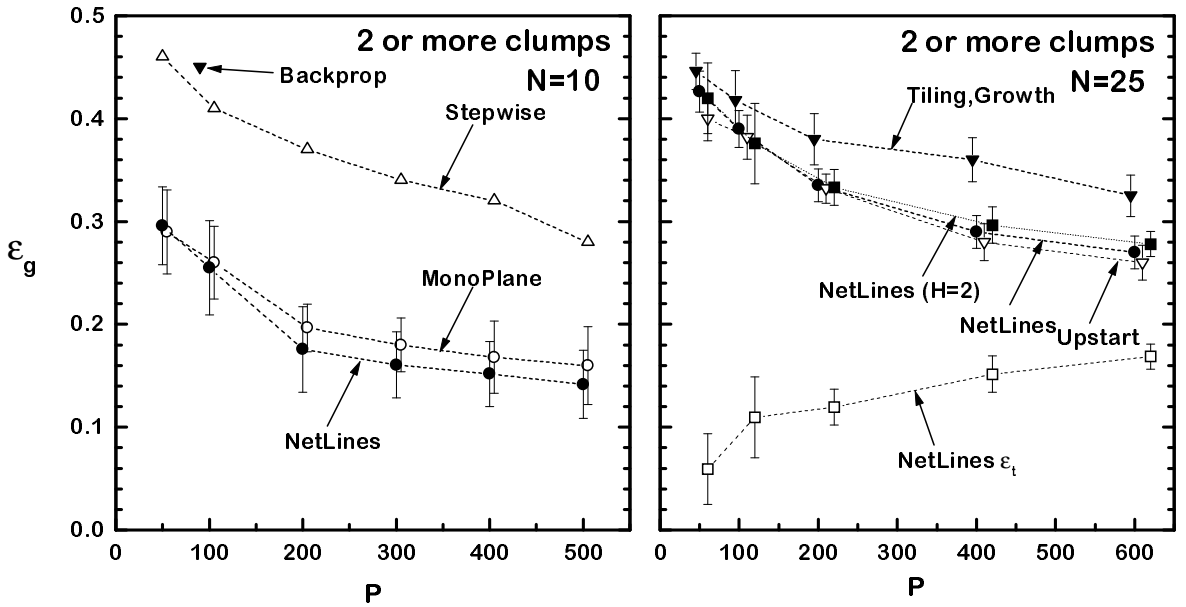

Figure 6.6: Deux domaines ou plus pour des chaînes de deux tailles, $N = 10$ et $N = 25$. Erreur de généralisation ε_g vs. taille de l'ensemble d'apprentissage P, pour différents algorithmes. $N = 10$: RPG [88], Stepwise [57]. $N = 25$: *Tiling* [68], *Upstart* [31]. Les résultats avec l'algorithme *Growth* [72] sont indiscernables de ceux de *Tiling* à l'échelle de la figure. Les résultats de Monoplan et NetLines sont des moyennes sur 25 tests.

L'erreur de généralisation correspondant à plusieurs algorithmes d'apprentissage, calculée sur des ensembles de test indépendants, mais de même taille que les ensembles d'apprentissage, *i.e.* $G = P$, sont montrés dans les figures 6.6 en fonction de P. Les points avec des barres d'erreur correspondent à des moyennes sur 25 essais. Les résultats avec NetLines, Monoplan et *Upstart* pour $N = 25$ ont presque les mêmes performances quand on fait l'apprentissage jusqu'à avoir zéro fautes. Nous avons essayé l'effet d'arrêter avant de tout apprendre (*early stopping*), en imposant un nombre maximal de H unités cachées. L'erreur résiduelle d'apprentissage ε_t est

montrée dans la même figure, comme une fonction de P. On voit que, arrêtant tôt l'apprentissage, on a un léger accroissement de l'erreur de généralisation ε_g, montrant que les unités cachées au-delà de 2 ne produisent pas de sur-apprentissage. Le nombre de poids employé par différents algorithmes en fonction de P est montré en échelle logarithmique, dans les figures 6.7. Il apparaît que la stratégie de NetLines est légèrement meilleure que celle de Monoplan, par rapport à la généralisation et taille du réseau. L'avantage de Monoplan, qui n'a pas besoin d'entraîner l'unité de sortie après chaque unité cachée, est perdue parce qu'il termine avec un plus grand nombre d'unités cachées.

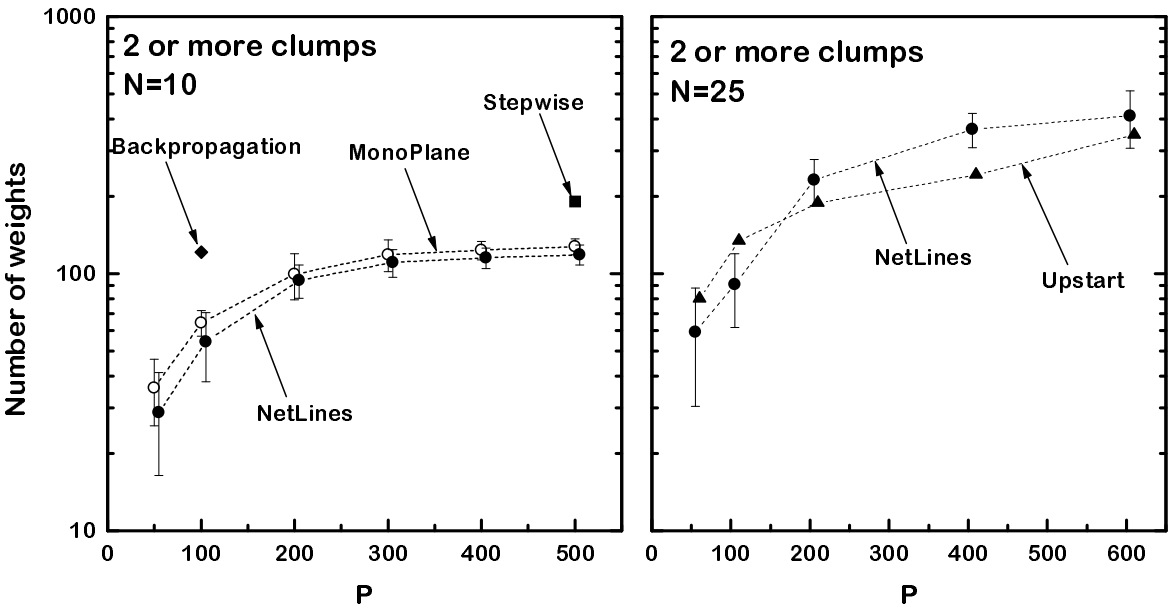

Figure 6.7: Deux domaines ou plus. Nombre de poids (échelle logarithmique) vs. taille de l'ensemble d'apprentissage P, pour $N = 10$ et $N = 25$. Les résultats de Monoplan et NetLines sont des moyennes sur 25 tests. Les références sont les mêmes que dans la figure 6.6.

6.3.3 Entrées réelles

Le diagnostic du cancer du sein (Wisconsin Data Base)

Les exemples de cette base [106] à $N = 9$ attributs servent à caractériser des échantillons cytologiques du sein, classés comme bénins ou malins (voir le tableau 2.1 du Chapitre 2). Nous excluons de l'apprentissage les 16 exemples qui ont l'attribut ξ_6 (*noyaux*) manquant. Parmi les $D = 683$ exemples restants, les deux classes sont représentées irrégulièrement : 65.5% des exemples étant bénins. Nous étudions la performance en généralisation des réseaux entraînés avec des ensembles de plusieurs tailles P. Les P exemples pour chaque test d'apprentissage étant choisis au hasard,

sans qu'il soit nécessaire d'équilibrer le nombre d'exemples de chaque classe. Dans la figure 6.8a, l'erreur de généralisation du classement de $G \equiv D - P$ exemples est montrée comme une fonction du nombre correspondant de poids dans une échelle logarithmique. Pour comparaison, les résultats obtenus avec un perceptron entraîné avec Minimerror-L avec $P = 75$ exemples est inclus dans la même figure. Les résultats, moyennés sur 50 tests indépendantes pour chaque P, montrent que les deux algorithmes NetLines et Monoplan ont de faibles ε_g et nécessitent moins de paramètres que d'autres algorithmes.

Le réseau entraîné peut être employé pour classer les exemples avec des attributs manquants. Le nombre d'exemples parmi les 16 cas avec l'attribut manquant, est montré comme une fonction de ξ_6 dans la figure 6.8b. Pour des grandes valeurs de ξ_6 il y a des décalages entre le diagnostic médical et le diagnostic du réseau sur la moitié des cas. C'est un exemple de la sorte d'information qui peut être obtenue dans des applications pratiques.

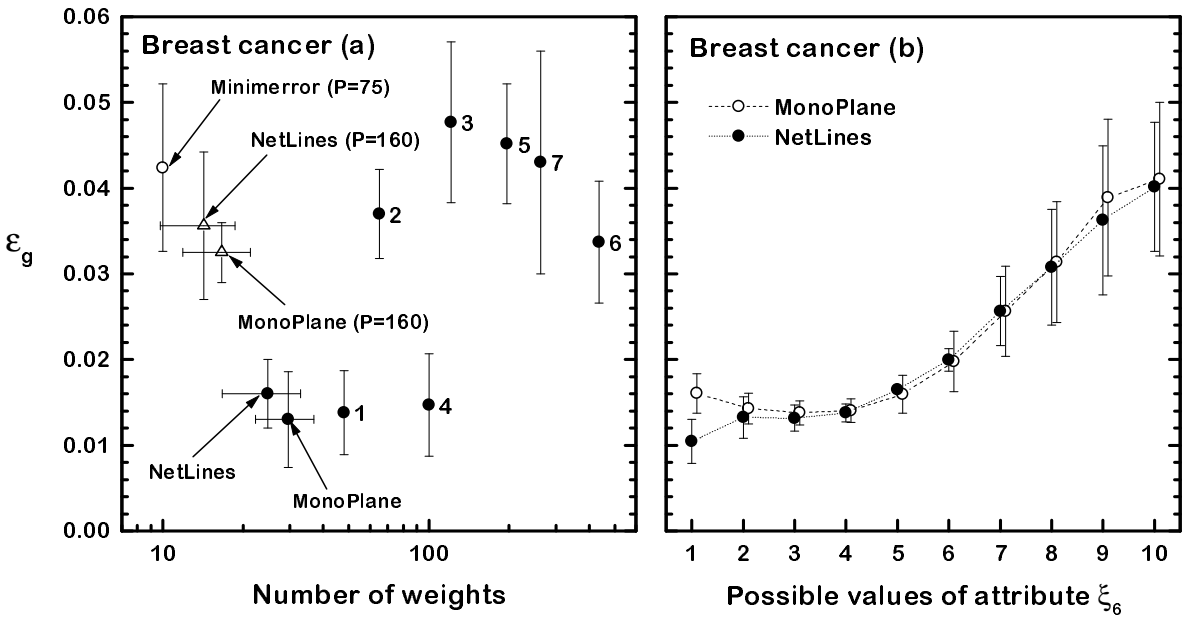

Figure 6.8: Classification du cancer du sein. (a) ε_g calculé par plusieurs algorithmes comme une fonction du nombre de poids (échelle logarithmique) des réseaux correspondants à $P = 525$. 1, 2, 3: RPG sans connexions de l'entre vers toutes les unités (court-circuits) [78] ; 4, 5, 6 : RPG avec des court-circuits [78] ; 7 : *Cascade Correlation* [23]. Des résultats avec d'ensembles d'apprentissage plus petits $P = 75$ et $P = 160$ (résolus avec un perceptron simple), sont montrés. (b) Erreurs de classification vs. valeurs possibles de l'attribut manquant pour les 16 exemples incomplets. Les résultats de Monoplan et NetLines sont moyennés sur 50 tests.

Dans la figure 6.9 nous montrons les performances obtenues par les trois heuris-

tiques incrémentales : Monoplan et NetLines ont obtenu les meilleurs performances en ε_g et poids, et NetSphères obtient un nombre plus important de poids, montrant que le réseau hypersphérique n'est pas optimal dans ce problème. Malgré tout, un perceptron hypersphérique obtient des performances comparables en ε_g à Monoplan et NetLines, mais avec un faible nombre de poids.

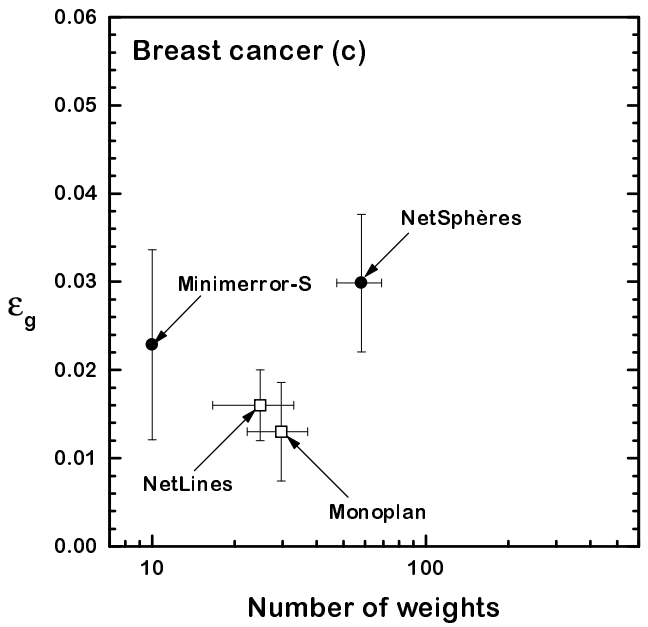

Figure 6.9: Classification du cancer de sein. (c) Erreur de généralisation ε_g calculée par Monoplan, NetLines et NetSphères comme une fonction du nombre de poids (échelle logarithmique) des réseaux correspondants à $P = 525$ exemples. Moyennes sur 50 réseaux indépendants.

Le diagnostic de diabète

Dans ce problème [66, 78] il y a $D = 768$ exemples décrits par $N = 8$ attributs réels (voir le tableau 6.4), correspondant à $\approx 35\%$ de femmes indiennes Pima souffrant de diabète, les 65% restantes étant saines.

Des ensembles d'apprentissage de $P = 576$ exemples étaient sélectionnés au hasard, et la généralisation était mesurée sur les $G = 192$ exemples restants. Une comparaison des résultats obtenus par d'autres algorithmes dans les mêmes conditions, présentée dans la figure 6.10, montre que NetLines a besoin de moins de paramètres, et atteint une meilleure performance.

Attributs	Interprétation
ξ_1	Nombre de fois enceinte
ξ_2	Concentration de glucose 2 heures après un test oral de tolérance
ξ_3	Pression diastolique du sang (mmHg)
ξ_4	Épaisseur de peau dans le triceps (mm)
ξ_5	2 heures sérum insuline (mu U/ml)
ξ_6	Indice de la masse du corps (kg/m^2)
ξ_7	Fonction de prédisposition au diabète
ξ_8	Âge (années)

Tables 6.4: Attributs des données pour le diagnostic de diabète des indiens Pima.

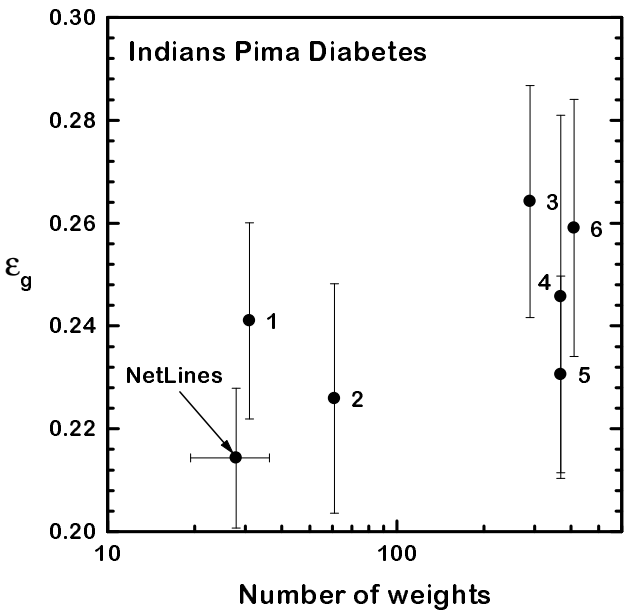

Figure 6.10: Diagnostic de diabète. Erreur de généralisation ε_g vs. le numéro de poids (échelle logarithmique). 1, 2, 3: RPG sans court-circuits [78] ; 4, 5, 6 : RPG avec des court-circuits [78]. Les résultats de NetLines sont des moyennes sur 50 tests.

6.3.4 Problèmes à plus de deux classes

Nous avons appliqué nos algorithmes à deux problèmes différents, tous les deux à trois classes. Nous comparons les résultats obtenus avec une structure du type *le gagnant prend tout* (*WTA*), où la classification est basée sur les résultats de trois réseaux, chacun entraîné indépendamment dédie à séparer une classe des deux autres, pour bâtir la structure de réseau en arbre. De même que le nombre de classes est réduit, nous avons construit les sous-réseaux avec toutes les séquences possibles

d'apprentissage. Le vote améliore la performance, comme on l'attendait.

Les trois formes d'ondes de Breiman

Ce problème a été introduit pour tester la méthode *CART* [6]. Les exemples d'entrée sont définis par $N = 21$ amplitudes réelles $x(t)$ échantillonnées régulièrement à intervalles $t = 1, 2, \cdots, N$. Chaque exemple est une combinaison linéaire, convexe et bruitée de deux parmi les trois ondes élémentaires $h_1(t), h_2(t)$, et $h_3(t)$ montrées dans la figure 6.11. Il y a trois combinaisons possibles, et la classe de l'exemple identifie de quelle combinaison il est émis. Les vecteurs d'entrée pour chaque classe sont engendrés de la manière suivante :

$$\begin{aligned} x(t) &= uh_1(t) + (1-u)h_2(t) + \eta(t) \ ; \ t = 1, \cdots, 21 \quad \text{Classe 1} \\ x(t) &= uh_1(t) + (1-u)h_3(t) + \eta(t) \ ; \ t = 1, \cdots, 21 \quad \text{Classe 2} \\ x(t) &= uh_2(t) + (1-u)h_3(t) + \eta(t) \ ; \ t = 1, \cdots, 21 \quad \text{Classe 3} \end{aligned}$$

où u est une variable aléatoire suivant une loi uniforme de distribution sur l'intervalle $[0, 1]$, et $\eta_t \ ; \ t = 1, \cdots, 21$ des variables aléatoires de moyenne nulle et variance unitaire. Les classes sont équiprobables.

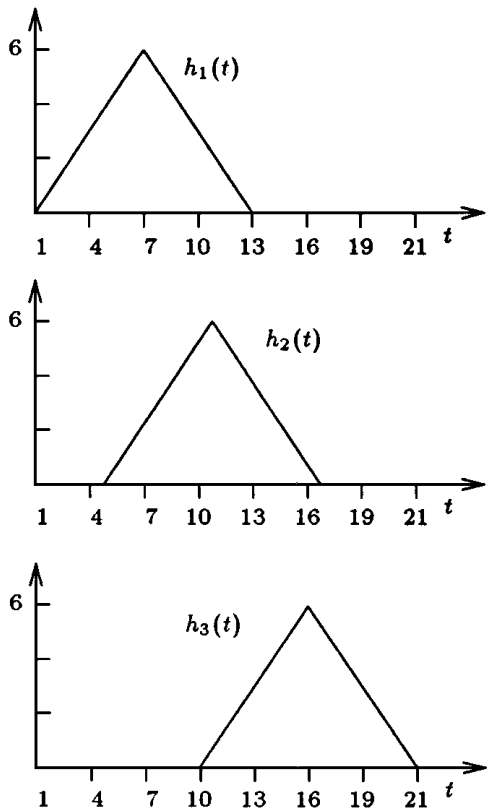

Figure 6.11: Les trois ondes de base. Ondes triangulaires centrées sur trois valeurs différentes de t.

Bien qu'il soit connu que, due au bruit, l'erreur de classification a une borne supérieure de $\varepsilon_g \approx 0.14$ [6], nous avons entraîné NetLines jusqu'à ce que l'erreur d'apprentissage soit zéro.

Nous avons entraîné les réseaux avec les mêmes 11 ensembles d'apprentissage de $P = 300$ exemples, et faisant pour la généralisation un test indépendant sur un ensemble de $G = 5000$ comme dans une récente et très complète étude [38]. Nos résultats, avec les meilleurs résultats d'autres algorithmes (qui atteignaient $\varepsilon_g < 0.25$ dans [38]) sont montrés dans la figure 6.12. Les résultats de NetLines et Monoplan [97, 96] correspondent à l'erreur d'apprentissage $\varepsilon_t = 0$. En fait, les résultats obtenus avec un perceptron simple entraîné avec Minimerror-L ou avec l'algorithme du Perceptron (qui peut être considéré comme le cas extrême de *early stopping*) ne sont guère améliorés par des réseaux plus complexes. Ici encore nous voyons que, contrairement à ce qu'on croyait, apprenant avec un algorithme à croissance ne produit pas de sur-apprentissage : le résultant ε_g reste le même que celui obtenu avec des réseaux plus petits. Les réseaux engendrés par NetLines ont entre 3 et 6 neurones cachés, dépendant des ensembles d'apprentissage. La structure du vote des réseaux en arbre réduit la variance, mais n'améliore pas la moyenne sur ε_g.

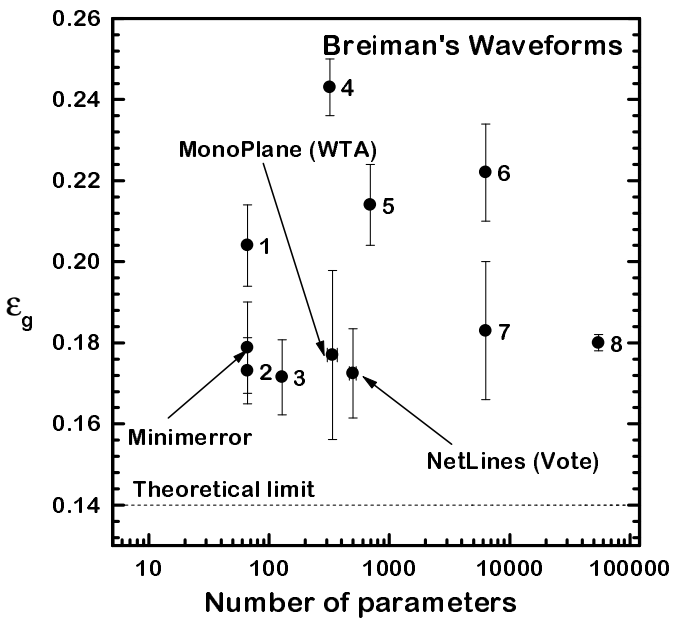

Figure 6.12: Les formes d'ondes de Breiman. Erreurs de généralisation de plusieurs algorithmes moyennés sur 11 tests vs. le nombre de poids (échelle logarithmique). Les barres d'erreur pour Monoplan et NetLines sont à peines visibles à l'échelle de la figure. 1 : Discrimination linéaire, 2 : Algorithme du Perceptron, 3 : RPG, 4 : Algorithme génétique, 5 : Discrimination quadratique, 6 : Fenêtres de Parzen. 7 : K-Plus proches voisins, 8 : Algorithme de Contraintes [38]

La classification des Iris

Dans ce problème classique à trois classes, on doit déterminer la classe de plantes d'iris, à partir des valeurs de $N = 4$ attributs réels. La base de données de $D = 150$ exemples, contient 50 exemples de chaque classe. Les réseaux étaient entraînés avec $P = 149$ exemples, et l'erreur de généralisation ε_g est la valeur moyenne de tous les 150 *leave-one-out* tests possibles. Les résultats de ε_g sont montrés comme une fonction du nombre de poids dans la figure 6.13. Les barres d'erreur sont disponibles seulement pour nos propres résultats. Dans ce problème, le vote des trois arbres, N_1, N_2 et N_3, formé avec les trois séquences possibles d'apprentissage, améliore la performance en généralisation.

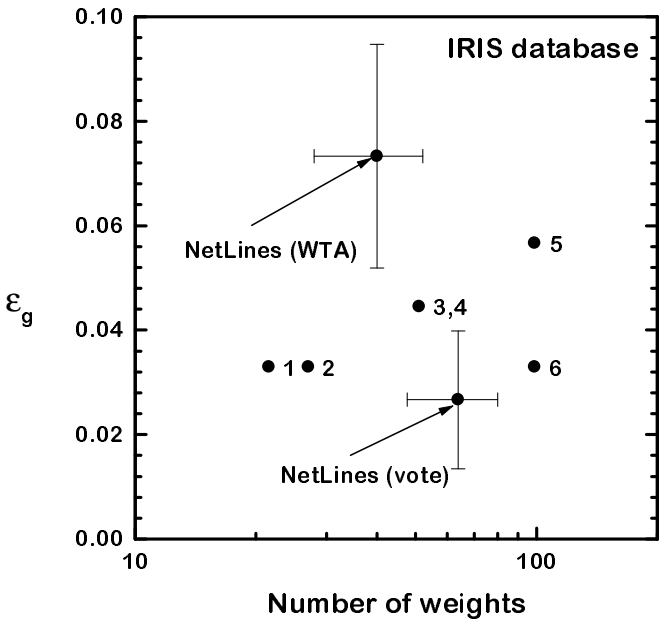

Figure 6.13: Iris. Erreur de généralisation ε_g vs. le nombre de paramètres (échelle logarithmique). 1 : $Offset$ [64], 2 : RPG [64] ; 4, 5 : RPG, 3 et 6 : GOT [104].

Autres applications

Outre ces résultats, nous avons fait aussi deux études préliminaires d'applications, l'une issue du domaine médical : la classification des données du Service de Toxicologie du *Centre Hospitalier Universitaire de Grenoble* [22, 26] et l'autre du domaine industriel : le diagnostic non destructif de failles dans des tuyaux métalliques à partir des mesures magnétiques [74]. Ces études sont encore en cours, mais les résultats de tests préliminaires sont prometteurs.

6.4 Conclusion

Dans ce chapitre, nous avons présenté des comparaisons avec des résultats obtenus par d'autres algorithmes d'apprentissage sur plusieurs problèmes étalon, sur la base de l'erreur de généralisation et du nombre de paramètres du réseau. Le problème des échos sonar, par exemple a montré la qualité de l'algorithme d'apprentissage Minimerror-L dans des problèmes à grande dimension. Le problème binaire de deux ou plus domaines, nous a appris quelque chose sur la capacité de généralisation et le sur-apprentissage : en effet, nous n'avons pas constaté d'amélioration sur l'erreur de généralisation en arrêtant la croissance avant que l'erreur d'apprentissage soit zéro. Dans les problèmes à entrées réelles, nous espérions une performance de NetLines supérieure à celle de Monoplan, fait que nous avons corroboré sur la classification de cancer du sein (où la performance de NetLines est légèrement inférieure mais le réseau est moins complexe) et dans le problème de classification des diabète, que nous n'avons pas résolu avec Monoplan.

Nous avons aussi constaté l'efficacité de la stratégie des réseaux en arbre dans les problèmes à trois classes des ondes de Breiman et de classification des iris. Ainsi, nous avons montré que les réseaux bâtis avec Monoplan et NetLines ont des performances en généralisation comparables (voire meilleures) à celles obtenues avec d'autres méthodes.

En ce qui concerne l'algorithme NetSphères, on ne peut pas conclure de choses spécifiques, car il est encore en test. D'autres voies pour l'apprentissage des cibles des unités cachées doivent être explorées, ce qui, en principe, devrait réduire le nombre d'unités nécessaires et donc, améliorer sa capacité de généralisation.

Chapitre 7

Conclusion et Perspectives

DAns ce mémoire, nous avons présenté des algorithmes d'apprentissage constructifs permettant de générer des réseaux de neurones binaires à une couche cachée, pour la classification de données.

Le développement d'algorithmes constructifs comporte deux aspects, également importants. Le premier est la mise au point de l'algorithme d'apprentissage des unités que l'on sera amenés à ajouter, et qu'on devra entraîner efficacement. Le deuxième est la mise au point d'une heuristique de construction intelligente. La défaillance de l'un ou de l'autre a pour conséquence la construction de réseaux trop grands, voire l'impossibilité d'apprendre et de généraliser correctement.

Dans le travail décrit dans cette thèse, nous avons contribué à ces deux aspects. Nous avons d'abord amélioré les performances d'un algorithme existant, Minimerror-L, pour l'apprentissage avec des neurones linéaires binaires, et nous avons proposé une généralisation non triviale, Minimerror-S, permettant l'apprentissage de séparations hypersphériques. Nous appelons perceptrons sphériques les unités binaires correspondantes. Ces deux algorithmes ont des propriétés théoriques intéressantes, et présentent des avantages sur d'autres algorithmes car ils permettent l'apprentissage d'ensembles séparables ou non.

Nous avons développé trois heuristiques pour la construction de réseaux à unités binaires pour les problèmes de classification. Les réseaux engendrés par ces méthodes sont sans rétroaction avec une seule couche cachée connectée aux entrées, et un neurone de sortie connecté aux unités cachées :

- Monoplan construit une machine à parité, bien adaptée à des problèmes à entrées binaires. Les unités cachées sont ajoutées successivement, pour corriger les erreurs de l'unité précédente.

- NetLines construit un réseau dont la décision de rajouter des unités cachées est pilotée par l'erreur d'apprentissage du neurone de sortie. Cette heuristique

est bien adaptée à des problèmes à entrées réelles.

- NetSphères utilise la même heuristique que NetLines, mais avec des unités cachées binaires sphériques.

Ces heuristiques couplent une stratégie d'addition successive de neurones cachés avec les algorithmes d'apprentissage Minimerror (L/S). Ces algorithmes permettent une adaptation automatique du classifieur à la complexité de la tâche. Les réseaux construits ont des unités cachées binaires dont les états constituent un codage fidèle des exemples. Chaque neurone caché définit une frontière (ou morceau de frontière) entre les classes dans l'espace des entrées. L'apprentissage est rapide, parce qu'il est réduit à celui de perceptrons simples. Des théorèmes de convergence (valables aussi bien pour des entrées binaires que réelles) garantissent qu'une solution avec un nombre fini de neurones cachés, existe.

Nous avons testé nos méthodes et comparé leurs résultats sur plusieurs problèmes étalon. Les réseaux engendrés par nos stratégies de croissance sont petits, bien que nous rajoutions des neurones cachés jusqu'à ce que le nombre d'erreurs d'apprentissage s'annule. Dans les cas où NetLines bâtit de plus grands réseaux que ceux construits par d'autres algorithmes, l'erreur de généralisation reste faible, montrant que nos algorithmes constructifs ne présentent pas de phénomène de surapprentissage.

Ce travail ouvre de nouvelles perspectives dans le domaine du développement constructif de réseaux pour la classification. Parmi celles qui nous semblent prometteuses, nous pouvons citer :

- Construction des classifieurs en arbre (comme *Neural Tree*) en utilisant Minimerror-L et/ou Minimerror-S comme algorithme d'entraînement des nœuds.

- Construction d'un classifieur hybride qui combine des hyperplans et des hypersphères.

- Construction d'un réseau à plusieurs couches cachées. Certains études montrent qu'un réseau à deux couches cachées peut être un meilleur approximateur qu'un réseau à une seule couche cachée. Des études empiriques, dans le cadre de la régression, montrent aussi que le nombre d'unités croit exponentiellement pour des réseaux à une couche cachée, mais polynomialement dans un réseau à 2 couches cachées.

Annexes A

Algorithmes

Dans cette annexe nous présentons le pseudo-code de nos deux algorithmes d'apprentissage de perceptrons linéaires Minimerror-L et sphériques Minimerror-S ; ainsi que les trois heuristiques incrémentales Monoplan, Netlines et Netsphères.

On rappelle que l'ensemble d'apprentissage $\mathcal{L}^\alpha = \{\vec{\xi}^\mu, \tau^\mu\}$, $\tau = \pm 1$, $\mu = 1, \cdots, P$ est composé de P exemples du type entrée-sortie, et que les représentations internes σ^μ et la sortie du réseau $\zeta^\mu = \pm 1$ sont binaires.

Algorithm 1 Minimerror-L

Require:

$\mathcal{L}^{\alpha}$, N, P

$w_i \Leftarrow \sum_{\mu} \xi_i^{\mu} \tau^{\mu}$; $i = 0, \cdots, N$ {Règle d'Hebb}

ϵ_0 {Pas du gradient}

0.9 {Facteur de décroissance de ϵ}

β_0; {Température initial}; $\delta\beta_0$ {Decrément de la température}

$\Psi \Leftarrow \frac{\beta_+}{\beta_-}$ {Rapport de températures}

a {Condition d'arrêt}

β_{MAX} {Maximum de bruit}; I_{MAX} {Maximum d'itérations}

INITIALISATION

$t \Leftarrow 0$

$\beta_+(0) \Leftarrow \beta_0$; $\delta\beta(t) \Leftarrow \delta\beta_0$

$\vec{w_*} \Leftarrow \vec{w}$

$\zeta^{\mu} \Leftarrow \text{signe}(\vec{w_*} \cdot \vec{\xi^{\mu}})$; $\mu = 1, \cdots, P$

$f_* \Leftarrow \sum_{\mu}(\zeta^{\mu} - \tau^{\mu})/2$ {Fautes}

$\beta_* \Leftarrow \beta_+$ {Température mémorisée de non changement du f_*}

repeat

$t \Leftarrow t + 1$

$\beta_+(t) \Leftarrow \beta_+(t) + \delta\beta(\text{t})$

$\beta_- \Leftarrow \beta_+(t)/\Psi$ {Vérifier toutes les t_0 itérations}

if $(\beta_+(t) < \beta_*)$ **then**

$\epsilon \Leftarrow \epsilon * 0.9$

else

$\beta_* \Leftarrow \beta_+(t)$

end if

$$\delta w_i \Leftarrow -\epsilon \left\{ \sum_{\gamma^{\mu} \leq 0} \frac{\xi_i^{\mu} \tau^{\mu}}{\cosh^2(\frac{\beta_- \gamma^{\mu}}{2})} + \sum_{\gamma^{\mu} > 0} \frac{\xi_i^{\mu} \tau^{\mu}}{\cosh^2(\frac{\beta_+ \gamma^{\mu}}{2})} \right\}$$

$\vec{w} \Leftarrow \vec{w} + \delta\vec{w}$

$\vec{w} \Leftarrow \vec{w}/\|\vec{w}\|$

$f \Leftarrow \sum_{\mu}(\zeta^{\mu} - \tau^{\mu})/2$ {Fautes}

if $(f < f_*)$ **then**

$f_* \Leftarrow f; \beta_* \Leftarrow \beta_+(t); \vec{w_*} \Leftarrow \vec{w}; \delta\beta(t) \Leftarrow \delta\beta_0$

else

$\delta\beta(t) \Leftarrow \log(t+1)\delta\beta_0$

end if

until $(t < I_{MAX}$ ET $\beta_- > a/\gamma$ ET $\beta_+ < \beta_{MAX})$

Minimiser avec gradient conjugué à la température $\beta = \beta_*$

$E^*(\gamma\ ; \beta) = \sum_{\mu} \frac{1}{2}(1 - \tanh\frac{\beta\gamma^{\mu}}{2}) + (\sqrt{N+1} - \|\vec{w}\|)^2$

Algorithm 2 Minimerror-S

Require:

$\mathcal{L}^{\alpha}$, N, P

{Centre de gravité et rayon calculé avec l'exemple ν de la classe opposée} $\vec{w_i}(0) = \frac{1}{P_-}\sum_{\mu|\tau^\mu=-1}\xi_i^\mu$; $i = 1, \cdots, N$; $\rho(0) = \min\{\sqrt{(\vec{w}-\vec{\xi^\nu})^2}\}$

ϵ_0 {Pas du gradient}; 0.9 {Facteur de décroissance de ϵ}; $\delta\beta_0$ {Decrément de la température}; $\Psi \Leftarrow \frac{\beta_+}{\beta_-}$ {Rapport de températures}; a {Condition d'arrêt}; β_{MAX} {Maximum de bruit}; I_{MAX} {Maximum d'itérations}

INITIALISATION

$t \Leftarrow 0$; $\delta\beta(t) \Leftarrow \delta\beta_0$; $\vec{w_*} \Leftarrow \vec{w}$; $\zeta^\mu \Leftarrow \text{signe}(\vec{w_*}\cdot\vec{\xi^\mu}); \mu = 1, \cdots, P$;

$f_* \Leftarrow \sum_\mu(\zeta^\mu - \tau^\mu)/2$; $\beta_* \Leftarrow \beta_+$

repeat

$t \Leftarrow t+1$

$\beta_+(t) \Leftarrow \beta_+(t) + \delta\beta(\text{t})$; $\beta_- \Leftarrow \beta_+(t)/\Psi$

Vérifier toutes les t_0 itérations :

if $(\beta_+(t) < \beta_*)$ **then**

$\epsilon \Leftarrow \epsilon * 0.9$

else

$\beta_* \Leftarrow \beta_+(t)$;

end if

- {a. Définition euclidienne (4.33)}

$$\delta w_i = +\frac{\epsilon}{2}\left\{\sum_{\lambda^\mu\leq 0}\frac{(w_i-\xi_i^\mu)}{\cosh^2(\frac{\beta_-\lambda^\mu}{2})}\tau^\mu + \sum_{\lambda^\mu>0}\frac{(w_i-\xi_i^\mu)}{\cosh^2(\frac{\beta_+\lambda^\mu}{2})}\tau^\mu\right\}$$

$$\delta\rho = -\frac{\epsilon}{2}\left\{\sum_{\lambda^\mu\leq 0}\frac{\tau^\mu}{\cosh^2(\frac{\beta_-\lambda^\mu}{2})} + \sum_{\lambda^\mu>0}\frac{\tau^\mu}{\cosh^2(\frac{\beta_+\lambda^\mu}{2})}\right\}$$

- {b. Définition logarithmique (4.37)}

$$\delta w_i = +\frac{\epsilon}{4}\left\{\sum_{\lambda^\mu\leq 0}\frac{(w_i-\xi_i^\mu)}{\cosh^2(\frac{\beta_-\lambda^\mu}{2})\|\vec{w}-\vec{\xi}\|^2}\tau^\mu + \sum_{\lambda^\mu>0}\frac{(w_i-\xi_i^\mu)}{\cosh^2(\frac{\beta_+\lambda^\mu}{2})\|\vec{w}-\vec{\xi}\|^2}\tau^\mu\right\}$$

$$\delta\rho = -\frac{\epsilon}{4}\left\{\sum_{\lambda^\mu\leq 0}\frac{\tau^\mu}{\rho\cosh^2(\frac{\beta_-\lambda^\mu}{2})} + \sum_{\lambda^\mu>0}\frac{\tau^\mu}{\rho\cosh^2(\frac{\beta_+\lambda^\mu}{2})}\right\}$$

$\vec{w} \Leftarrow \vec{w} + \delta\vec{w}$

$f \Leftarrow \sum_\mu(\zeta^\mu - \tau^\mu)/2$ {Fautes}

if $(f < f_*)$ **then**

$f_* \Leftarrow f; \beta_* \Leftarrow \beta_+(t); \vec{w_*} \Leftarrow \vec{w}; \delta\beta(t) \Leftarrow \delta\beta_0$

else

$\delta\beta(t) \Leftarrow \log(t+1)\delta\beta_0$

end if

until $(t < I_{MAX}$ ET $\beta_- > a/\gamma$ ET $\beta_+ < \beta_{MAX})$

Algorithm 3 Monoplan

Require: $\mathcal{L}^{\alpha}$

$h = 0$

Mettre la classe à apprendre $\{\tau^{\mu}_{h+1} = \tau^{\mu}\}, \mu = 1, \cdots, P$

repeat

repeat

Construire la couche cachée :

$h \Leftarrow h + 1$

Connecter l'unité cachée h aux $N + 1$ entrées

Apprendre l'ensemble $\mathcal{L} \Leftarrow \{\vec{\xi}^{\mu}, \tau^{\mu}_{h}\}_{\mu=1,\cdots,P}$ avec Minimerror-L

Soit σ^{μ}_{h} la sortie calculé de l'exemple μ après l'apprentissage

Mettre les nouveaux objetives à apprendre : $\{\tau^{\mu}_{h+1} = \tau^{\mu}_{h}\sigma^{\mu}_{h}\}_{\mu=1,\cdots,P}$

if $(h = 1$ ET $\tau^{\mu}_{h+1} = 1; \forall\mu)$ **then**

Stop $\{\mathcal{L}$ est linéairement séparable$\}$

end if

$e_h \Leftarrow \sum_{\mu}(1 - \tau^{\mu}_{h+1})/2$; {Compter le nombre d'erreurs d'apprentissage}

until $(e_h > 0)$

Apprendre la sortie :

Connecter l'unité de sortie aux h unités cachées

Apprendre l'ensemble de RI $\mathcal{L} \Leftarrow \{\vec{\sigma}^{\mu}(h), \tau^{\mu}\}_{\mu=1,\cdots,P}$ avec Minimerror-L

Soit ζ^{μ} la sortie finale à l'exemple μ après l'apprentissage;

Mettre $\{\tau^{\mu}_{h+1} \Leftarrow \tau^{\mu}\zeta^{\mu}\}_{\mu=1,\cdots,P}$

$e_{\zeta} \Leftarrow \sum_{\mu}(1 - \tau^{\mu}_{h+1})/2$ {Compter le nombre d'erreurs d'apprentissage}

until $(h \leq H_{max}$ ET $e_{\zeta} > E_{max})$ {Condition d'arrêt}

Algorithm 4 NetLines

Require: $\mathcal{L}^{\alpha}$, N, P

$h = 0$

Mettre la classe à apprendre $\{\tau^{\mu}_{h+1} = \tau^{\mu}\}$ pour $\mu = 1, \cdots, P$

repeat

repeat

Entraîner les unités cachées :

$h \Leftarrow h + 1$ {Connecter l'unité cachée h aux $N + 1$ entrées}

Apprendre l'ensemble $\mathcal{L} \Leftarrow \{\vec{\xi}^{\mu}, \tau^{\mu}_{h}\}, \mu = 1, \cdots, P$ avec Minimerror-L

Soit σ^{μ}_{h} la sortie calculé de l'exemple μ après l'apprentissage

Mettre les nouveaux objectifs à apprendre : $\{\tau^{\mu}_{h+1} = \tau^{\mu}_{h}\sigma^{\mu}_{h}\}_{\mu=1,\cdots,P}$

if $(h = 1)$ **then**

{Pour le premier neurone caché}

if $(\tau^{\mu}_{h+1} = 1;\ \forall\mu)$ **then**

Stop {$\mathcal{L}$ est linéairement séparable}

else

mettre $\tau^{\mu}_{h+1} = \sigma^{\mu}_{h}\tau^{\mu}$ pour $\mu = 1, \cdots, P$; aller à 1;

end if

end if

$e_h \Leftarrow \sum_{\mu}(1 - \tau^{\mu}_{h+1})/2$; {Compter le nombre d'erreurs d'apprentissage}

until $(e_h > 0)$

Apprendre la relation entre les RI et les sorties :

Connecter le neurone de sortie aux h unités cachées entraînées

Apprendre l'ensemble de RI $\mathcal{L} \Leftarrow \{\vec{\sigma}^{\mu}(h), \tau^{\mu}\}, \mu = 1, \cdots, P$ avec Minimerror-L

Soit ζ^{μ} la sortie finale à l'exemple μ après l'apprentissage;

Mettre $\{\tau^{\mu}_{h+1} \Leftarrow \tau^{\mu}\zeta^{\mu}\}_{\mu=1,\cdots,P}$

$e_{\zeta} \Leftarrow \sum_{\mu}(1 - \tau^{\mu}_{h+1})/2$ {Compter le nombre d'erreurs d'apprentissage}

until $(h = H_{max}$ OU $e \leq E_{max})$; {Condition d'arrêt}

Algorithm 5 NetSphères

Require: $\mathcal{L}^{\alpha}$, N, P

$h = 0$

Mettre la classe à apprendre $\{\tau^{\mu}_{h+1} = \tau^{\mu}\}$ pour $\mu = 1, \cdots, P$

repeat

repeat

Entraîner les unités cachées sphériques :

$h \Leftarrow h + 1$ {Connecter l'unité cachée h aux $N + 1$ entrées}

if $(h = 1)$ **then**

$\vec{w}_1 = \frac{1}{P_-} \sum_{\mu|\tau^{\mu}=-1} \xi^{\mu}_i$

else

$\vec{w}_h = \vec{\xi^{\mu}}|\sigma^{\mu}_{h-1} \neq \tau^{\mu}_{h-1}$

end if

Apprendre l'ensemble $\mathcal{L} \Leftarrow \{\vec{\xi^{\mu}}, \tau^{\mu}_h\}, \mu = 1, \cdots, P$ avec Minimerror-S

Soit σ^{μ}_h la sortie calculé de l'exemple μ après l'apprentissage

Mettre les nouveaux objetives à apprendre : $\{\tau^{\mu}_{h+1} = \tau^{\mu}_h \sigma^{\mu}_h\}_{\mu=1,\cdots,P}$

if $(h = 1)$ **then**

{Pour le premier neurone caché}

if $(\tau^{\mu}_{h+1} = 1; \forall \mu)$ **then**

Stop {$\mathcal{L}$ est séparable}

else

mettre $\tau^{\mu}_{h+1} = \sigma^{\mu}_h \tau^{\mu}$ pour $\mu = 1, \cdots, P$; aller à 1;

end if

end if

$e_h \Leftarrow \sum_{\mu}(1 - \tau^{\mu}_{h+1})/2$; {Compter le nombre d'erreurs d'apprentissage}

until $(e_h > 0)$

Apprendre la relation entre les RI et les sorties :

Connecter le neurone de sortie aux h unités cachées entraînées

Apprendre l'ensemble de RI $\mathcal{L} \Leftarrow \{\vec{\sigma}^{\mu}(h), \tau^{\mu}\}, \mu = 1, \cdots, P$ avec Minimerror-L

Soit ζ^{μ} la sortie finale à l'exemple μ après l'apprentissage;

Mettre $\{\tau^{\mu}_{h+1} \Leftarrow \tau^{\mu}\zeta^{\mu}\}_{\mu=1,\cdots,P}$

$e_{\zeta} \Leftarrow \sum_{\mu}(1 - \tau^{\mu}_{h+1})/2$ {Compter le nombre d'erreurs d'apprentissage}

until $(h = H_{max}$ OU $e \leq E_{max})$; {Condition d'arrêt}

Annexes B

La convergence de NetLines

DAns cette annexe, nous allons montrer une solution particulière à la stratégie d'apprentissage de NetLines. Cette solution est bâtie de manière telle que le cardinal d'un sous-ensemble convexe d'exemples bien appris, L_h, grandit de façon monotone avec l'addition d'unités cachées. Comme ce cardinal ne peut pas être plus grand que le nombre total d'exemples d'apprentissage, l'algorithme doit s'arrêter avec un nombre fini d'unités cachées.

On suppose que h unités cachées ont été ajoutées au réseau et que le neurone de sortie fait encore des erreurs de classification sur les exemples de $\mathcal{L}^{\alpha}$, (erreurs d'apprentissage). Parmi ces exemples mal appris, soit ν l'exemple le plus proche de l'hyperplan normal à $\vec{w}_h$, ci-après hyperplan-h. On définit L_h le sous-ensemble d'exemples (correctement appris) placé entre l'hyperplan-h et l'exemple $\vec{\xi}^{\nu}$. Les exemples $\in L_h$ ont $0 < \gamma_h < |\gamma_h^{\nu}|$. Le sous-espace L_h et au moins l'exemple ν sont bien appris si l'unité cachée, $h+1$, a les poids :

$$\vec{w}_{h+1} = \tau_h^{\nu}\vec{w}_h - (1-\epsilon_h)\tau_h^{\nu}(\vec{w}_h \cdot \vec{\xi}^{\nu})\hat{e}_0 \tag{B.1}$$

où $\hat{e}_0 \equiv (1, 0, \cdots, 0)$. Les conditions que tous les deux L_h et l'exemple ν aient des stabilités positives (*i.e.* correctement appris) imposent que :

$$0 < \epsilon_h < \min_{\mu \in L_h} \frac{|\gamma_h^{\nu}| - \gamma_h^{\mu}}{|\gamma_h^{\nu}|} \tag{B.2}$$

Les poids suivants entre les unités cachées et la sortie peuvent donner la sortie correcte à l'exemple ν *et* aux exemples L_h :

$$W_0(h+1) = W_0(h) + \tau^{\nu} \tag{B.3}$$

$$W_i(h+1) = W_i(h) \; for \; 1 \leq i \leq h \tag{B.4}$$

$$W_{h+1}(h+1) = -\tau^{\nu} \tag{B.5}$$

Alors, $card(L_{h+1}) \geq card(L_h) + 1$. Comme le nombre d'exemples $\in L_h$ augmente de façon monotone avec h, la convergence est garantie avec moins de P unités cachées.

Annexes C

Algorithmes incrémentaux

DAns cette annexe nous présentons en pseudo-code quelques algorithmes incrémentales pour la classification. Pour tous ceux-là, l'ensemble d'apprentissage $\mathcal{L}^\alpha = \{\vec{\xi}^\mu, \tau^\mu\}$, $\tau = \pm 1$, $\mu = 1, \cdots, P$ est composé de P exemples du type entrée-sortie, les représentations internes RI par σ^μ et la sortie du réseau $\zeta^\mu = \pm 1$. Nous avons indiqué pour chacune des méthodes, l'algorithme d'apprentissage des unités individuelles utilisé par leurs auteurs.

C.1 L'algorithme *Tiling*

Algorithme proposé par Mézard et Nadal [68].

Algorithm 6 Tiling

1. Début. Couche $L = 1$
2. Mettre $h = 0$. Entraîner l'unité maîtresse $\sigma_{L,0}$ avec un algorithme comme *pocket* sur $\mathcal{L}^\alpha$.
3. Tous les exemples sont-ils bien classés ?, si oui aller à 9.
4. Faire $h = h+1$ pour ajouter une unité auxiliaire σ_L^h qui va corriger les exemples mal classées par l'unité $\sigma_{L,0}$
5. Calculer les RI $\vec{\sigma}_L = (\sigma_{L,0}^\mu, \sigma_{L,1}^\mu, \cdots, \sigma_{L,h}^\mu)$ $\mu = 1, \cdots, P$.
6. Toutes les RI sont-elles fidèles ? sinon aller à 4.
7. Ajouter une nouvelle couche :
faire $L = L + 1$. Apprendre l'ensemble $\mathcal{L} = \{\vec{\sigma}_{L-1}^\mu, \tau^\mu\}$; $\mu = 1, \cdots, P$ (La couche L est construite en considérant comme entrées, les sorties de la couche $L - 1$).
8. Aller à 2.
9. Fin.

C.2 *Sequential Learning*

L'algorithme *Sequential Learning* a été crée par de Marchand *et al.* [65].

Algorithm 7 Sequential Learning

1. Commencer avec une seule unité. Mettre deux compteurs $p = P, h = 1$ pour quantifier le nombre d'exemples non appris et le nombre d'unités cachées.
2. Apprentissage. L'unité cachée h est entraîné sur les p exemples.
3. Réduire l'ensemble d'apprentissage. Les p_h avec $\tau^\mu = -1$ exemples qui ont été correctement classés sont enlevés de l'ensemble d'apprentissage. Faire $p = p_h$. Si tous les p exemples ont le même τ^μ, aller à 5.
4. Insérer une nouvelle unité. Geler les poids de l'unité h, ajouter une nouvelle unité connecté à toutes les entrées. Faire $h = h + 1$. Aller à 2
5. Connecter l'unité de sortie ζ.
6. Apprendre la sortie avec l'algorithme du perceptron.
7. FIN

C.3 L'algorithme *Cascade Correlation*

L'algorithme *Cascade Correlation* a été crée par Fahlman et Lebière [29].

Algorithm 8 *Cascade Correlation*

1. Commencer avec un perceptron simple avec des poids choisis au hasard. Entraîner jusqu'à une performance choisi avec l'algorithme *quickprop*. Si la performance est trouvée, aller à 5, autrement ajouter des unités cachées une à une.
2. Une queue d'unités candidates est générée. Chaque unité fera une recherche dans l'espace de poids. Cette queue permettra de diminuer le risque d'insérer une unité attrapée dans un minimum local. Chaque unité candidat est entraîné pour maximiser la corrélation entre sa sortie et la signal d'erreur résiduelle du réseau.
3. Insérer l'unité. L'unité candidate avec le score plus haut est inséré dans le réseau. On gèle ses poids. La nouvelle unité est connectée à la sortie avec des poids aléatoires.
4. Entraîner le réseau. L'unité de sortie est entraînée en calculant tous ses poids. Si la performance est satisfaisante, aller a 5, autrement aller à 2.
5. STOP

C.4 L'algorithme *Offset*

Cet algorithme a été crée par Martinez et Estève [64].

Algorithm 9 *Offset*

Couche cachée 1 :
1. Faire $h = 0$
Repeter

- Connecter le neurone h aux $N + 1$ entrées
- Apprendre l'ensemble $\mathcal{L}^{\alpha}$ avec l'algorithme *pocket*
- Soit ζ_h^{μ} la sortie calculé de l'exemple μ après l'apprentissage
- Mettre les nouveaux objectives à apprendre : $\tau^{\mu} = \tau^{\mu} \oplus \zeta^{\mu}$ (Les erreurs faites par l'unité H sont transformées en $\tau^{\mu} = +1$ et les exemples bien appris en $\tau^{\mu} = -1$, avec $\oplus$ défini comme la fonction booléenne OU-exlusif).
- Si il n'y a plus d'erreurs aller à la couche cachée 2

Tandis qu'il y ait des erreurs d'apprentissage.
Couche cachée 2 :
L'algorithme *Offset* réduit n'importe quelle fonction binaire à un problème de la parité à dimension h, en calculant les poids de façon géométrique pour donner la sortie correcte.
Créer h unités cachés binaires.
Soit P_h le nombre des RI *distinctes*; alors :

- Si $P_h = 2^h$ on a le problème de la parité complet.
- Si $P_h < 2^h$ on a la parité incomplète, on peut donc effacer des unités redondantes en gardant la fidélité des RI.

C.5 L'algorithme *Upstart*

L'algorithme *Upstart* a été crée par Frean [31].

Algorithm 10 *Upstart*

1. Apprendre l'ensemble $\mathcal{L}^\alpha$ avec un perceptron, qui sera l'unité mère Z.
2. Tester. Si tous les exemples sont bien classées alors STOP.
3. Si $(\zeta_Z^\mu \neq \tau^\mu)$ pour au moins un exemple μ, alors de façon recursive, intercaler deux neurones X et Y entre les entrées et l'unité mère :

- Enseigner à X $\begin{cases} \tau^\mu = +1 & \text{si } \tau^\mu = -1; \zeta_Z^\mu = +1 \textit{ (WON)} \\ \tau^\mu = -1 & \text{autrement} \end{cases}$
- Enseigner à Y $\begin{cases} \tau^\mu = +1 & \text{si } \tau^\mu = +1; \zeta_Z^\mu = -1 \textit{ (WON)} \\ \tau^\mu = -1 & \text{autrement} \end{cases}$

Apprendre la sortie à Z avec l'algorithme *Pocket*.
L'unité X devienne une unité mère
L'unité Y devienne autre unité mère
Aller à 2.

C.6 Arbre neuronal

Il y a deux versions d'arbres : de décision [6] ou neuronaux [87, 30]. Nous allons décrire un arbre neuronal *binaire*, où chaque noeud t peut être terminal (feuille associée à une classe $\tau = \pm 1$) ou un sous-arbre (nœud interne avec deux nœuds fils : gauche et droite). Chaque nœud est associé à un neurone, mais plusieurs nœuds peuvent être associés à une même classe.

Construction de l'arbre. Soit $\mathcal{L}^\alpha$ l'ensemble d'apprentissage. Soient $\mathcal{L}(t)$ l'ensemble d'apprentissage pour le nœud t et σ_t un neurone associé à t si celui-ci n'est pas une feuille ou bien une étiquette ± 1 dans le cas contraire.

Classification. La classe d'un exemple $\vec{\xi}^\mu$ peut être établie par :

Algorithm 11 Arbre neuronal. Construction

Require:

$\mathcal{L}^{\alpha}$, N, P

$t \Leftarrow racine$ et $\mathcal{L}(t) \Leftarrow \mathcal{L}^{\alpha}$

SI $(\mathcal{L}(t) = 0)$ alors STOP

Entraîner le neurone σ_t à séparer $\mathcal{L}(t)$ en deux sous ensembles :

$\mathcal{L}_{-}(t)$ où la sortie de $\sigma_t = -1$ et $\mathcal{L}_{+}(t)$ où la sortie de $\sigma_t = +1$

Ajouter à t un nœud gauche t_{-} :

- SI $\mathcal{L}_{-}(t)$ contient seulement exemples de la classe $\tau = -1$, alors t_{-} est une feuille, avec $\mathcal{L}(t_{-}) = 0$;
- SINON sera un nœud interne avec $\mathcal{L}(t_{-}) = \mathcal{L}_{-}(t)$.

Ajouter à t un nœud droite t_{+} :

- SI $\mathcal{L}_{+}(t)$ contient seulement exemples de la classe $\tau = +1$, alors t_{+} est une feuille, avec $\mathcal{L}(t_{+}) = 0$;
- SINON sera un nœud interne avec $\mathcal{L}(t_{+}) = \mathcal{L}_{+}(t)$.

Répéter le même algorithme pour t_{-} et t_{+}; commençant par le pas 2.

Algorithm 12 Arbre neuronal. Classification

Require:

$\vec{\xi}^{\mu}$

1. Soit $t = racine$
2. Si le nœud t est une feuille, alors la classe de $\vec{\xi}^{\mu}$ est donné par la classe associé à ce nœud, sinon :
   - SI $\sigma_t(\vec{\xi}^{\mu}) = -1$ alors $t = t_{-}$
   - SINON $t = t_{+}$
3. Aller à 2.

C.7 *Restricted Coulomb Energy (RCE)*

Cet algorithme a été crée par *Cooper et al.* [80].

Algorithm 13 Restricted Coulomb Energie

1. Présenter le premier exemple $\{\vec{\xi^1}, \tau^1\}$. Créer un neurone caché G_1 et un neurone de décision H_1, avec des paramètres $\vec{w}_1 = \vec{\xi^1}$ et rayon d'influence ρ_0.

La sortie de $G_1 = \begin{cases} 1 & \text{si } \sum_i \|\vec{\xi}_i^1 - \vec{w}_i^1\|^2 \leq \rho_0 \\ 0 & \text{autrement} \end{cases}$

Connecter le neurone H_1 à la sortie de G_1 par un poids de 1.

Sortie de $H_1 = \begin{cases} 1 & \text{si } \sum_i \vec{\xi_i} \geq 1 \Rightarrow \zeta = 1 \\ 0 & \text{autrement } \zeta = -1 \end{cases}$

2. Présenter un deuxième exemple $\{\vec{\xi^2}, \tau^2\}$. On va tomber sur l'une des quatre situations suivantes :

- a) $\vec{\xi^2} \in$ classe $\tau = +1$. On active la sortie de G_1, la sortie de H_1 vaut donc 1 : la classification est correcte, rien à faire.
- b) $\vec{\xi^2} \in$ classe $\tau = +1$ mais n'active pas le neurone G_1, donc $H_1 = 0$: l'exemple n'a pas été reconnu. Rajouter une unité cachée G_2, avec des paramètres $\vec{w}_2 = \xi^2$, rayon d'influence ρ_0. Connecter le neurone H_1 à la sortie de G_2 par un poids de 1.
- c) $\vec{\xi^2} \in$ classe $\tau = -1$, mais la sortie de H_1 vaut donc 0 : classe inconnue. Créer une unité G_2 et une autre H_2, avec des paramètres : $\vec{w}_2 = \xi^2$, Rayon d'influence ρ_0
  Connecter le neurone H_2 à la sortie de G_2 par un poids de 1.
- d) $\vec{\xi^2} \in$ classe $\tau = -1$, mais le réseau lui donne la classe $\tau = +1$. Il active le neurone G_1 et donc de H_1. Il faut créer une unité G_2 et une autre H_2 pour la nouvelle classe mais aussi limiter les rayons d'influence : $\vec{\xi^2}$ ne devra pas activer G_1 ni H_1, $\rho_1 \leq \|\vec{\xi^2} - \vec{w}_1\|$. $\rho_2 \leq \|\vec{\xi^2} - \vec{w}_2\|$.

3. Répéter les pas 1 et 2 autant de fois qui soient nécessaires pour bien classer tous les exemples.

Annexes D

Décomposition en biais et variance

Récemment il y a eu une grande activité sur le thème du biais et de la variance : [40, 34, 11, 25, 93, 58], mais il n'existe pas un consensus général. En grande partie parce qu'on ne peut pas traiter un problème d'approximation où la mesure d'erreur est continue, comme par exemple :

$$e(\vec{\xi}^{\mu}) = (\tau^{\mu} - \zeta^{\mu})^2 \tag{D.1}$$

comme un problème de classification, où la mesure d'erreur est une fonction de Heaviside (2.8). D'après la littérature, il est possible de séparer les définitions en celles [11, 25, 93, 58] qu'essaient de faire une décomposition de biais+variance à la façon (D.4) et l'approche due à Friedman [34], qui montre que le biais et la variance suivent une forte interaction de façon assez différente en classification qu'en approximation.

D.1 Cas de la régression

Geman *et al.* ont calculé le biais et la variance pour des problèmes d'approximation de fonctions. Soit $\mathcal{L}^{\alpha}$ un ensemble d'apprentissage où :

$$\tau^{\mu} = f(\vec{\xi}^{\mu}) + \eta^{\mu} \tag{D.2}$$

L'objectif est de construire un modèle de la fonction $f(\vec{\xi})$ en utilisant les exemples de l'ensemble $\mathcal{L}^{\alpha}$. Si on désigne par $\hat{f}(\vec{\xi}, \mathcal{L})$ l'estimation obtenue en entraînant un algorithme sur $\mathcal{L}^{\alpha}$, une estimation possible de $\hat{f}(\vec{\xi}, \mathcal{L})$ est l'erreur quadratique moyenne sur le point $\vec{\xi}$:

$$e(\vec{\xi}) = \langle (f(\vec{\xi}) - \hat{f}(\vec{\xi}, \mathcal{L}))^2 \rangle_{\mathcal{L}} \tag{D.3}$$

où $\langle \cdots \rangle_{\mathcal{L}}$ représente l'espérance sur l'ensemble $\mathcal{L}^\alpha$. Compte tenu du fait que la fonction f est inconnue, la valeur (D.3) ne peut pas être calculée directement.

Soient $\tau^\mu = f(\vec{\xi}^\mu)$ la vraie fonction à approximer, $\zeta^\mu = \hat{f}(\vec{\xi}^\mu, \mathcal{L})$ l'estimation obtenue en utilisant $\mathcal{L}^\alpha$, et $f(\vec{\xi}^\mu) = \langle \tau | \vec{\xi}^\mu \rangle$, le tout dépendant de $\vec{\xi}^\mu$ qu'on ne l'écrit plus. Alors, si on utilise la mesure d'erreur (D.3) moyennée sur plusieurs ensembles d'apprentissage $\mathcal{L}_i^\alpha$; $i = 1, \cdots, l$, on peut calculer :

$$\begin{aligned} \langle (f - \hat{f})^2 \rangle &= \langle (f - \langle \hat{f} \rangle + \langle \hat{f} \rangle - \hat{f})^2 \rangle \\ &= \langle (f - \langle \hat{f} \rangle)^2 \rangle + \langle (\langle \hat{f} \rangle - \hat{f})^2 \rangle + 2 \langle (f - \langle \hat{f} \rangle) \cdot (\langle \hat{f} \rangle - \hat{f}) \rangle \end{aligned}$$

alors, l'erreur d'approximation (D.3) sur le point $\vec{\xi}$ peut être décomposée comme :

$$\langle (f - \hat{f})^2 \rangle = \underbrace{\left[f - \langle \hat{f} \rangle \right]^2}_{(biais)^2} + \underbrace{\langle \left[\hat{f} - \langle \hat{f} \rangle \right]^2 \rangle}_{v = variance} \tag{D.4}$$

Le premier terme est le carré du *biais* et renseigne sur l'écart entre la valeur de la fonction f évaluée au point $\vec{\xi}$ et la valeur donnée par l'estimateur $\hat{f}$ moyennée sur tous les l ensembles d'apprentissage possibles. Le deuxième terme correspond à la *variance* v, qui mesure la sensibilité de l'estimateur $\hat{f}$ face au choix d'un ensemble d'apprentissage particulier $\mathcal{L}_i^\alpha$.

Comme on a du bruit η, si l'estimation fournie par $\hat{f}(\vec{\xi}^\mu)$ passe par tous les P points de l'ensemble $\mathcal{L}^\alpha$, on aura :

$$\langle \hat{f}(\vec{\xi}^\mu) \rangle = \langle f(\vec{\xi}^\mu) + \eta^\mu \rangle = f(\vec{\xi}^\mu) \tag{D.5}$$

$$\langle (\hat{f}(\vec{\xi}^\mu) - \langle f(\vec{\xi}^\mu) \rangle)^2 \rangle = \langle (f(\vec{\xi}^\mu) + \eta^\mu - f(\vec{\xi}^\mu))^2 \rangle = \langle (\eta^\mu)^2 \rangle \tag{D.6}$$

Le biais de $\hat{f}$ est nul et v égal à celle du bruit. Cette situation (ou interpolation) correspond à un *sur-apprentissage*, et il faut l'éviter. L'estimateur $\hat{f}$ approche mal la fonction cible f. Si on lisse l'estimateur $\hat{f}$ (ce qui correspond à éliminer certains degrés de liberté), ceci peut améliorer l'approximation, car on a réduit la variance. Cependant, si on le lisse trop, certains détails risquent d'être perdus, ce qui implique une augmentation du terme de biais, qui entraîne à son tour une augmentation de l'erreur. D'après l'équation (D.4), on voit que quand le biais et la variance sont tous deux nuls, $\hat{f} = f$. Ceci est la situation optimale, ou le réseau modélise parfaitement la fonction f. En réalité comme on ne dispose que d'un ensemble fini d'exemples on pourra faire seulement une bonne approximation, dont le biais et la variance seront petits. La question de comment trouver le meilleur compromis reste malgré tout ouverte.

D.2 Cas de la classification

Dans les problèmes de classification ($\tau = \pm 1$) l'expression (D.2) n'est pas tout à fait correcte, parce que le bruit η^μ pourrait ne pas être négligeable par rapport à τ^μ et changer la valeur de τ^μ à $-\tau^\mu$. Le bruit se présente ici plutôt dans les entrées $\vec{\xi}^\mu$.

Soit un problème de classification à deux classes $\tau = \pm 1$, et soit $\tau = f(\vec{\xi}) = \pm 1$, la *vraie* classe de $\vec{\xi}$. Soit $\zeta_i(\vec{\xi})$ la classe de $\vec{\xi}$ estimée avec l'ensemble d'apprentissage $\mathcal{L}^\alpha$, $i = 1, \cdots, l$. Soit :

$$f_+(\vec{\xi}) = \frac{1}{l}\sum_{i=1}^{l}\Theta(\zeta_i(\vec{\xi})) = \frac{1}{l}\sum_{i=1}^{l}\Theta(\zeta_i(\vec{\xi}) = 1) \tag{D.7}$$

la fraction des machines qui estiment que la classe de $\vec{\xi}$ est 1. Si on classe $\vec{\xi}$ avec un *comité de machines* [1] :

$$\hat{\zeta}(\vec{\xi}) = \begin{cases} +1 & \text{si } f_+(\vec{\xi}) > \frac{1}{2} \\ -1 & \text{autrement} \end{cases} \tag{D.9}$$

Alors, l'erreur de généralisation (erreur de prédiction) pour l'exemple $\vec{\xi}$ va être :

$$\varepsilon_g(\vec{\xi}) = \mathcal{P}(\tau = +1)\cdot\mathcal{P}(\hat{\zeta} = -1) + \mathcal{P}(\tau = -1)\cdot\mathcal{P}(\hat{\zeta} = +1) \tag{D.10}$$

Si on suppose que $f_+(\vec{\xi})$ est une variable gaussienne de moyenne $\overline{f_+}$ et de variance v_+^2 (la dépendance en $\vec{\xi}$ est omise), alors :

$$\mathcal{P}(f_+) = \frac{1}{\sqrt{2\pi}v_+}\exp\left\{-\frac{(f_+ - \overline{f_+})^2}{2v_+^2}\right\} \tag{D.11}$$

[1] Un *comité de machines* est un ensemble de L réseaux (classifieurs) $\hat{f}_l; l = 1, \cdots, L$. On prend la sortie du comité sur le point $\vec{\xi}$, $\hat{f}_{COM}(\vec{\xi})$ comme la moyenne des sorties des réseaux qui forment le comité, sur ce point :

$$\hat{f}_{COM}(\vec{\xi}) = \Theta\left(\frac{1}{L}\sum_{l=1}^{L}\hat{f}_l(\vec{\xi})\right) \tag{D.8}$$

Or, en utilisant (D.9) et (D.11), la probabilité que $\hat{\zeta}$ soit égal à -1 est :

$$\begin{aligned}
\mathcal{P}(\hat{\zeta}=-1) &= \mathcal{P}(f_+ < \frac{1}{2}) && \text{(D.12)}\\
&= \int_{-\infty}^{\frac{1}{2}} \mathcal{P}(f_+)df_+ && \text{(D.13)}\\
&= 1 - \int_{\frac{1}{2}}^{+\infty} \mathcal{P}(f_+)df_+ && \text{(D.14)}\\
&= 1 - \int_{\frac{1}{2}}^{+\infty} \frac{df_+}{\sqrt{2\pi\upsilon_+^2}} \exp\left\{-\frac{(f_+ - \overline{f_+})^2}{2\upsilon_+^2}\right\} && \text{(D.15)}
\end{aligned}$$

Si on prend : $u = (x - \overline{f_+})/\upsilon_+$ alors $du = dx/\upsilon_+$, et $u(x = \frac{1}{2}) = (\frac{1}{2} - \overline{f_+})/\upsilon_+$, d'où, on aura :

$$\mathcal{P}(\hat{\zeta}=-1) = 1 - \frac{1}{\sqrt{2\pi}} \int_{(\frac{1}{2}-\overline{f_+})/\upsilon_+}^{\infty} e^{-u^2/2}du \qquad \text{(D.16)}$$

Et finalement :

$$\begin{aligned}
\mathcal{P}(\hat{\zeta}=-1) &= 1 - \phi\left(\frac{\frac{1}{2} - \overline{f_+}}{\upsilon_+}\right)\\
&= \phi\left(-\frac{(\frac{1}{2} - \overline{f_+})}{\upsilon_+}\right) && \text{(D.17)}
\end{aligned}$$

en utilisant le fait que :

$$\phi(-u) = 1 - \phi(u) \qquad \text{(D.18)}$$

De façon analogue, et en suivant un raisonnement similaire, pour $\mathcal{P}(\hat{\zeta}=+1)$ on aura :

$$\begin{aligned}
\mathcal{P}(\hat{\zeta}=+1) &= p(f_+ > \frac{1}{2})\\
&= \phi\left(\frac{\frac{1}{2} - \overline{f_+}}{\upsilon_+}\right) && \text{(D.19)}
\end{aligned}$$

De (D.17) et (D.19), l'erreur de généralisation sur le point $\vec{\xi}$, est :

$$\begin{aligned}
\varepsilon_g(\vec{\xi}) &= p(\tau=1)\cdot\phi\left(-\frac{\frac{1}{2} - \overline{f_+}}{\upsilon_+}\right) + p(\tau=-1)\cdot\phi\left(\frac{\frac{1}{2} - \overline{f_+}}{\upsilon_+}\right) && \text{(D.20)}\\
&= \phi\left[\text{signe}(f - \frac{1}{2})\cdot\left(\frac{\overline{f_+} - \frac{1}{2}}{\upsilon_+}\right)\right] && \text{(D.21)}
\end{aligned}$$

f étant la *vrai* probabilité que $\tau = 1$.

Maintenant l'erreur de généralisation ε_g dépend de la vraie probabilité f et de $\overline{f_+}$ à travers :

$$\mathrm{B}(f, \overline{f_+}) = \mathrm{sign}(\frac{1}{2} - f)(\overline{f_+} - \frac{1}{2}) \tag{D.22}$$

Si $v_+^2 > 0$, l'erreur $\varepsilon_g(\vec{\xi})$ (D.21) est une fonction monotone et croissante de B (D.22), qu'on appellera le *biais à la frontière*. Alors, $\overline{f_+}$ et v_+^2 affectent la classification de façon très différente que dans les problèmes d'approximation. Etant donné v_+^2, l'erreur d'approximation (D.4) est proportionnelle à la distance $(f - \overline{f_+})^2$, tandis que, en classification, la dépendance est seulement à travers le signe de $(f - \frac{1}{2})$. A partir de cela, deux situations peuvent se présenter :

- Si B < 0 l'erreur de classification décroît si on augmente $|f - \frac{1}{2}|$ indépendamment de l'estimation du biais $(f - \overline{f_+})^2$.
- Si B > 0 l'erreur de classification augmente avec la distance entre $\overline{f_+}$ et $\frac{1}{2}$.

D'un autre côté, étant donné $\overline{f_+}$, pour les problèmes de régression, l'erreur (D.4) est proportionnelle à v_+^2, mais pour la classification l'effet de v_+^2 dépend surtout du signe de B :

- Si (B < 0) l'erreur de classification décroît si on décroît v_+^2 (mais non de façon linéaire).
- Si (B > 0) l'erreur de classification augmente si on *décroît* v_+^2

Ce qu'il faut retenir de cette discussion c'est que pour les problèmes de régression, de faibles variances v_+^2 ne donnent pas forcement de faibles ε_g, car le carré du biais peut être grand. En ce qui concerne la classification, $v_+^2 = 0$ correspond à avoir une classification optimale si $B < 0$, mais à une erreur maximale si $B > 0$. En approximation, la dépendance de l'erreur par rapport à $\overline{f_+}$ et v_+^2 est *additive*, tandis qu'en classification il y a une *forte interaction non linéaire* : l'influence du biais à la frontière peut être atténuée si v_+^2 est petite, et en même temps, l'influence de v_+^2 dépend du signe du biais. Pour la classification donc, $v_+^2 \to 0$ est important mais le carré du biais ne l'est pas [34] (on dit que la variance tend à dominer le biais) : il est suffisant que $\overline{f_+}$ et f soient du même côté par rapport à $\frac{1}{2}$ (étant donné que $B < 0$), et alors on peut réduire ε_g en réduisant *seulement* la variance v_+^2.

D'autres auteurs (Dietterich et Kong [25], Breiman [11], Tibshirani [93] et Kohavi et Wolpert [58]) essaient de faire une décomposition du biais+variance similaire à (D.4), mais de ceci résultent parfois des mesures de variance négatives [11, 25] ou des expressions qui ne permettent pas de faire une décomposition additive [93].

Bibliographie

[1] E.A.I. Alpaydin. *Neural models of supervised and unsupervised learning.* PhD thesis, EPFL 863, Lausanne, Suisse., 1990.

[2] J.A. Anderson and E. Rosenfeld, editors. *Neurocomputing: Foundations of Research.* MIT Press, Cambridge, 1988.

[3] A. Assoum, J.M. Torres-Moreno, N.E. Radi, R. Ecoffet, R. Velazco, and M.B Gordon. Robustness of ann hardware implementations : A case study. In *International Conference on Artificial Neural Networks*, pages 666–666, Paris, October 9-13, 1995.

[4] M.R. Berthold and J. Diamond. Boosting the performance of RBF networks with dynamic decay adjustment. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 521–528. The MIT Press, 1995.

[5] M. Berthold. A probabilistic extension for the DDA algorithm. In *IEEE International Conference on Neural Networks*, pages 341–346, Washington, 1996.

[6] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth and Brooks/Cole, Monterey, CA, 1984.

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

[8] C.M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford, England, 1995.

[9] M. Biehl and M. Opper. Tilinglike learning in the parity machine. *Physical Review A*, 44:6888, 1991.

[10] R. Baron, H. Paugam-Moisy, J.M. Torres-Moreno, and M.B. Gordon. Etude comparative de trois modèles connexionnistes sur un problème de classification. In O. Gascuel, editor, *Journées Acquisition, Apprentissage*, pages 275–287, Sète, FRANCE, 8-10 mai 1996.

[11] L. Breiman. Bias, variance and arcing classifiers. Technical Report 460, Department of Statistics. University of California at Berkeley, April, 1994.

[12] J. Bruske and G. Sommer. Dynamic cell structures. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 497–504. The MIT Press, 1995.

[13] D. Binder, E.C. Smith, and A.B. Holman. Satellite anomalies from galactic cosmic rays. *IEEE Tran. Nucl. Sc.*, NS 22(6):2675–2680, 1975.

[14] A. Buhot, J.-M. Torres Moreno, and M.B. Gordon. Numerical simulations of an optimal algorithm for supervised learning. In Michel Verleysen, editor, *European Symposium on Artificial Neural Networks*, pages 151–156, Brussels, 1997. D facto.

[15] A. Buhot, J.M. Torres Moreno, and M.B. Gordon. Finite size scaling of the bayesian perceptron. *Physical Review E*, 55:7434–7440, 1997.

[16] S. Canu. Des réseaux de neurones à la régression flexible. Diplôme DHDR. Université de Paris 6-Pierre et Marie Curie, Paris, France, 1996.

[17] M. Cottrell, B. Girard, I. Girard, and M. Mangeras. Time series and neural networks: a statistical method for weight elimination. In Michel Verleysen, editor, *European Symposium on Artificial Neural Networks*, pages 157–164, Brussels, 1993. D facto.

[18] R. Chentouf. *Construction de réseaux de neurones multicouches pour l'approximation.* PhD thesis, Institut National Polytechnique de Grenoble, Grenoble, France, 1997.

[19] T.M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, EC-14:326–334, 1965.

[20] B. Chakraborty and Y. Sawada. Fractal connection structure: Effect on generalization supervised feed-forward networks. In *IEEE International Conference on Neural Networks*, pages 264–269, Washington, 1996.

[21] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

[22] V. Danel, P. Cony, and V. Rialle. Coma due to psychotropes: how can multifactorial analysis help make diagnosis. In *Second Meeting of the European Association of Poison Centres and Clinical Toxicologists*, Lille, 6-9 Décembre 1995.

[23] J. Depenau. *Automated design of neural network architecture for classification.* PhD thesis, EF-448, DAIMI, Computer Science Department, Aarhus University., 1995.

[24] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis.* John Wiley, New York, 1973.

[25] T.G. Dietterich and E.B. Kong. Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Department of Computer Science. Oregon State University, 1995.

[26] V. Danel and V. Rialle. L'analyse factorielle, aide au diagnostic des comas toxiques. In *XXVe Congrès de la Société de la Réanimation de Langue Française*, Janvier 1997.

[27] J. Denker, D. Schwartz, B. Wittner, S. Solla, R. Howard, L. Jackel, and J. Hopfield. Large automatic learning, rule extraction, and generalization. *Complex Systems*, 1:877–922, 1987.

[28] S.E. Fahlman. Faster-learning variations on back-propagation: an empirical study. In D. Touretzky, G.E. Hinton, and T.J. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 38–51, San Mateo, CA, 1988. Morgan Kaufman.

[29] S.E. Fahlman and C. Lebiere. The Cascade-Correlation learning architecture. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532, San Mateo, 1990. (Denver 1989), Morgan Kaufmann.

[30] K.R. Farrell and R.J. Mammone. Speaker recognition using neural tree networks. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 1035–1042. Morgan Kaufmann Publishers, Inc., 1994.

[31] M. Frean. The Upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, 2(2):198–209, 1990.

[32] M. Frean. A "thermal" perceptron learning rule. *Neural Computation*, 4(6):946–957, 1992.

[33] Bernd Fritzke. Supervised learning with growing cell structures. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 255–262. Morgan Kaufmann Publishers, Inc., 1994.

[34] J.H Friedman. On bias, variance, 0/1 - loss, and the curse-of-dimensionality. Technical report, Department of Statistics. Stanford University, 1996.

[35] S.I. Gallant. Optimal linear discriminants. In *Proc. 8th. Conf. Pattern Recognition, Oct. 28-31, Paris*, volume 4, pages 849–852, 1986.

[36] S.I. Gallant. Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1:179–191, 1990.

[37] E. Gardner. Maximum storage capacity in neural networks. *Europhysics Letters*, 4:481–485, 1987.

[38] O. Gascuel. Symenu. collective paper (gascuel o. coordinator). Technical report, 5èmes Journées Nationales du PRC-IA Teknea, (Nancy), 1995.

[39] M.B. Gordon and D. Berchier. Minimerror: A perceptron learning rule that finds the optimal weights. In Michel Verleysen, editor, *European Symposium on Artificial Neural Networks*, pages 105–110, Brussels, 1993. D facto.

[40] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the Bias/Variance dilemma. *Neural Computation*, 4(1):1–58, 1992.

[41] E. Gardner and B. Derrida. Optimal storage properties of neural network models. *Journal of Physics A*, 21:271–284, 1988.

[42] M.B. Gordon and D. Grempel. Optimal learning with a temperature dependent algorithm. *Europhysics Letters*, 29(3):257–262, 1995.

[43] M. Golea and M. Marchand. A growth algorithm for neural network decision trees. *Europhysics Letters*, 12(3):205–210, 1990.

[44] M.B. Gordon. A convergence theorem for incremental learning with real-valued inputs. In *IEEE International Conference on Neural Networks*, pages 381–386, Washington, 1996.

[45] M.B. Gordon, P. Peretto, and D. Berchier. Learning algorithms for perceptrons from statistical physics. *Journal of Physics I France*, 3:377–387, 1993.

[46] R.P. Gorman and T.J. Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1:75–89, 1988.

[47] M. Hoehfeld and S. Fahlman. Learning with limited numerical precision using the cascade correlation algorithm. Technical Report CMU-CS-91-130, Carnegie Mellon University, 1991.

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

[49] J. Hertz, A. Krogh, and G. Palmer. *Introduction to the theorie of Neural Computation.* Addison Wesley, Redwood City, CA, 1991.

[50] B. Hassibi and D.G. Stork. Second order derivatives for network pruning: Optimal Brain Surgeon. In Stephen José Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 164–171. Morgan Kaufmann, San Mateo, CA, 1993.

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

[52] C. Jutten and O. Fambon. Pruning methods: a review. In Michel Verleysen, editor, *European Symposium on Artificial Neural Networks*, pages 365–370, Brussels, 1995. D facto.

[53] C. Jutten. Apprntissage dans les réseaux neuronaux à architecture évolutive. In Marie Cottrell and Jean-Pierre Rospars, editors,

Huitièmes Journées Neurosciences et Sciences de l'Ingénieur, pages 185–188, Marly-le-Roi, FRANCE, Mai 6-9, 1996.

[54] M. Karouia, R. Lengellé, and Denoeux T. Performance analysis of a MLP weight initialization algorithm. In Michel Verleysen, editor, *European Symposium on Artificial Neural Networks*, pages 347–352, Brussels, 1995. D facto.

[55] W. Krauth and M. Mézard. Learning algorithms with optimal stability in neural networks. *Journal of Physics A*, 20:L745–L752, 1987.

[56] W. Krauth, J.P. Nadal, and M. Mézard. The roles of stability and symmetry in the dynamics of neural networks. *Journal of Physics A*, 21:2995–3011, 1988.

[57] S. Knerr, L. Personnaz, and G. Dreyfus. Single-layer learning revisited: a stepwise procedure for building and training a neural network. In *Neurocomputing, Algorithms, Architectures and Applications*. Elsevier, 1990.

[58] R. Kohavi and D.H. Wolpert. Bias plus Variance decomposition for zero-one loss functions. Technical report, Department of Computer Science. Stanford University, 1996.

[59] C. Kwon. Exact asymptotic estimates of the storage capacities of the committe machines with overlapping and non-overlapping receptive fields. In Michel Verleysen, editor, *European Symposium on Artificial Neural Networks*, pages 157–162, Brussels, 1997. D facto.

[60] Y. LeCun, B. Boser, J. S. Denker, B. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[61] Y. Le Cun. Une procédure d'apprentissage pour réseau à seuil asymétrique. In *Cognitiva 85: A la Frontière de l'Intelligence Artificielle des Sciences de la Connaissance des Neurosciences*, pages 599–604, Paris, 1985. (Paris 1985), CESTA.

[62] Y. Le Cun, J.S. Denker, and S.A. Solla. Optimal brain damage. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 598–605, San Mateo, 1990. (Denver 1989), Morgan Kaufmann.

[63] G.J. Mitchison and R.M. Durbin. Bounds on the learning capacity of some multi-layer networks. *Biological Cybernetics*, 60:345–356, 1989.

[64] D. Martinez and D. Estève. The Offset algorithm: building and learning method for multilayer neural networks. *Europhysics Letters*, 18:95–100, 1992.

[65] M. Marchand, M. Golea, and P. Ruján. A convergence theorem for sequential learning in two-layer perceptrons. *Europhysics Letters*, 11:487–492, 1990.

[66] C.J. Merz and P.M. Murphy. Uci repository of machine learning databases [http://www.ics.uci.edu/ mlearn/mlrepository.html]. Technical report, University of California, Department of Information and Computer Science, 1996.

[67] K. Mehrotra, C.K Mohan, and S. Ranka. *Elements of Artificial Neural Networks*. The MIT Press, Cambridge, Mass., 1997.

[68] M. Mézard and J.-P. Nadal. Learning in feedforward layered networks: the tiling algorithm. *J. Phys. A: Math. and Gen.*, 22:2191–203, 1989.

[69] W.S. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, 1943. Reprinted in [2].

[70] M.L. Minsky and S.A. Papert. *Perceptrons*. MIT Press, Cambridge, 1969. Partially reprinted in [2].

[71] S. Mukhopadhyay, A. Roy, L. S. Kim, and S. Govil. A polynomial time algorithm for generating neural networks for pattern classification: Its stability properties and some test results. *Neural Computation*, 5(2):317–330, 1993.

[72] J.-P. Nadal. Study of a growth algorithm for a feedforward neural network. *Int. Journ. of Neur. Syst.*, 1:55–9, 1989.

[73] P. Peretto. *An introduction to the Modeling of Neural Networks*. Cambridge, University Press, Cambridge, Mass., 1992.

[74] Perazzo. Craking in steel tubes, 1995. Communication privé.

[75] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T Vetterling. *Numerical Recipes*. Cambridge University Press, Cambridge, England, 1986.

[76] T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, 1990.

[77] D. A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[78] L. Prechelt. PROBEN1 - A set of benchmarks and benchmarking rules for neural network training algorithms. Technical Report 21/94, University of Karlsruhe, Faculty of Informatics, September 1994.

[79] S. Raymond. The space radiation environment for electronics, 1988. *IEEE Tran. Nucl. Sc.* NS 76.

[80] D.E Reilly, L.N. Cooper, and C. Elbaum. A neural model for category learning. *Biological Cybernetics*, 45:35–41, 1982.

[81] B. Raffin and M.B. Gordon. Learning and generalization with Minimerror, a temperature dependent learning algorithm. *Neural Computation*, 7(6):1206–1224, 1995.

[82] A. Roy, S. Govil, and R. Miranda. An algorithm to generate radial basis function (RBF)-like nets for classification problems. *Neural Networks*, 8(2):179–201, 1995.

[83] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. Reprinted in [2].

[84] A. Roy, L. Kim, and S. Mukhopadhyay. A polynomial time algorithm for the construction and training of a class of multilayer perceptron. *Neural Networks*, 6(1):535–545, 1993.

[85] F. Rosenblatt. *Principles of Neurodynamics*. Spartan, New York, 1962.

[86] P. Rujan. A fast method for calculating the perceptron with maximal stability. *Journal of Physics I France*, 3:277–290, 1993.

[87] J. A. Sirat and J.-P. Nadal. Neural trees: a new tool for classification. *NETWORK*, 1:423–38, 1990.

[88] S.A. Solla. Learning and generalization in layered neural networks: The contiguity problem. In L. Personnaz and G. Dreyfus, editors, *Neural Networks from Models to Applications*, pages 168–177, Paris, 1989. (Paris 1988), I.D.S.E.T.

[89] J. Sejnowski, T. and C.R. Rosenberg. Parallel networks that learn to pronunce english text. *Complex Systems*, 1(1):145–168, 1987.

[90] McGarrity Srour. Radiation effects on microelectronics in space, 1988. *IEEE Tran. Nucl. Sc.*. NS 76.

[91] Y. Shang and B.W. Wha. A global optimization method for neural networks training. In *IEEE International Conference on Neural Networks*, pages 7–11, Washington, 1996.

[92] S.B. Trhun and 23 co authors. The Monk's problems. A performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, 1991.

[93] R. Tibshirani. Bias, variance and prediction error for classification rules. Technical report, Department of Preventive Medicine and Biostatistics and Departament of Statitistics. University of Toronto, 1996.

[94] J.M. Torres Moreno. Monoplan: un réseau constructiviste avec la règle Minimerror. Rapport DEA. Institut National Polytechnique de Grenoble, 1997.

[95] J.-M. Torres Moreno and M. Gordon. An evolutive architecture coupled with optimal perceptron learning for classification. In Michel Verleysen, editor, *European Symposium on Artificial Neural Networks*, pages 365–370, Brussels, 1995. D facto.

[96] J.M. Torres-Moreno and M.B. Gordon. Classification par apprentissage : une étude comparative. In O. Gascuel, editor, *Journées Acquisition, Apprentissage*, pages 338–341, Sète, FRANCE, 8-10 mai 1996.

[97] J.M. Torres Moreno and M.B. Gordon. Application de l'algorithme incrémental Monoplan à deux problèmes de classification. In Marie Cottrell and Jean-Pierre Rospars, editors, *Huitièmes Journées Neurosciences et Sciences de l'Ingénieur*, pages 185–188, Marly-le-Roi, FRANCE, Mai 6-9, 1996.

[98] J.M. Torres Moreno and M.B Gordon. Characterization complete of the sonar benchmark. Neural Processing Letters, 1:1–4. In press, 1998.

[99] J.M. Torres Moreno and M.B Gordon. Efficient adaptive learning for classification tasks with binary units. Neural Computation, In Press, 1997.

[100] V.N. Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, Berlin, 1982.

[101] V. Vapnik. Principles of risk minimization for learning theory. In John E. Moody, Steve J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 831–838. Morgan Kaufmann Publishers, Inc., 1992.

[102] V. L. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, New York, 1995.

[103] V.N. Vapnik and A.Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16:264–280, 1971.

[104] K.B. Verma and J.J. Mulawka. A new algorithm for feedforward neural networks. In Michel Verleysen, editor, *European Symposium on Artificial Neural Networks*, pages 359–364, Brussels, 1995. D facto.

[105] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, 1974.

[106] W.H. Wolberg and O.L. Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. In *Proceedings of the National Academy of Sciences, USA*, volume 87, pages 9193–9196, 1990.

[107] A.S. Weigend, D.E. Rumelhart, and B.A. Huberman. Generalization by weight-elimination with application to forecasting. In Richard P. Lippmann, John E. Moody, and David S. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 3, pages 875–882. Morgan Kaufmann Publishers, Inc., 1991.