Statistiques et réseaux de neurones pour un système de diagnostic : application au diagnostic de pannes automobiles

Hervé Poulard

Année 1996

Thèse

Présentée au Laboratoire d'Analyse et
d'Architecture des Systèmes du CNRS
en vue de l'obtention du Titre de
Docteur de l'Université Paul Sabatier de Toulouse
Spécialité : ***Intelligence Artificielle***

par Hervé POULARD
Ingénieur ENSICA

STATISTIQUES ET RÉSEAUX DE NEURONES POUR UN SYSTÈME DE DIAGNOSTIC : APPLICATION AU DIAGNOSTIC DE PANNES AUTOMOBILES

Soutenue le Jeudi 9 Mai 1996 devant le jury :

A. Titli	**Président**
B. Dubuisson	**Rapporteur**
M. Weinfeld	**Rapporteur**
D. Estève	**Directeur de thèse**
M. Samuelides	**Examinateur**
G. Zwingelstein	**Examinateur**
H. Caussinus	**Examinateur**
G. Peltier	**Examinateur**

Rapport LAAS N° 96177
Thèse préparée au LAAS-CNRS
7, avenue du Colonel Roche
31077 Toulouse Cédex (France)

A ma mère, qui aurait tant aimé faire de longues études mais qui n'a pas pu,
A mon père, qui m'a permis de faire de très longues études,
A son amour des choses bien faites,
A mon grand frère, A mes petites soeurs jumelles,
A ceux qui m'aiment, A ceux que j'aime,
A toi Sophie.

Remerciements

Le travail présenté dans ce mémoire a été effectué dans le cadre d'une convention CIFRE au sein de la société ACTIA et du Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS) du CNRS, dans le groupe Microstructures silicium et Microsystèmes Intégrés (M2I). Je remercie à ce titre Monsieur Alain COSTES directeur du LAAS de m'avoir accueilli dans son établissement et Monsieur Augustin MARTINEZ qui dirige actuellement le groupe M2I.

J'exprime toute ma gratitude à mon directeur de recherche Monsieur Daniel ESTÈVE, directeur adjoint du LAAS, de m'avoir témoigné toute sa confiance dès notre première rencontre et de m'avoir accueilli dans sa naissante équipe de recherche sur les réseaux de neurones. Durant mes travaux de DEA et de thèse, j'ai eu la chance immense de travailler avec un directeur de thèse qui fut toujours à l'écoute, toujours disponible malgré ses intenses activités, et qui, certainement, a su me transmettre sa passion de la recherche. En le remerciant de tous les conseils avisés, de toutes les idées fortes et de toutes les orientations scientifiques judicieuses qu'il m'a donnés, je tiens à lui exprimer ma profonde reconnaissance et toute mon estime.

Je remercie Monsieur Guy PELTIER, directeur de la société ACTIA, de s'être engagé dans des activités de recherche fondamentale, démarche souvent difficile pour une PME, de m'avoir accueilli dans sa société et de m'avoir donné tous les moyens de mener à bien les objectifs que nous nous étions fixés. Je le remercie également pour l'intérêt qu'il a témoigné envers mes travaux et pour sa participation à mon jury de thèse.

Je tiens à remercier Monsieur André TITLI, Professeur à l'Institut National des Sciences Appliquées de Toulouse, d'avoir accepté la présidence de mon jury, de nos diverses discussions et de ses conseils.

Monsieur Bernard DUBUISSON, Professeur à l'Université de Technologie de Compiègne, m'a fait un grand honneur en acceptant de prendre sur son temps précieux pour juger mon travail en tant que rapporteur. Grand nom du diagnostic par reconnaissance de formes, les concepts qu'il a développés imprègnent mon travail. Je lui témoigne tout mon respect et ma reconnaissance.

Je suis particulièrement reconnaissant à Monsieur Michel WEINFELD, Directeur de recherche à l'Ecole Polytechnique, pour le professionnalisme et la minutie avec lesquels il a relu et jugé mon manuscrit. La multitude de critiques constructives qu'il a faites, m'a permis d'améliorer de manière substantielle la qualité de cette thèse.

Je remercie Monsieur Manuel SAMUELIDES, Professeur à l'Ecole Nationale Supérieure de l'Aéronautique et de l'Espace d'avoir étudié mes travaux en tant qu'examinateur, et de m'avoir donné de son temps pour discuter sur mes développements "hypercubiques".

Spécialiste des techniques de supervision et de diagnostic, Monsieur Gilles ZWINGELSTEIN, Directeur de recherche associé au CNRS, me fait un grand honneur en acceptant d'être examinateur. Je lui en suis très reconnaissant.

Je suis très honoré de la présence dans mon jury de Monsieur Henri CAUSSINUS, Professeur à l'Univer-

sité Paul Sabatier. Lors de nos rencontres, nous avons particulièrement abordé les aspects géométriques et statistiques développés dans ma thèse, et ses remarques m'ont été fort utiles pour affiner certaines parties de mon manuscrit.

Monsieur Saïd LABRECHE, Docteur de l'Université Paul Sabatier, a travaillé avec moi sur ce projet pendant 8 mois. Grâce à une totale compatibilité de caractères, une complémentarité des connaissances et de nombreux intérêts communs, c'est une des premières fois où l'expression *synergie des compétences* a pris pour moi tout son sens, "$1 + 1 = 3$", disait un intervenant lors d'un cours de gestion auquel j'ai assisté. Animés d'une cohésion et d'un esprit d'équipe inébranlables, nous avons réalisé un travail important. J'ai beaucoup appris avec Saïd dans des domaines qui ne m'étaient pas familiers. Cette thèse ne serait pas ce qu'elle est sans son apport. J'estime aussi qu'il a apporté beaucoup à l'ensemble de l'équipe de recherche sur les réseaux de neurones. Outre ces aspects purement professionnels, Saïd est bien entendu pour moi un grand ami. Pour toutes ces raisons, je lui suis infiniment reconnaissant. J'adresse mille excuses à sa femme Anne-Marie et à son petit Alexandre pour les nocturnes improvisées.

Depuis le début de mon DEA jusqu'à aujourd'hui, il m'a été particulièrement agréable de travailler avec Monsieur Dominique MARTINEZ, Chargé de recherche au LAAS. En constante communication, même lors de ses deux séjours au USA grâce au miracle d'Internet, j'ai pu profiter de son expérience et de ses conseils. Notre voyage en Angleterre dans le cadre du projet DRIVE-DETER a été très sympathique, malgré la charge de travail à réaliser en peu de temps. Je le remercie aussi pour tout le temps qu'il a passé à relire mes papiers et pour nos multiples discussions.

Au sein de la société ACTIA, j'adresse mes remerciements à :

- Messieurs Philippe GUYARD et Didier CALMELS respectivement ancien et nouveau directeur du bureau d'études de la société ACTIA, et à Monsieur Jean-Claude FONTET, Ingénieur, pour leur accueil chaleureux dans une équipe jeune, dynamique et sympathique, pour la confiance qu'ils ont témoignée envers un "savant" aux idées quelques fois "fumeuses" et pour l'intérêt qu'ils ont montré en suivant de très près l'évolution de mes travaux.

- Monsieur Laurent MALBERTI, Ingénieur, qui m'a beaucoup aidé au début de mes travaux en développant la version modifiée du Testeur Universel constituant la centrale d'acquisition du système de diagnostic et en mettant au point le protocole de communication. J'ai été impressionné par sa rapidité, son professionnalisme et sa patience. Il s'est beaucoup intéressé à mes développements et je lui en suis reconnaissant.

- toute l'équipe du bureau d'études : Robin, Jacquot, Gilles, Marc, Fabi, Pascal, Alain, Daniel, Françoise, Nadine, Didier, Bernard, Jojo... pour leur sympathie. Toutes mes excuses à ceux que j'oublie et à ceux dont le prénom m'échappe.

Au LAAS, j'exprime ma reconnaissance à :

- toute l'équipe neurone : Marie, Thierry, Michel, Armando, Bruno... avec qui j'ai travaillé quotidiennement. Un remerciement particulier à Neil pour nos travaux communs et nos nombreuses discussions et à Amine pour son aide dans l'étude de l'intégration VLSI.

- tous les autres que j'ai pu côtoyer tous les jours : Amal, Adama, Noureddine, Rachid, Salah, Stéphane, Zouhaïr... Un remerciement particulier à Zakaria pour nos diverses sorties nocturnes, sa bonne humeur de tous les jours et la gentillesse de sa famille qui m'a accueilli au Maroc.

- nos secrétaires devouées et aux personnels des services administratif, informatique, documentation-édition, du magasin, pour leur constante disponibilité et leur gentillesse.

- tous les autres que je n'ai pas cités et que j'ai pu fréquenter dans ce grand laboratoire et dont la liste serait trop longue.

Merci à toi, Sophie, pour m'avoir soutenu et supporté dans des moments parfois difficiles, et pour le temps très important passé à corriger ma prose.

Durant ces trois années, j'ai alterné entre deux mondes : celui de l'industrie et celui de la recherche. Du fait des réalités économiques, le milieu industriel impose de penser en termes de produit, il faut rester dans le domaine du *réalisable*. De plus les expériences sur le terrain révèlent la véritable complexité de la réalité et l'existence de paramètres non pris en compte dans les développements théoriques. Ce qui conduit souvent à reformuler les problèmes, à affiner les modèles. Quant à lui, le milieu de la recherche fait travailler la rigueur et l'imagination. Etre au confluent de ces deux mondes, en travaillant sur un sujet particulièrement pluridisciplinaire, a été une expérience forte et enrichissante. Bien que la dernière année fut épuisante, ce travail de thèse a été passionnant.

Table des matières

Introduction 5

1 Méthodes de diagnostic et définition des objectifs 9
- 1.1 Introduction . . . 9
- 1.2 Les méthodes de diagnostic de systèmes : état de l'art . . . 10
  - 1.2.1 Les méthodes symboliques de diagnostic . . . 11
  - 1.2.2 Les méthodes internes de diagnostic . . . 14
  - 1.2.3 Les méthodes externes de diagnostic . . . 15
  - 1.2.4 Diagnostic et réseaux de neurones . . . 23
- 1.3 Le diagnostic de pannes automobiles . . . 24
- 1.4 Objectifs de l'étude . . . 25

2 Développement de la maquette ANDI 27
- 2.1 Introduction . . . 27
- 2.2 Description générale de la maquette . . . 28
  - 2.2.1 Le système vu de l'extérieur . . . 28
  - 2.2.2 La partie matérielle . . . 29
  - 2.2.3 Les données captées & le protocole . . . 30
  - 2.2.4 Environnement logiciel . . . 31
- 2.3 Description du logiciel ANDI . . . 32
  - 2.3.1 Introduction . . . 32
  - 2.3.2 Le générateur de données . . . 33
  - 2.3.3 Les outils d'analyse . . . 37
  - 2.3.4 Les outils de diagnostic . . . 38
  - 2.3.5 Les réseaux de neurones dans ANDI . . . 43
- 2.4 Les essais réalisés . . . 45
  - 2.4.1 Motronic, Lada & ZX . . . 45
  - 2.4.2 Peugeot 605V6 . . . 45
- 2.5 Conclusion . . . 48
- 2.6 Quelques aspects du logiciel ANDI . . . 50
  - 2.6.1 Oscilloscope . . . 50
  - 2.6.2 Analyse temporelle de signaux . . . 50
  - 2.6.3 Tracé de densité de probabilité . . . 50
  - 2.6.4 Outil d'analyse des défauts élémentaires . . . 50
  - 2.6.5 Outil de diagnostic avec identification des pannes . . . 51

3 Apprentissage des réseaux de neurones à couches **63**
3.1 Introduction . . . 63
3.2 L'origine biologique et le premier modèle de neurone . . . 64
3.2.1 Le neurone biologique . . . 64
3.2.2 Le neurone formel . . . 65
3.2.3 Le règle de Hebb . . . 66
3.3 Du Perceptron à la rétropropagation . . . 67
3.3.1 Le Perceptron . . . 67
3.3.2 Adaline et Madaline . . . 69
3.3.3 La rétropropagation . . . 71
3.4 Les méthodes de simplification de réseau . . . 76
3.5 Les algorithmes de construction . . . 77
3.5.1 Généralités . . . 77
3.5.2 Les extensions du Perceptron pour minimiser le nombre d'erreurs . . . 79
3.5.3 Deux algorithmes de construction . . . 82
3.6 Conclusion . . . 86

4 Une étude de l'hypercube **89**
4.1 Introduction . . . 89
4.2 Motivations . . . 89
4.3 Définitions . . . 90
4.4 Stratification de l'hypercube . . . 93
4.5 Caractérisation des parties linéairement séparables . . . 96
4.6 Automorphismes affines et simulations . . . 99
4.7 Conclusion . . . 103

5 BCP : une méthode efficace d'apprentissage d'une unité **105**
5.1 Introduction . . . 105
5.2 L'algorithme général . . . 106
5.2.1 Rappels et notations . . . 106
5.2.2 Description de haut niveau de l'algorithme général . . . 107
5.2.3 Quelques résultats théoriques . . . 110
5.3 Les diverses méthodes . . . 111
5.3.1 BCP Prouvé . . . 112
5.3.2 BCP Heuristique . . . 116
5.3.3 Minimisation du nombre d'exemples mal classés : l'algorithme BCP Min . . . 118
5.3.4 Maximisation du nombre d'exclus : l'algorithme BCP Max . . . 121
5.4 Simulations pour apprendre une unité sur des problèmes LS . . . 122
5.4.1 Fonctions booléennes complètes LS . . . 122
5.4.2 Problèmes analogiques . . . 123
5.4.3 Justification de la convergence du BCP Heuristique . . . 124
5.5 Simulations pour apprendre une unité sur des problèmes NLS . . . 125
5.5.1 BCP Prouvé sur des fonctions booléennes aléatoires . . . 125
5.5.2 BCP Prouvé sur des distributions gaussiennes avec chevauchement . . . 126
5.5.3 Comparaison de l'évolution de $||w^t||$ pour des problèmes NLS . . . 127
5.5.4 BCP Max sur des fonctions booléennes aléatoires . . . 127
5.5.5 BCP Max sur deux distributions gaussiennes . . . 128
5.5.6 BCP Min sur des fonctions booléennes aléatoires . . . 129

5.5.7 BCP Min sur des problèmes analogiques 130
5.5.8 BCP Min sur le problème deux spirales 131
5.5.9 Comparaison des temps de calcul 132
5.6 Le BCP et les algorithmes de construction : simulations 132
5.6.1 Fonctions booléennes aléatoires en dimension 6 133
5.6.2 Problème symétrie miroir 134
5.6.3 Problème de détection de décalage 134
5.6.4 Problème de parité en dimension 8 135
5.7 Amélioration du BCPSL : l'algorithme BCPSLhd 136
5.7.1 BCPSL et BCPSLml : leurs limitations 136
5.7.2 La méthode 137
5.7.3 Simulations 138
5.8 MOBCPSL : un algorithme de construction multisorties 140
5.8.1 Principes 140
5.8.2 BCP Max à cible fixe 140
5.8.3 L'algorithme 141
5.8.4 Amélioration du MOBCPSL : l'algorithme MOBCPSLhd 144
5.8.5 Simulations 146
5.9 Vers une implémentation VLSI 148
5.9.1 Implémentation des réseaux fournis par le MOBCPSLhd 148
5.9.2 Implémentation de l'algorithme d'apprentissage 153
5.9.3 Conclusion sur l'implémentation 153
5.10 Conclusion 154

6 Retour au diagnostic : la méthodologie STANDING **159**
6.1 Introduction 159
6.2 La méthodologie STANDING 160
6.2.1 Principes 160
6.2.2 Contraintes 161
6.2.3 Choix des composantes et implantation du système 162
6.3 Les méthodes statistiques 165
6.3.1 Généralités 165
6.3.2 Analyse en Composantes Principales 166
6.3.3 Estimation de densité de probabilité : algorithmes EM et SEM 167
6.3.4 Filtrage des points extrêmes 169
6.3.5 Alternance des deux algorithmes EM et SEM 170
6.3.6 Remarques sur l'initialisation de l'association EM - SEM 171
6.4 Génération automatique des réseaux de neurones 173
6.4.1 Généralités 173
6.4.2 Les réseaux de neurones gaussiens 174
6.4.3 Les réseaux de neurones binaires 175
6.5 Organe d'identification 182
6.5.1 Généralités 182
6.5.2 Calcul des défautgrammes 182
6.5.3 ACP des défautgrammes et identifieurs 183
6.6 Conclusion et perspectives 184
6.7 Résultats de simulations 186

Conclusion générale **205**

Bibliographie **209**

Annexe A : Théorème de convergence du BCP **221**

A.1 Variations des barycentres et du vecteur w 221
A.2 $\mathcal{F}_t$ est une bijection, calcul de $\mathcal{F}_t^{-1}$ 222
A.3 Majoration de $\Delta^t(x^t)$ 223
A.4 Minimisation du majorant de $\Delta^t(x^t)$ dans U_t', méthode AL_0 223
A.5 Prise en compte du paramètre de relaxation 224
A.6 Démonstration du théorème de convergence avec paramètre de relaxation variable . . 225

Les savants sont optimistes parce que leur passion leur donne des joies fréquentes en leur épargnant les chagrins ; ils ne désespèrent pas de ne jamais trouver la vérité et ils s'en consolent aisément puisqu'ils ne sont jamais privés du plaisir de la recherche.

La plupart d'entre eux restent jeunes de coeur. Peut-être n'ont-ils pas été aussi jeunes que d'autres, mais ils l'ont été plus longtemps. Leur naïveté même, qui éclate à tous les yeux, est un signe de jeunesse. C'est sans doute que le chagrin seul vieillit et leur passion n'engendre que des joies sans douleur.

HENRI POINCARÉ

Introduction

Si un terme peut caractériser l'aube du troisième millénaire, d'un point de vue scientifique et technologique, c'est *la maîtrise de la complexité*. Cette notion de complexité surgit de toutes parts et inonde notre environnement naturel ou artificiel. On la redécouvre ou on la crée.

Avec le développement de la théorie du chaos, de plus en plus de systèmes qui pouvaient apparaître bien modélisés, se révèlent être d'une extrême complexité car complètement imprévibles à long terme, chaotiques. La nature aujourd'hui mieux modélisée que jamais, se révèle toujours plus complexe.

Du côté des réalisations humaines, grâce aux bouleversements apportés par l'ère électronique, et la foudroyante évolution de l'informatique, nos sociétés technologiques engendrent des systèmes de plus en plus performants, rapides, optimisés, automatisés et ainsi de plus en plus complexes. Mais parallèlement à cette complexité grandissante et au nombre des systèmes ainsi qualifiés, il y a un domaine qu'il est fondamental de faire évoluer, c'est celui de la sécurité. Premièrement, il est impératif d'accroître les normes de sécurité, particulièrement dans des domaines critiques comme le nucléaire, l'automobile, et l'aéronautique.... Deuxièmement cet impératif de sécurité, lié aux enjeux économiques en cas d'accident ou de panne, impose une grande maîtrise des techniques de surveillance, de diagnostic de pannes et de diagnostic préventif.

Dans cette perspective, on peut distinguer deux types de systèmes : ceux pour lesquels les fautes ne sont pas tolérées et les autres. Pendant longtemps, la sûreté de fonctionnement était un domaine privilégié des secteurs de pointe tels que l'aéronautique, le nucléaire, le spatial, pour des raisons évidentes de sécurité et de coût en cas d'accident. Dans de tels secteurs, les erreurs ne sont pas admissibles, et les contraintes de sécurité sont prioritaires dans les dépenses de développement. Différentes techniques ont été étudiées pour accroître la fiabilité des systèmes et notamment les techniques de redondance (matérielle et logicielle pour les systèmes informatisés). Mais l'assurance de la défaillance-nulle coûte évidemment très cher et est restreinte à peu de domaines très critiques. Dans la majorité des systèmes, où la fiabilité coûte relativement cher, il faut quand même savoir maîtriser les défaillances : détecter les pannes et leurs origines, et agir en conséquence. C'est notre champ privilégié de travail.

Le diagnostic de systèmes a longtemps été considéré comme faisant partie des tâches de dépannage, et a souvent été traité par des experts du domaine qui résolvaient les problèmes grâce à l'expérience acquise, avec une documentation comme seul outil d'aide au diagnostic. Mais du fait de la croissance de

la complexité des systèmes mis en jeu, du principe de disponibilité maximale et de la compétitivité, la sécurité de fonctionnement et le diagnostic sont devenus des techniques très importantes et s'insèrent dans toute la chaîne de production d'un produit, de la conception à la maintenance. Dès la conception, de plus en plus de systèmes intègrent des fonctionnalités de diagnostic interne : les machines s'autodiagnostiquent. Elles ne sont donc plus constituées uniquement d'organes propres à la fonction devant être réalisée, mais elles comportent aussi un système de surveillance, de contrôle, de diagnostic... Quand les fonctions d'autodiagnostic sont inexistantes pour des raisons de coût, ou ne sont pas suffisantes pour déterminer la panne, c'est au niveau de la maintenance que des outils d'aide au diagnostic sont développés.

Ces quinze dernières années, l'automobile a été marquée par *l'introduction massive de l'électronique* à de multiples niveaux : allumage, injection, ABS... Le nombre de calculateurs par véhicule, déjà important sur les véhicules haut de gamme, est encore en croissance. Cette évolution n'est pas terminée. La longueur totale des câbles et connexions diverses commence à se chiffrer en kilomètres. L'intégration des réseaux VAN ou CAN, véritable réseau informatique au sein de la voiture, devrait permettre de limiter la complexité du câblage. Mais une telle croissance de la complexité des systèmes, si elle apporte de fabuleux progrès, rend la tâche *du diagnostic de pannes* de plus en plus difficile.
Il est nécessaire de développer des outils de plus en plus puissants pour connaître l'origine d'une panne et réparer le véhicule. La société ACTIA développe et commercialise de tels produits. Mais le développement de ces outils lui aussi devient de plus en plus complexe. La diversité des véhicules, des calculateurs et des systèmes à diagnostiquer rend le développement de nouveaux produits de plus en plus difficile et cher. Cette évolution a donc incité la société ACTIA à investir dans la recherche de nouvelles méthodologies répondant à plusieurs contraintes, dont la généricité est la plus importante.

Cette thèse a été effectuée dans le cadre d'une convention CIFRE entre la société ACTIA et le LAAS-CNRS. L'objectif de ce travail de thèse était la conception d'un système de diagnostic de pannes automobiles *générique* sans informations symboliques sur les systèmes ni modélisation. L'outil de diagnostic doit être développé uniquement à partir d'observations en bon fonctionnement et dans les divers cas de pannes que l'on désire diagnostiquer. C'est donc une approche du diagnostic par une stratégie de type reconnaissance de formes. Un des objectifs est de n'avoir à modifier qu'un minimum de choses dans le coeur du système pour développer un produit pour une nouvelle voiture et même pour le diagnostic d'une nouvelle fonction prise en main par le raz de marée électronique. L'ensemble des systèmes est considéré comme une boîte noire et ses entrées-sorties sont analysées dans diverses situations. Un tel système peut être considéré, non plus comme un système de diagnostic, mais plutôt comme *générateur de systèmes de diagnostic*. Les avantages d'un tel système sont importants : diminution du temps et des coûts de développement et indépendance vis-à-vis des constructeurs.
Pour atteindre cet objectif, assez ambitieux, notre idée d'origine était d'utiliser les capacités d'apprentissage des *réseaux de neurones artificiels*. Après une première phase d'étude qui a prouvé l'intérêt des réseaux de neurones, l'utilisation couplée d'autres techniques parut nécessaire et notamment *des méthodes statistiques*. De plus, l'utilisation des réseaux de neurones a posé de nombreux problèmes théoriques. Ce travail a finalement abouti au développement de nouveaux algorithmes neuronaux et à une nouvelle méthodologie de diagnostic. Cette méthodologie dénommée STANDING (STAtistiques et Neurones artificiels pour un système de DIagNostic Générique) est basée sur deux principes : la modélisation des zones de bon fonctionnement pour une détection de défauts avec une sensibilité réglable et la discrimination des effets des pannes dans un espace caractéristique du mauvais fonctionnement pour l'identification des pannes. STANDING a été testé sur des données réelles.

Dans un *premier chapitre* nous ferons un état de l'art non exhaustif des méthodes de diagnostic de

systèmes. Nous insisterons plus particulièrement sur les méthodes de diagnostic par reconnaissance de formes. Après avoir abordé le problème du diagnostic de pannes automobiles nous définirons les objectifs de cette étude.

Le *second chapitre* est consacré à la description du système d'acquisition et de traitement qui a été conçu. La première maquette d'un système de diagnostic générique avec réseaux de neurones sera détaillée. Bien que l'identification des pannes n'a pas toujours été facile à mettre au point et dans certains cas très difficile, cette maquette a montré l'intérêt des réseaux de neurones et la possibilité de développer un système de diagnostic uniquement à partir d'observations. Ce premier développement a pourtant soulevé de nombreux problèmes.

Le *chapitre trois* présente un état de l'art de l'apprentissage des réseaux de neurones à couches. Ce domaine s'étant particulièrement enrichi ces dernières années, l'exhaustivité n'était pas possible. Nous avons essayé de développer certains modèles importants, historiquement, ou pour la compréhension de la suite de ce manuscrit, et nous avons simplement abordé les autres techniques.

Un développement assez particulier fait l'objet du *chapitre quatre* : une étude des structures de l'hypercube pour caractériser les fonctions booléennes linéairement séparables. Dans le domaine des réseaux de neurones binaires, ce problème est fondamental car les fonctions booléennes sont en nombre très faible par rapport au nombre total de fonctions pour une dimension donnée, et ces fonctions définissent de manière exhaustive les positions pertinentes que peut prendre un hyperplan qui coupe l'hypercube, c'est-à-dire la valeur des coefficients synaptiques d'un neurone binaire travaillant sur des données binaires en entrées. Bien que cette étude n'ait pas complètement abouti, plusieurs résultats théoriques ont été établis.

Dans le *chapitre cinq*, nous présenterons une nouvelle famille d'algorithmes pour l'apprentissage des réseaux de neurones à couches et plus particulièrement pour les algorithmes de construction de réseaux de neurones binaires. Plusieurs algorithmes d'apprentissage d'une unité binaire sont proposés pour des applications diverses. Les nombreuses simulations qui ont été réalisées pour comparer ces algorithmes à ceux utilisés le plus souvent, montrent l'efficacité des méthodes proposées. Basés sur une stratégie de construction particulière, plusieurs algorithmes de construction de réseaux sont proposés. La dernière évolution est apparemment le premier algorithme de ce genre aussi général : les entrées peuvent être quelconques et le nombre de sorties aussi (mais binaires). N'ayant *a priori* pas de concurrent à neurones binaires, ce dernier sera comparé avec un algorithme basé sur des neurones continus : Cascade Correlation. Une possible implantation VLSI des réseaux fournis par cet algorithme sera proposée.

Le *dernier chapitre* sera consacré à l'aboutissement de ce travail : la méthodologie STANDING. Nous détaillerons successivement les nouvelles techniques employées dans la détection de défauts et dans l'identification des pannes : neuronales, statistiques et géométriques. Nous terminerons le chapitre par une série de résultats obtenus sur des données réelles qui illustre les diverses phases de la conception du système.

Les sciences n'essaient pas d'expliquer ; c'est tout juste si elles tentent d'interpréter ; elles font essentiellement des modèles. Par modèle, on entend une construction mathématique qui, à l'aide de certaines interprétations verbales, décrit les phénomènes observés. La justification d'une telle construction mathématique réside uniquement et précisement dans le fait qu'elle est censée fonctionner.

JOHN VON NEUMANN

Chapitre 1

Méthodes de diagnostic et définition des objectifs

1.1 Introduction

Nos sociétés technologiques engendrent des systèmes de complexité croissante dont nous sommes de plus en plus dépendants. Cette dépendance nécessite de pouvoir leur faire confiance. Or les catastrophes qui arrivent encore trop souvent montrent qu'une défaillance mineure peut avoir des conséquences dramatiques : humaines, économiques ou environnementales. La maîtrise de *la sécurité de fonctionnement* et des *méthodes de diagnostic* des systèmes complexes est donc un enjeu fondamental.

Dans nos civilisations occidentales, la qualité d'un médecin est estimée en fonction de ses capacités à guérir ses patients, donc à faire un bon diagnostic et à prendre les bonnes décisions. En revanche dans la médecine chinoise traditionnelle le médecin intervient avant que survienne la maladie car son activité est principalement préventive, *il vaut mieux prévenir que guérir*. Ce principe peut être transposé au domaine qui nous intéresse : les défaillances, même détectées et réparées rapidement coûtent cher et peuvent être dangereuses. La maîtrise des méthodes de diagnostic ne se cantonne pas au domaine du diagnostic curatif, mais concerne aussi le diagnostic préventif. Le diagnostic des systèmes est ainsi un domaine scientifique en plein essor impliqué dans de nombreux domaines industriels.

L'automobile n'échappe pas à cet accroissement de la complexité des systèmes. En effet, l'automobile subit depuis le début des années 80 une mutation similaire à l'évolution de l'aéronautique vers ce que l'on nomme le *fly by wire* : commandes de vol électriques, navigation et pilotage automatique, atterrissage assisté... Les voitures d'aujourd'hui sont pourvues d'une multitude de systèmes électroniques dirigés par un nombre grandissant de calculateurs. Cette évolution bouleverse le métier de garagiste, particulièrement dans la phase de diagnostic. Le développement d'outils de diagnostic automatique est donc indispensable, et leur puissance doit suivre l'évolution de la complexité des systèmes.

Le mot diagnostic vient du grec *diagnosis* signifiant "action de discerner" dont la racine *gnosis* signifie "connaissance". Le diagnostic est l'utilisation des connaissances pour reconnaître un état défaillant ou non. Quand on observe un homme réalisant une tâche de diagnostic, il utilise des informations

très diverses : issues de ses sens, de son expérience, de sa connaissance du système... et les traite avec des processus cognitifs variés. Cette variété d'informations et de méthodes utilisées consciemment ou inconsciemment par l'homme, peut expliquer la multitude de méthodes de diagnostic qui ont été développées. Le diagnostic est ainsi un domaine de recherche difficile et passionnant car pluridisciplinaire. Les domaines particulièrement impliqués sont l'automatique et l'intelligence artificielle, dont les principes constituent le coeur d'une grande partie des méthodes de diagnostic. Mais des techniques de nombreux autres domaines sont aussi utilisées, en particulier les statistiques et le traitement du signal pour le prétraitement des données.
Dans un premier paragraphe, nous ferons un état de l'art des méthodes de diagnostic. On définira trois catégories de méthodes de diagnostic qui se différencient par le type d'information qui est traité : les méthodes symboliques, internes et externes. Les deux premières catégories seront simplement abordées. La présentation de la dernière catégorie sera plus précise, car c'est une méthodologie de ce type qui sera développée dans cette thèse. Une classification précise n'est pas aisée, et certaines techniques appartiennent à plusieurs catégories. Les réseaux de neurones sont une de ces méthodes. Ils seront présentés séparément car leur développement très important ces dernières années a fourni diverses possibilités d'utilisation de cette technique dans les trois catégories. Nous développerons ensuite le problème du diagnostic de pannes automobiles, ce qui nous amènera dans un dernier paragraphe à une définition précise des objectifs de ces travaux.

1.2 Les méthodes de diagnostic de systèmes : état de l'art

Le diagnostic des systèmes industriels a fait l'objet d'une normalisation nationale et internationale (AFNOR et CEI). La définition du diagnostic qui a été adoptée est la suivante :

> *Le diagnostic est l'identification de la cause probable de la (ou des) défaillance(s) à l'aide d'un raisonnement logique fondé sur un ensemble d'informations provenant d'une inspection, d'un contrôle ou d'un test.*

Un système de diagnostic est donc un procédé de mise en relation d'un effet observé et de sa cause. Il doit donc comporter deux fonctionnalités :

- *la détection d'une défaillance* : toute modification du comportement du système par rapport à celui attendu doit être détectée.
- *l'identification* : la cause de la défaillance doit être déterminée à partir d'informations diverses.

Il est évident que l'objectif final d'identification ne peut être atteint que si on possède assez d'informations : la causalité doit implicitement se trouver dans l'ensemble des informations traitées.
Nous classerons les méthodes de diagnostic en trois catégories, suivant le type d'information utilisé.

- *les méthodes symboliques de diagnostic* : les informations utilisées sont principalement des données de haut niveau, la plupart du temps symboliques, au sens large, et non numériques. Elles sont en général basées sur une description symbolique du système et/ou de son comportement. Les informations utilisées sont des connaissances structurelles et comportementales. Les liens de cause à effet entre les défaillances et leurs causes, sont souvent explicites dans les données.
- *les méthodes internes de diagnostic* : ce type de diagnostic est aussi appelé *diagnostic par modèle.* C'est principalement le domaine de l'automatique. Les informations utilisées sont numériques et le fonctionnement interne du système est modélisé mathématiquement. Les liens de cause à effet se trouvent dans les divers modèles utilisés.

- *les méthodes externes de diagnostic* : les informations utilisées sont aussi numériques, mais aucun modèle mathématique n'est utilisé. C'est principalement le domaine de l'analyse de données. Les liens de cause à effet doivent être définis par un ensemble d'observations du système en bon et en mauvais fonctionnement.

Nous n'aborderons pas explicitement les méthodes de diagnostic préventif. Celui-ci n'est possible que si on a une estimation de l'évolution du comportement du système, ce qui nécessite des informations numériques. Il est donc envisageable seulement avec les deux dernières catégories de méthodes. Il peut être réalisé en fixant plusieurs seuils de détection, ou en quantifiant l'éloignement d'un mode de fonctionnement encore estimé normal par rapport au fonctionnement nominal.
Deux ouvrages ont servi de base à la constitution de cet état de l'art : un livre très complet sur les diverses méthodes de diagnostic [201], et un autre dédié aux méthodes de diagnostic par reconnaissance de formes [58]. Deux rapports de recherche de synthèse et un article assez général ont également été utilisés [22, 8, 134], ainsi que divers articles de synthèse mais dans des domaines plus précis, que l'on citera successivement. Il convient enfin de signaler que cet état de l'art n'est pas du tout exhaustif. Nous avons plutôt essayé de dégager les méthodes les plus significatives afin de montrer la grande diversité de celles-ci, des domaines scientifiques impliqués et des données utilisées.

1.2.1 Les méthodes symboliques de diagnostic

Les techniques d'analyse fonctionnelle

Ces méthodes d'analyse de systèmes ne sont pas à proprement parler des méthodes de diagnostic. Mais elles sont d'une importance capitale dans la chaîne de développement d'un produit et peuvent servir à l'élaboration d'une méthode de diagnostic basée sur des informations symboliques. Ce sont des méthodes d'analyses et de description en amont de l'étude des défaillances et du développement de stratégies de détection et d'identification [19]. Parmi ces méthodes on peut citer : SADT, RELIASEP, FAST, MERISE...
Nous allons expliquer brièvement la méthodologie SADT (Structured Analysis and Design Technique) car elle est assez générale et permet une description structurée de systèmes très complexes et hétérogènes. De plus elle intègre une notion importante de gestion de documentation qui est un problème difficile à gérer sur de grands systèmes. Le principe de SADT [98] est de décrire le système par un modèle symbolique. Cette méthodologie est graphique et utilise des boîtes connectées entres elles. Une utilisation avancée nécessite donc un outil informatique.
L'application de SADT part de la description la plus abstraite du système. Si nous considérons cette description comme contenue dans un seul module, représenté par une boîte, nous pouvons alors éclater cette boîte en plusieurs boîtes, chacune représentant une composante de la boîte initiale. Chacune des nouvelles boîtes est à son tour décomposée pour exposer l'information qu'elle récèle. Cette approche d'abstraction hiérarchique permet de ne pas considérer trop de détails trop tôt. La précision de l'analyse est progressive. A chaque étape, le nombre de boîtes admissible sur le même diagramme est limité. Cette contrainte permet une exposition systématique et uniforme des niveaux successifs de détail.
Deux décompositions complémentaires peuvent être réalisées. La décomposition en *actigrammes* et la décomposition en *datagrammes*. La première se focalise sur les fonctions à réaliser : les fonctions sont dans les boîtes et les données alimentent ou contrôlent les boîtes ou en sortent, ce sont les divers arcs qui relient les boîtes. La seconde est une approche plus orientée objet. Les données sont dans les boîtes, et ce sont les fonctions qui relient les données par les arcs.
Bien que cette méthodologie soit particulièrement bien adaptée au génie logiciel, elle est très générale et peut être utilisée pour décrire de manière aussi précise que l'on veut un système aussi complexe qu'il soit. La décomposition fonctionnelle (hiérarchique et descendante) obtenue peut alors être utilisée dans

une AMDE (Analyse des Modes de Défaillance et de leur Effets) et dans le développement d'arbres de défaillances ou d'arbres de tests.

AMDE et AMDEC

L'AMDE (Analyse des Modes de Défaillance et de leur Effets) et l'AMDEC (Analyse des Modes de Défaillance, de leur Effets et de leur Criticité) sont des méthodes très utilisées dans les études de sûreté de fonctionnement lors de la conception de systèmes industriels. Bergot dans [19] décrit l'AMDEC comme une méthode d'analyse critique consistant à identifier de façon inductive et systématique les risques de dysfonctionnement d'un système puis à en rechercher les origines et leurs conséquences. Une AMDEC procède en deux étapes. Premièrement, elle consiste à examiner comment et pourquoi les fonctions du système étudié risquent de ne plus être assurées correctement, en définissant les modes de défaillances potentielles, en recherchant les causes possibles à l'origine de chaque mode, et enfin, en recherchant pour chaque combinaison cause-mode de défaillance, les effets sur le système et sur l'utilisateur, ainsi que les niveaux de détection possibles. La seconde étape, qui concerne uniquement l'AMDEC consiste à évaluer la criticité des modes de défaillance à partir de trois critères : gravité, fréquence d'apparition et probabilité de non-détection. Cette dernière étude permet de hiérarchiser les défaillances potentielles, et de proposer des actions correctives. Les résultats de cette analyse sont présentés sous forme d'un tableau à colonnes. L'AMDEC est particulièrement utile pour le diagnostic des systèmes opérationnels. Si elle a été bien faite, elle contient toute les défaillances possibles reliées à leurs causes. Le problème du diagnostic est alors la démarche déductive qui consiste à analyser le tableau et à déterminer toutes les causes possibles d'une défaillance observée. L'avantage de cette méthode, qui est son exhaustivité dans la détermination des liens de cause à effet, devient aussi un inconvénient : son extrême lourdeur d'utilisation. Pour des systèmes industriels complexes, une AMDEC peut conduire à la réalisation de milliers de tableaux. Dans ces conditions, l'utilisation de l'AMDEC pour le diagnostic impose l'utilisation de méthodes informatisées qui permettent de générer automatiquement les connaissances nécessaires à la conception d'un outil d'aide au diagnostic. L'article [186] traite de ce sujet et plus particulièrement de la génération automatique d'une base de connaissances pour un système expert à partir des données de l'AMDEC.

Arbres de défaillances

La méthode des arbres de défaillances est une méthode inductive avec laquelle on identifie toutes les combinaisons d'événements possibles qui entraînent la réalisation d'un événement indésirable. Les conditions et les événements sont organisés sous la forme d'un arbre utilisant les connecteurs logiques *et* et *ou*. L'arbre de défaillances est constitué de niveaux successifs d'événements tel que chaque événement à un niveau donné est généré à partir de combinaisons logiques d'événements de niveaux inférieurs. Cette procédure est itérée jusqu'à atteindre les événements élémentaires appelés événements de base. Ces derniers qui sont les feuilles de l'arbre, correspondent à la défaillance d'un élément de base du système qui ne peut plus être expliquée plus finement. Ce sont les causes possibles de la défaillance observée. Il est possible d'associer à chaque événement une probabilité d'apparition, qui peut être utilisée dans l'exploitation de ces arbres.

Arbres de tests

Les arbres de tests ressemblent aux arbres de défaillances. C'est la même représentation mais enrichie et présentée à des fins opérationnelles. Un arbre de tests [62] est constitué de deux types de noeuds. Les premiers représentent l'état du système d'un point de vue diagnostic, c'est-à-dire l'état de défaillance observé et les hypothèses sur l'origine de cette défaillance (par exemple plusieurs composants sont

suspects et d'autres non). Le second type de noeuds correspond aux tests qui doivent être réalisés pour augmenter la connaissance et permettre de déterminer la cause de la défaillance parmi les causes possibles. L'arbre est constitué de manière à avoir une alternance de noeuds d'état et de noeuds de tests. Un noeud d'état a plusieurs noeuds de test fils connectés par un *ou* qui correspondent à un choix possible de tests qui permettent de faire avancer la recherche, et un noeud de test a plusieurs noeuds d'état fils connectés par un *et* qui sont les sous-problèmes qui devront être résolus. Comme dans les arbres de défaillances, les feuilles correspondent à un élément de base du système, un composant élémentaire. On parcourt l'arbre en faisant les tests jusqu'à ce que tous les problèmes soient résolus, c'est-à-dire que tous les composants soient connus comme non-défaillants ou défaillants. Les arbres de tests sont particulièrement utilisés dans l'automobile où ils sont construits à partir des données d'une AMDEC. Ils permettent de présenter de manière simple toutes les actions à mener pour faire un diagnostic. Le problème de cette méthode est la construction de ces arbres. C'est un travail très long et complexe, car divers critères d'optimisation interviennent dans leur élaboration (trouver les tests, minimisation de la profondeur de l'arbre, prise en compte de l'inaccessibilité d'un élément...). C'est un problème combinatoire qui nécessite une méthodologie de génération automatique. C'est dans ce cadre qu'une thèse a été réalisée par M. Olivier Duffaut à ACTIA [62] : une approche multi-modèles pour la génération automatique d'arbres de tests. Le problème est de considérer différentes modélisations complémentaires du système, dans un même processus de génération d'arbres de tests : modèle de système en bon fonctionnement, modèle en mauvais fonctionnement, modèle fonctionnel, modèle électrique...

Les méthodes de diagnostic par simulations

Ces méthodes utilisent une approche comportementale du système. Celui-ci est modélisé par un langage spécifique et son fonctionnement est simulé. Les résultats de cette simulation sont ensuite comparés à l'état réel du système. Les plus couramment utilisés sont ceux qui s'intéressent aux changements d'états des différentes entités : *l'approche par événements*. Les modèles utilisés sont principalement : *les graphes à automates finis*, *les réseaux de Pétri* et *les Grafcet*.
Les réseaux de Pétri constituent un domaine de recherche actif dans l'automatique séquentielle. Diverses extensions au formalisme classique ont été développées, entre autres les réseaux de Pétri généralisés, temporisés, colorés, continus...
Dans [50], l'auteur fait un état de l'art détaillé de ces nouvelles extensions des réseaux de Pétri pour la modélisation des systèmes dynamiques. Une utilisation des réseaux de Petri temporisés pour le diagnostic est présentée dans [185].

Les systèmes experts de première génération

Avec les systèmes experts, l'intelligence artificielle tente de calquer les raisonnements humains de haut niveau à partir d'une logique formelle, et notamment les processus déductifs utilisés par exemple dans le diagnostic réalisé par un médecin. Un système expert est constitué de deux composants : la *base de connaissances* et le *moteur d'inférence*. La base de connaissances peut être décomposée en deux parties : la *base de faits* et la *base de règles*. La base de faits représente l'état du système observé. La base de règles est un ensemble de règles logiques qui permettent de déduire, d'*inférer* de nouveaux faits à partir d'autres faits déjà établis dénommés les *prémisses*. Le moteur d'inférence a la lourde tâche de déduire tous les faits susceptibles d'être déduits à partir de la base de connaissances. Plusieurs techniques sont possibles : chaînage avant, chaînage arrière, recherche en profondeur, en largeur... et plusieurs logiques peuvent être utilisées : logique des prédicats, logiques non monotones, logiques floues, logiques probabilistes ...

Les premiers systèmes experts, dont le célèbre MYCIN (pour le diagnostic de maladies infectieuses), ont été largement médiatisés et paraissaient très prometteurs. Mais l'engouement pour ce type d'approche en diagnostic a largement diminué. Les raisons en sont assez nombreuses. Premièrement il faut codifier toutes les règles de diagnostic que l'expert avait acquis par son expérience. Ce travail est extrêmement délicat et nécessite non seulement l'expert du domaine mais aussi la participation d'un cognitien, qui code les connaissances dans la logique formelle adoptée. C'est la méthode de diagnostic de l'être humain qui est modélisée et la performance du diagnostic réalisé par le système expert dépend complètement des capacités de l'expert et du succès de la codification. De plus les systèmes de diagnostic obtenus sont très spécialisés et leur utilisation est complètement restreinte aux applications couvertes par l'expérience qui a été implantée.

Les nouvelles approches de l'intelligence artificielle

Ces nouvelles approches du diagnostic par intelligence artificielle ont été précisément formalisées par Reiter [167]. Ces nouvelles méthodes sont appelées *méthode de diagnostic du premier principe* et aussi *diagnostic par modèle.* Au lieu de modéliser les règles de diagnostic utilisées, c'est la structure du système qui est modélisée : ses composants et leurs connexions. Deux approches sont possibles : l'approche par *maintien de consistance* et l'*approche abductive.*
Dans l'approche par maintien de consistance, Reiter définit le modèle par $< SD, COMPS >$ avec SD la description du système et COMPS l'ensemble des composants qui est un ensemble de constantes. La description du système est un ensemble de propositions du premier ordre qui définit comment les composants du système sont connectés entre eux, et comment ils se comportent en fonctionnement normal. Il définit alors un prédicat unaire $AB(c)$ avec $c \in COMPS$ qui signifie que le composant du système c est en défaillance. Une observation OBS est un ensemble de propositions du premier ordre. Un diagnostic Δ pour le triplet $(SD, COMPS, OBS)$ est alors défini comme le sous-ensemble minimal $\Delta \in COMPS$ tel que

$$SD \cup OBS \cup \{AB(c) \,/\, c \in \Delta\} \cup \{\neg AB(c) \,/\, c \in COMPS - \Delta\}$$

est consistant. Une vérification systématique pour tous les sous-ensembles de $COMPS$ est totalement inefficace dès que le nombre de composants du système est un peu important. Un système beaucoup plus performant a été réalisé en 1987, dénommé GDE (General Diagnosis Engine). D'autres méthodes récentes ont été proposées basées sur les TMS (Truth Maintenance Systems) et les ATMS (Asumption Truth Maintenance System).
Dans l'approche abductive, le système est décrit uniquement par SD qui contient différentes règles de fonctionnement sans distinction entre les modes de fonctionnement normaux et anormaux. Un diagnostic abductif est alors un ensemble minimal d'hypothèses sur le mode de fonctionnement de certains composants qui permet d'inférer les observations, soit

$$SD \cup \{MODE(c) \,/\, c \in \Delta\} \quad \models \quad OBS$$

Cette approche a aussi été étudiée récemment et a fourni diverses méthodes de génération de diagnostic (voir [141] pour des références).

1.2.2 Les méthodes internes de diagnostic

Ces méthodes sont basées sur la connaissance d'un modèle paramétrique du système et de la valeur de ses paramètres en fonctionnement normal. Ce modèle traduit la réalité physique du système et il a une expression analytique. Cette connaissance permet alors de calculer des quantités qui doivent être toujours nulles en fonctionnement normal et non nulles en cas de défailance. Ces quantités sont

appelées résidus. Une analyse de ces résidus basée sur la théorie de la décision, sur la théorie de la détection de changements [12] ou sur des tests statistiques simples permet de détecter les défauts ; l'identification du défaut se faisant par correspondance de la composante d'un résidu avec un organe. Parmi ces méthodes, on en distingue deux types : les *méthodes de redondance analytique* [70] et les *méthodes d'estimation paramétrique* [100, 99].
Avant de décrire brièvement les principes de ces deux types de méthodes, il convient de mentionner le nombre important d'études, depuis le début des années 80, sur l'identification des processus non linéaires [96, 83, 183, 103, 51].

Les méthodes de redondances analytiques

Par utilisation de relations redondantes dans le modèle, on réécrit les expressions analytiques du modèle afin de générer des quantités qui doivent être toujours nulles en fonctionnement normal et dont le calcul ne nécessite que des grandeurs mesurables ou estimables. Ces quantités constituent les résidus à étudier.
Le cas particulier où le vecteur d'état est calculable et ainsi la sortie estimable grâce au modèle est le domaine de l'*estimation d'état*. Le résidu est alors simplement la différence entre la sortie estimée et la sortie calculée (un exemple linéaire est le filtre de Kalman).

Les méthodes d'estimation de paramètres

A partir des quantités mesurables on fait une estimation des paramètres du modèle. Le résidu est alors la différence entre cette estimation et les valeurs des paramètres du modèle (un exemple est la modélisation ARMA d'un processus stationnaire).

1.2.3 Les méthodes externes de diagnostic

Contrairement aux deux catégories précédentes, ces méthodes de diagnostic ne se basent sur aucune modélisation du système. Le système est observé de l'extérieur, et on le considère comme une boîte noire. Les informations traitées sont numériques et la prise de décision est basée sur une étude de la valeur d'un signal observé par rapport à un ensemble de valeurs connues et stockées dans une base de données représentant le système en bon fonctionnement ou dans divers modes de défaillance. Le problème à résoudre est donc de décider si un signal est dans une certaine zone ou plutôt dans une autre. C'est donc un problème essentiellement statistique et les règles de décision font souvent intervenir les lois de probabilité des signaux.
Après avoir décrit quelques tests simples qui permettent une première approche de ce problème, nous présenterons le problème général de reconnaissance de formes. Nous décrirons très précisément la théorie probabiliste de la décision bayésienne, nous aborderons ensuite les techniques d'estimation de densité de probabilité et les diverses méthodes de classification supervisée et non supervisée.

Les méthodes statistiques simples

La méthode la plus simple consiste à comparer la valeur du signal à des seuils déterminés en observant le système en bon fonctionnement. Il peut y avoir plusieurs intervalles de validité, ainsi que deux types de seuil : les *seuils de prévention* et les *seuils de défaillance*. Cette méthode est souvent utilisée car extrêmement simple à mettre en place. Son problème est son imprécision. Les signaux sont souvent bruités et une valeur peut temporairement ne pas être dans les intervalles de confiance sans avoir une défaillance. Si les intervalles de confiance sont trop étroits il y a alors des risques de fausses alarmes et si les intervalles sont trop larges il y des risques de défaillances non détectées.

Une solution est d'estimer des caractéristiques statistiques simples des signaux, moyenne et écart-type, en les calculant dans une fenêtre glissante pour éliminer une partie du bruit : c'est un filtrage passe-bas. Ainsi on peut calculer une estimation de la moyenne d'un signal y_t par

$$\widehat{\overline{y}} = \frac{1}{n} \sum_{i=t-n+1}^{t} y_i$$

C'est ensuite cette moyenne qui sera comparée au seuil. On peut aussi tester un estimateur de l'écart-type $\widehat{\sigma}_y$ calculé par

$$\widehat{\sigma}_y^2 = \frac{1}{n} \sum_{i=t-n+1}^{t} (y_i - \widehat{\overline{y}})^2$$

ou alors par l'estimateur non biaisé

$$\widehat{\sigma}_y^2 = \frac{1}{n-1} \sum_{i=t-n+1}^{t} (y_i - \widehat{\overline{y}})^2$$

Cet estimateur est ensuite comparé à la valeur calculée sur un très grand nombre d'échantillons de fonctionnement sans défaillance.
Il existe une grande quantité d'autres tests simples : basés sur la médiane, basés sur la différence entre deux mesures qui est une estimation de la dérivée du signal (dans le cas où l'échantillonnage se fait à intervalle régulier)...
La méthode générale est de calculer un estimateur d'une quantité statistique représentative du signal et de la comparer à sa valeur en bon fonctionnement. Un moyen de fixer les seuils de détection en fonction de la base de données de bon fonctionnement est l'utilisation de la loi de Tchebychev-Bienaymé, qui s'exprime par

$$P(|x - \overline{x}| \geq k\sigma) \leq \frac{1}{k^2}$$

Si la base de données est assez renseignée, on aura une bonne estimation de l'espérance de x et de son écart-type. Dans ce cas si on se fixe une probabilité maximale de fausse alarme à $\frac{1}{k^2}$ (c'est-à-dire que les valeurs extrêmes sont considérées comme défauts), on peut fixer les seuils de détection à $\overline{x} \pm k\sigma$.
Il convient de préciser que cette méthode est très imprécise car elle ne tient compte que des moments du premier et du second ordre de la distribution de probabilité. Par exemple, elle ne tient absolument pas compte d'une asymétrie possible de la distribution. Mais c'est une première approche très simple à implanter.

Les méthodes de reconnaissance de formes

Le problème de reconnaissance de formes peut être défini comme suit. Après avoir choisi un espace de représentation du système, c'est-à-dire un ensemble de signaux qui peuvent subir divers prétraitements, on réalise l'acquisition d'un grand nombre de données de divers modes de fonctionnement. Nous appellerons ces observations l'ensemble d'apprentissage. On a connaissance de l'existence de M modes de fonctionnement appelés classes. Pour chaque observation, on sait à quel mode de fonctionnement elle appartient. Le problème qui se pose est de définir à partir de ces connaissances, un procédé de discrimination qui permette de déterminer la classe d'une observation non contenue dans la base d'apprentissage. Ce problème est équivalent à la recherche des frontières entre les classes qui minimise l'erreur de classification. Il y a diverses approches de ce problème et une littérature très abondante. Nous donnerons un aperçu de ces diverses méthodes.

Théorie bayésienne de la décision [61]. La théorie probabiliste de ce problème est très intéressante car elle formule la solution optimale par minimisation du risque conditionnel, que l'on peut souvent assimiler au risque de mauvaise classification. Cette théorie est nommée théorie bayésienne de la décision car elle est basée sur le calcul des probabilités conditionnelles grâce à la règle de Bayes. Cette méthode est souvent prise comme référence pour estimer la qualité d'autres procédés. Nous allons décrire précisément les principes de cette théorie.
On définit trois ensembles.

- Premièrement l'espace des classes $\Omega = (\omega_1, \ldots, \omega_M)$, qui est muni d'une loi de probabilité *a priori* notée ν. On a

$$\forall \omega \in \Omega \quad \nu(\omega) \geq 0 \qquad \text{et} \qquad \sum_{i=1}^{M} \nu(\omega_i) = 1$$

- L'espace des observations Y. Les observations de chaque classe ω sont régies par la loi de probabilité conditionnelle P_ω. La probabilité d'avoir l'observation y sachant qu'elle appartient à la classe ω est $P_\omega(y)$ que l'on note aussi $P(y/\omega)$. D'un point de vue global, la probabilité d'avoir l'observation y est alors

$$P(y) = \sum_{i=1}^{M} \nu(\omega_i)\, P_{\omega_i}(y)$$

- L'espace de décision D. C'est l'ensemble des décisions que l'on peut prendre quand on est en présence d'une certaine observation. On définit alors une fonction notée s de O dans D qui est la *stratégie* de décision. On définit aussi une fonction de coût λ (ou fonction de perte) de $\Omega \times D$ dans $\mathbb{R}$. Ainsi $\lambda(\omega, \delta)$ représente le coût de la décison δ alors que l'on a la classe ω.

Les stratégies optimales utilisent un critère nommé *le risque conditionnel* : c'est-à-dire le coût moyen de la décision δ lorsqu'on est en présence de l'observation y. Cette fonction de risque est définie par

$$R(\delta, y) = \sum_{i=1}^{M} \lambda(\omega_i, \delta)\, P(\omega_i/y)$$

Avec $P(\omega_i/y)$ la probabilité *a posteriori* d'avoir la classe ω_i sachant que l'on a l'observation y. La stratégie optimale est alors celle qui minimise ce risque conditionnel soit

$$s^*(y) = \arg\min_{\delta \in D} \{R(\delta, y)\}$$

Le problème du calcul de cette fonction de risque réside dans le calcul des probabilités $P(\omega_i/y)$, car la fonction de risque est définie par l'utilisateur. Mais d'après la loi de Bayes, nous avons

$$P(\omega_i/y) = \frac{\nu(\omega_i)\, P_{\omega_i}(y)}{P(y)}$$

dans le cas discret. Dans le cas continu, les lois de probabilité P et P_{ω_i} sont alors munies de leurs densités p et p_{ω_i}, et on peut montrer simplement que l'on a alors

$$P(\omega_i/y) = \frac{\nu(\omega_i)\, p_{\omega_i}(y)}{p(y)}$$

La fonction de coût s'exprime alors dans le cas discret et dans le cas continu par

$$R(\delta, y) = \frac{1}{P(y)} \sum_{i=1}^{M} \lambda(\omega_i, \delta)\, P_{\omega_i}(y)\, \nu(\omega_i) \qquad \text{et} \qquad R(\delta, y) = \frac{1}{p(y)} \sum_{i=1}^{M} \lambda(\omega_i, \delta)\, p_{\omega_i}(y)\, \nu(\omega_i)$$

Nous allons détailler deux types de décision importante : la décision sans ambiguïté (une observation est classée systématiquement) et la décision avec rejet d'ambiguïté (une observation peut être rejetée). Nous aborderons aussi une extension de cette dernière, à laquelle est ajoutée une notion de rejet de distance, mais qui ne rentre pas exactement dans le cadre de la théorie du coût minimal.

- *Décision sans ambiguïté :*

  Dans ce type de décision nous avons une bijection entre les ensembles Ω et D. C'est-à-dire qu'une décision d_i correspond à une classe ω_i, et de manière univoque. On notera b cette bijection de Ω dans D. La fonction λ est alors nommée l'indicatrice d'erreurs et on a

$$\begin{aligned} \lambda(\omega, \delta) &= 0 \quad \text{si} \quad \delta = b(\omega) \\ \lambda(\omega, \delta) &= 1 \quad \text{si} \quad \delta \neq b(\omega) \end{aligned}$$

  La fonction de risque peut alors s'exprimer par

$$\begin{aligned} R(\delta, y) &= \sum_{i=1}^{M} \lambda(\omega_i, y)\, P(\omega_i/y) \\ &= \sum_{i=1}^{M} P(\omega_i/y) \; - \; P(b^{-1}(\delta)/y) \\ &= 1 \; - \; P(b^{-1}(\delta)/y) \end{aligned}$$

  La stratégie optimale est donc donnée par la minimisation de cette fonction donc

$$\begin{aligned} s^*(y) &= \arg\min_{\delta \in D} R(\delta, y) \\ &= \arg\max_{\delta \in D} P(b^{-1}(\delta)/y) \\ &= \arg\max_{\omega \in \Omega} P(\omega/y) \\ &= \arg\max_{\omega \in \Omega} [\, P_\omega(y)\, \nu(\omega)\,] \end{aligned}$$

- *Décision avec rejet d'ambiguïté :*

  Pour cette stratégie, on introduit une décision de rejet, que l'on notera $\diamond$. On a donc une bijection seulement entre Ω et $D - \{\diamond\}$. La fonction de coût est alors définie par

$$\begin{aligned} \lambda(\omega, \delta) &= 0 && \text{si} \quad \delta = b(\omega) \\ \lambda(\omega, \delta) &= 1 && \text{si} \quad \delta \neq b(\omega) \\ \lambda(\omega, \delta) &= 1 - r && \text{si} \quad \delta = \diamond \end{aligned}$$

  avec le coefficient r dans $[0, 1]$. On verra plus loin sa signification. Dans ce cas, la fonction de risque s'exprime par

$$\begin{aligned} &\text{si} \quad \delta = \diamond \quad \text{alors} \quad && R(\delta, y) = 1 - r \\ &\text{si} \quad \delta \neq \diamond \quad \text{alors} \quad && R(\delta, y) = 1 - P(b^{-1}(\delta)/y) \end{aligned}$$

  On peut alors montrer simplement que la stratégie optimale est définie par

$$\begin{aligned} s^*(y) &= \diamond && \text{si} \quad \max_{\omega \in \Omega} P(\omega/y) \leq r \\ s^*(y) &= \arg\max_{\omega \in \Omega} P(\omega/y) && \text{si} \quad \max_{\omega \in \Omega} P(\omega/y) > r \end{aligned}$$

On a donc un rejet lorsque toutes les probabilités *a posteriori* $P(\omega/y)$ sont inférieures au seuil r. Ceci peut arriver dans les zones de chevauchement des classes, il y a ambiguïté sur la décision, d'où le nom de cette stratégie de décision. On peut montrer simplement que le rejet d'ambiguïté peut se produire seulement si

$$\frac{1}{M} \leq r \leq 1$$

- *Décision avec rejet d'ambiguïté et rejet de distance :*

  Dans [58], l'auteur introduit aussi une notion de rejet de distance. Ce rejet de distance intervient, si la probabilité d'apparition d'une observation y est globalement trop faible. On teste donc d'abord la probabilité $P(y)$ et on fait un rejet de distance si elle est inférieure à un seuil, sinon on applique la règle de décision avec rejet d'ambiguïté.

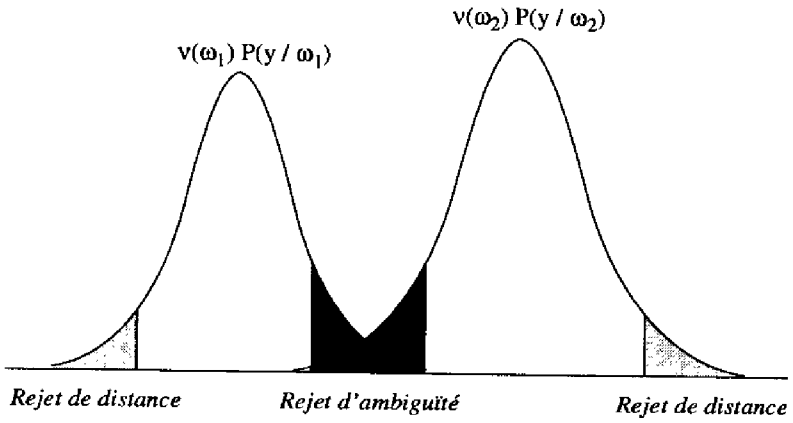

FIG. 1.1: Rejet d'ambiguïté et rejet de distance

La figure 1.1 symbolise ces deux types de rejet dans le cas monodimensionnel de deux classes régies par des lois gaussiennes.

Le problème de l'application de cette théorie de la décision bayésienne est qu'en général on ne connaît pas les lois de probabilité. Il faut donc les estimer à partir d'un ensemble d'observations, le plus grand possible. L'estimation de loi de densité de probabilité est un problème d'ordre statistique qui a donc une grande importance en reconnaissance de formes. Il existe de nombreuses méthodes pour résoudre ce problème. On peut les classer en deux catégories : les estimations non-paramétriques et les estimations paramétriques.

Les méthodes non-paramétriques d'estimation de densité. Le principe de ces méthodes [120] est de considérer qu'une densité $p(x)$ ne présente pas de variations brusques dans une région R assez petite. Si N est le nombre total d'observations et k le nombre d'observations dans cette petite région, alors un bon estimateur de la probabilité $P(R)$ de se trouver dans cette région est simplement k/N. Donc une estimation de la densité de probabilité en tout point x de cette région est

$$\widehat{p}(x) \approx \frac{P(R)}{V} \approx \frac{k}{VN}$$

avec V le volume de la région R. Pour estimer $p(x)$ en un point x_0 on définit une suite de régions $(R_n(x_0))_{n=1,\ldots,N}$ contenant x_0 et utilisées respectivement avec $1 \ldots N$ observations. Avec la méthode

précédente on obtient alors une suite d'estimateurs de la densité en x_0, que l'on notera $(p_n(x_0))_{n=1,\ldots,N}$. Si on note k_n le nombre d'observations parmi n qui se trouvent dans la région $R_n(x_0)$ de volume V_n nous avons

$$p_n(x_0) = \frac{k_n}{nV_n}$$

Il y a deux stratégies possibles pour générer les régions R_n [58] :

- La première consiste à lier la taille de R_n au nombre d'observations n. Ce sont les méthodes des *histogrammes*, du *noyau* ou des *fenêtres de Parzen*. Dans les trois cas la région considérée est un hypercube. Pour la méthode des histogrammes on fait simplement un quadrillage de la zone, alors que dans les deux autres on définit un hypercube centré sur chaque point. Il convient de noter que dans [148], les auteurs proposent une méthode de détection de défaillance utilisant les histogrammes.

- La seconde est de fixer le nombre d'observations k_n en fonction du nombre d'observations n et de construire les régions R_n de manière à ce qu'elles contiennent k_n observations. C'est la méthode des *k-plus proches voisins*. On utilise généralement des hypersphères.

Avec ces deux stratégies, on montre sous certaines conditions [61] que l'estimateur $p_n(x_0)$ est asymptotiquement sans biais et consistant : il converge vers la densité de probabilité réelle quand n tend vers l'infini.
Malgré leur grande généricité, ces méthodes d'estimation non-paramétrique de densité posent quelques problèmes : le nombre d'observations nécessaires pour avoir une bonne approximation est généralement important et l'utilisation de ces méthodes demande beaucoup de calculs. Elles sont malgré tout largement utilisées pour la classification, notamment la méthode des k-plus proches voisins, dont plusieurs variantes ont été développées [58].

Les méthodes paramétriques d'estimation de densité. Le principe de ces méthodes est de supposer que la loi de probabilité suit un modèle analytique $p(x)$ défini par un ensemble de paramètres. On procède alors à une estimation des paramètres du modèle par la *méthode des moments* ou du *maximum de vraisemblance*, grâce à la base de données des observations.
La méthode des moments consiste à calculer les moments empiriques sur les observations que l'on possède et à écrire les égalités avec les moments théoriques calculés analytiquement à partir des modèles. Sauf dans des cas simples, cette méthode généralement conduit à un système d'équations non linéaires. La méthode du maximum de vraissemblance consiste à écrire la vraissemblance du modèle, qui est le produit des $p(x)$ pour toutes les observations, et à maximiser cette vraissemblance par rapport aux paramètres du modèle.
L'utilisation des densités gaussiennes multi-dimensionnelles est particulièrement attrayante car celles-ci ne sont définies que par leurs deux premiers moments : le vecteur moyenne μ et la matrice de covariance V. De plus, il est montré qu'une somme pondérée de gaussiennes peut approcher une fonction avec la précision voulue si elle a suffisamment de composantes (ce nombre peut bien sûr être infini). D'une manière générale, le modèle utilisé est donc une somme pondérée de q gaussiennes multidimensionnelles, appelé mélange de gaussiennes. On a alors

$$p(x) = \sum_{k=1}^{q} \alpha_k \, p_k(x) \quad \text{avec} \quad \sum_{k=1}^{q} \alpha_k = 1$$

$$\text{et} \quad p_k(x) = \frac{1}{\sqrt{2\pi}^{\,d} \sqrt{\det V_k}} \exp[-\frac{1}{2} (x - \mu_k)^T \, V_k^{-1} \, (x - \mu_k)]$$

Très souvent ce mélange est restreint à une seule gaussienne de moment (μ, V). Dans ce cas les équations du maximum de vraissemblance conduisent aux estimateurs classiques du vecteur moyenne et de la matrice de covariance soit

$$\widehat{\mu} = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad \text{et} \qquad \widehat{\mu} = \frac{1}{N}\sum_{i=1}^{N} (x_i - \widehat{\mu})^t (x_i - \widehat{\mu})$$

On peut noter que la maximisation de la vraissemblance fournit l'estimateur biaisé de la matrice de covariance.
Mais dès le cas bidimensionnel, l'estimation des paramètres du modèle pose des problèmes. Les équations du maximum de vraissemblance sont insolubles de manière directe (problème du type *poule et oeuf*) et la méthode des moments aboutit à une équation non linéaire complexe qui peut s'écrire, après de laborieux calculs, sous la forme d'un polynôme du neuvième degré [164]. Avec certaines hypothèses simplificatrices sur les matrices de covariance, les calculs sont plus simples, mais on perd en généralité. Ces dernières années ont vu le développement d'une nouvelle méthode appelée EM (*Expectation Maximization*) [164, 54]. Cette méthode est *a priori* parmi les plus efficaces actuellement, car elle permet l'estimation de tous les paramètres d'un mélange de gaussiennes quelconque : les coefficients α_k de pondération des gaussiennes, les vecteurs moyens μ_k et les matrices de covariance V_k. Nous décrirons cette méthode, ainsi qu'une méthode dérivée dans le chapitre 6. Il est très important de noter que cette méthode d'estimation n'est absolument pas restrictive sur le choix du modèle de densité. Elle peut être utilisée pour un mélange de densités quelconques. Il convient enfin de noter que le problème de maximisation de la fonction de vraissemblance peut aussi être abordé par les méthodes classiques d'optimisation : descente de gradient, méthodes du second ordre...

Les méthodes de classification supervisée. En général, ces méthodes de classification n'utilisent pas d'estimation de densité de probabilité et ne font donc pas partie du cadre de la théorie bayésienne. Cependant, certaines réalisent une classification optimale au sens bayésien quand le nombre de points devient très grand. Ce sont des stratégies de classification qui utilisent la base d'apprentissage afin de définir au mieux les frontières entre les classes. Le problème de l'optimalité au sens de la théorie bayésienne est alors appelé *le problème de généralisation*. Le pouvoir de généralisation est défini comme la capacité du discriminant à donner la bonne décision sur des données non contenues dans la base d'apprentissage.
Parmi ces méthodes le cas du discriminant linéaire entre deux classes a particulièrement été étudié. Une multitude de méthodes a été proposée. Elles font partie du cadre général de l'apprentissage d'une unité d'un réseau de neurones. On en détaillera certaines dans le chapitre 3 : le perceptron et ses diverses extensions, l'adaline et la rétropropagation du gradient. Dans le chapitre 5, nous présenterons une autre méthode : l'algorithme BCP avec ses diverses extensions, dont une version qui converge vers la position optimale dans le cas linéairement séparable. Il existe beaucoup d'autres méthodes ([61] pour les références manquantes) : d'autres extensions du perceptron, la méthode de l'*Optimal Margin Classifier* [190] et sa récente modification nommée *Soft Margin Classifier* [46], le discriminant de *Fisher*, la méthode de la *pseudo-inverse*, la *programmation linéaire*, la méthode *Ho-Kashyap*, la méthode des *fonctions potentielles*, la méthode du *perceptron avec maximum de stabilité* [171], l'algorithme *Window Training* [26], et bien d'autres encore. Ces diverses méthodes se différencient sur plusieurs critères : le comportement sur des bases d'apprentissage non linéairement séparables (nous définirons précisément cette notion dans le chapitre 3 mais intuitivement cela signifie qu'avec les données de la base on ne peut pas trouver un hyperplan qui sépare les deux classes sans erreurs), la convergence vers 0 erreur dans le cas linéairement séparable, la convergence vers la position optimale au sens de la théorie bayésienne dans les deux cas linéairement séparables ou non. Parmi ces méthodes, beaucoup ont été développées récemment.

Le problème de classification en plusieurs classes est un des domaine de prédilection des réseaux de neurones. Suivant les algorithmes employés, les frontières entre les classes peuvent être linéaires ou non. Nous développerons largement ces techniques dans le chapitre 3. Quelques techniques non neuronales ont été proposées, notamment une extension de la méthode *Ho-Kashyap* à plusieurs classes [58], dont la technique d'opposition d'une classe au reste est un des principes des algorithmes MOBCPSL et MOBCPSLhd, qui sont des algorithmes de construction de réseaux de neurones binaires pour la classification multi-classes que nous présenterons dans le chapitre 5.
On peut enfin citer la méthode de classification avec des discriminants quadratiques présentée dans [58].

Les méthodes de classification automatique. Jusqu'à présent nous avons supposé que nous connaissions la classe de chaque observation. Or dans un même mode de défaillance, un système peut avoir plusieurs modes de fonctionnement, représentés dans l'espace des observations par des zones distinctes. De plus, même sans défaillance, un système a souvent plusieurs modes de fonctionnement. Donc le problème qui se pose est de pouvoir déterminer ces divers groupements d'observations sans connaissance *a priori* des classes. C'est le problème *de classification automatique* de données, ou de classification non supervisée.
Beaucoup d'algorithmes ont été proposés. On les regroupera en cinq catégories :

- *les méthodes basées sur le critère d'inertie* : le principe de ces méthodes est de minimiser l'inertie intraclasse qui est définie comme la somme sur toutes les classes de la somme des distances au carré de chaque point au centre de sa classe. Intuitivement, on cherche à trouver des classes compactes. L'algorithme le plus connu est sans doute l'algorithme *k-mean* [57, 65]. Un des problèmes de cet algorithme, et de beaucoup d'autres d'ailleurs, est qu'il faut donner le nombre de classes *a priori*. Dans l'algorithme des *Clusters Principaux* [136], il n'est pas nécessaire de donner cette information, c'est un résultat de l'algorithme. Nous verrons dans le chapitre 6 que ces algorithmes posent un autre problème lié au critère optimisé.

- *l'apprentissage non supervisé par réseaux de neurones* : une multitude d'algorithmes a été proposée pour faire de la classification automatique par apprentissage non supervisé de réseaux de neurones [113, 168]. Les réseaux de neurones s'auto-organisent suivant diverses stratégies souvent liées à un critère de proximité. On peut citer notamment : *les réseaux de Kohonen* [110], *Competitive Learning*, *Adaptative Resonance Theory* (ART1 et ART2), *Reinforcement Learning*, *Learning Vector Quantization* (LVQ1, LVQ2 et LVQ3)...

- *les méthodes arborescentes ou hiérarchiques* : ces méthodes construisent un arbre, où chaque noeud représente une division du problème en sous-problèmes. Les différentes profondeurs de l'arbre correspondent à des partitions successives des observations. Au cas limite, les feuilles sont des classes constituées d'un seul élément. Deux approches sont utilisées : l'une montante dite *approche par agglomération* et l'autre descendante dite *approche par division*. On peut citer les algorithmes [61] : *Agglomerative Hierarchical Clustering*, *Nearest Neighbor Algorithm*, *Furthest Neighbor Algorithm* et *Stepwise Optimal Hierarchical Clustering*. On peut aussi citer l'algorithme plus récent DYSECT [4].

- *les méthodes basées sur les graphes* (voir [61]) : le principe est de construire un graphe dont les sommets sont les observations. Deux observations seront liées par un arc si la distance qui les sépare est inférieure à un certain seuil. Les classes sont alors définies comme les composantes connexes de ce graphe. Avant de déterminer ces composantes connexes, on procède à l'élimination d'arcs inconsistants, définis comme les arcs dont la longueur est bien supérieure à la moyenne des arcs qui partent de ce sommet.

- *les méthodes basées sur un mélange de densités* : une de ces méthodes sera décrite dans le chapitre 6. Le principe est d'utiliser un mélange de densités (par exemple un mélange de gaussiennes), et d'associer à chaque observation la classe qui maximise la probabilité *a posteriori* d'appartenance, soit $P(\omega_i/y)$.

Les méthodes de prétraitement en reconnaissance de formes. Tout type de prétraitement peut être utilisé avant d'appliquer les techniques de reconnaissance de formes. On peut grossièrement en distinguer deux classes : les *méthodes statistiques* et les *méthodes de traitement du signal.*
Parmi les méthodes statistiques, une particulièrement utilisée est l'analyse en composantes principales (ACP) (par exemple [59]) que nous détaillerons et utiliserons dans le chapitre 6. Mais d'autres techniques peuvent être appliquées, notamment les *projection pursuits* ou *projections révélatrices* [74, 73, 35, 36]. Il y a aussi toutes les méthodes d'extraction de caractéristiques simples d'un signal (moyenne, écart-type...) présentées précédemment.
Une grande partie des méthodes de traitement du signal peuvent aussi être utilisées pour extraire des caractéristiques, notamment le calcul des autocorrélations et intercorrélations, la *transformée de Fourier*, les *analyses temps-fréquence* (Gabor-Ville, ondelettes...). La *théorie de la transformée en ondelettes* [132, 44, 43] est assez récente et fort prometteuse dans de multiples domaines où les signaux n'ont pas de stationnarité dans leur spectre. Cette théorie a de multiples applications : étude de signaux tels que la parole, compression d'informations, étude d'images [30], analyse des turbulences en mécanique des fluides [130]... On peut enfin citer les méthodes de recherche de raies pour l'analyse des modes vibratoires, entre autres les méthodes de *Prony* et *Pisarenko*. On peut bien sûr utiliser toutes les méthodes d'identification paramétriques de l'automatique et traiter les estimateurs ou les résidus par une approche de reconnaissance de formes (notamment la modélisation ARMA pour les processus stationnaires).

1.2.4 Diagnostic et réseaux de neurones

Les réseaux de neurones peuvent être décrits comme un ensemble d'unités simples (les neurones) connectées entre elles en un réseau à couches, comme le montre la figure 1.2. Il existe aussi des réseaux dits bouclés mais nous ne les aborderons pas dans cette thèse. Chaque unité calcule la somme de ses entrées pondérées par les coefficients synaptiques et réalise une certaine fonction d'activation, en général non linéaire, pour calculer sa sortie : une fonction à seuil (neurone binaire) ou une sigmoïde (approximation continue d'un neurone binaire).

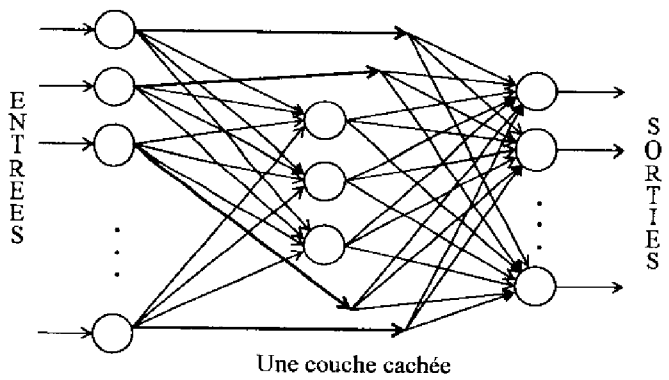

FIG. 1.2: Un réseau de neurones à couches

La spécificité du réseau réside dans la valeur des poids synaptiques. Ce sont des coefficients attribués à chaque connexion liant deux neurones et qui sont déterminés grâce à un apprentissage. Nous décrirons précisément ces techniques dans le chapitre 3 où nous ferons un état de l'art des méthodes d'apprentissage supervisé des réseaux de neurones à couches, qui sont les plus utilisées dans le diagnostic. Bien que le domaine d'application de prédilection des réseaux de neurones dans le diagnostic soit les approches de type reconnaissance de formes, donc les approches externes, ils sont aussi utilisés dans les deux autres catégories de diagnostic.
Dans les méthodes symboliques, des études ont été faites pour implanter un système expert avec un réseau de neurones. Ce type d'implantation est un palliatif à l'explosion combinatoire de la recherche à effectuer par le moteur d'inférence [153]. De plus, les approches symboliques par la logique floue peuvent aussi s'intégrer sous la forme d'un réseau de neurones flous.
Dans les méthodes de diagnostic interne, les réseaux de neurones sont actuellement beaucoup étudiés [28, 111, 82]. Ils peuvent notamment être utilisés pour l'identification, le contrôle et le diagnostic des systèmes non linéaires [97].
Dans les méthodes de diagnostic externe, nous avons vu que c'est une technique fondamentale principalement pour la classification multi-classes, mais aussi pour la classification non supervisée (par exemple [60]).

1.3 Le diagnostic de pannes automobiles

La construction automobile a été marquée ces deux dernières décennies par l'arrivée massive de l'électronique dans les véhicules. Ceux qui n'étaient que de petits modules avec quelques transistors (allumages transistorisés) sont maintenant de puissants calculateurs architecturés autour de microprocesseurs. Les faits les plus marquants sont sans doute l'injection-allumage électronique qui permet une grande optimisation du fonctionnement du moteur, et le système de freinage ABS qui a accru la sécurité. Depuis quelques années l'augmentation du nombre de calculateurs est permanente. La Safrane en possède déjà douze. Comme dans les avions de dernière génération, toute commande, tout contrôle passe par une puce en silicium. Il est évident qu'une telle évolution bouleverse le métier de mécanicien automobile, en particulier la phase de diagnostic. Il n'est plus question de tendre l'oreille à l'écoute du moindre bruit suspect comme le faisaient les bons mécaniciens après des années d'expérience. En parallèle à cette évolution, il était donc nécessaire de développer des outils de diagnostic de pannes performants, capable de trouver les moindres pannes dans cet imbriglio de câbles et de boîtes noires qu'est devenu le dessous d'un capôt. La société ACTIA s'est imposée sur ce marché, et a développé toute une gamme de produits de plus en plus puissants, ces produits devant suivre l'évolution de la complexité des véhicules.
Les calculateurs eux aussi de plus en plus performants, font maintenant leurs propres diagnostics qu'ils délivrent sur une ligne série, avec d'autres paramètres. On distingue donc deux types de diagnostic : le *diagnostic série* et le *diagnostic parallèle*. Le diagnostic série est la lecture et l'affichage des informations que délivrent les calculateurs. Il est donc restreint à ce que voit le calculateur et surtout à ce qu'il veut bien donner. Le diagnostic parallèle est lui beaucoup plus exhaustif mais aussi beaucoup plus complexe. Il est réalisé grâce aux informations qui sont captées entre les calculateurs et les organes moteurs mais aussi en d'autres points de mesure (par exemple une mesure de haute tension avec une pince pour le contrôle de la présence d'étincelles). Un outil de diagnostic parallèle doit donc traiter tous ces signaux et déterminer la panne s'il y en a une. La société ACTIA a développé un outil de diagnostic parallèle : le testeur universel (TU). Nous allons décrire les principes de son fonctionnement.
Le diagnostic réalisé par le TU repose sur deux types de méthodes : des méthodes simples de diagnostic externe pour la détection de défauts sur les divers signaux séparément et une méthode d'identification

similaire à un arbre de défaillances. Le module d'identification est appelé le Gestionnaire de Hiérarchie (GH).
Le TU reçoit une grande quantité de signaux principalement relatifs aux fonctions d'allumage et d'injection. Comme nous le verrons dans le chapitre suivant, ces signaux sont très hétérogènes et certains subissent des prétraitements. En général les défauts sont détectés avec de simples seuils. Il y a quelques exceptions qui nécessitent un algorithme particulier (comme la présence d'une étincelle sur chaque front montant d'un signal de commande). Le rôle du module d'identification est de déduire la panne avec les défauts élémentaires sur les signaux. La méthode est en fait un arbre de défaillances. Ce sont les défauts élémentaires sur les signaux qui constituent toutes les feuilles de l'arbre. Ainsi une liaison est faite entre des pannes (données symboliques) et leurs manifestations sur les signaux. Cet arbre est assez simple : à part quelques exceptions il comporte deux niveaux. La majorité des connecteurs logiques sont des *ou* (quelques pannes correspondent à des noeuds *et*). Cet arbre est construit par une expertise des systèmes diagnostiqués. Il nécessite une analyse fontionnelle des systèmes. La détection de certaines pannes nécessite aussi une étude expérimentale.
Les seuils de détection sont fixés aussi à partir d'une analyse des systèmes : par exemple si le signal issu du capteur de température d'eau a une dynamique de 1 à 2,5 volts pour des températures de -30 à $+40$ degrés, on pourra fixer les seuils à 0,5 et 3 volts.

1.4 Objectifs de l'étude

Du fait de l'augmentation des fonctionnalités gérées par l'électronique, la conception d'outils comme le TU devient de plus en plus difficile, du fait des informations symboliques nécessaires à leur conception qui doivent être fournies par les constructeurs. De plus, les défauts élémentaires sont détectés sur les signaux traités séparément, les détections sont mono-dimensionnelles. Enfin, le TU est particulièrement dédié au diagnostic des fonctions allumage-injection, les plus importantes dans le fonctionnement d'une voiture. Pour développer un produit sur un nouveau véhicule, en général il faut changer des tables de paramètres (pour les seuils, le GH et les prises de mesures). Mais pour des véhicules très différents (calculateurs "exotiques" ou passage d'une voiture à essence à une voiture au diesel) ou pour diagnostiquer d'autres fonctions il est très souvent nécessaire de modifier les programmes.

> *Le point de départ de ce travail est l'utilisation des Réseaux de Neurones Artificiels (RNA) et de leur capacité d'apprentissage pour réaliser un système générique, qui s'adapte à tout véhicule, à tout calculateur, et tout type de fonctions, sans avoir à modifier les programmes, mais seulement des paramètres, et sans avoir de connaissances symboliques sur les systèmes. Le principe d'un tel produit est donc un diagnostic externe par reconnaissance de formes qui n'utilise aucune modélisation mais seulement des observations en bon et en mauvais fonctionnement.*

Les avantages potentiels d'un tel système sont importants : diminution des coûts de développement, indépendance vis-à-vis des constructeurs, plus grande sensibilité de détection grâce aux réseaux de neurones, généricité du produit, potentialités pour le diagnostic préventif...
Il est évident qu'un tel objectif n'est réalisable qu'avec une plus grande puissance de calculs que celle disponible sur le TU (deux micro-contrôleurs). Actuellement les produits de diagnostic évoluent vers des micro-ordinateurs portables de type PC, donc avec une puissance de calcul suffisante (qui peut facilement évoluer) pour le système de diagnostic que nous nous sommes proposés de développer. En outre, la possibilité d'utiliser des langages de programmation évolués est un avantage considérable par rapport à l'assembleur utilisé sur les micro-contrôleurs : puissance, plus grande rapidité de développement, plus grande facilité de maintenance...

Enfin, dans la mesure du possible, on essayera de concevoir un système de diagnostic dont l'implémentation est assez rapide pour pouvoir effectuer le diagnostic en temps réel (comme le font certains des outils déjà existants) et dont la mise au point ne nécessite qu'un minimum d'intervention humaine : c'est-à-dire que l'on essayera d'automatiser un maximum d'étapes dans l'élaboration de l'outil de diagnostic, et notamment lors de la construction des réseaux de neurones.

Je me fais l'impression de n'avoir été qu'un enfant jouant sur la plage et s'y amusant à y trouver de temps en temps un galet particulièrement lisse ou un coquillage plus joli que les autres, tandis que s'étendait devant moi, inconnu, le grand océan de la vérité.

ISAAC NEWTON

Chapitre 2

Développement de la maquette ANDI

2.1 Introduction

Le développement de cette première maquette avait trois buts : mettre au point le système d'acquisition (aux niveaux matériel et logiciel de bas niveau), prouver l'intérêt des réseaux de neurones pour le diagnostic et concevoir un système de diagnostic parallèle entièrement paramétrable avec des observations, sans modélisation quelconque.
Cette maquette est constituée de deux organes : un boîtier d'acquisition des signaux et un PC pour les analyser et faire le diagnostic en temps réel ou en temps différé. D'un point de vue logiciel, le boîtier d'acquisition intègre un petit logiciel très générique et en partie commandable par le PC, qui sert uniquement à la conformation des signaux et à gérer la communication avec le PC. La partie logicielle du système est donc essentiellement constituée par ANDI (ANalyse et DIagnostic), qui fonctionne sur PC.
Pour bien traiter les nombreux signaux très hétérogènes issus des organes du véhicule et définir les prétraitements adéquats, le développement de plusieurs outils d'analyse en temps réel ou en temps différé parut nécessaire. La partie diagnostic est basée sur deux modules : l'un détecte des défauts dits élémentaires, c'est-à-dire un défaut d'un certain type sur un signal, ou un défaut sur plusieurs signaux détecté par un réseau de neurones, et le second détermine le défaut père, c'est-à-dire l'origine de la panne, la cause symbolique. Le premier module réalise donc la détection alors que le second réalise l'identification.
L'ensemble du logiciel ANDI est complètement paramétrable : les outils d'analyse comme les outils de diagnostic. Il n'y a pratiquement aucune ligne de programme à modifier pour passer d'un véhicule à l'autre. Le paramétrage du module de détection est réalisé après analyse des données de bon fonctionnement et le module d'identification est réglé avec des données de mauvais fonctionnement caractéristiques des pannes que l'on désire identifier. Les outils d'analyse peuvent être paramétrés suivant les besoins (les outils couramment utilisés sont paramétrés par défaut).
Après avoir décrit le système d'une manière très générale (la partie matérielle, le protocole de communication et les divers logiciels), nous détaillerons les diverses fonctionnalités du logiciel ANDI, et particulièrement les méthodes de détection de défauts élémentaires et la méthode d'identification. Avant de conclure ce chapitre, nous présenterons les essais réalisés avec cette maquette.

2.2 Description générale de la maquette

2.2.1 Le système vu de l'extérieur

Comme on peut le voir sur la figure 2.1 qui suit, le système se compose d'un boîtier d'acquisition qui est un TU (Testeur Universel : un produit ACTIA existant, modifié pour ne servir que de boîtier d'acquisition) lequel recueille les informations sur le véhicule, et d'un PC qui traite ces informations.

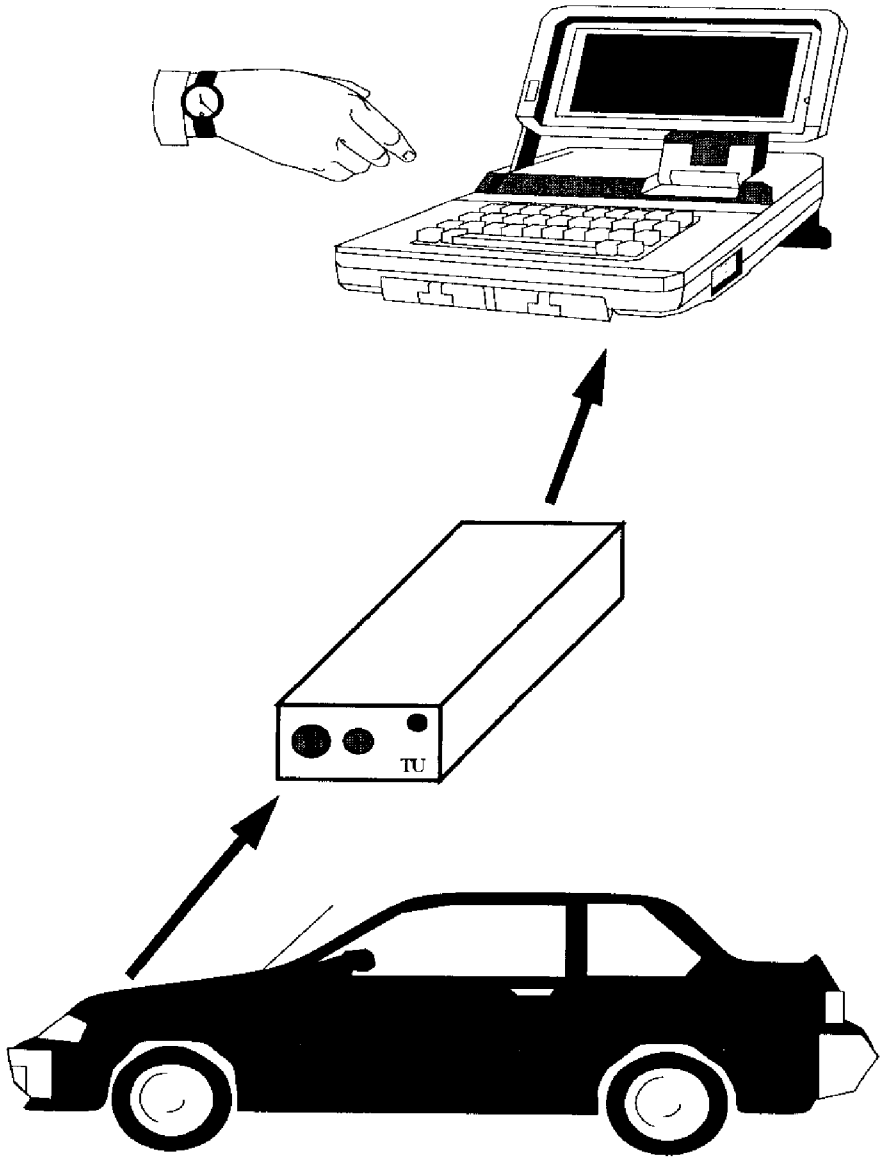

FIG. 2.1: Vue générale du système

2.2.2 La partie matérielle

Comme le montre la figure 2.2, les signaux sont collectés au niveau du moteur par des dérivateurs, sous haute impédance pour ne pas les altérer. Pour une configuration classique de diagnostic injection-allumage, il y a un dérivateur principal au niveau du calculateur où la majorité des signaux sont captés. Il y a aussi d'autres dérivateurs au niveau des MPA (Module de Puissance Allumage) et des captures d'informations diverses telle que la pince haute tension pour capter la présence des étincelles...

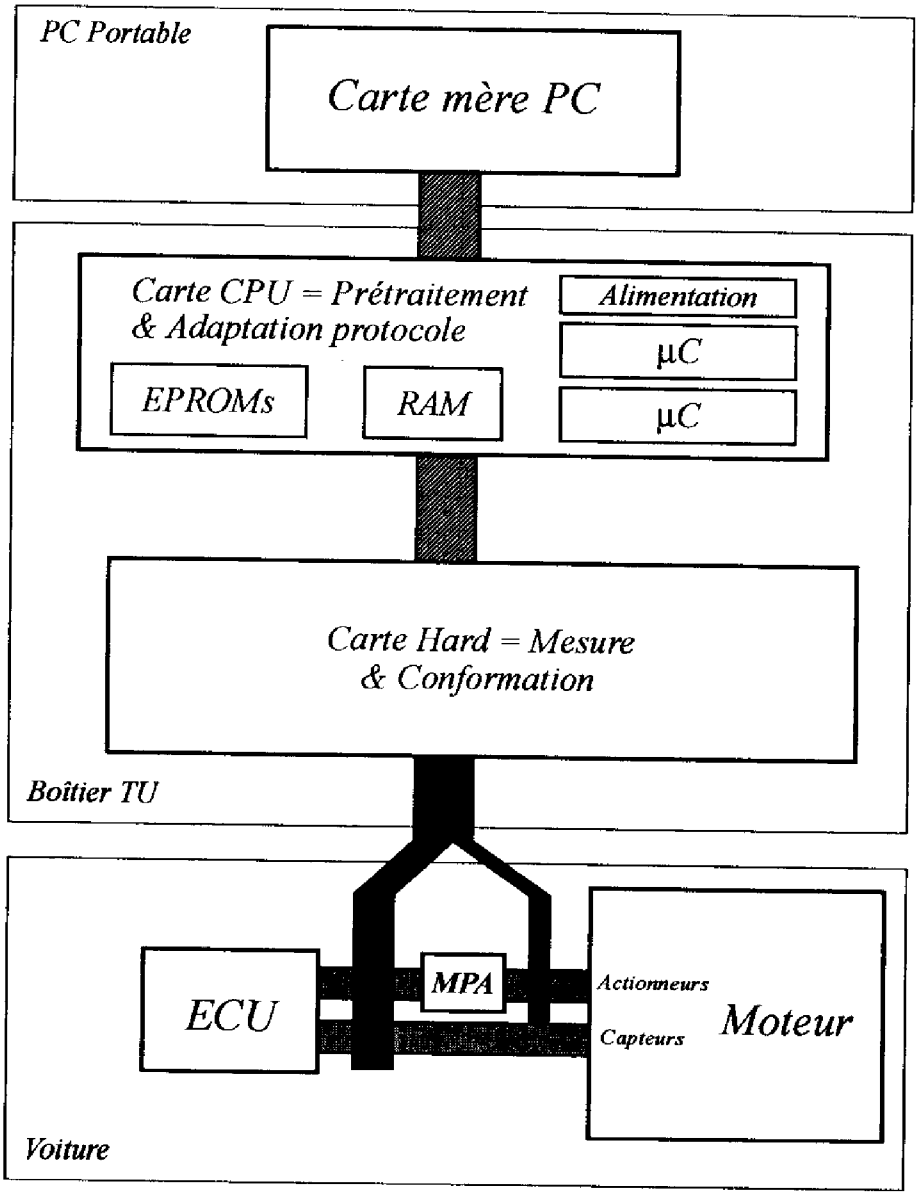

FIG. 2.2: Partie matérielle

La carte *Hard* du TU a pour but de conformer les signaux, de les numériser. Certains signaux analogiques sont numérisés par des CAN, d'autres sont rendus booléens... Cette carte ne comporte que de l'électronique passive, sans microprocesseurs.

Les informations numérisées de la carte *Hard* sont fournies à la seconde carte qui réalise certains prétraitements très simples (et le même quelque soit le véhicule), intègre ces informations dans la RAM double accès et gère le protocole de communication avec le PC. Cette seconde carte comporte entre autres deux microcontrôleurs, deux EPROMs (où sont stockés leurs programmes) et l'alimentation du TU.
Cette dernière carte est connectée au PC par un port parallèle. Le PC récupère les données suivant un protocole de communication que l'on détaillera ci-après, et les traite.

2.2.3 Les données captées & le protocole

Le TU envoie continuellement des informations au PC qui les utilise ou non selon les besoins de l'utilisateur. Il les envoie par paquets : les trames. Une trame comporte les informations sur ce qui s'est passé pendant un cycle moteur, soit deux tours moteur. En général, le TU capte les informations pendant deux tours moteur, puis les traite et les place dans la RAM double accès pendant un tour. Ce n'est pas rigoureusement exact car il peut y avoir des problèmes de synchronisation pouvant lui faire sauter un tour. La fréquence avec laquelle le TU envoie les trames est donc fonction du régime moteur, elle peut varier approximativement de 1 Hz à 35 Hz. Les contraintes temps réel sont donc assez dures : il faut faire le diagnostic en moins de 25ms pour ne pas manquer de trames.
Les données de la trame sont de quatre types :

- *les signaux analogiques* : température, pression, tension..., ces données ne variant pas très rapidement, le TU ne donne qu'une seule valeur par trame.
- *les signaux booléens* : alimentation démarreur, allumage, masses correctes ... , ces données sont en général des défauts qui sont déjà diagnostiqués électroniquement par la carte *Hard* du TU, par des comparateurs.
- *les signaux fréquentiels* : PMH (Point Mort Haut), commande allumage .... , ce sont des signaux booléens mais qui varient à des fréquences élevées dépendant du régime moteur. Le TU donne donc la valeur initiale, et les temps (en micro secondes) de tous les fronts. Le TU gère 5 signaux fréquentiels. Ce sont des signaux très importants car ils commandent tout le sytème d'injection et d'allumage. Le signal PMH (c'est un abus de langage, il se nomme en fait signal dent) est tout à fait particulier. Il est en fait généré avec une roue dentée reliée à l'arbre moteur qui possède une dent manquante. C'est ce signal qui permet à tout le système électronique de se synchroniser. Ce signal est prétraité par le TU, qui le simplifie et le vérifie.
- *les signaux spéciaux* : il y a enfin d'autres signaux assez divers tels que le signal du contrôle haute tension qui est en fait une succession de présence / non présence d'étincelles lors des fronts d'un des signaux fréquentiels, celui du primaire bobine. Il y a aussi sur la carte *Hard* deux entrées pour lesquelles le TU calcule la période du signal et son rapport cyclique, qu'il donne dans la trame...

Cette trame comporte enfin un compteur, qui est incrémenté à chaque nouvelle trame pour vérifier si on n'en a pas manqué, à cause d'une erreur de protocole ou parce que le traitement a été trop lent, et un octet qui sert dans le logiciel à mettre des marqueurs, c'est-à-dire des repères lors d'une acquisition par exemple.
La RAM double accès ne doit jamais être adressée en même temps par le TU et par le PC, il a donc fallu établir un protocole de communication pour se synchroniser et éviter tout conflit, et pour définir la manière dont le PC va envoyer les commandes de mesures au TU. Ce protocole est basé sur deux principes : la RAM est par défaut au PC, et le TU informe le PC par une interruption matérielle

lorsqu'il en a besoin (il en a besoin en fait uniquement lorsqu'il veut mettre une nouvelle trame, car il fait ses calculs en interne et prépare sa trame dans une autre RAM). Il a aussi fallu définir un système de resynchronisation, et un mode d'envoi de mots de commande au TU.
On peut symboliser le protocole sur la figure 2.3 qui suit :

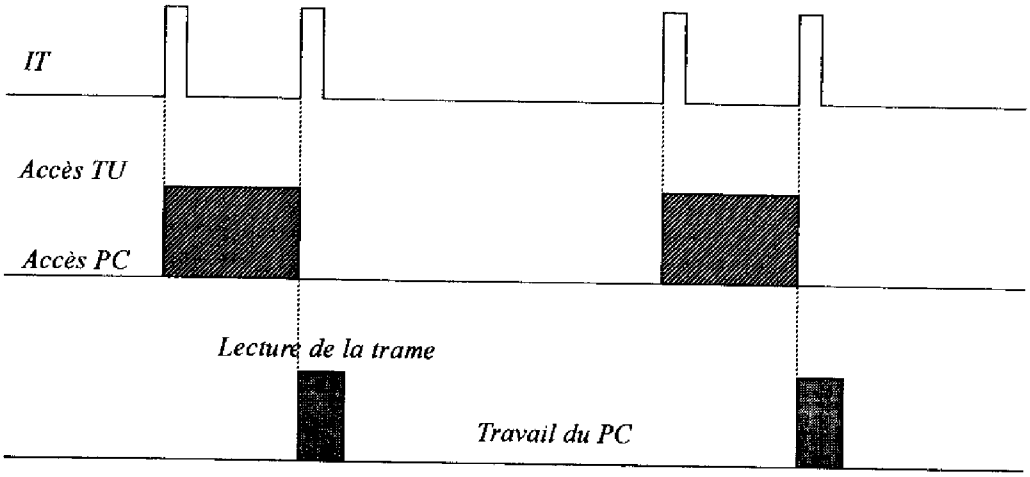

FIG. 2.3: Symbolisation du protocole de communication

2.2.4 Environnement logiciel

L'ensemble des logiciels du système est représenté sur la figure 2.4.

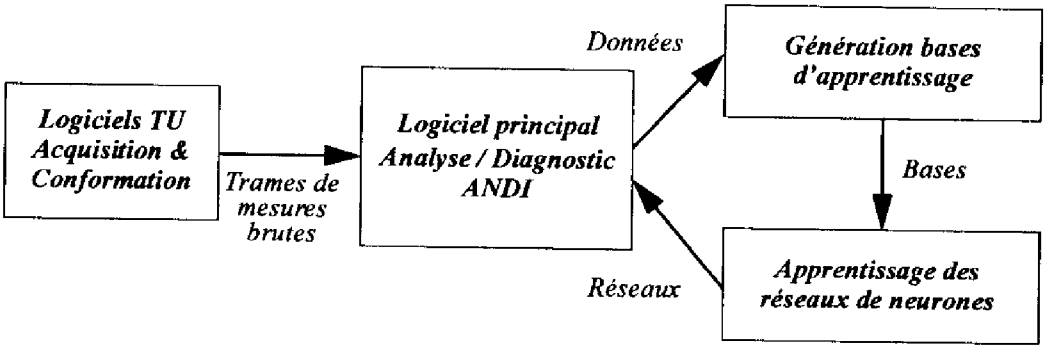

FIG. 2.4: Les diverses couches logicielles mise en jeu

Le TU comporte deux logiciels pour les deux microcontrôleurs. Ils sont écrits en assembleur, mais sont très généraux et ne devraient pratiquement jamais être modifiés. Les diverses options de mesures qui pourraient se présenter en passant d'un véhicule à l'autre sont envoyées sous forme de commandes au TU par le PC.
Sur PC trois logiciels ont été développés en C :

- Le logiciel principal d'Analyse / Diagnostic : ANDI.
- Un logiciel de création de bases de données pour l'apprentissage des réseaux de neurones à partir de données générées par le logiciel principal en temps différé.

- Un logiciel de création de réseaux de neurones, qui apprend les réseaux à partir des bases d'apprentissage générées par le précédent, en vue de les intégrer dans le module de diagnostic. L'apprentissage est réalisé avec la rétropropagation du gradient, qui fournit des réseaux de neurones continus. Nous détaillerons cet algorithme classique dans le chapitre 3.

2.3 Description du logiciel ANDI

2.3.1 Introduction

Les principaux modules du logiciel ANDI sont représentés sur la figure 2.5 qui suit :

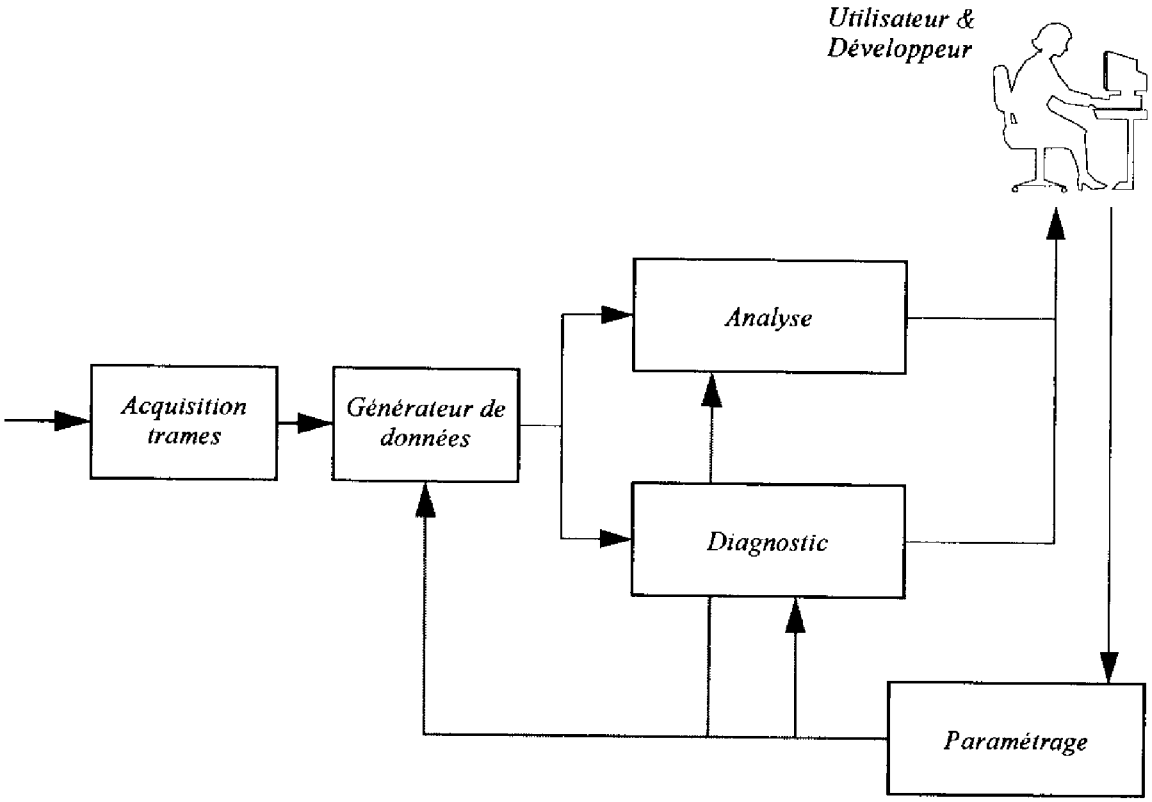

FIG. 2.5: Schéma général du logiciel ANDI

Ce logiciel est à la fois un outil d'analyse des données moteur et un outil de diagnostic de pannes, en temps réel et en temps différé. En effet, dès les premiers développements et essais sur véhicule, il nous a paru nécessaire de munir ce programme de toute une gamme d'outils d'analyse intégrés au logiciel, et surtout de rendre tous ces outils d'analyse ou de diagnostic complètement paramétrables, pour ne pas avoir à modifier le programme quand on passe d'un véhicule à un autre. Il y a donc toute une série de modules qui servent à paramétrer l'application.
Au départ de beaucoup de procédures, d'analyse ou de diagnostic, il faut extraire des informations spécifiques des trames envoyées par le TU. Il nous est donc apparu essentiel de centraliser ces calculs dans un module appelé le générateur de signaux. Cette centralisation permet ainsi de rendre le système très ouvert comme nous le verrons.
Tous les outils du logiciel accessibles par l'utilisateur peuvent donc être regroupés en quatre catégories :

- Les outils d'analyse
- Les outils de diagnostic
- Les outils de paramétrisation
- Les autres outils divers (gestion de fichiers, shell DOS, aide en ligne en hypertexte...)

Après avoir présenté les fonctionnalités du générateur de signaux, nous présenterons brièvement les différents outils d'analyse et plus précisément les outils de diagnostic avec les méthodes employées. Nous finirons cette section en décrivant l'utilisation des réseaux de neurones et son intérêt.

2.3.2 Le générateur de données

Introduction

Le but de ce module, représenté sur la figure 2.6, est d'extraire une information de la trame et si on le désire, lui faire subir un prétraitement (que l'on nommera post-calcul).

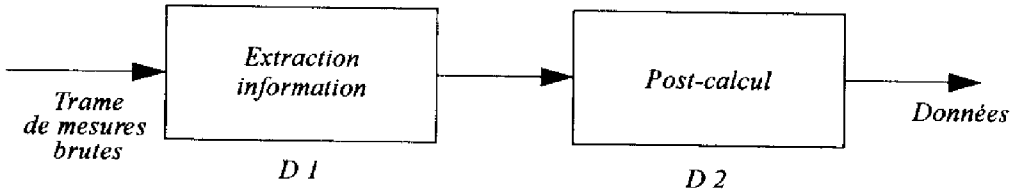

FIG. 2.6: Le générateur de données

Après avoir détaillé le cas très important des données extraites des signaux fréquentiels, nous décrirons de façon générale ces deux blocs.

Données extraites des signaux fréquentiels

Nous avons vu plus haut que les signaux fréquentiels sont envoyés dans la trame sous la forme d'un état initial et du temps d'apparition de chaque front (changement d'état). Le but est de transformer ces informations temporelles en informations atemporelles qui représentent au mieux ce qui s'est passé pendant le temps d'analyse, sachant qu'en général, s'il n'y a pas de problèmes, ces signaux sont assez réguliers sur un cycle moteur. Il s'agit d'une forme de compression d'informations.
Quatre caractéristiques de chaque signal fréquentiel sont calculées :

- le nombre de fronts filtrés : nbf_f
- la moyenne du rapport cyclique : $\overline{RC}$
- l'écart-type du rapport cyclique : σ_{RC}
- la phase par rapport au signal PMH : ϕ_{PMH}.

Le nombre de fronts filtrés est le nombre de fronts apparus uniquement pendant le cycle moteur, donc entre le premier et le dernier front du PMH. Dans le cas d'un moteur classique avec une roue dentée qui n'a qu'une dent manquante, représenté sur la figure 2.7, le signal PMH a toujours 5 fronts. Un tour moteur correspond exactement à un cycle du signal PMH et un cycle moteur correspond à deux cycles du signal PMH. Dans le cas d'un V6, la roue dentée a trois dents manquantes. Un tour moteur correspond à 3 cycles du signal PMH, et ce signal a alors 13 fronts pour un cycle moteur. Donc dans ce cas on prendra tous les fronts entre les fronts 1 et 13 du PMH (le nombre de dents manquantes est

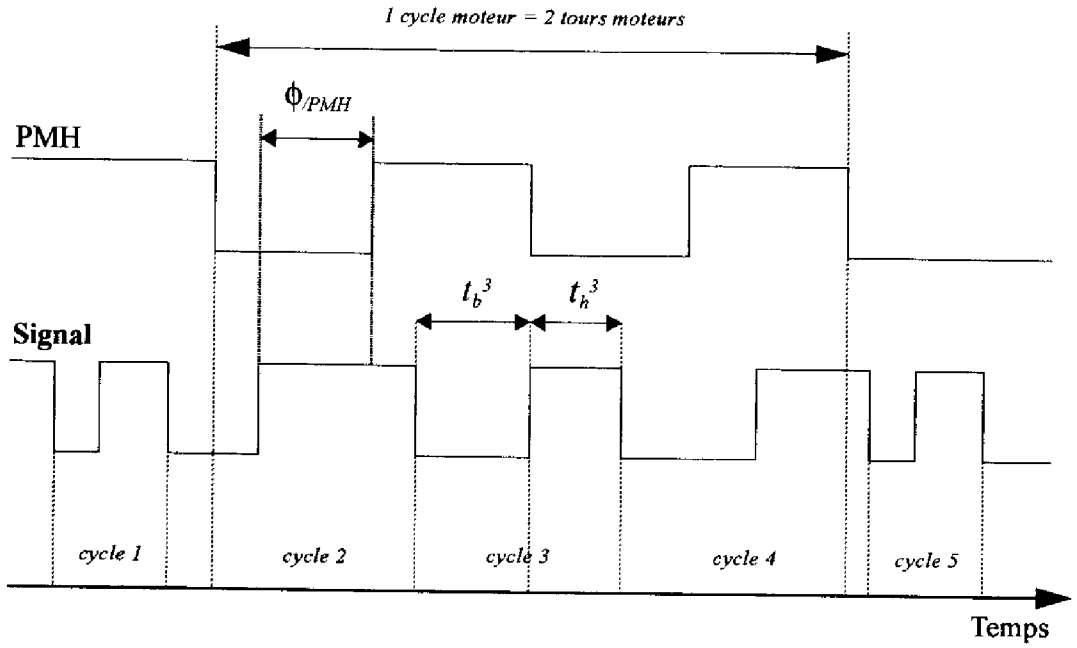

FIG. 2.7: Données extraites des signaux fréquentiels

un paramètre du programme). Ce nombre de fronts filtrés est une information très intéressante car il ne varie pratiquement pas, donc il permet déjà de détecter un grand nombre de défauts.
Pour un cycle du signal (trois fronts, succession palier haut / palier bas ou inversement, cela dépend de l'état initial du signal), on définit le rapport cyclique par la durée du palier haut divisée par la durée totale du cycle, multiplié par 100, pour obtenir un pourcentage :

$$RC_i = 100 \frac{t_h^i}{t_h^i + t_b^i} \tag{2.1}$$

Si n_c est le nombre de cycles du signal, on calcule alors la moyenne et l'écart-type du rapport cyclique par :

$$\overline{RC} = \frac{\sum_{i=1}^{n_c} RC_i}{n_c} \tag{2.2}$$

$$\sigma_{RC} = \sqrt{\frac{\sum_{i=1}^{n_c} (RC_i - \overline{RC})^2}{n_c}} \tag{2.3}$$

Ces deux quantités sont également intéressantes pour diagnostiquer un défaut sur le signal fréquentiel. L'écart-type du rapport cyclique doit être en temps normal très proche de zéro (empiriquement rarement supérieur à 3 sauf dans de très fortes accélérations où il peut monter jusqu'à 5). Ceci signifie que lorsque tout va bien, sur un cycle de mesure les cycles du signal sont constants. Mais dès qu'il y a un défaut, le signal peut perdre complètement cette régularité. Il suffit de fixer une limite acceptable et de considérer qu'au-delà il y a défaut sur ce signal.
La moyenne du rapport cyclique est constante uniquement pour le PMH (elle oscille légérement entre 49,5 et 50,5), mais absolument pas pour les autres signaux. En revanche, on peut observer généralement

qu'elle varie en fonction du régime (surtout pour les signaux relatifs à l'allumage). On peut fixer dans le logiciel un intervalle de valeurs acceptables. Mais comme nous le verrons plus loin, un diagnostic beaucoup plus fin sera réalisé avec les réseaux de neurones.
La phase du signal par rapport au PMH est définie comme le temps entre le signal 2 du PMH (donc toujours un front montant) et le front montant du signal le plus proche. Cette quantité que l'on norme (en pourcentage par rapport au cycle moteur complet) est signée (différence de deux temps). De même que la moyenne du rapport cyclique, on peut utiliser cette donnée pour diagnostiquer le signal en fixant un intervalle de valeurs acceptable, mais l'utilisation des réseaux de neurones sera préférable, beaucoup plus précise.

Remarque : Si le nombre de fronts du signal est pair, cela signifie que le dernier cycle est incomplet. Dans ce cas, on affine le calcul en considérant que le dernier palier, dont nous ne possédons que le début, est identique à l'avant-dernier de même type. Enfin si le nombre de fronts est inférieur strictement à 3, on force l'écart-type à une valeur que l'on peut paramétrer, et la moyenne à zéro.

Les données que l'on peut générer

Nous allons présenter toutes les données que l'on peut générer, en décomposant le diagramme D 1 de la figure 2.6 en sous-diagrammes successifs (le petit rond symbolise les choix possibles). La décomposition du bloc D 1 peut être représentée sur la figure 2.8.

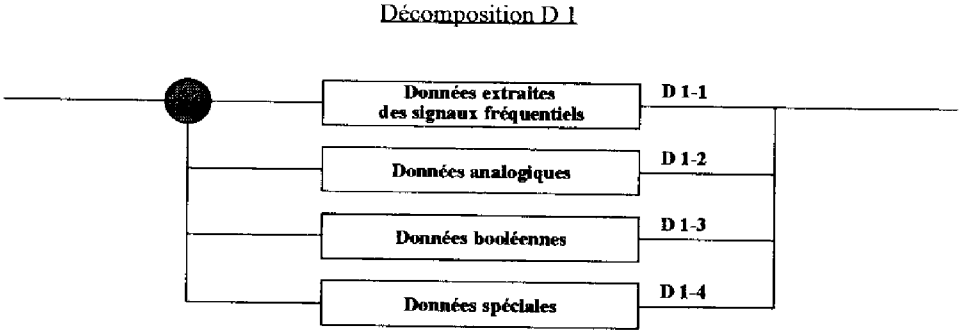

FIG. 2.8: Les divers types de données extraites des trames

La figure 2.9 représente la décomposition des blocs de niveaux 2 et 3.
Les données du bloc D 1-1 ont été décrites dans le paragraphe précédent. Dans la décomposition du bloc D 1-2, les données analogiques $\mu 1$ et $\mu 2$ correspondent aux signaux des entrées analogiques du TU. Pour le bloc D 1-3, les données des ports des micros correspondent aux différents ports de sortie des deux microcontrôleurs (pour ce bloc il faut spécifier l'octet et le numéro du bit que l'on veut extraire). Les données du bloc D 1-2-3 sont calculées avec les signaux fréquentiels : le PMH pour calculer le régime (en tours/min), et les signaux d'allumage pour l'avance (en degrés) et le temps de charge (en micro secondes).

Les post-calculs

Après avoir extrait une information de base, il est possible d'effectuer un post-calcul sur cette donnée. La figure 2.10 représente le détail du diagramme D 2 qui décrit les calculs implantés dans le logiciel.

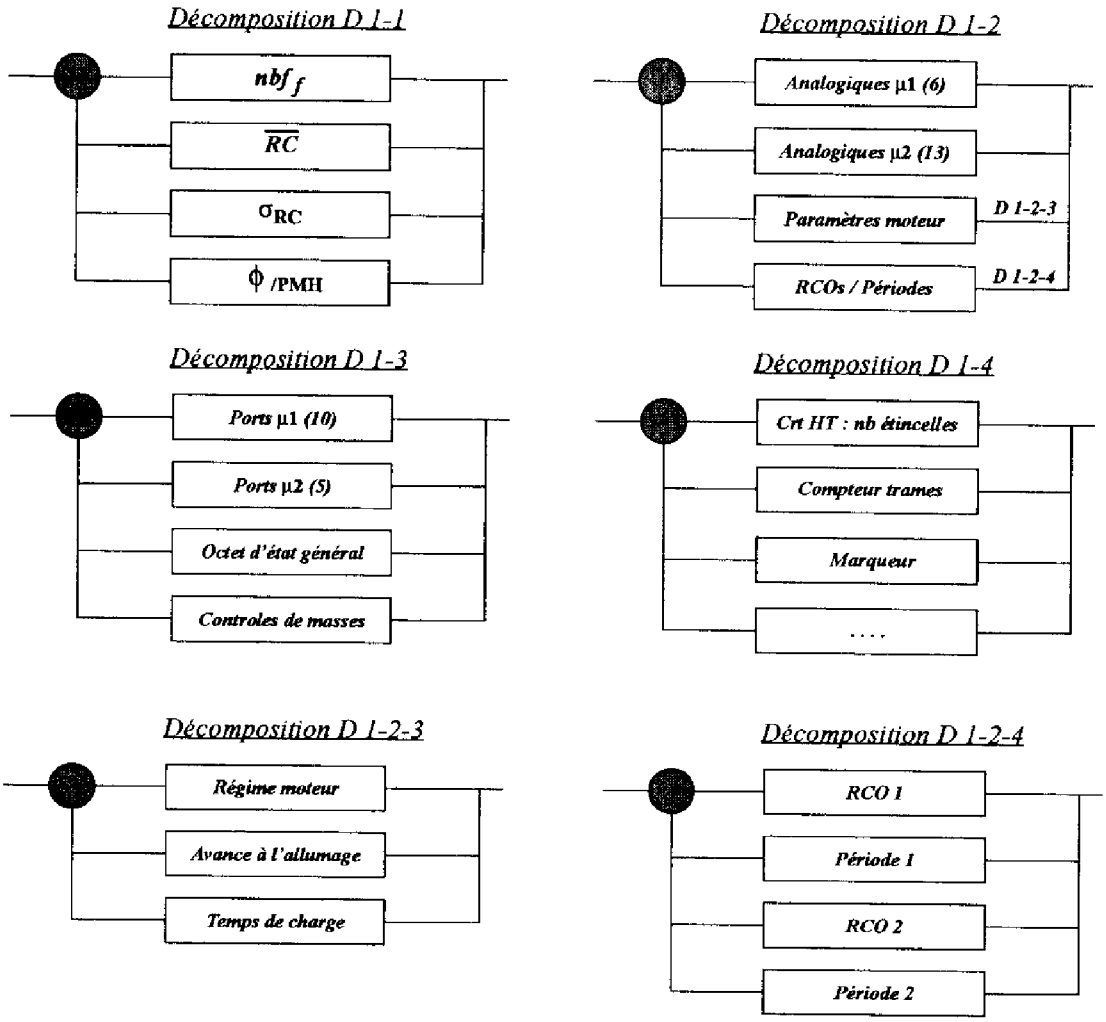

FIG. 2.9: Décomposition des blocs de niveaux 2 et 3

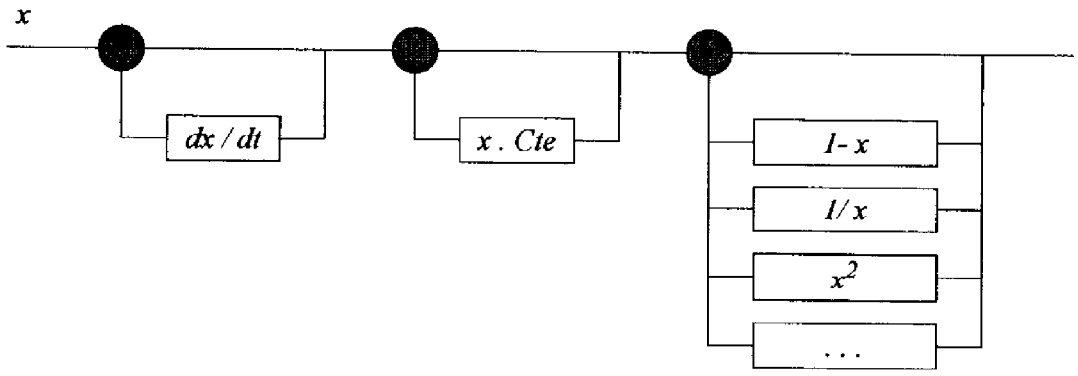

FIG. 2.10: Les post-calculs implantés

2.3.3 Les outils d'analyse

Il y a deux types d'analyse : celle du développeur d'application, et celle de l'utilisateur qui veut observer le comportement d'une certaine donnée moteur. L'analyse du développeur devra se faire sur un PC puissant à partir de fichiers de trames acquis sur le véhicule, grâce à deux outils d'acquisitions que l'on détaillera plus loin. L'ensemble des fichiers de trames devra couvrir au mieux toutes les conditions de fonctionnement de la voiture (charge, type de route, température, vitesse ...), pour pouvoir fixer les seuils le mieux possible et apprendre les réseaux de manière précise.
Il y a quatre outils d'analyse :

- Visualisation de type oscilloscope (voir figure 2.18),
- Tracé de signaux en fonction du temps (voir figure 2.19),
- Tracé d'un signal en fonction d'un autre,
- Etude statistique : estimation de Pdf (Probability density function) (voir figure 2.20).

Comme le montrent les figures 2.11 et 2.12, l'oscilloscope et l'estimation de Pdf fonctionnent en temps réel et en temps différé, alors que les tracés de signaux ne fonctionnent qu'en temps différé. L'oscilloscope est un outil très important dans la première phase d'analyse du fonctionnement d'une voiture et il était impératif qu'il puisse fonctionner en temps réel sans manquer une seule trames de données. Ces contraintes temps réels ont imposées une programmation particulière : toutes les données sont calculées directement à partir des trames sans passer par le générateur de données, qui prend un peu plus de temps. Du fait de ces contraintes de temps, l'oscilloscope en temps réel ne permet pas d'enregistrer en même temps les trames, il n'a pas assez de temps. En revanche il travaille sur un *buffer* tournant d'une centaine de trames (4 secondes au plus haut régime) que l'on peut sauvegarder pour analyser en temps différé.

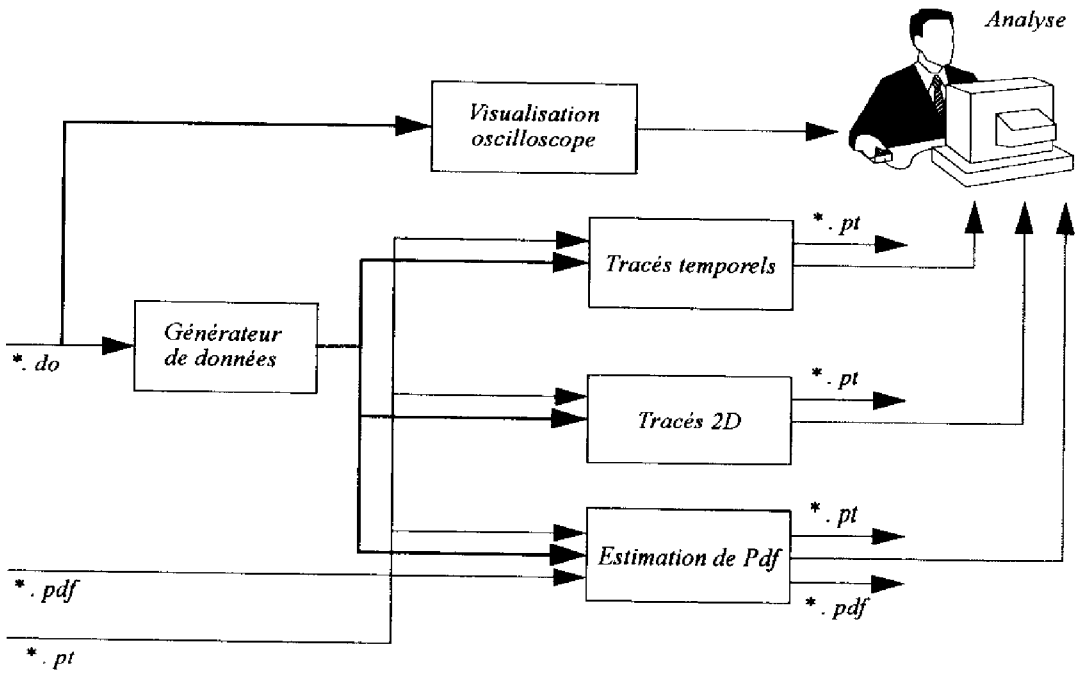

FIG. 2.11: Les outils d'analyse en temps différé

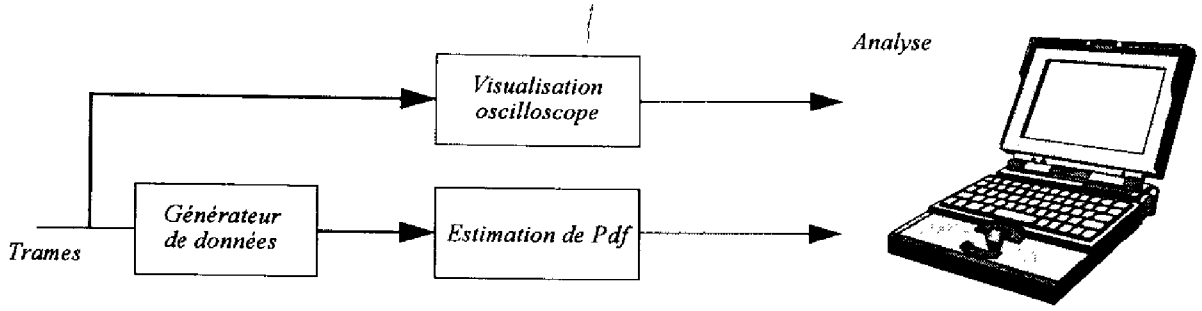

FIG. 2.12: Les outils d'analyse en temps réel

Comme on peut le voir sur la figure qui suit, les outils d'analyse en temps différé peuvent être utilisés de plusieurs manières, avec plusieurs entrées :
Les tracés temporels et les tracés 2D peuvent fonctionner soit à partir d'un fichier de trames, soit à partir d'un fichier de points (*.pt). Ces fichiers de points sont générés par ces même outils si on le désire. Cela évite d'effectuer plusieurs fois les mêmes calculs sur les mêmes trames. La génération des fichiers de points peut se faire suivant deux formats : un format avec en-tête pour les réutiliser dans ANDI, et un format sans en-tête si on veut les utiliser dans d'autres logiciels (grapheurs, tableurs...).
Le format de ces fichiers est tout à fait classique : format ASCII avec des tabulations comme séparateurs des données dans un vecteur, et un retour chariot pour séparer les vecteurs.
L'analyse statistique calcule la moyenne, l'écart-type, et une approximation de la fonction de densité de probabilité d'une donnée. La qualité de cette approximation dépend du nombre de pas de discrétisation choisi. Cet outil peut générer un fichier (*.pdf), où sont stockées toutes les informations nécessaires pour tracer cette Pdf : le nombre d'éléments dans chaque intervalle, la moyenne et l'écart-type exact. Ces fichiers sont écrits en ASCII, dans un format un peu spécial, mais sont facilement réutilisables.
Avant de poursuivre, il convient de signaler que les fichiers sur lesquels travaille le logiciel sont de 5 types :

- *.cnf : ce sont les fichiers de configuration (ASCII)
- *.do : ce sont des fichiers de trames (Binaire)
- *.pt : ce sont des fichiers de points (ASCII)
- *.pdf : ce sont des fichiers de Pdf (ASCII)
- *.hlp : ce sont des fichiers d'aide (ASCII)

2.3.4 Les outils de diagnostic

Généralités

Une panne réelle, due à la défaillance d'un organe mécanique, électrique ou autre peut générer un ensemble de défauts élémentaires. Quand on a une certaine combinaison de défauts élémentaires, il faut pouvoir déterminer l'origine de ces défauts : le défaut père. De plus, des défauts peuvent apparaître aléatoirement à cause de bruits de mesure, ou de perturbations diverses. Avant de déclarer qu'il y a eu effectivement un défaut, il faut filtrer. A un instant donné, un défaut non filtré est dit présent, et un défaut filtré est dit permanent.
Deux outils de diagnostic, nommés simple et complet, sont utilisables soit en temps réel, soit en temps différé.
Le premier détecte les défauts élémentaires, les filtre mais ne génère pas de défauts pères. En revanche il calcule des informations statistiques sur l'apparition des défauts élémentaires : il calcule la probabilité

d'apparition de chaque défaut sur l'ensemble de l'expérience (ou du fichier si on travaille en temps différé) et dans une fenêtre glissante. Il calcule aussi la moyenne et l'écart-type des intervalles de temps entre deux apparitions successives, ce qui peut être intéressant pour analyser des défauts qui apparaissent peu mais régulièrement (voir figure 2.21).
Le second filtre les défauts élémentaires et calcule les défauts pères, grâce au GH (Gestionnaire de Hiérarchie). A la fin d'un diagnostic complet, on informe l'utilisateur de l'apparition comparative des défauts pères par un histogramme (voir figure 2.22), pour lever toute ambiguïté s'il y en a.
La méthode de détection des défauts pères peut donc être symbolisée par le schéma de la figure 2.13.

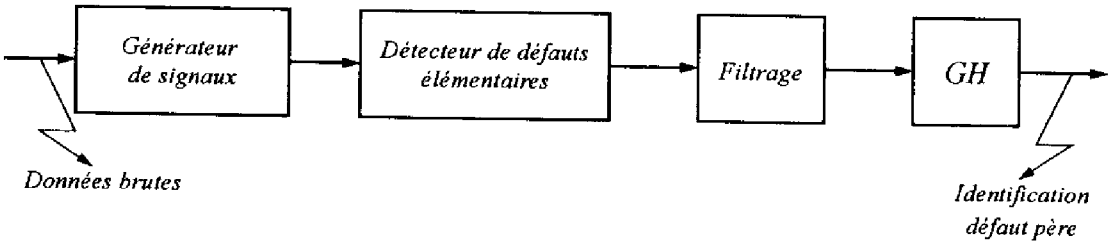

FIG. 2.13: Principe de détection des défauts pères

Les défauts élémentaires

Il y a 6 catégories de défauts élémentaires :

- *les défauts sur les signaux fréquentiels* : l'utilisateur peut définir un défaut élémentaire pour chacun des 5 signaux fréquentiels de la trame. La détection de ce défaut peut se faire en testant une ou plusieurs des données extraites (expliquées plus haut). Tester le nombre de fronts non filtrés est moins intéressant que le nombre de fronts filtrés, ce dernier étant plus constant. L'utilisateur va donc choisir une combinaison de tests sur : le nombre de fronts filtrés, la moyenne du rapport cyclique, l'écart-type du rapport cyclique et la phase. Si une de ces caractéristiques est en défaut, alors on dira qu'il y a un défaut élémentaire sur ce signal fréquentiel. Pour tester le nombre de fronts filtrés, le module de détection vérifie si celui-ci est égal à une des valeurs acceptables, lesquelles auront été définies après analyse. Pour la moyenne du rapport cyclique et la phase, il y a défaut si la valeur n'est pas comprise dans un intervalle défini par l'utilisateur (avec un seuil bas et un seuil haut), alors que pour l'écart-type il n'y a qu'un seuil haut, le seuil bas étant 0. Ces signaux sont de loin les plus difficiles à diagnostiquer, particulièrement celui d'injection.

- *les défauts sur les signaux analogiques* : l'utilisateur peut définir un défaut élémentaire pour chacun des signaux analogiques définis dans le générateur de données. Il y a deux options de détection : sur la valeur et sur la valeur absolue de sa dérivée. Pour la première, un défaut est déclaré si la valeur n'est pas dans un intervalle défini par l'utilisateur (seuil bas - seuil haut), et pour la seconde, si la valeur absolue de la dérivée est supérieure à un seuil, le seuil bas étant mis à zéro.

- *les défauts sur les signaux booléens* : un défaut élémentaire peut être défini pour chaque donnée booléenne définie dans le générateur de données. La mise en défaut se fait sur un état 0 ou 1 selon ce qui est défini par l'utilisateur. Celui-ci peut aussi demander d'effectuer un premier filtrage pour cette catégorie de défauts élémentaires, en entrant un nombre de mises à défaut successives égal à n. Le module de détection attendra d'avoir n mauvaises valeurs successives avant de déclarer un défaut présent.

- *les défauts spéciaux* : c'est la catégorie dans laquelle on a classé les détections particulières qui n'ont pas de procédures génériques. Par exemple, le contrôle haute tension par pince. Il faut réaliser une détection avec un algorithme spécialisé. Celui-ci va vérifier que pour chaque front montant du signal primaire bobine il y a une étincelle, et qu'il n'en existe pas d'autres non synchronisées. C'est le seul cas rencontré pour l'instant. L'implémentation logicielle a été assez générale, pour pouvoir définir facilement de nouveaux défauts spéciaux et programmer facilement leur algorithme de détection.

- *les défauts détectés avec les réseaux de neurones* : c'est la catégorie de défauts qui nous intéresse particulièrement. Ce principe de détection a été implanté d'une manière très générale. Le principe est de définir un réseau de neurones avec comme entrées des données quelconques issues du générateur de données (en nombre quelconque) et en sortie le réseau répond défaut ou fonctionnement normal. Nous détaillerons plus loin cette méthode de détection.

- *les défauts sur un signal quelconque* : cette catégorie, implantée en dernier lieu, est aussi très générale. On peut définir un défaut sur une donnée quelconque issue du générateur de données. L'avantage de ce type de défauts est que l'utilisateur peut définir plusieurs intervalles (seuil bas - seuil haut) de validité. Le diagnostic est donc beaucoup plus fin dans le cas d'une donnée qui ne varie pas de façon continue. Le défaut sera déclaré si la donnée ne se trouve dans aucun des intervalles.

Le filtrage

Le filtrage consiste à attendre que le défaut apparaîsse suffisamment de fois avant de déclarer qu'il y a effectivement un défaut : s'il persiste on le nommera *permanent*. Un défaut qui existe à un instant est nommé défaut *présent*. Le but est d'éviter les bruits de mesure.
Le principe est très simple, il consiste à dire qu'il y a un défaut permanent D_P si le nombre d'apparitions de ce défaut pendant une certaine période est supérieur à un seuil. En terme de probabilités cela consiste à dire que la probabilité d'apparition du défaut D_{Pr} dans une fenêtre glissante est supérieure à un certain seuil. Si on note par 0 et 1, respectivement l'absence et la présence d'un défaut (présent ou permanent), alors on a

$$D_P(t) = 1 \iff \frac{\sum_{i=t-T+1}^{t} D_{Pr}(i)}{T} \geq P_a$$

La largeur de la fenêtre glissante T est paramétrable, ainsi que la probabilité d'apparition Pa.
On peut visualiser ce principe sur le schéma de la figure 2.14. Les graphiques représentent la présence ou l'absence d'un défaut en fonction du temps. Les deux premiers correspondent au défaut présent et le dernier au défaut permanent. Sur le premier graphique, le défaut n'a pas été présent suffisamment longtemps dans la fenêtre glissante, donc il n'est pas permanent au temps t_1. Sur le second graphique, il a été assez persistant, sa probabilité d'apparition dans la fenêtre dépasse le seuil de 50%, il est donc déclaré permanent au temps t_2.
Un défaut qui aura été permanent mais qui ne l'est plus est appelé *fugitif*. On appellera enfin un défaut qui est devenu permanent lors d'une expérience, et qui l'est toujours resté jusqu'à la fin, un défaut *établi*.

Remarques

- Ce principe de filtrage est une forme de filtrage passe-bas.

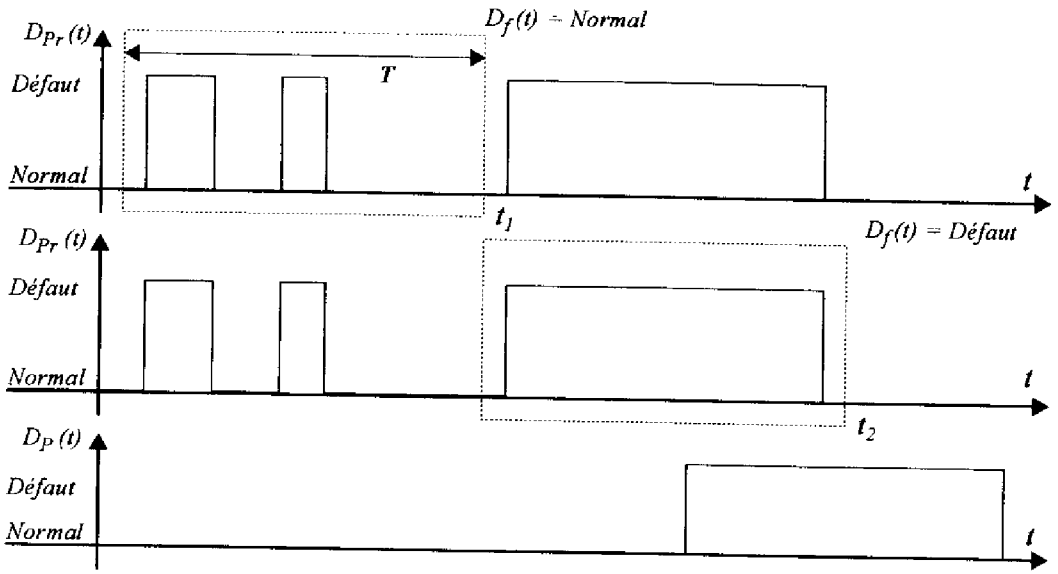

FIG. 2.14: Principe du filtrage

- La notion de défaut permanent est relative à une fenêtre glissante.
- Comme on peut le voir sur le troisième graphique de la figure qui suit, il y a un phénomène d'inertie. Le défaut peut être encore permanent alors qu'il n'est plus présent. La probabilité peut être supérieure au seuil sans que le défaut soit présent. On peut éliminer ce phénomène en exigeant de plus que le défaut soit présent pour le déclarer permanent (ET logique du premier et du troisième graphique). Mais ce n'est pas forcément intéressant.

Gestionnaire de Hiérarchie

Une fonction mécanique ou électronique de haut niveau qui est défaillante va engendrer des défauts sur tout un ensemble de signaux élémentaires. Il faut donc pouvoir déterminer à partir d'un ensemble de défauts élémentaires les défauts pères responsables. Mais une combinaison donnée de défauts élémentaires peut être engendrée par plusieurs défauts pères. Il faut donc déterminer parmi ceux-ci le véritable responsable : c'est le rôle du Gestionnaire de Hiérarchie (GH).
Le GH fonctionne donc en deux phases. Une première phase détermine tous les défauts pères susceptibles d'être responsables, et une seconde phase calcule ceux qui sont les véritables pères, donc élimine les défauts pères qui sont engendrés par d'autres pères, qui sont des conséquences et non des causes.
Définitions :

- $\xi = \{e_1, \ldots, e_n\}$ l'ensemble de tous les défauts élémentaires que l'on détecte,
- $\mathcal{P} = \{p_1, \ldots, p_n\}$ l'ensemble des défauts pères définis,
- Les défauts pères sont définis comme des combinaisons de défauts élémentaires, des sous-ensembles de défauts élémentaires, donc nous avons $\forall i \; p_i \subset \xi$,
- l'ensemble $\mathcal{P}$ est muni d'une relation d'ordre partiel, pour traduire les dépendances hiérarchiques. Nous aurons alors (p_i fils de p_j) $\Longleftrightarrow$ ($p_i \subset p_j$). Ce qui veut dire que lorsqu'un père p_i est fils de p_j, donc engendré par p_j, les défauts élémentaires que génère p_i sont aussi engendrés par p_j,

- E est un diagnostic élémentaire, c'est-à-dire l'ensemble des défauts élémentaires qui sont présents à un instant donné,
- $\underline{P}$ est l'ensemble des pères potentiellement responsables, c'est le résultat de la première phase de calcul,
- P est l'ensemble des pères responsables, ce que l'on cherche.

Première phase
Pour calculer $\underline{P}$, il suffit de prendre tous les pères dont les défauts élémentaires sont inclus dans l'ensemble des défauts élémentaires courants E, on peut donc écrire :

$$\underline{P} = \{p_i \in P / p_i \subset E\}$$

Seconde phase
Cette phase est aussi très simple, il suffit d'enlever de $\underline{P}$ tous les pères qui sont fils d'un autre père, qui est lui aussi potentiellement responsable. Donc mathématiquement cela s'écrit :

$$P = \underline{P} \ / \ \{p_i \in \underline{P} \ / \ \exists p_j \in \underline{P} \text{ avec } (p_i \text{ fils de } p_j)\}$$

Le principe de GH est illustré par un exemple sur la figure 2.15.

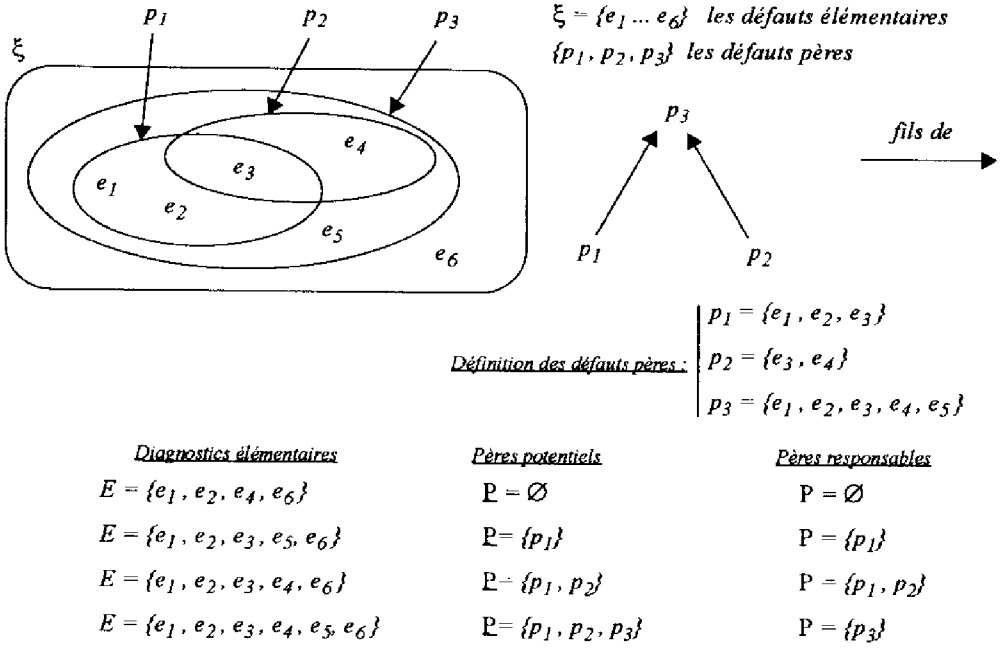

FIG. 2.15: Gestionnaire de Hiérarchie : exemple didactique

Remarque
Cette description de la méthode est un peu simplifiée par rapport à celle effectivement implantée dans le logiciel. En réalité on peut associer plusieurs vecteurs de défauts élémentaires à un père, ceci pour essayer de traduire les états transitoires à l'établissement d'un défaut père, et la non unicité des vecteurs de défauts élémentaires pour un défaut père...

Cette méthode d'identification est une forme d'arbre de défaillances. Cet arbre est plutôt un graphe orienté. Les pannes sont détectées avec des combinaisons de défauts élémentaires. Mais il y a de multiples différences entre ce nouveau GH et celui développé pour le TU. La différence la plus importante est la génération automatique des liens de dépendance entre les pannes. La hiérarchie est calculée grâce à l'hypothèse combinatoire que nous avons faite sur la notion de filialité. De plus la structure de cet arbre peut être quelconque, elle dépend des diverses combinaisons de défauts élémentaires qui définissent les pères. La première étape recherche les pères potentiels. La seconde étape de la recherche de pannes a été formalisée de manière combinatoire comme la fermeture transitive de l'ensemble des pères potentiels par rapport à la relation binaire de filialité. Le calcul de ces causes d'un point de vue graphe consiste à partir de tous les pères trouvés par la première étape et à remonter le graphe en ne passant que par des causes potentielles jusqu'à ce que ce ne soit plus possible. Les véritables pannes sont en définitive tous les points d'arrêt de cette remontée. Contrairement à celui du TU, le paramétrage de ce nouveau GH ne nécessite aucune information sur les systèmes, ce qui est un avantage primordial.

2.3.5 Les réseaux de neurones dans ANDI

L'utilisation des réseaux de neurones dans ce système pose un gros problème : celui de la constitution de la base d'apprentissage, ceci parce qu'il est impossible d'acquérir une base d'apprentissage équilibrée avec assez d'informations, c'est-à-dire des frontières entre le fonctionnement normal et le fonctionnement avec défauts bien définis. Pourquoi ?
L'ensemble des points de bon fonctionnement peut être bien défini. Par contre, le mauvais fonctionnement est en fait défini comme tout ce qui n'est pas du bon fonctionnement (voir figure 2.16). Il est donc beaucoup plus difficile de créer des points de fonctionnement avec défauts de manière à entourer correctement l'espace de bon fonctionnement, pour avoir des frontières précises. Les défauts que l'on peut générer sont trop francs et peu nombreux. Il est donc impossible de construire une base d'apprentissage correcte de manière purement empirique.

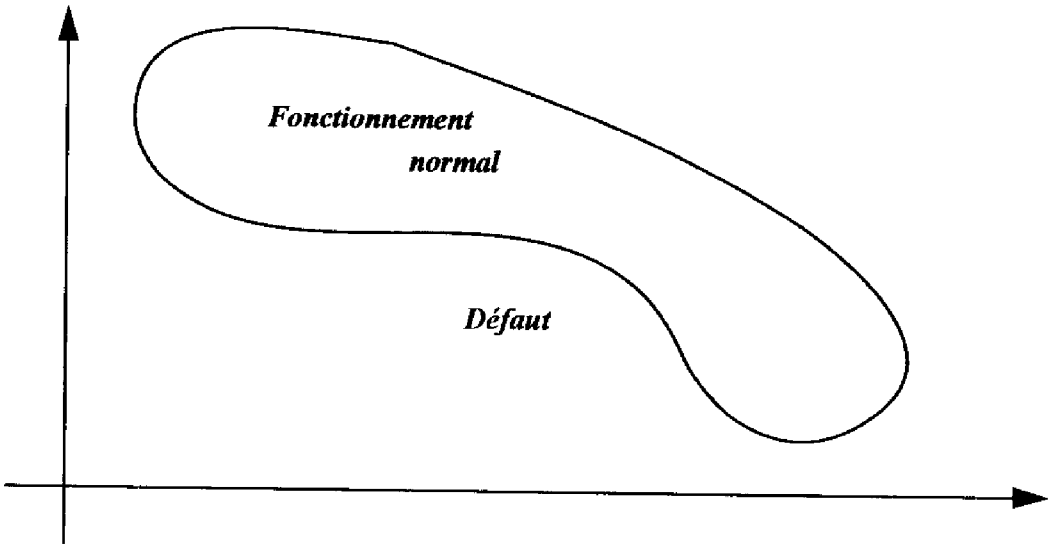

FIG. 2.16: Le mauvais fonctionnement défini comme complémentaire du bon fonctionnement

Au début de cette étude nous pensions résoudre ce problème en faisant l'hypothèse suivante : l'en-

semble des points de bon fonctionnement est convexe (c'est-à-dire sans trous). Dans cette hypothèse, il semblait possible de créer une base d'apprentissage satisfaisante en calculant l'enveloppe convexe de ce nuage et en générant arbitrairement des points de mauvais fonctionnement tout autour de cette enveloppe. On aurait pu en plus limiter le nombre de points de bon fonctionnement et avoir ainsi une base d'apprentissage parfaitement renseignée et équilibrée. Mais l'expérience a montré que les nuages de fonctionnement normal sont rarement convexes. Cette première idée sera reprise dans la méthodologie présentée dans le chapitre 6, mais avec un traitement préliminaire important pour règler le problème de non-convexité des nuages de points.
Dans ce premier système, nous avons alors contourné le problème pour les RNA à deux entrées. Dans ce cas on peut visualiser graphiquement l'ensemble des points de bon fonctionnement et créer manuellement la base d'apprentissage très rapidement. C'est ce que nous avons implanté. Voici la succession des étapes à suivre pour créer un défaut sur 2 signaux corrélés avec les RNAs :

- choisir les deux données que l'on veut diagnostiquer,
- générer un fichier de points qui couvre au maximum l'ensemble de bon fonctionnement,
- visualiser cet ensemble dans le logiciel de génération de bases,
- créer la base d'apprentissage manuellement (outil logiciel avec souris),
- créer un embryon de réseau (permet de dimensionner de manière presque optimale, d'initialiser une partie des poids et de faciliter l'apprentissage),
- apprendre le réseau avec le logiciel d'apprentissage,
- définir le défaut dans ANDI et intégrer le réseau à la configuration.

Les expériences menées dans ANDI avec les réseaux de neurones ont montré le grand intérêt du diagnostic multi-dimensionnel réalisé par les réseaux, même en deux dimensions. La figure 2.17 en est un exemple. Sur ce graphique, nous avons tracé le temps de charge des bobines en fonction du régime moteur pour 42000 trames de bon fonctionnement (les petits points) et un ensemble de trames de mauvais fonctionnement correspondant à une panne générée artificiellement sur l'alimentation après relais (APR) (les plus). Comme on peut le voir, le temps de charge dépend du régime, et cette dépendance suit une loi non évidente qui doit faire intervenir d'autres variables. Mais la zone de bon fonctionnement dans ce plan n'est pas du tout quelconque. Elle semble très bien définie. C'est une forme complexe et absolument pas convexe. Les points de mauvais fonctionnement sont en dehors de la zone de bon fonctionnement (les plus dans la zone de bon fonctionnement correspondent au début et à la fin de l'expérience où la panne n'est pas présente mais où on enregistre les données). Un tel défaut est impossible a détecter avec des méthodes monodimensionnelles, car le temps de charge n'est pas possible à ces régimes mais il est courant à des régimes plus élevés. Le diagnostic multidimensionnel est donc nécessaire pour détecter ce défaut. Un réseau de neurones a alors été conçu avec ces deux variables, et il s'est montré très sensible à ce défaut qui n'était pas détecté autrement.
Le problème de la génération de bases d'apprentissage n'a pas été résolu pour les réseaux à plus de deux entrées. Ces réseaux sembleraient bien adaptés au diagnostic de l'injection par exemple, où il est nécessaire de corréler de multiples signaux. Il est donc nécessaire de s'engager dans des recherches théoriques.
Les essais sur véhicule ont aussi révélé un autre problème : l'utilisation des réseaux de neurones continus (nécessitant le calcul d'une fonction de transfert assez complexe) utilise trop de temps de calcul. Le temps de calcul avec seulement trois réseaux (totalisant environ 50 neurones) devient rapidement trop important et on manque des trames.
Une solution à ce problème est l'utilisation de neurones binaires qui ne peuvent être constitués qu'avec les algorithmes de construction (sauf des cas simples avec le perceptron) que nous aborderons dans le chapitre suivant.

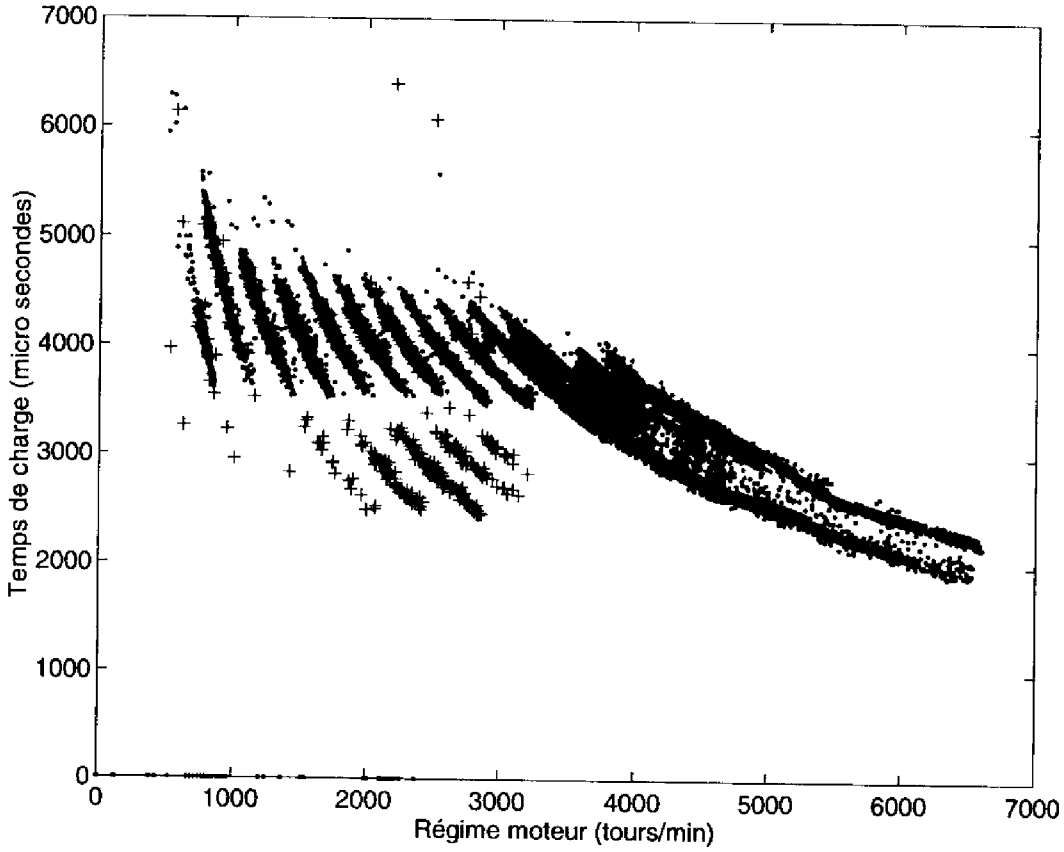

FIG. 2.17: Intérêt du diagnostic multi-dimensionnel

2.4 Les essais réalisés

2.4.1 Motronic, Lada & ZX

La totalité du logiciel a pu être développée sur un PC classique. La valise Motronic (qui est un système qui simule une injection avec un calculateur de marque Motronic) nous a servi à développer l'embryon du logiciel : la communication avec le TU et l'acquisition de données ; et a servi constamment à tester les outils de détection, temps réel. Les essais sur la Lada et la ZX ont montré l'intérêt des outils d'analyse et surtout la nécessité de paramétrer au maximum le logiciel.
Une première validation de l'ensemble du système a été réalisée sur la valise Motronic (détection élémentaire, GH...) et a fourni d'excellents résultats.

2.4.2 Peugeot 605V6

Les essais sur ce véhicule avaient pour but de valider l'ensemble du système. Bien que le passage d'un système artificiel à un véhicule réel révèle une autre complexité, on peut considérer les résultats comme assez satisfaisants. Voici comment s'est déroulée cette campagne d'essais.
Après avoir branché le système, et paramétré les ordres de mesures à envoyer au TU, nous avons enregistré des données à l'arrêt. Nous avons commencé à paramétrer les divers outils d'analyse (définition des données, des tracés temporels et 2D intéressants...). Ensuite nous avons roulé sur autoroute,

en ville et en campagne en essayant de couvrir toute la gamme de vitesses. Nous avons pu enregistrer 42000 trames, soit environ 1 heure d'enregistrement. Après avoir fait une première analyse des données (répartition statistique et recherche de corrélations), nous avons branché des *buggers* en série avec les prises de signaux. Ce sont des sytèmes d'interrupteurs pour générer des défauts (on peut faire des circuits ouverts, à la masse, au plus de l'alimentation...). Nous avons généré toute une série de défauts en enregistrant en même temps. Avec ces derniers fichiers nous avons pu définir les défauts élémentaires, régler leurs seuils de détections, définir les réseaux de neurones... Nous avons enfin défini les défauts pères et leurs combinaisons de défauts élémentaires, et vérifié la détection de ces défauts pères.
Après avoir détaillé les défauts élémentaires (au nombre de 29) de la configuration que nous avons définie, nous présenterons les 18 défauts pères, et les résultats de l'identification des pannes, avec une note entre 1 et 10.

Description des défauts élémentaires

Les défauts sur signaux fréquentiels :

- FR1 Référence commande allumage
- FR2 Signal d'injection
- FR3 Variation commande allumage
- FR4 Variation primaire bobine
- FR5 Signal PMH

Les défauts sur signaux analogiques :

- AN1 Tension batterie
- AN2 Richesse
- AN3 Pression collecteur
- AN4 Température eau
- AN5 Température air
- AN6 Position papillon
- AN7 Alimentation capteur pression
- AN8 Alimentation capteur papillon
- AN9 Rapport cyclique 2 (RCO2)
- AN10 Période 2

Les défauts sur signaux booléens :

- BO1 Problèmes dents
- BO2 Second enroulement EVR
- BO3 Chauffe sonde O2
- BO4 Commande relais puissance
- BO5 Alimentation après relais (APR)
- BO6 Tension Batterie Permanent (Vbat)
- BO7 Alimentation allumage
- BO8 Masse puissance
- BO9 Référence masse
- BO10 Masse allumage

Les défauts spéciaux :

- SP1 Contrôle haute tension par pince

Les défauts détectés avec réseaux de neurones :

- NN1 Avance = f(régime)
- NN2 Temps de charge = f(régime)
- NN3 Phase primaire bobine = f(régime)

Remarques

- la détection de défauts sur le signal PMH a été définie parmi les défauts sur signaux fréquentiels. Ces défauts sont normalement déjà détectés par le TU (défaut BO1), mais on s'est aperçu que l'on détectait des défauts non détectés par BO1 (des trames où le nombre de fronts était correct mais leur apparition non régulière).

- les réseaux de neurones que l'on a intégrés à ces essais ont montré leur puissance de détection. Ainsi on a pu détecter des défauts très fins sur l'avance, le temps de charge et le primaire bobine, qu'il était impossible de détecter avec un système de seuils monodimensionnels. La dépendance du régime, même si elle n'est pas toujours claire comme une droite ou un ensemble de droites, est présente pour ces trois signaux et en tenir compte apporte une grande finesse du diagnostic.

Description des défauts pères

Voici les défauts pères qui ont été définis, avec la note attribuée estimée subjectivement en tenant compte de la difficulté de mise au point et de la qualité d'identification, c'est-à-dire si la panne a été identifiée sans équivoque.

- P01 Contact Coupé : 6/10
- P02 Moteur arrêté : 6/10
- P03 Electrovanne : 9/10
- P04 Panne d'essence : 6/10
- P05 Commande relais puissance : 7/10
- P06 Alimentation potentiomètre papillon : 6/10
- P07 Vbat permanent : 10/10
- P08 Capteur température eau : 10/10
- P09 Capteur température air : 10/10
- P10 Papillon, curseur potentiomètre : 7/10
- P11 Alimentation allumage, MPA : 10/10
- P12 Commande allumage : 10/10
- P13 Primaire bobine : 10/10
- P14 Injecteurs : 8/10
- P15 Masses : 1/10
- P16 Référence masse eau + air : 9/10
- P17 Référence masse pression + papillon : 9/10
- P18 Alimentation capteur pression : 9/10

Le défaut le plus délicat à détecter est le défaut de masse. Lorsque l'on créé un défaut sur les masses (puissance et référence) séparément, il n'y a aucun défaut élémentaire généré. Il faut donc créer un défaut sur les deux en même temps (circuits ouverts). Mais dès que l'on crée ces défauts, la voiture cale. Il faut donc les créer brièvement. Ainsi il n'y a pas de permanence dans la combinaison de défauts générés. De plus ces défauts génèrent beaucoup de défauts élémentaires. Il est donc très difficile de définir des combinaisons pour le GH, et on a des interférences avec les défauts pères P1 et P2. Normalement ces deux défauts de masses sont traités par le TU, mais le temps de présence de ces défauts n'était vraissemblablement pas assez important, car le TU ne les détectait que rarement.

Les défauts P7, P8 et P9 sont simplement une copie des défauts élémentaires, d'où une parfaite détection.
Les trois défauts pères relatifs à l'allumage P11, P12 et P13 sont détectés parfaitement.
Le défaut P4 a été défini suite à l'enregistrement de données de fonctionnement normal pendant lequel nous sommes tombés en panne d'essence. Après avoir analysé les données enregistrées pendant cette panne, on s'est aperçu que pendant les 30 secondes qui ont précédé la panne, les réseaux de neurones ont détecté des défauts sur le temps de charge et sur l'avance à l'allumage (en fait on avait un retard à l'allumage). On a donc essayé de définir un défaut père qui détecte une panne d'essence juste avant qu'elle n'arrive. On a une bonne détection pendant les 30 secondes avant la panne. Mais le problème est que ce défaut est généré pendant des transitoires d'autres défauts pères car il est assez simple, il y a peu de défauts élémentaires dans sa définition.
Le défaut P14 sur les injecteurs fut assez difficile à mettre au point. En fait on a affiné le diagnostic élémentaire du signal fréquentiel. Ce défaut père est ainsi une simple copie du défaut élémentaire.
La détection des autres défauts pères a assez bien fonctionné.

2.5 Conclusion

Le développement de cette maquette s'est basé sur un produit ACTIA de diagnostic parallèle déjà existant et largement distribué chez les garagistes. La partie matérielle a été un peu modifiée pour pouvoir servir essentiellement de centrale d'acquisition d'un maximum d'informations et les communiquer au PC pour leur traitement. Une telle association nous a donc ouvert la voie au développement d'outils beaucoup plus complexes que ceux développés en assembleur sur des microcontrôleurs : taille mémoire, rapidité, espace disque, langage évolué... Certaines informations sont déjà traitées au niveau du TU. Celui-ci délivre alors des informations booléennes qui sont déjà des défauts élémentaires (ce sont des contacts, des contrôles de masses réalisés avec des comparateurs, la validité du signal dent...). D'autres signaux sont simplement numérisés, ce sont les données analogiques. Les 5 signaux fréquentiels sont très importants car ils commandent l'injection et l'allumage. Les données correspondantes envoyées au PC sont donc très précises.
Par rapport au système de diagnostic existant, nous avons développé de nouvelles méthodes de détection de défauts élémentaires et une nouvelle méthode d'identification. Sachant que les signaux fréquentiels sont assez réguliers sur un cycle moteur, la détection de défauts par le calcul de l'écart-type du rapport cyclique s'est montrée très pertinente. Ce simple test statistique a pu détecter des défauts sur le signal PMH non détectés par le TU qui normalement détecte ce type de défaut. Le fait de pouvoir tester plusieurs caractéristiques d'un signal (2 pour les signaux analogiques et 4 pour les signaux fréquentiels) offre une large gamme de sensibilité de détection.
L'introduction des réseaux de neurones, qui permettent de prendre en compte les relations entre plusieurs signaux, accroît de manière substancielle la puissance de détection du système. Ils peuvent détecter des défauts impossibles à détecter avec des méthodes monodimensionnelles. Ils sont donc particulièrement intéressants.
Le Gestionnaire de Hiérarchie a été développé de manière à pouvoir être paramétré avec des observations du système en défaillance. Son principe est un peu similaire à celui du système existant au niveau de sa représentation mais sa génération est complètement automatique à partir d'observations de mauvais fonctionnement. De plus son exploitation est différente.

Cette première étude a donc abouti à une maquette d'un système Analyse - Diagnostic complètement paramétrable par expérimentation, sans modification de programmes. Mais plusieurs problèmes liés au Gestionnaire de Hiérarchie et à l'utilisation des réseaux de neurones restent à résoudre.

Pour le paramétrage du GH, pour des défauts francs générés artificiellement, on a réussi à définir une configuration assez satisfaisante. Mais avec des défauts sur des voitures réellement en panne, le système peut ne pas répondre correctement. La raison en est simple. En définissant le fonctionnement du GH de manière combinatoire, en fonction des effets des défauts francs générés artificiellement, celui-ci est complètement dépendant de ces défauts, et il semble extrêmement difficile de créer un ensemble de défauts représentatifs de toutes les pannes réelles possibles de manière purement empirique. De plus le principe combinatoire de la définition des dépendances pose de gros problèmes d'interférence lors des transitoires. Ce système fonctionne bien lorsque les défauts sont permanents. Cette méthode ne paraît donc pas idéale.

En ce qui concerne l'utilisation des réseaux de neurones, de nombreux problèmes ont été soulevés. Premièrement la génération automatique d'une base d'apprentissage pertinente est un problème important qui doit être résolu. Une méthode automatique doit être étudiée, qui puisse fonctionner en dimension quelconque. Deuxièmement, comme nous l'avons signalé dans le paragraphe sur les essais, les temps de calcul des réseaux de neurones continus peuvent être longs quand le nombre de neurones devient important. Le système ne respecte alors plus les contraintes temps réel. Il est vrai que pour faire du diagnostic de pannes automobiles, ce n'est pas un problème fondamental, on peut enregistrer les données et faire le diagnostic en temps différé. Mais si c'est possible, il vaut mieux avoir un système temps réel. Dans cette optique, *les réseaux de neurones binaires* apparaissent plus intéressants. Un second problème plus délicat posé par l'utilisation de réseaux de neurones continus appris par la rétropropagation du gradient ou des algorithmes d'optimisation du second ordre (voir chapitre 3) est l'automatisation de l'apprentissage. En effet comme nous le verrons dans le chapitre suivant, ces algorithmes posent de nombreux problèmes (choix de la structure, détermination des paramètres d'apprentissage, lenteur de convergence ou convergence dans un minimum local) qui nécessitent la supervision de l'apprentissage par un expert du domaine. Il semble assez difficile de réaliser un système d'apprentissage entièrement automatisé. Et si c'était réalisable, le processus serait certainement très long. Les réseaux de neurones intégrés dans ANDI lors des essais ont été appris avec la rétropropagation. Malgré une bonne initialisation et une structure de réseau proche de l'optimum (réalisées grâce à l'outil graphique de génération de bases et d'embryons de réseaux), l'apprentissage a été relativement long et délicat. Comme nous le verrons dans le chapitre suivant, une solution à ces problèmes est l'utilisation des *algorithmes de construction.* Finalement le dernier problème, qui est en fait le plus important, est celui de la qualité de la réponse du réseau : la généralisation. Celle-ci doit être parfaite, c'est-à-dire que le réseau doit répondre bon fonctionnement uniquement à l'intérieur du nuage de points et mauvais fonctionnement uniquement à l'extérieur. En général, avec un réseau de neurones continus la sortie du réseau n'est parfaitement connue que dans le voisinage des points constituant la base d'apprentissage. Donc pour satisfaire à cette contrainte avec les modèles classiques (même appris avec les quelques algorithmes de construction qui existent pour ces modèles) il est pratiquement nécessaire de couvrir de points tout l'espace dans lequel les variables diagnostiquées peuvent prendre leurs valeurs, ce qui n'est pas envisageable. Avec une base d'apprentissage contenant essentiellement des points de la frontière ou des points de bon fonctionnement, une solution est d'utiliser des réseaux de neurones à activité localisée : les *réseaux de neurones à base radiale.* Dans le cadre des algorithmes de construction de réseaux de neurones binaires, aucun algorithme ne permet de respecter cette contrainte. Une autre solution serait de générer le réseau de manière purement géométrique à partir de la base d'apprentissage.

Finalement il nous a semblé plus facile de développer un algorithme satisfaisant à ces contraintes dans le cadre des réseaux de neurones binaires. Mais comme nous le verrons dans le chapitre 6, les trois solutions seront en fait utilisées : réseaux de neurones à activité localisée construits par un procédé particulier, réseaux de neurones binaires construits grâce à un nouvel algorithme développé dans le

chapitre 5 et la solution géométrique sera utilisée temporairement en cas de non-convergence de ce dernier.

Enfin, l'utilisation des réseaux de neurones est intéressante lorsque les signaux mis en relation ont une certaine corrélation (plus ou moins évidente). Dans la maquette, les trois réseaux ont été conçus en faisant une recherche de corrélation par une analyse visuelle, grâce aux outils d'analyse d'ANDI. Mais il convient de se pencher sur une méthode moins empirique.

Nous verrons dans le dernier chapitre comment ces problèmes ont été résolus. Mais avant cela, après le paragraphe suivant qui présente quelques aspects du logiciel ANDI, nous allons présenter un état de l'art succinct et non exhaustif de l'apprentissage des réseaux de neurones à couches et les développements théoriques réalisés dans ce domaine.

2.6 Quelques aspects du logiciel ANDI

2.6.1 Oscilloscope

Comme on peut le voir sur la figure 2.18 cet outil permet de visualiser une grande partie des informations de la trame, et surtout les signaux fréquentiels. Cette fenêtre est divisée en 4 parties. La première qui occupe la majeure partie de l'oscilloscope est l'affichage des signaux fréquentiels. Cet affichage qui n'est pas très précis car en mode texte (il est nécessaire de respecter les contraintes temps réel à haut régime) est toujours cadré sur un cycle moteur (sauf exceptions pour lesquelles il y a des problèmes avec le signal PMH). Le signal Variations Primaire Bobine est un peu spécial, c'est sur ce signal que sont symbolisées les étincelles détectées par la pince haute tension (les petites étoiles sur les fronts montants). La seconde fenêtre contient les données analogiques, la troisième les données booléennes et la dernière des données diverses. A part pour les signaux fréquentiels, l'affichage des informations est entièrement paramétrable.

2.6.2 Analyse temporelle de signaux

La figure 2.19 représente l'outil de tracé temporel. La fenêtre principale contient les signaux en fonction du temps. La fenêtre du bas, des informations suivant le mode de fonctionnement. Entre les deux une petite fenêtre indique de manière graphique quelle est la partie du fichier qui est actuellement visualisée, car on peut visualiser la partie que l'on veut avec un système de zoom. Un mode curseur permet de voir la valeur des signaux et de basculer sur l'oscilloscope en temps différé si on le désire. Le nombre de signaux peut aller de 1 à 7.

2.6.3 Tracé de densité de probabilité

La fonction de densité de probabilité est visualisée sous forme d'histogramme (figure 2.20). On peut donc voir la répartition de cette donnée sur l'intervalle que l'on désire. Cet outil a un seul mode qui est un mode curseur. Ceci permet de visualiser à une position donnée des informations sur les valeurs et les probabilités.

2.6.4 Outil d'analyse des défauts élémentaires

Cet outil fonctionne en temps différé et en temps réel (figure 2.21). De gauche à droite les informations visualisées sont :

- la liste des défauts élémentaires,

- un feu vert ou rouge pour visualiser le défaut présent,
- la probabilité d'apparition depuis le début de l'expérience ou du fichier,
- la probabilité d'apparition dans la fenêtre glissante (propre au défaut),
- la moyenne du temps séparant deux apparitions du défaut présent,
- l'écart-type de ce temps,
- le cumul du nombre d'apparitions depuis le début de l'expérience ou du fichier,
- la présence ou la non présence du défaut permanent, après filtrage.

2.6.5 Outil de diagnostic avec identification des pannes

Quand le diagnostic est en cours, cette fenêtre affiche les informations suivantes de gauche à droite :

- la liste des défauts élémentaires
- visualisation des défauts élémentaires présents (sans filtrage)
- visualisation des défauts élémentaires permanents (filtrage sur les défauts élémentaires)
- la liste des défauts pères
- visualisation des défauts pères présents (sans filtrage)
- visualisation des défauts pères permanents (filtrage sur les défauts pères)

A la fin de l'expérience de diagnostic avec cet outil, le logiciel affiche l'apparition des défauts pères présents par un histogramme. Le défaut qui est apparu le plus souvent définit la dynamique du système de barres. Ceci est utile pour lever les ambiguïtés lorsque il y a des interférences entre les défauts pères (figure 2.22).

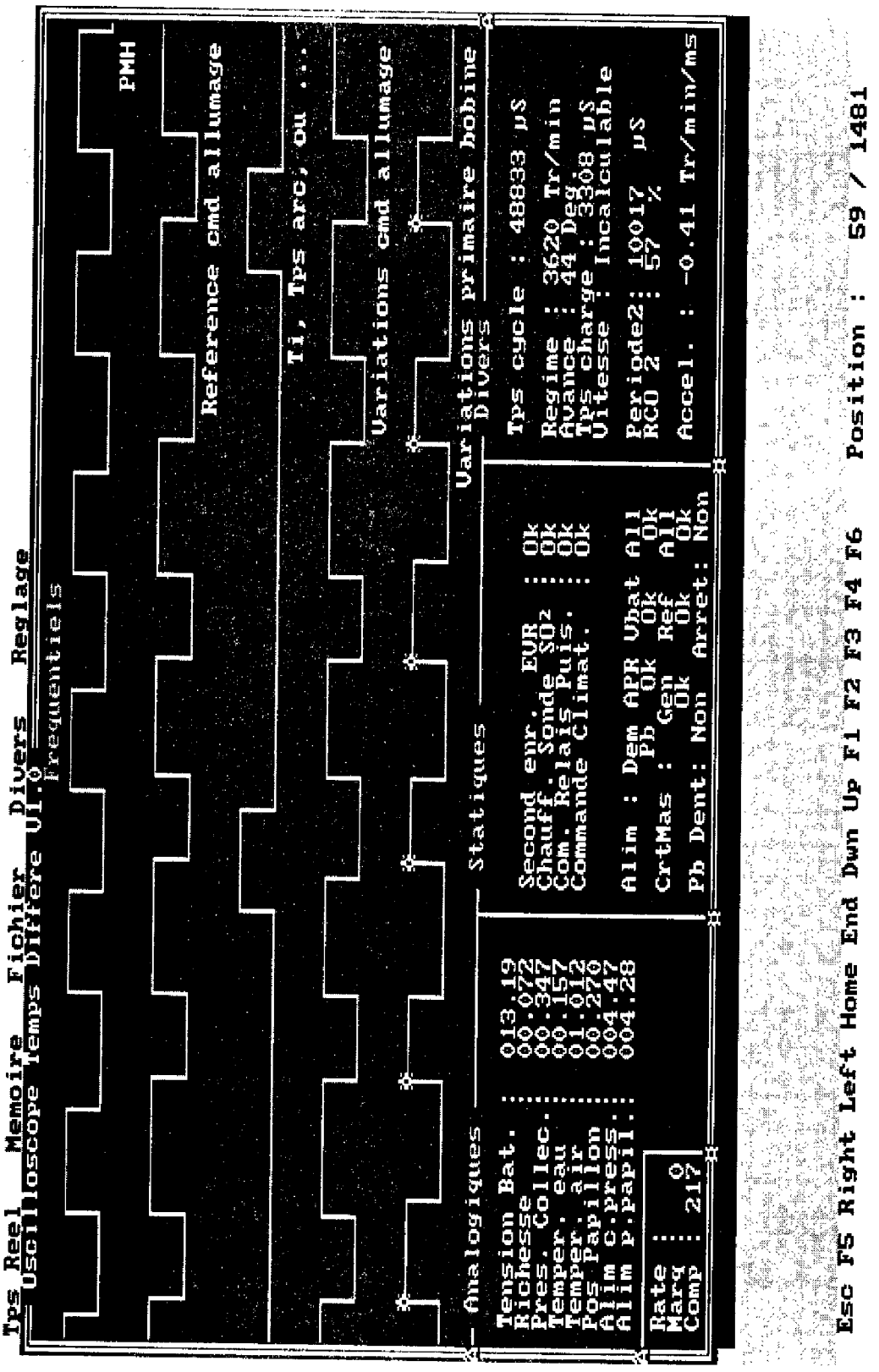

FIG. 2.18: Outil d'analyse d'ANDI : oscilloscope.

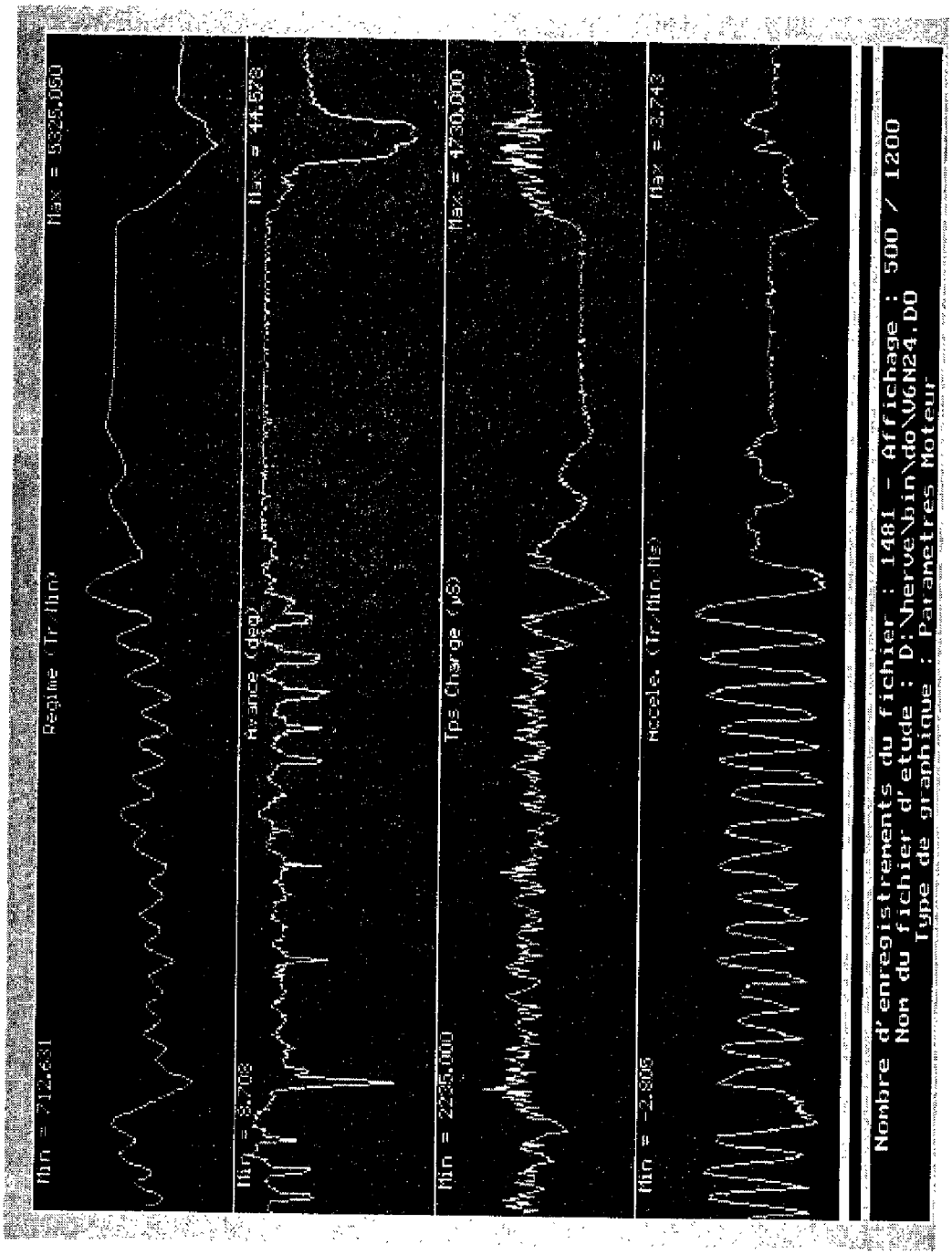

FIG. 2.19: Outil d'analyse d'ANDI : analyse temporelle des signaux.

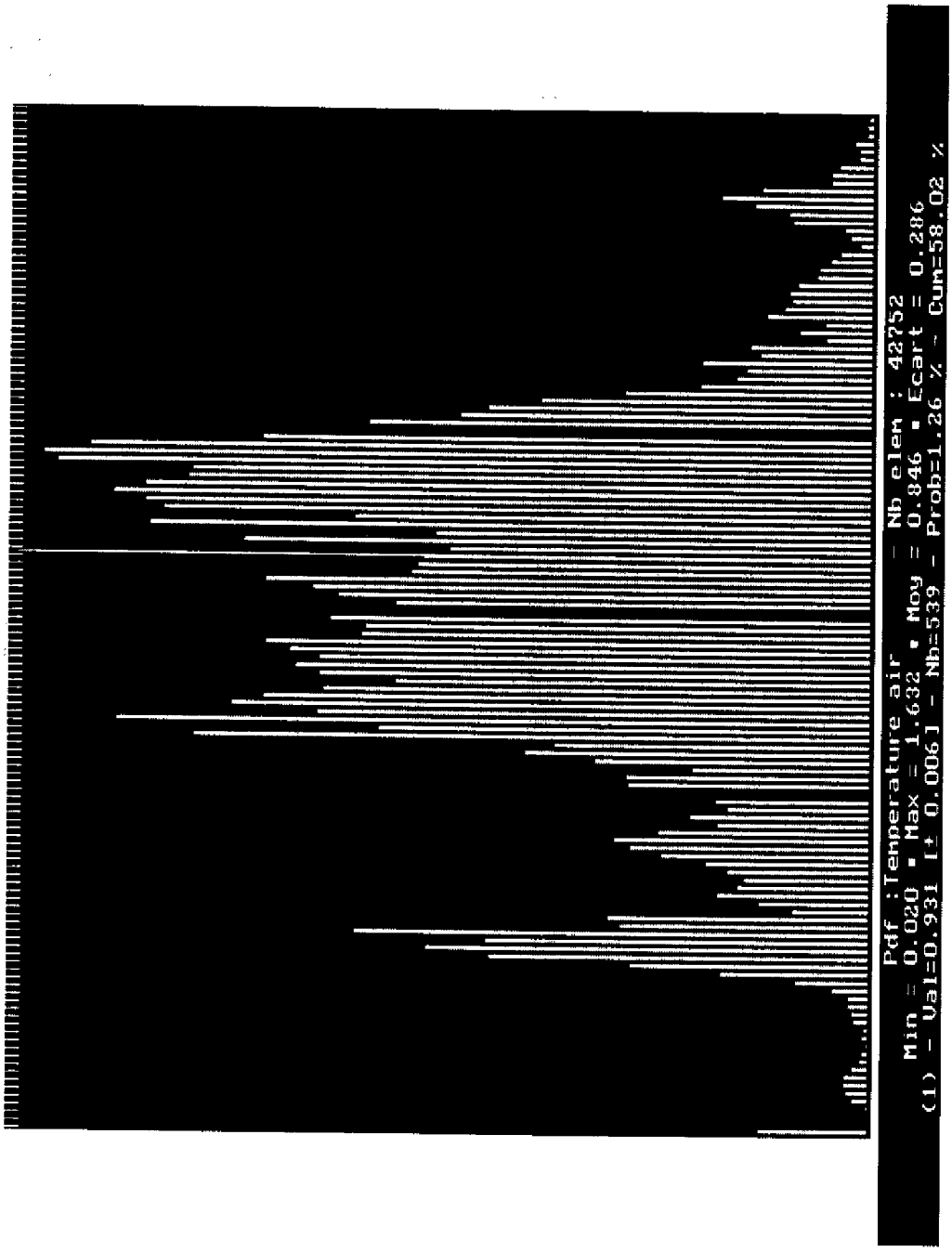

FIG. 2.20: Outil d'analyse d'ANDI : tracé de densité de probabilité (approximation par histogrammes).

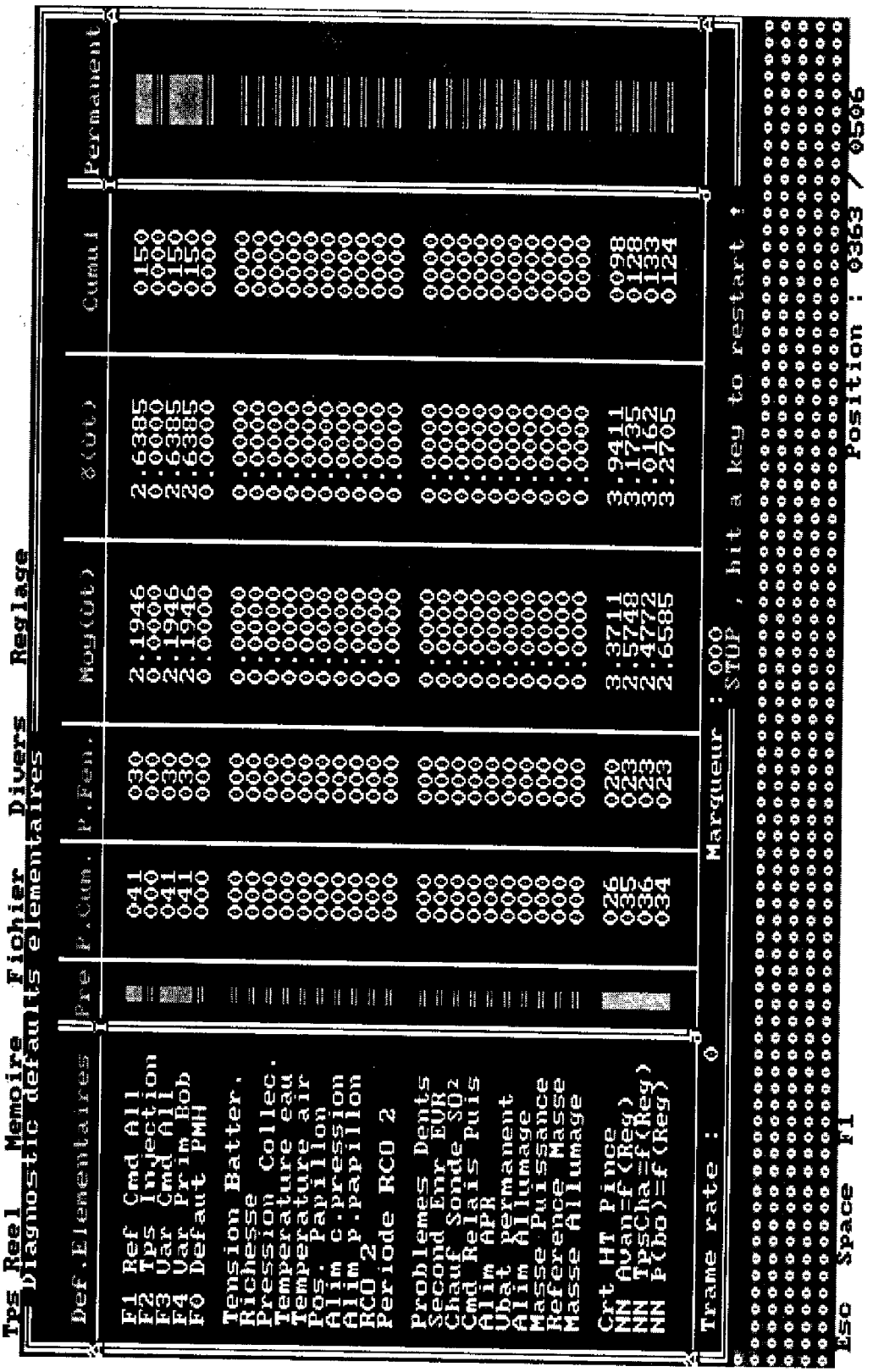

FIG. 2.21: Outil de diagnostic d'ANDI : apparition des défauts élémentaires.

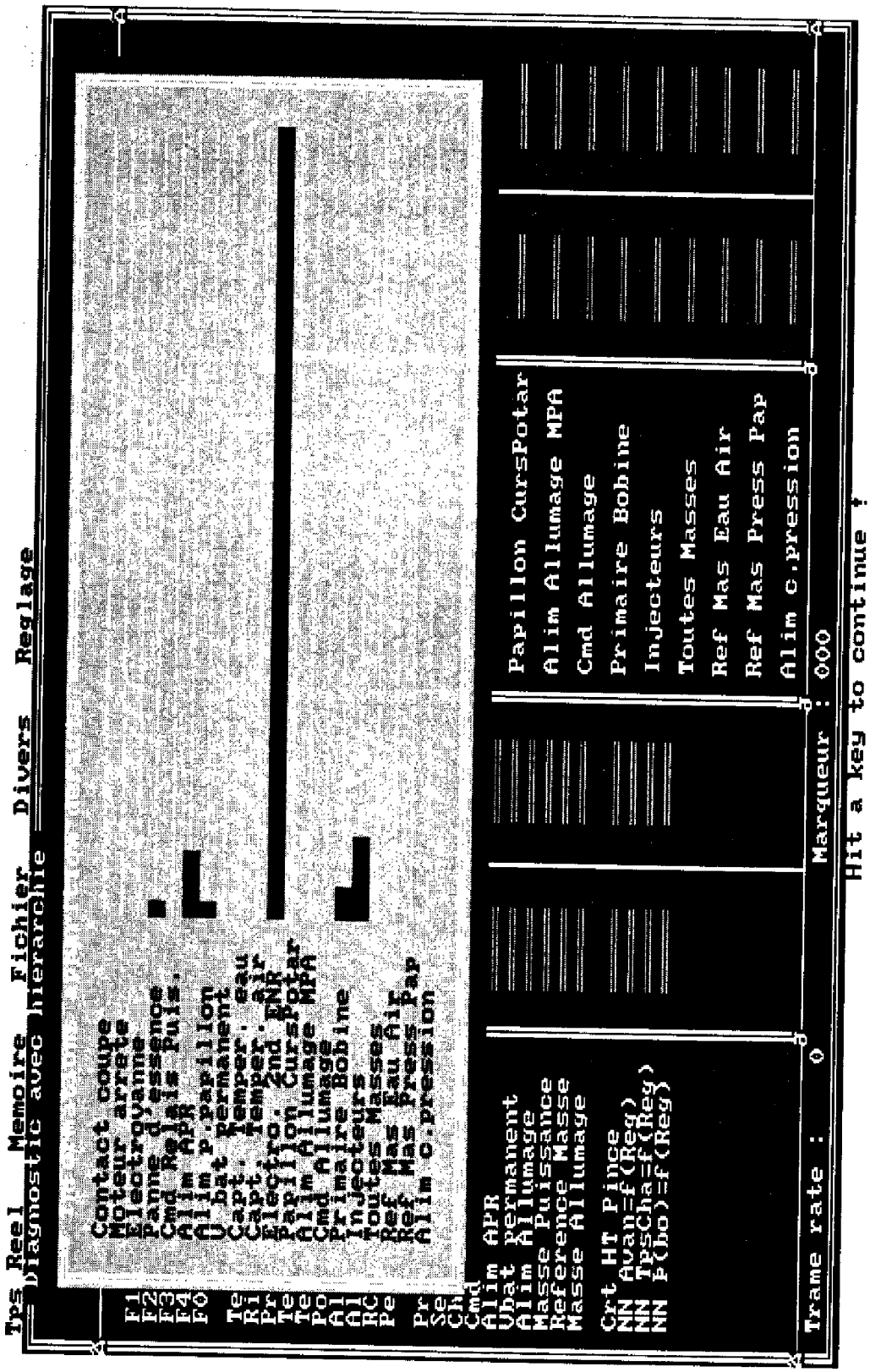

FIG. 2.22: Outil de diagnostic d'ANDI : identification des pannes.

L'univers n'était pas gros de la vie, ni la biosphère de l'homme. Notre numéro est sorti au jeu de Monte-Carlo. Quoi d'étonnant à ce que, tel celui-ci qui vient d'y gagner un milliard, nous éprouvions l'étrangeté de notre condition ?

JACQUES MONOD, *Le Hasard et la Nécessité*

Chapitre 3

Apprentissage des réseaux de neurones à couches

3.1 Introduction

Le système nerveux central des mammifères supérieurs et particulièrement celui des humains est *le chef d'oeuvre* de la nature, le fruit de milliards d'années d'évolution. Les merveilleuses capacités de notre cerveau, ce centre de traitement de la multitude d'informations sensorielles que nous recevons constamment de l'univers qui nous entoure, ce centre de décision qui dirige nos actions, cette machine à penser qui nous permet de communiquer, rêver, écrire, raisonner, aimer... , font du cerveau un organe exceptionnel, siège de la conscience, dont on commence seulement à cerner quelques propriétés.
Face à l'échec des ordinateurs dans la réalisation de tâches triviales pour un jeune enfant, ces trente dernières années ont vu naître un mouvement scientifique dont le but est d'essayer de comprendre, de modéliser des mécanismes propres au cerveau : *les sciences cognitives* (du latin *cognitio* : acte de reconnaître). Ce mouvement prend aujourd'hui beaucoup d'ampleur, touche de plus en plus de domaines et paraît très prometteur. Les domaines d'études de prédilection en sont la perception, le raisonnement, la représentation des connaissances et la linguistique. *" L'enjeu est fondamental car il s'agit de mettre en valeur l'intelligence humaine, de l'assister et d'en amplifier les pouvoirs"*, écrit André Holley dans l'éditorial d'un numéro du Courrier du CNRS dédié aux sciences de la cognition [92].
Cet axe de recherche peut être abordé par ses deux extrêmes : l'approche descendante, globale, extérieure, observationnelle, c'est l'étude des logiques (IA, logique floue ...), des structures linguistiques ... , et l'approche montante. Cette dernière, basée sur la connaissance des structures biologiques du cerveau et de nos organes des sens, tente d'expliquer et de copier la nature en modélisant ses moyens d'action. C'est dans ce cadre que s'est développée l'étude des réseaux de neurones.
Les capacités extraordinaires du cerveau semblent émerger du travail parallèle de 100 milliards de neurones fortement interconnectés. Ainsi chaque neurone est connecté à 10000 autres cellules nerveuses. Les scientifiques du domaine essayent de comprendre comment un assemblage de neurones (plus ou moins modélisés) peut avoir les propriétés observées. A l'origine les modèles de réseaux de neurones formels ont été créés par la biologie pour la biologie. Mais très vite, ces systèmes constitués d'unités simples travaillant en parallèle et connectées en réseaux complexes, se sont avérés très intéressants

dans de multiples domaines. Bien que les études biologiques inspirent le développement de nouveaux modèles, la majorité des études et algorithmes développés est probablement éloignée des réalités biologiques. Les réseaux de neurones sont devenus un nouveau système de traitement de l'information et on lui donne d'ailleurs le nom plus systémique de *réseaux d'automates.*

Ce qui différencie le plus les réseaux de neurones des autres systèmes de traitement de l'information, c'est la capacité d'apprentissage. L'information n'est plus modélisée mais elle est apprise grâce à un ensemble d'exemples appelé base d'apprentissage. On distingue deux types d'apprentissage : le *supervisé* et le *non-supervisé.* L'apprentissage non-supervisé est une forme d'extraction de caractéristiques des données. L'apprentissage supervisé consiste à forcer la réponse du réseau pour les éléments de la base d'apprentissage en modifiant la structure du réseau ou ses paramètres, ou les deux. Ce type d'apprentissage nécessite donc la connaissance de la sortie pour les éléments de la base d'apprentissage. Ces deux types d'apprentissage semblent coexister dans le cerveau : l'apprentissage non-supervisé dans les tâches sensorielles d'assez bas niveau et le supervisé dans des tâches de plus haut niveau, où la conscience de la réussite ou de l'échec, du bien ou du mal, ou d'un autre critère de qualité permet la supervision.

D'un point de vue connectivité, on distingue deux grands types de réseaux : *les réseaux bouclés* et *les réseaux à couches.* Les derniers ont des entrées et des sorties bien définies, et l'information se propage sans passer deux fois par la même unité. On peut en théorie fournir une expression analytique des sorties du réseau en fonction des entrées, indépendante du temps. Les réseaux bouclés peuvent être caractérisés par l'existence d'au moins un couple de neurones (i, j) pour lequel il existe un chemin pour aller de i à j, et un pour aller de j à i. La propagation de l'information est alors plus complexe à cause de ces bouclages. Le temps intervient et on dit qu'il y a une *dynamique des états.* L'expression analytique des sorties en fonction des entrées est forcément dépendante du temps.

Les réseaux de neurones constituent maintenant un domaine vaste, et particulièrement les réseaux à couches. Nous restreindrons donc notre présentation à certains algorithmes d'apprentissage supervisé. Après une rapide présentation des origines biologiques, nous détaillerons le Perceptron qui fut le premier modèle de réseaux de neurones doté d'un mécanisme d'apprentissage. En suivant le cheminement historique, nous présenterons ensuite l'Adaline et finalement la célèbre rétropropagation du gradient. Cet algorithme et ses versions améliorées est encore le plus utilisé actuellement. C'est celui qui a été employé dans ANDI. Ensuite nous présenterons brièvement les méthodes de simplification de réseaux, et enfin plus précisément certains algorithmes de construction, basés sur des versions améliorées du Perceptron pour apprendre une unité, que nous présenterons en détails.

3.2 L'origine biologique et le premier modèle de neurone

3.2.1 Le neurone biologique

La naissance des réseaux de neurones est directement issue des recherches neuro-biologiques. Avant de décrire les réseaux formels, il convient donc de s'intéresser aux cousins biologiques. Le système nerveux central [39] est constitué de quelques cent milliards de neurones fortement interconnectés entre eux. Chaque neurone est connecté à environ une dizaine de milliers d'autres par l'intermédiaire de systèmes, en général constitués d'un axone, d'une synapse et d'une dendrite (la synapse étant en réalité l'espace entre le bout de l'axone et le corps de la dendrite et dans lequel circulent les neuro-transmetteurs), comme on peut le voir sur la figure 3.1.

Chaque neurone reçoit des informations, sous forme d'impulsions électriques : les influx nerveux, provenant de toutes les ramifications des dendrites. A partir de tous ces signaux, il décide d'émettre ou non un signal. S'il émet un signal, il le fait grâce à la sortie du neurone, l'axone. La synapse est donc le lieu de passage des impulsions d'un neurone à l'autre. Ce passage d'informations est réalisé

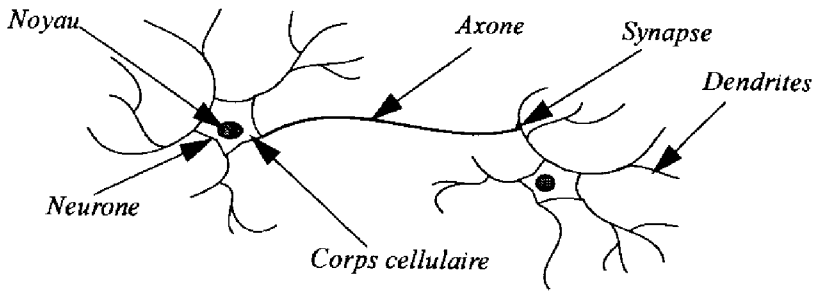

FIG. 3.1: Neurone biologique et chemin des influx nerveux

par des procédés chimiques, car un influx nerveux ne peut pas passer directement d'un corps cellulaire à un autre. Les vecteurs de cette transmission sont les neuro-transmetteurs. Une synapse peut être inhibitrice ou excitatrice. Dans le premier cas le neurone émetteur aura tendance à inhiber le neurone récepteur, alors que dans le second cas il aura tendance à l'activer. De plus, par sa géométrie, son lieu de ramification, les structures des processus chimiques..., une synapse est caractérisée par l'efficacité avec laquelle elle assure la liaison.
On considère qu'à chaque ramification de dendrites, la somme des influx est effectuée. La somme des influx des neurones émetteurs pondérés par les efficacités synaptiques semble donc arriver à l'organe central de décision du neurone. Celui-ci est caractérisé par un seuil. Si le signal reçu est supérieur à ce seuil, le neurone génère alors un influx nerveux qu'il émet par l'intermédiaire de ses axones vers d'autres neurones, sinon le neurone reste inactif. On peut donc représenter le fonctionnement d'un neurone par le schéma de la figure 3.2 :

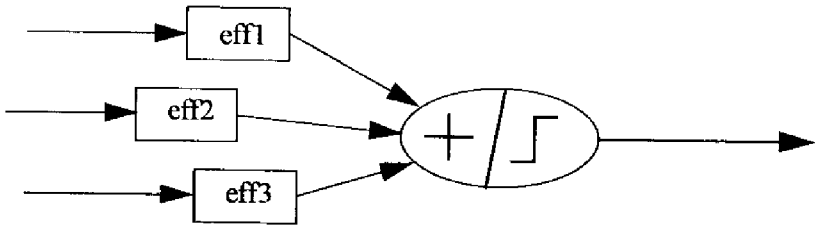

FIG. 3.2: Symbolisation du fonctionnement d'un neurone biologique

Comme nous le verrons dans la suite de ce chapitre, dans le cadre des réseaux de neurones à couches, les procédés d'apprentissage développés depuis le début du connexionisme n'ont en général plus rien à voir avec la biologie. Le modèle de neurone comme unité élémentaire de calcul a été plus ou moins conservé, mais les processus d'apprentissage sont des développements mathématiques abstraits et aucunement basés sur des réalités biologiques.

3.2.2 Le neurone formel

Le premier modèle de neurone formel fut présenté en 1943, par Mc Culloch et Pitts [131], deux chercheurs de l'université de Chicago. Ce modèle reproduit exactement les caractéristiques du modèle de neurone biologique présenté ci-dessus. C'est une unité à sortie binaire, soit il est actif, soit il est inactif (sortie à 1 ou à 0). Pour calculer sa sortie, le neurone calcule d'abord son entrée totale en faisant la somme de toutes ses entrées pondérées par les poids synaptiques correspondants. Ensuite

il compare l'entrée totale à un seuil qui est une valeur caractéristique de chaque neurone. Si l'entrée totale est supérieure au seuil alors le neurone s'active, sinon il est inactif.

Mathématiquement, si on note x_i les n entrées, w_i le poids de chaque connexion (les efficacités synaptiques) alors l'unité calcule sa sortie en faisant la somme des entrées pondérées par les poids plus la quantité θ et seuillée par une fonction de Heaviside ϕ.

$$s = \phi\left(\sum_{i=1}^{n} w_i x_i + \theta\right) = \phi\left(w.x + \theta\right) \tag{3.1}$$

$$\phi(A) = 1 \text{ si } A \geq 0 \quad \text{et} \quad \phi(A) = 0 \text{ si } A < 0 \tag{3.2}$$

En général θ est appelé le biais (*bias* en anglais), et le seuil est $-\theta$.

3.2.3 Le règle de Hebb

Historiquement, le premier mécanisme d'évolution des synapses, donc le premier mécanisme d'apprentissage, fut proposé par le chercheur D. Hebb en 1949 [90]. Le principe biologique est le suivant : *"Lorsqu'une connexion entre deux cellules est très forte, si la cellule émettrice s'active alors la cellule réceptrice s'active aussi. Pour lui permettre de jouer ce rôle déterminant lors du mécanisme d'apprentissage il faut donc augmenter le poids de cette connexion. En revanche, si la cellule émettrice s'active sans que la cellule réceptrice le fasse, ou si la cellule réceptrice s'active alors que la cellule émettrice ne s'était pas activée, cela traduit le fait que la connexion entre ces deux cellules n'est pas prépondérante dans le comportement de la cellule réceptrice. On peut donc, dans la phase d'apprentissage, laisser un poids faible à cette connexion"*. La règle pour les neurones formels est : "Si deux neurones connectés entre eux sont activés en même temps, alors on renforce la connexion qui les relie. Dans le cas contraire elle n'est pas modifiée." Cette règle est symbolisée sur le schéma de la figure 3.3.

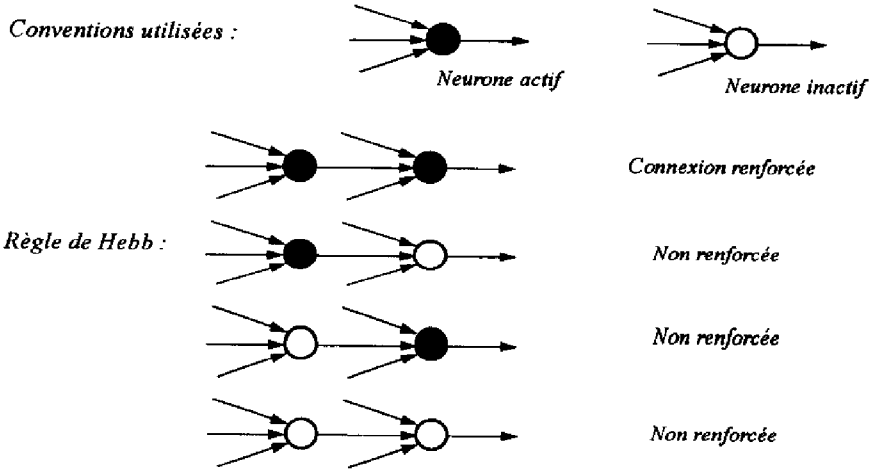

FIG. 3.3: Règle de Hebb

Cette règle d'apprentissage est le début de l'histoire des réseaux de neurones, qui servira au début des années 80 sous sa forme originelle à l'apprentissage des réseaux bouclés.

3.3 Du Perceptron à la rétropropagation

3.3.1 Le Perceptron

Le Perceptron, proposé par F. Rosenblatt en 1962 [169], fut le premier modèle de réseau de neurones doté d'un mécanisme d'apprentissage. Comme le montre la figure 3.4, il comprend trois parties :

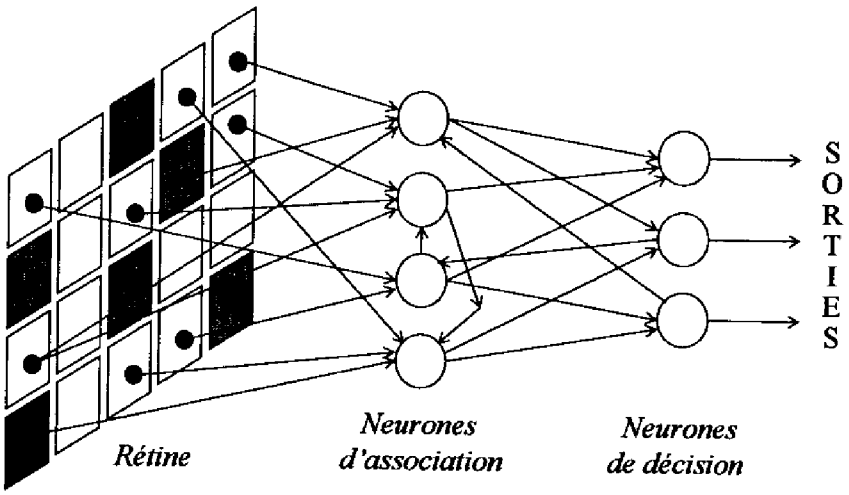

FIG. 3.4: Le Perceptron

- *la rétine* : elle est constituée de cellules sur lesquelles s'inscrivent les stimuli. Dans la plupart des modèles, ces cellules sont binaires mais elles peuvent aussi fournir une réponse modulée suivant l'intensité du stimulus.

- *une couche de neurones d'association* : chacun de ces neurones peut être connecté à des cellules de décision (voir ci-dessous). Les connexions reliant les cellules de la rétine aux neurones d'association ont des poids non modifiables. Ces cellules ne servent donc qu'à effectuer un prétraitement sur les données d'entrée fournies à la rétine. Les véritables entrées du réseau sont donc les sorties de cette couche.

- *une couche de neurones de décision* : elle représente la sortie du Perceptron. Chaque neurone de décision est connecté à des neurones d'association. Ce sont des unités à seuil et les valeurs des connexions des neurones de cette couche vont être modifiées pendant l'apprentissage.

L'apprentissage du Perceptron est supervisé. Il consiste donc à modifier les poids des neurones de décision pour que les sorties calculées sur un ensemble d'exemples en entrées, que l'on nommera vecteurs d'entrée ou plus simplement exemples, correspondent aux sorties désirées nommées les cibles. Comme la couche de cellules d'association a ses poids figés, on peut la considérer comme un prétraitement spécifique. Les véritables entrées du système adaptatif sont les sorties de cette couche.
Considérons un Perceptron simple constitué d'un seule cellule de décision. Ce Perceptron peut dissocier deux classes de vecteurs d'entrée : ceux pour lesquels la sortie est 1 et ceux pour lesquels la sortie est 0. La base d'apprentissage est constituée de N paires (ξ^μ, t^μ), où ξ^μ est l'entrée des cellules de décision (donc le vecteur de sortie de la couche d'association dans le Perceptron général), et t^μ est la sortie désirée.

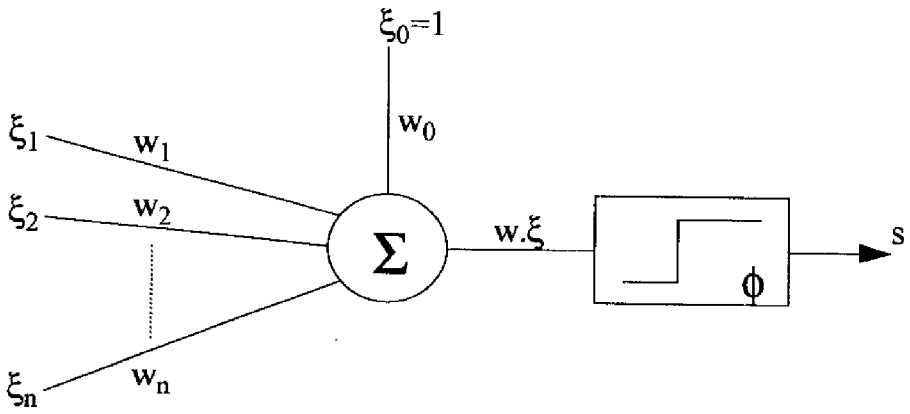

FIG. 3.5: Un Perceptron simple

L'apprentissage de ce Perceptron consiste alors à présenter les éléments de la base d'apprentissage, calculer la sortie du réseau et modifier les poids jusqu'à ce que le nombre d'erreurs soit acceptable. Le seuil θ de l'unité de sortie est aussi un paramètre modifiable, et on le considère généralement comme la composante $w_0 = \theta$ de w qui est alors un vecteur de $\mathbb{R}^{n+1}$, et on ajoute une cellule d'association d'indice 0 dont la sortie est toujours 1, ce qui revient à ajouter une composante 1 à chaque vecteur d'entrée de la base d'apprentissage. Dans ces conditions, la sortie de l'unité pour l'exemple d'indice μ se calcule alors par $s = \phi(w.\xi^\mu)$. Ce Perceptron simple est représenté sur la figure 3.5. Si on suppose que l'on a n cellules d'association nous avons $\xi^\mu \in \mathbb{R}^{n+1}$ et $t^\mu \in \{0, 1\}$.
Si on note s^μ la sortie pour l'exemple μ, la règle de modification des poids est alors

$$\Delta w_i = \nu \left(s^\mu - t^\mu\right) \xi_i^\mu \tag{3.3}$$

Cette règle sera nommée PLR (Perceptron Learning Rule). Le paramètre ν positif est appelé taux d'apprentissage (ou *learning rate*). Ce paramètre peut varier au cours de l'apprentissage, mais dans ce cas il doit être décroissant.

Définition 3.1 *On dira que la base d'apprentissage (ξ^μ, t^μ) est linéairement séparable si et seulement si $\exists w \in \mathbb{R}^{n+1}$ tel que*

$$\forall \mu \in \{1, \ldots, N\} \qquad \begin{cases} \phi(w.\xi^\mu) \geq 0 & \Longleftrightarrow \quad t^\mu = 1 \\ \phi(w.\xi^\mu) < 0 & \Longleftrightarrow \quad t^\mu = 0 \end{cases} \tag{3.4}$$

Cette définition est en fait très simple intuitivement. Si on représente les exemples de la base d'apprentissage comme des points dans l'espace d'origine $\mathbb{R}^n$ (sans la composante mise à 1), une base linéairement séparable signifie que l'on peut trouver un hyperplan affine qui sépare les points de cible 1 de ceux de cible 0. Le vecteur normal à cet hyperplan correspond au vecteur des poids de l'unité et la constante au seuil.
Nous avons alors le thérorème suivant :

Théorème 3.1 *Si pour une base d'apprentissage (ξ^μ, t^μ), les poids d'un Perceptron simple sont mis à jour avec la PLR (équation 3.3) alors le Perceptron converge vers une solution, c'est-à-dire un vecteur $w \in \mathbb{R}^{n+1}$ qui vérifie 3.4, si et seulement si la base est linéairement séparable.*
Ce théorème peut être visualisé simplement sur les schémas de la figure 3.6.

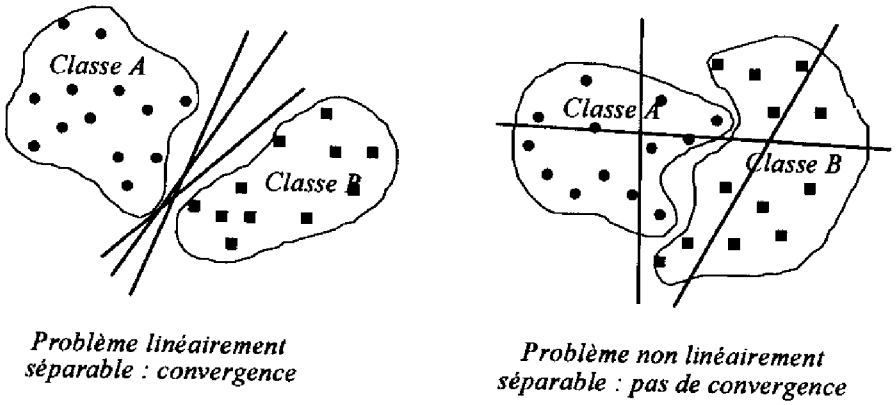

FIG. 3.6: Convergence du Perceptron

Démonstration 3.1 *Dans le cas où la base n'est pas linéairement séparable, il est évident que le Perceptron ne peut pas converger vers une solution car il n'en existe pas par définition de la séparabilité linéaire. En effet, il n'existe pas de vecteur w qui vérifie l'équation 3.4. Dans le cas d'une base linéairement séparable, on montre que le Perceptron converge vers une solution en un nombre fini de présentations des exemples de la base d'apprentissage. Le lecteur pourra trouver plusieurs démonstrations de ce théorème dans [135, 146, 61].*

Le Perceptron fut développé à l'origine pour essayer d'exhiber certaines propriétés des organes sensoriels. Son nom est d'ailleurs un néologisme né de l'union de perception et électronique. Il eut beaucoup de succès jusqu'à ce que Minsky et Papert en aient démontré les sévères limitations [135], notamment l'impossibilité de reconnaître la connexité d'une forme avec un Perceptron à champ limité. En fait un Perceptron général avec ses deux couches (association et décision) peut résoudre n'importe quelle fonction booléenne exprimée sous sa forme normale disjonctive. Les neurones de décision peuvent réaliser les OU et ceux d'association les ET. Mais les problèmes sont de deux ordres : il faut que les cellules d'association soient connectées à toutes les cellules de la rétine pour résoudre certaines fonctions (notamment la fonction parité) et deuxièmement les poids des cellules d'association sont figés car l'algorithme d'apprentissage ne permet que l'adaptation des poids des cellules de décision.
Cette étude mathématique pertinente des incapacités du Perceptron brisa l'élan scientifique pour les réseaux de neurones pendant plus de dix années. Mais les bases des futurs algorithmes d'apprentissage étaient en place. De plus, comme nous allons le voir, l'arrivée des algorithmes de construction à la fin des années 80, replaça l'algorithme d'apprentissage du Perceptron dans l'actualité connexioniste.

3.3.2 Adaline et Madaline

L'Adaline (ADAptative LINear element) [198] fut développé pour pallier un problème de robustesse des solutions trouvées par le Perceptron dans le cas de la discrimination de deux classes de vecteurs d'entrée. En effet, l'apprentissage du Perceptron doit être arrêté lorsque plus aucune erreur n'est commise, car une autre modification des poids pourrait générer à nouveau des erreurs. Or l'hyperplan séparateur ainsi obtenu peut être très proche de certains points. Ce qui implique une robustesse limitée. Lors de la présentation de nouveaux exemples non pris en compte dans la base d'apprentissage, le pouvoir de généralisation sera d'autant plus faible que l'hyperplan est proche de certains points : le moindre bruit pourra faire passer l'exemple de l'autre côté de l'hyperplan.
De plus, dans le cas d'une base non linéairement séparable, aucune solution n'est trouvée et le Perceptron va passer par un nombre important (mais fini) de positions différentes [135]. On montre même

qu'il passera par au moins une position pour chaque partition des vecteurs d'entrée en deux classes [29]. Sans un critère de qualité, le Perceptron est donc sans intérêt pour les problèmes non linéairement séparables. Nous verrons par la suite plusieurs méthodes pour utiliser le Perceptron sur ce type de problèmes.

Pour pallier ces deux problèmes, l'idée est de minimiser une fonction d'erreur qui est la somme sur les exemples de la différence entre la sortie désirée et la sortie calculée, mais avant le seuillage. Pour cela, l'unité utilisée est un peu différente de l'unité utilisée dans le Perceptron. La fonction d'activation du neurone est $2\phi - 1$, c'est-à-dire que les sorties ne sont plus 0 et 1 mais -1 et 1. Donc les cibles t^μ sont dans $\{1, -1\}$.

On définit la fonction d'erreur E^μ pour un vecteur d'entrée ξ^μ et la fonction d'erreur E sur la base entière par

$$E^\mu = \frac{1}{2}\left(w.\xi^\mu - t^\mu\right)^2 \qquad \text{et} \qquad E = \sum_{\mu=1}^{N} E^\mu \tag{3.5}$$

Il convient de signaler que la valeur absolue du produit scalaire $w.\xi^\mu$ est la distance du point ξ^μ à l'hyperplan multipliée par la norme du vecteur normal à cet hyperplan. Cette fonction d'erreur pourra être égale à zéro si et seulement si tous les points de cible -1 sont sur un hyperplan et tous les points de cible 1 se trouvent sur un autre hyperplan parallèle au premier. L'hyperplan solution sera alors la position intermédiaire entre ces deux hyperplans, soit la solution optimale.

Cette fonction d'erreur peut alors être minimisée par une descente de gradient. Deux implémentations sont possibles. Soit on réalise une modification des poids après le passage de toute la base, donc proportionnelle à $\frac{\partial E}{\partial w}$, soit une modification après le passage de chaque élément de la base, donc proportionnelle à $\frac{\partial E^\mu}{\partial w}$. La première est une optimisation par descente de gradient standard et la seconde une optimisation par descente de gradient stochastique.

La règle d'apprentissage de la version stochastique s'exprime alors par

$$\Delta w_i = -\nu \frac{\partial E^\mu}{\partial w_i} = -\nu \left(w.\xi^\mu - t^\mu\right) \xi^\mu \tag{3.6}$$

Cette règle d'apprentissage est généralement appelée la règle *Delta*, ou règle de *Widrow-Hoff*. Elle converge vers la solution des moindres carrés qui minimise la fonction d'erreur E.

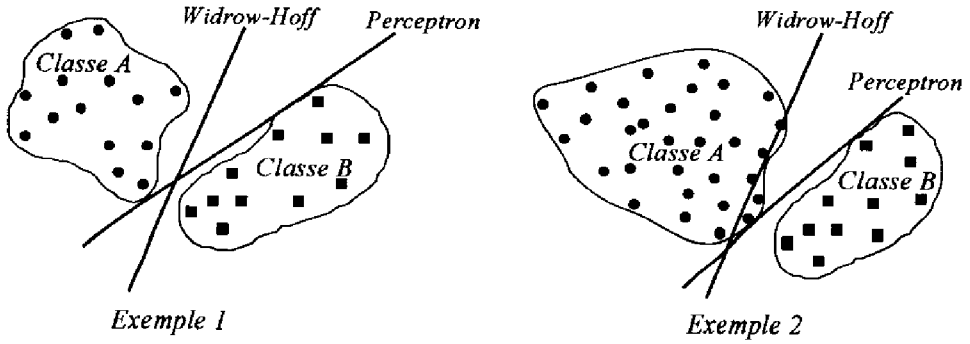

FIG. 3.7: Comparaison des solutions fournies par la règle du Perceptron et la règle de Widrow-Hoff

Dans certains cas de bases linéairement séparables, la solution des moindres carrés est une solution au problème de séparation des deux classes. Et dans de telle situations, la règle de Widrow-Hoff donne une solution plus robuste que celle du Perceptron. Mais elle ne donne pas systématiquement une solution

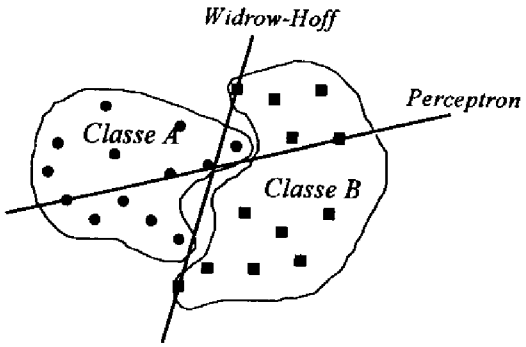

FIG. 3.8: Comparaison Perceptron - Widrow-Hoff pour un problème non linéairement séparable

car la fonction d'erreur définie en 3.5 n'est pas une vraie fonction d'erreur (par rapport au but de séparation des classes). Ainsi, la figure 3.7 représente un exemple de ces deux situations.

Dans les cas non linéairement séparables, la réponse de l'Adaline est en général beaucoup plus intéressante que le Perceptron simple qui peut donner une position sans intérêt, comme le montre la figure 3.8.

L'Adaline fut utilisé deux années plus tard pour la création de réseaux de neurones à deux couches : Madaline (Multiple Adaline) [197]. La première couche est constituée d'unités adaptatives Adaline et la couche de sortie est constituée d'unités non adaptatives réalisant une fonction logique (ET, OU, majorité ...). L'algorithme d'apprentissage nommé MRI permet l'apprentissage des éléments de la première couche simultanément. C'est une structure similaire à celle du Perceptron général dans le sens où une couche est constituée d'éléments adaptatifs et la seconde d'unités figées.

Plus récemment, deux autres modèles fut développés. MRII [200] permet l'apprentissage d'un réseau de deux couches d'unités à seuil adaptatives. Et enfin, MRIII [3] permet l'apprentissage de n'importe quel réseau de neurones à couches constitué d'unités continues. En fait, cet algorithme récent est tout à fait similaire à la rétropropagation que l'on va présenter maintenant.

3.3.3 La rétropropagation

La rétropropagation constitue en fait la suite logique de l'enchaînement Perceptron - Adaline. Le problème du Perceptron est qu'il minimise une erreur en tout ou rien, sans prendre en compte une notion de distance. Il est donc très peu robuste mais c'est une véritable fonction d'erreur. L'Adaline ne travaille plus en tout ou rien mais minimise une fonction d'erreur quadratique, donc est plus robuste et donne des solutions approchées lorsqu'il n'y a pas de solutions. Mais le problème est que la fonction d'erreur de l'Adaline n'est pas une véritable fonction d'erreur, car le signal est pris avant le seuillage. De plus, et c'est ceci qui est très limitatif, le Perceptron et l'Adaline ne fonctionnent que sur des réseaux avec une couche de poids adaptatifs.

Le problème était donc de trouver une fonction d'erreur véritable qui ne travaille pas en tout ou rien et qui permette l'apprentissage des réseaux multi-couches. La solution à ce problème était de changer de neurone formel, de passer d'automates à seuil à des automates continus, et de trouver un mécanisme simple de calcul des dérivées composées, car la sortie du réseau devient une fonction continue et dérivable de tous les poids.

La fonction d'activation d'un neurone continu est une approximation d'une unité à seuil par une

fonction continue. La fonction utilisée le plus couramment est la *sigmoïde*, c'est-à-dire

$$\sigma(x) = \frac{1}{1+e^{-\beta x}} \tag{3.7}$$

Plus le coefficient β est grand, plus cette fonction se rapproche de la fonction de Heaviside. Dans le cas où on désire une approximation de la fonction $2\phi - 1$ (la fonction signe) on prend $\sigma(x) = \tanh(\beta x)$. En fait, l'algorithme que l'on va présenter n'est pas du tout restrictif, on peut prendre une fonction quelconque tant qu'elle est dérivable. La figure 3.9 représente l'allure de la fonction $\sigma(x)$ définie en 3.7 (la pente en 0 est égale à $\beta/4$).

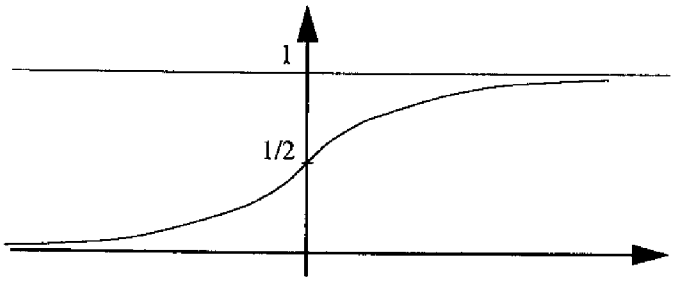

FIG. 3.9: Allure de la sigmoïde

D'un point de vue connectivité, le réseau peut être quelconque mais à couches : on peut avoir un nombre de couches quelconque avec des connexions pontées (non directement connectées d'une couche à l'autre). Les couches de neurones qui ne sont ni la couche de sortie ni la couche d'entrée sont appelées couches cachées (exemple sur la figure 1.2). Les fonctions d'activation ne sont pas forcément les mêmes pour chaque neurone, mais elles doivent toutes être des fonctions continues.
La base d'apprentissage est encore notée (ξ^μ, t^μ) avec $\xi^\mu \in \mathbb{R}^{n+1}$ (la première composante de chaque vecteur d'entrée étant un 1 que l'on ajoute pour traiter le seuil de la même manière que les connexions). La sortie désirée est un vecteur de dimension égale au nombre de neurones de sortie. Si tous les neurones de sortie ont la même fonction d'activation σ, il faut avoir $t^\mu \in (\sigma(\mathbb{R}))^p$ (ce n'est pas strictement obligatoire mais préférable). On introduit alors une unité dont la sortie est toujours à 1 et connectée à toutes les cellules qui ne sont pas des cellules d'entrées du réseau. On notera alors w_{ij} le poids de la connexion entre la cellule émetrice j et la cellule réceptrice i. Si on présente l'exemple d'indice μ en entrée du réseau, l'entrée totale d'un neurone i sera notée A_i^μ, et sa sortie $s_i^\mu = \sigma_i(A_i^\mu)$, avec σ_i sa fonction d'activation. Pour la suite, on n'indiquera pas l'indice μ de l'exemple pour alléger les écritures, mais il y est toujours implicitement.

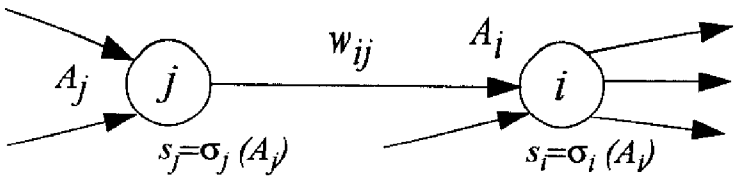

FIG. 3.10: Notations pour la rétropropagation

Si on note S l'ensemble des indices des neurones de sortie, les fonctions d'erreur partielle et globale sont alors définies par

$$E^\mu = \frac{1}{2}\sum_{j\in S}(s_j - t_j)^2 \qquad \text{et} \qquad E = \sum_{\mu=1}^{N} E^\mu \tag{3.8}$$

La minimisation de la fonction d'erreur globale va se faire par une descente de gradient. Donc après la présentation de tous les vecteurs d'entrée de la base d'apprentissage nous modifierons la valeur de chaque connexion par la règle

$$\Delta w_{ij} = -\nu \frac{\partial E}{\partial w_{ij}} = -\nu \sum_{\mu=1}^{N} \frac{\partial E^\mu}{\partial w_{ij}} \tag{3.9}$$

Cette règle d'apprentissage est généralement appelée la règle *Delta généralisée*. Dans l'expression de E^μ, seule la sortie s_i dépend du paramètre w_{ij}. Or ce paramètre ne sert que dans le calcul de l'entrée totale du neurone d'indice i. Nous pouvons donc décomposer la dérivée par

$$\frac{\partial E^\mu}{\partial w_{ij}} = \frac{\partial E^\mu}{\partial s_i}\frac{\partial s_i}{\partial A_i}\frac{\partial A_i}{\partial w_{ij}} = \frac{\partial E^\mu}{\partial s_i}\,\sigma_i'(A_i)\,s_j \tag{3.10}$$

Tous le problème réside donc dans le calcul de $\dfrac{\partial E^\mu}{\partial s_i}$ que l'on notera D_i^μ pour tous les neurones du réseau sauf les neurones d'entrée.
Deux cas se présentent :

- Si le neurone d'indice i est un neurone de sortie, alors le calcul est très simple. Nous avons

$$D_i^\mu = \frac{1}{2}\sum_{j\in S}\frac{\partial}{\partial s_i}(s_j - t_j)^2 = \frac{1}{2}\frac{\partial}{\partial s_i}(s_i - t_i)^2 = s_i - t_i \tag{3.11}$$

- Dans le cas où le neurone d'indice i n'est pas un neurone de sortie, alors le signal s_i va servir au calcul de l'entrée totale de tous les neurones connectés au neurone i en aval. Donc si on note F_i l'ensemble des indices des neurones connectés à i en aval, nous pouvons alors décomposer cette dérivée par

$$D_i^\mu = \sum_{k\in F_i}\frac{\partial E^\mu}{\partial s_k}\frac{\partial s_k}{\partial s_i} = \sum_{k\in F_i} D_k^\mu \frac{\partial s_k}{\partial A_k}\frac{\partial A_k}{\partial s_i} = \sum_{k\in F_i} D_k^\mu\,\sigma_k'(A_k)\,w_{ki} \tag{3.12}$$

Après avoir calculé D_i^μ pour tous les neurones de sortie avec la relation 3.11, le calcul de D_i^μ pour les autres neurones se fait grâce à la relation de récurrence 3.12, en allant de la sortie vers l'entrée : *on rétropropage le signal d'erreur*. C'est de là que vient le nom de cet algorithme. Ce calcul est simplement une formulation récurrente du calcul de dérivées composées. Dans l'algorithme MRIII [3], la dernière évolution de Madaline à des neurones continus, le calcul de D_i^μ ne se fait pas par la rétropropagation, mais par un procédé de perturbation qui s'apparente en fait à un calcul de dérivées partielles par la méthode des différences finies. D'après l'auteur [199], MRIII serait moins sensible que la rétropropagation classique, nécessite un peu plus de calculs mais ils sont moins compliqués et plus facilement intégrables sur silicium.
La règle de modification des poids décrite par l'équation 3.9 est la règle dite du gradient total. La convergence de l'algorithme avec cette règle est très difficile, car la fonction d'erreur totale possède de nombreux minima locaux. La version stochastique, similaire à la règle 3.6 pour l'Adaline, est préférable

car elle a beaucoup moins tendance à converger dans des minima locaux. Ainsi, la modification des poids se fait après la présentation de chaque exemple suivant la règle

$$\Delta w_{ij} = -\nu \frac{\partial E^{\mu}}{\partial w_{ij}} \tag{3.13}$$

Le problème majeur de cette version stochastique est que la modification des poids après la présentation d'un exemple ne prend pas en compte les informations calculées sur le précédent, il y a un phénomène d'oubli. Pour résoudre ce problème, on ajoute un terme d'inertie. Ainsi la règle d'apprentissage utilisée le plus est *la règle du gradient stochastique avec inertie*, soit

$$\Delta w_{ij}^{t} = -\nu \frac{\partial E^{\mu}}{\partial w_{ij}} + \lambda \Delta w_{ij}^{t-1} \tag{3.14}$$

La modification d'un poids lors de la présentation d'un vecteur d'entrée quelconque au temps t prend en compte la modification réalisée lors de la présentation au temps $t-1$ de l'exemple précédent.

La paternité de cet algorithme est un sujet de polémiques. Il semblerait d'après [199] que la première personne à avoir utilisé cette méthode de calcul des dérivées partielles est P. Werbos en 1971. Il publia ses résultats en 1974 dans son manuscrit de thèse [194]. Mais son travail ne fut pas remarqué par la communauté scientifique. Par la suite D. Parker redécouvrit cette technique en 1982 qu'il publia dans un rapport de recherche du MIT en 1985 [150]. Une équipe française [47] a aussi découvert cette procédure à la même époque. Quelques temps plus tard, D.E. Rumelhart, G.E. Hinton et R.J. Williams publièrent le fameux PDP (Parallele Distributed Processing), le livre qui fit connaître mondialement l'algorithme de rétropropagation du gradient [174].

Malgré la possibilité d'apprentissage de problèmes complexes, que peut illustrer la célèbre application NetTalK [178], l'utilisation de la rétropropagation pose de nombreux problèmes :

- une des limitations importantes est *le temps de calcul* : l'apprentissage est très long. Dans [91] une complexité algorithmique empirique de $O(N^3)$ est donnée.
- de nombreux problèmes sont dûs à la géométrie de la fonction d'erreur : *minima locaux, ravins et plateaux*. Le problème des minima locaux est en partie résolu avec le gradient stochastique, mais il subsiste quand même. La forme de la fonction sigmoïde implique une dérivée très faible hors de la zone autour de 0. Ainsi dans des zones où toutes les sigmoïdes sont saturées, les composantes du vecteur gradient sont toutes très faibles. Ce sont les plateaux, et il est très difficile de s'en dégager. Dans le cas où certaines dérivées ne sont pas saturées et d'autres oui, on nomme ces zones des ravins. Si le paramètre d'apprentissage n'est pas très bien réglé, on observe alors des oscillations très importantes (comme une bille lâchée dans un cylindre sans vitesse initiale). Dans [31], les auteurs proposent plusieurs exemples très simples de classification en deux classes où la rétropropagation est beaucoup moins performante que le simple Perceptron. La cause de cet échec est la géometrie de la fonction d'erreur, qui possède de larges minima locaux.
- l'algorithme est très sensible aux conditions initiales, c'est-à-dire à la manière dont sont initialisés les poids du réseau.
- Enfin, le problème le plus important est celui du *dimensionnement du réseau*. La rétropropagation apprend une base d'apprentissage sur un réseau dont la structure est fixée *a priori*. La structure est définie par le nombre de couches cachées, le nombre de neurones par couches et la topologie des connexions. Il s'agit donc de définir cette structure en fonction de la base

d'apprentissage à apprendre. C'est un problème extrêmement difficile qui n'a pas de solutions pertinentes actuellement. De plus, une mauvaise structure dégrade considérablement les performances du réseau. S'il est sous-dimensionné, l'algorithme n'arrivera pas à apprendre. S'il est surdimensionné, la base d'apprentissage sera apprise mais ses performances en généralisation seront faibles, car le nombre de paramètres libres du réseau étant trop grand, la base d'apprentissage sera apprise *par coeur*.

La fin des années 80 a vu le développement d'une multitude de variantes de la rétropropagation afin de résoudre les deux premiers problèmes liés à la difficulté de convergence. En général, le principe est de prendre en compte les informations du second ordre, c'est-à-dire les dérivées secondes. Ainsi de nombreux algorithmes ont été développés basés sur des méthodes d'optimisation plus efficaces que la descente de gradient : gradient conjugué, quasi-newton et toutes leurs méthodes dérivées... Ils se basent tous sur le calcul des dérivées secondes (le hessien), soit de manière exacte, soit par une approximation. Deux articles de synthèse sur ce domaine sont particulièrement intéressants et complémentaires. L'un [32] fait une synthèse des méthodes de calculs des dérivées secondes, et l'autre [14] une synthèse des méthodes qui les utilisent. D'autres algorithmes basés sur des ajustements heuristiques du paramètre d'apprentissage ont aussi été proposés. En général, tous ces algorithmes dérivés accélèrent de manière significative la convergence.
Parmi les algorithmes les plus connus, on peut citer l'algorithme *QuickProp* [66] qui utilise une heuristique de calcul simple avec une méthode de style Newton, la méthode *Bold Driver* [13] utilisant une heuristique d'adaptation du paramètre d'apprentissage, le *Scaled Conjugate Gradient* [139] qui est une version du gradient conjugué qui ne nécessite aucun réglage de paramètres et la méthode de style quasi-Newton de *Broyden-Fletcher-Goldfarb-Shanno* [15]. On peut enfin citer une méthode assez particulière *Least Squares Backpropagation* [24] qui minimise une erreur couche par couche par les moindres carrés, en utilisant la fonction inverse de la sigmoïde. Il semblerait que parmi ces méthodes, le *Scaled Conjugate Gradient* soit dans les plus efficaces.
En général, ces méthodes utilisent des approximations des dérivées secondes. Le calcul exact du Hessien, dont un algorithme a été proposé par Bishop [25], est assez complexe et long (avec un réseau sans connexion pontée). Afin de diminuer ces calculs complexes, une méthode très performante a été proposée par Pearlmutter [152], qui permet de calculer sans approximation le produit du Hessien par un vecteur. Dans les méhodes du second ordre, le Hessien intervient généralement sous la forme d'un tel produit. Ces méthodes de calcul exact ou approximatif des dérivées secondes sont importantes, car outre leurs applications dans les algorithmes précédemment cités, elles sont souvent utilisées dans les méthodes de simplification de réseau que l'on abordera dans le paragraphe suivant.

La sensibilité à l'initialisation a aussi été étudiée, et quelques techniques ont été proposées pour initialiser le réseau de manière assez satisfaisante, dont certaines fournissent aussi la structure du réseau. Ces méthodes sont généralement basées sur une analyse des données de la base d'apprentissage (sélection de prototypes, classification, analyse discriminante, statistiques...) [55, 195, 105, 187]. On peut aussi citer une méthode géométrique basée sur les diagrammes de Voronoi [18].

Le problème du dimensionnement du réseau est pour l'instant sans réponse pertinente, mis à part les quelques techniques d'initialisation qui fournissent aussi une structure, et quelques résultats théoriques difficilement applicables [17, 16]. Les utilisateurs procèdent alors par une série *essais - validation* en modifiant la structure à chaque essai et en validant le réseau sur une base de test.
Pour pallier ce problème de dimensionnement, à la fin des années 80 deux axes de recherches se sont développés :

- *Les méthodes de simplification de réseau ou d'élagage* : on a au départ d'un réseau sur-dimensionné et on essaye d'enlever les connexions et les neurones superflus.

- *Les algorithmes de construction* : la construction du réseau est réalisée de manière incrémentale. Des unités sont ajoutées lorsque cela est nécessaire, suivant diverses stratégies.

Après avoir abordé très succinctement les méthodes de simplification de réseau, nous développerons plus précisément les méthodes constructives, et particulièrement celles basées sur des neurones binaires, car c'est dans ce domaine que de nouveaux algorithmes ont été développés et qui seront présentés dans le chapitre 5.
Il convient enfin de signaler que les réseaux de neurones continus ont une application de prédilection qui est l'approximation de fonctions avec une base d'exemples, car toute fonction peut être approchée d'aussi près que l'on désire par un réseau de neurones : ce sont *des approximateurs universels* [123, 95, 93, 94]. Ce domaine d'application ne peut être abordé par les réseaux de neurones binaires, à moins de concevoir un réseau avec des neurones continus en sortie (linéaires par exemple) et binaires dans les couches cachées. Une telle architecture n'a pas été trouvée dans la littérature.

3.4 Les méthodes de simplification de réseau

De multiples méthodes d'élagage de réseau ont été développées depuis le début des années 90. Un excellent article de R. Reed [165] en présente une synthèse que nous allons résumer sans entrer dans les détails.
Certaines de ces méthodes sont basées sur le fait que lorsqu'une connexion a un poids nul, elle peut être retirée. Les unités sont ensuite enlevées du réseau si et seulement si elles n'ont plus de connexions en amont ou plus de connexion en aval. D'autres méthodes retirent directement des unités du réseau. Mise à part la méthode triviale, peu efficace et longue, qui consiste à successivement enlever les connexions et évaluer les modifications de la réponse du réseau, ces méthodes peuvent être classées en trois catégories :

- les méthodes qui calculent sous diverses formes *la sensibilité du réseau* à une connexion ou à un neurone. La méthode proposée par Mozer *et al* [140] calcule l'importance des neurones durant l'apprentissage et enlève directement un neurone si son importance est faible. La technique développée par Karnin [104] calcule après apprentissage la sensibilité du réseau à la suppression des connexions. Cette sensibilité est estimée en fonction des valeurs initiales et finales des connexions, et des variations de poids qui ont été calculées. Dans la méthode *Optimal Brain Damage*, Le Cun *et al* [48] estime la *pertinence* d'une connexion après apprentissage par un calcul basé sur la différentielle de la fonction d'erreur au point solution trouvé par l'algorithme. Il utilise une hypothèse simplificatrice en considérant le Hessien comme une matrice diagonale. Une technique similaire est utilisée dans la méthode *Optimal Brain Surgeon* développée par Hassibi et Stork [89], mais sans hypothèse simplificatrice sur le Hessien, car d'après les auteurs des termes non diagonaux peuvent être aussi importants que les termes diagonaux.

- les méthodes qui *ajoutent un terme à la fonction d'erreur* afin de faire décroître la valeur des poids vers 0, que l'on peut grouper sous le terme de *Weight Decay*. Dans cette catégorie on peut citer [40, 193, 102, 101] qui se différencient dans le type de terme ajouté à la fonction d'erreur. La méthode *Soft Weight-Sharing* proposée par Hinton et Nowlan [147] utilise un terme d'erreur plus complexe en modélisant la distribution de probabilité de la valeur des poids par un mélange de gaussiennes. Une méthode originale a aussi été proposée par Poulard *et al* [154]. Elle est dédiée aux problèmes à sorties booléennes. Elle utilise une fonction d'erreur quadratique

spéciale, avec un intervalle non réduit à un point sur lequel l'erreur est nulle. Cette méthode est justifiée par le caractère binaire des sorties, et permet intuitivement d'élargir l'espace des solutions en créant des *bassins* solutions. Après la convergence dans un bassin, un second terme d'erreur est introduit pour faire décroître les poids vers 0. Ce terme d'erreur est uniquement fonction des poids. Plusieurs type de termes ont été utilisés dont un est similaire à celui utilisé dans [193]. Cette méthode s'est révélée assez efficace, et optimale sur certains problèmes simples (XOR par exemple) même en ayant au départ des structures largement sur-dimensionnées. Le manque de temps n'a pas permis de comparer cette méthode avec les autres.

- les autres méthodes utilisent des stratégies assez diverses. La méthode *Interactive Pruning* proposée par Sietsma et Dow [181] repose sur une analyse numérique des sorties des neurones. Si un neurone a une sortie constante sur l'ensemble de la base, alors il ne sert à rien et il peut être retiré en modifiant le seuil des unités connectées en aval. Si plusieurs unités ont des sorties fortement corrélées sur l'ensemble de la base, alors il y a de grandes chances pour qu'elles soient redondantes. Elle peuvent donc être remplacées par une seule unité en additionnant leurs poids. D'autres heuristiques sont proposées. La méthode *Local Bottlenecks* proposée par Kruschke [114] est basée sur une sorte de compétition des unités cachées pour survivre. Une quantité nommée le *gain* (le score) est calculée pour chaque unité. Cette quantité est nulle pour un neurone qui a une sortie constante. Elle est toujours positive, mais plus elle est faible plus le neurone est superflu. Après chaque cycle de la rétropropagation, une compétition est engagée entre les cellules. Cette compétition se résume à réajuster le gain des unités i en fonction du gain des autres unités $j \neq i$, et des angles entre le vecteur des poids de i et les différents vecteurs des poids des unités j. Cette modification du gain est toujours négative et lorsque le gain d'une unité devient négatif ou nul, l'unité est éliminée. Une autre méthode nommée *Distributed Bottlenecks* a été proposée par Kruschke [114] dans le même article. Cette stratégie est très particulière car elle est destinée à atteindre les objectifs de simplification du réseau, c'est-à-dire l'optimisation du pouvoir de généralisation, sans retirer effectivement des neurones ou des connexions. Le principe est en fait de restreindre l'espace dans lequel les poids peuvent prendre leur valeur en ajoutant des contraintes. Ces contraintes ont le même effet que l'élimination de degrés de liberté par suppression d'unités ou de poids. Intuitivement, le principe est le suivant : lorsque les vecteurs des poids de deux unités sont proches ils vont être encore rapprochés, et lorsqu'ils sont loin ils vont être éloignés. Ce principe est implémenté par une modification des poids particulière après chaque cycle de la rétropropagation, et est équivalent à une descente de gradient sous contraintes. La méthode proposée par Whitley et Bogart [196] utilise des algorithmes génétiques. Ces derniers ont aussi été utilisés par Paugam-Moisy [151] pour la recherche d'un réseau ayant de bonnes performances grâce à un espion qui estime et sélectionne la qualité de réseaux dans une population de réseaux appris simultanément sur une architecture massivement parallèle.

En parallèle à ces multiples développements de méthodes de simplification de réseau, que l'on pourrait qualifier de *destructives*, une multitude de méthodes *constructives* ont été proposées : ce sont les algorithmes de construction que nous allons présenter maintenant.

3.5 Les algorithmes de construction

3.5.1 Généralités

Comme on l'a écrit précédemment, les méthodes de construction de réseaux sont de récents algorithmes particulièrement intéressants car ils permettent la construction de réseaux sans avoir à définir de

structure *a priori*. D'autre part, la construction est généralement beaucoup plus rapide qu'un apprentissage avec la rétropropagation, car elle est réalisée de manière incrémentale et chaque phase d'apprentissage est réalisée sur un sous-problème beaucoup plus simple que le problème global.
Une très grande quantité d'algorithmes de construction a été proposée. On peut classer ces algorithmes en trois catégories, suivant le type de neurones utilisés, et les structures des réseaux obtenus sont très variées suivant le type d'algorithme.

- les méthodes basées sur la formation de prototypes utilisant des *neurones à activité localisée*. Les algorithmes les plus connus de cette catégorie sont GAL (*Grow And Learn*) [2] et RCE (*Restricted Coulomb Energy*) [166]. Ce dernier algorithme appartient à la classe des réseaux de neurones à base radiale (*Radial Basis Function Neural Networks*).

- les méthodes basées sur un apprentissage de type rétropropagation utilisant des *neurones continus* (généralement des neurones à sigmoïde). L'algorithme DNC (Dynamic Node Creation) [7] est une utilisation assez simple de la rétropropagation pour un apprentissage constructif. Le plus connu de cette catégorie est sans aucun doute Cascade Correlation [67]. Cet algorithme a encore beaucoup de succès aujourd'hui pour sa rapidité. Il convient de signaler que cet algorithme peut utiliser deux types d'unités : des neurones à sigmoïde et des neurones à activation gaussienne. Récemment un autre algorithme nommé FNNCA (Feedforward Neural Network Construction Algorithm) [179] a été proposé. Ce dernier utilise une stratégie similaire à l'algorithme DNC mais avec une méthode d'optimisation du second ordre. On peut aussi citer l'algorithme IRO (Internal Representation Optimization) [121], qui n'utilise pas la rétropropagation mais un procédé d'optimisation d'un critère sur les représentations internes.

- les méthodes basées sur un apprentissage de type Perceptron, purement géométrique ou autre utilisant des *neurones à seuil*. C'est cette dernière catégorie qui nous intéresse particulièrement. Une multitude d'algorithmes de ce type ont été développés : Stepwise [109], Tiling [133], Upstart [71], Offset [128, 126, 129], Pyramid [77], Tower [78], Bincor [182], Perceptron Cascade [33], Sequential Learning [125], BLTA (Boolean Like Training Algorithm) [80], Target Switch [34], SWL (Sequential Window Learning) [145], méthode basée sur les diagrammes de Voronoi [27] ...

Nous nous intéresserons particulièrement à cette dernière catégorie, car ce sont les plus intéressants d'un point de vue implémentation silicium et du fait de la simplicité de la fonction d'activation, ils sont plus rapides. Dans ANDI, l'utilisation de neurones à sigmoïdes posa des problèmes de temps de calcul. Les contraintes temps réel n'étaient pas respectées à cause des sigmoïdes à calculer. Pour la conception de puces neuronales, deux voies sont possibles : l'implantation analogique et l'implantation numérique. La première possibilité pose des problèmes au niveau du stockage de la valeur des connexions. Ce stockage peut se faire de manière capacitive ou résistive. La première solution est imprécise et occupe de la place sur le circuit, la seconde occupe aussi de la place et pose des problèmes d'échauffement. De plus ces deux types d'intégration n'échappent pas aux différents inconvénients liés à la technologie analogique : les instabilités électriques, la sensibilité aux bruits... Donc la solution numérique paraît beaucoup plus intéressante. Et dans ce cadre, la conception de neurones binaires est beaucoup plus simple que tout type de neurones, notamment les neurones à sigmoïde, d'où l'intérêt des algorithmes de construction de réseaux de neurones binaires.
Les algorithmes de construction de réseaux de neurones binaires peuvent aussi être classés en trois catégories, suivant la méthode utilisée pour apprendre une unité :

- chaque unité est apprise afin de minimiser le nombre d'erreurs sur une certaine base d'apprentissage. C'est la catégorie qui compte le plus d'algorithmes : Stepwise, Tiling, Upstart, Offset,

Pyramid, Tower, Bincor, Perceptron Cascade ... Les stratégies sont diverses et fournissent des structures de réseaux variées. Les variantes du Perceptron utilisées sont principalement le Pocket algorithme [76], la version Ratchet [78] de ce dernier et le Thermal [72] Perceptron. Nous présenterons dans le chapitre 5 un autre algorithme utilisable par ces procédures de construction, le BCP Min.

- chaque unité est apprise afin de maximiser le nombre de vecteurs d'entrée exclus, c'est-à-dire que d'un côté de l'hyperplan, un nombre maximum d'exemples de même classe est séparé des autres. C'est la stratégie développée dans le Sequential Learning. L'algorithme SWL utilise aussi cette stratégie mais avec des neurones spéciaux appelés *Window Neuron*. Ce sont en fait des unités à double seuil dont la sortie est à un dans une zone délimitée par deux hyperplans parallèles. L'algorithme Target Switch est un peu particulier car il utilise une procédure de minimisation du nombre d'erreurs et modifie le seuil de l'hyperplan pour obtenir un ensemble d'exemples exclus. Dans le chapitre 5 nous présenterons une nouvelle méthode efficace pour trouver un ensemble de vecteurs d'entrée exclus, le BCP Max. Et apparemment c'est la première méthode de ce genre.

- les unités sont construites par d'autres procédés souvent géométriques (diagrammes de Voronoi...). Le BLTA est basé sur une recherche dans des tables de vérités (il ne peut apprendre que des fonctions booléennes).

Après avoir détaillé les extensions du Perceptron les plus utilisées pour minimiser le nombre d'erreurs commises par une unité, nous décrirons deux algorithmes de construction : l'Upstart et le Sequential Learning.
Il convient de signaler que cette liste n'est pas du tout exhaustive car les algorithmes de construction sont un sujet particulièrement étudié et de nouveaux algorithmes sont proposés regulièrement.

3.5.2 Les extensions du Perceptron pour minimiser le nombre d'erreurs

L'algorithme 3.1 décrit une procédure très simple de calcul du nombre d'erreurs pour une position donnée de l'hyperplan. Cette procédure servira dans la présentation des divers algorithmes. La base d'apprentissage est toujours notée (ξ^μ, t^μ) avec $\xi^\mu \in \mathbb{R}^{n+1}$ (la première composante de chaque vecteur d'entrée étant 1).

Algorithme 3.1 Fonction $Err(w)$ qui fournit le nombre d'erreurs commises par l'hyperplan w

$Err(w) = 0$
Boucle sur les exemples ξ^μ
    $s = \phi(w.\xi^\mu)$
    Si $(s \neq t^\mu)$ alors $Err(w) \longleftarrow Err(w) + 1$
Fin Boucle $[\xi^\mu]$

Les extensions du Perceptron que l'on va présenter sont les plus utilisées, mais là encore, cette liste n'est pas exhaustive. D'autres algorithmes ont été proposés, dont certains assez récemment, que nous n'avons pas eu le temps de tester : *l'Adatron* [5], *le Minimerror* [163], et *le Minover* [112]. On peut aussi citer *le Cone Perceptron* [191], qui est une version généralisée du Perceptron pour les bases d'apprentissage linéairement séparables, mais qui ne minimise pas le nombre d'erreurs.

Extension simple du Perceptron

Algorithme 3.2 Extension simple du Perceptron

$N_{err} = N + 1$, $t = 0$ et initialisation de w au hasard
Tant que $(t < t_{max})$ et $(N_{err} \neq 0)$ faire
    Boucle sur les exemples ξ^μ (présentation aléatoire)
        $s^\mu = \phi(w.\xi^\mu)$
        $\varepsilon = s^\mu - t^\mu$
        Si $(\varepsilon \neq 0)$ alors $w \longleftarrow w + \nu\varepsilon\xi^\mu$
    Fin Boucle $[\xi^\mu]$
    Si $(Err(w) < N_{err})$ alors $N_{err} = Err(w)$ et $w_m = w$
Fin Tant que

Comme on l'a déjà abordé plus haut, sur des problèmes non linéairement séparables, le Perceptron va passer successivement par une multitude de positions. Afin de trouver la position qui minimise le nombre d'erreurs, l'idée la plus simple est de tester la qualité de la position après la présentation des exemples de la base d'apprentissage, et on conserve en mémoire la meilleure position trouvée. L'algorithme 3.2 décrit cette méthode.

Le Pocket algorithm

Le Pocket algorithm [76] est basé sur l'idée que dans le cas de problèmes non linéairement séparables, le Perceptron va passer par au moins une position pour toutes les partitions en deux ensembles de la base d'apprentissage. Il va donc passer par la position optimale, celle qui minimise le nombre d'erreurs. Pour ne pas la manquer, un système de mémorisation (une *mise en poche*) de la meilleure position trouvée est utilisé comme dans l'algorithme précédent. Mais le test est différent. L'auteur de cet algorithme propose de tester le nombre d'exemples qui sont bien classés successivement. Il montre alors que cette méthode tend vers la solution optimale.

Algorithme 3.3 L'algorithme Pocket

$N_{ok} = 0$, $N_{ok}^m = 0$, $t = 0$ et initialisation de w au hasard
Tant que $(t < t_{max})$ et $(N_{ok}^m < 2N)$ faire
    Boucle sur les exemples ξ^μ (présentation aléatoire)
        $s^\mu = \phi(w.\xi^\mu)$
        $\varepsilon = s^\mu - t^\mu$
        Si $(\varepsilon \neq 0)$ alors $w \longleftarrow w + \nu\varepsilon\xi^\mu$ et $N_{ok} = 0$
        Sinon faire
            $N_{ok} \longleftarrow N_{ok} + 1$
            Si $(N_{ok} > N_{ok}^m)$ alors $N_{ok}^m = N_{ok}$ et $w_m = w$
        Fin Sinon
    Fin Boucle $[\xi^\mu]$
Fin Tant que

La version Ratchet du Pocket algorithm

C'est une amélioration directe du dernier algorithme, afin de trouver plus rapidement la position optimale [78]. Le problème du Pocket est qu'il peut passer par cette position mais ne pas y rester, car le nombre successif de présentations d'exemples sans erreur dépend de l'ordre dans lequel ils sont présentés. Ainsi une autre position pourrait remplacer la position optimale car sur un sous-ensemble d'exemples présentés successivement, elle ne commet pas d'erreur, alors que globalement elle n'est pas optimale. Le principe est donc de calculer le nombre d'erreurs lorsqu'une position est susceptible d'être mémorisée. Finalement, elle ne le sera que si le nombre d'erreurs commises est inférieur à celui de la position déjà en mémoire.
Comme nous le verrons dans les simulations présentées dans le chapitre 5, cette version est beaucoup plus efficace que le Pocket simple. Mais son problème majeur est le temps de calcul qui croît très rapidement avec la taille de la base d'apprentissage. La complexité algorithmique du Perceptron simple et de l'extension présentée précédemment est de $O(nN)$. Pour le Pocket et le Ratchet, le calcul de la complexité algorithmique est beaucoup plus complexe et nous n'avons pas trouvé une réponse à cette question dans la littérature. Etant donnée la croissance des temps de calculs, le Pocket est *peut-être* en $O(nN)$ mais certainement pas le Ratchet.

Algorithme 3.4 La version Ratchet du Pocket

$N_{err} = N + 1$, $N_{ok} = 0$, $N_{ok}^m = 0$, $t = 0$ et initialisation de w au hasard
Tant que $(t < t_{max})$ et $(N_{err} \neq 0)$ faire
  Boucle sur les exemples ξ^μ (présentation aléatoire)
    $s^\mu = \phi(w.\xi^\mu)$
    $\varepsilon = s^\mu - t^\mu$
    Si $(\varepsilon \neq 0)$ alors $w \longleftarrow w + \nu\,\varepsilon\,\xi^\mu$ et $N_{ok} = 0$
    Sinon faire
      $N_{ok} \longleftarrow N_{ok} + 1$
      Si $(N_{ok} > N_{ok}^m)$ faire
        Si $(Err(w) < N_{err})$ alors $N_{err} = Err(w)$, $N_{ok}^m = N_{ok}$ et $w_m = w$
      Fin Si
    Fin Sinon
  Fin Boucle $[\xi^\mu]$
Fin Tant que

Le Thermal Perceptron

Le Thermal Perceptron développé par M. Frean [72] est une extension relativement simple du Perceptron. La pertinence de la méthode réside dans la valeur du paramètre d'apprentissage qui dépend à la fois de la position du vecteur d'entrée considéré et du temps. Le but de la technique est double :

- faire décroître l'amplitude des modifications du vecteur w pour atteindre en fin d'algorithme une position stabilisée,
- favoriser les modifications de position pour les vecteurs d'entrée qui sont proches de l'hyperplan. Intuitivement, cela signifie que les exemples qui sont mal classés, proches de l'hyperplan, ne nécessitent que de faibles variations du vecteur w donc il y a peu de risques d'altérer la qualité de la position déjà prise, alors que les exemples loin de l'hyperplan nécessitent de fortes variations et donc risquent de fortement modifier le nombre d'erreurs.

La règle d'apprentissage proposée nommée Thermal PLR, est alors la suivante

$$\Delta w_i = \nu_t \left(s^\mu - t^\mu\right) \xi_i^\mu \, e^{-\frac{|w.\xi^\mu|}{T_t}} \tag{3.15}$$

Dans cette équation, le terme $|w.\xi^\mu|$ représente la distance euclidienne du vecteur d'entrée ξ^μ multipliée par la norme du vecteur normal à l'hyperplan. Le paramètre T_t agit comme une température qui contrôle l'amplitude des modifications au cours de l'apprentissage. Pour les exemples très proches de l'hyperplan, l'exponentielle sera pratiquement égale à 1, donc la modification de la position ne sera pas atténuée. En revanche, pour les vecteurs d'entrée éloignés de l'hyperplan, l'exponentielle sera proche de 0 et l'hyperplan ne bougera que très peu.
La stabilisation de la solution est assurée par la convergence vers 0 du paramètre ν_t et de la température T_t. L'auteur suggère simplement une décroissance linéaire vers 0 à partir de ν_0 et de T_0. Le paramètre d'apprentissage initial ν_0 peut être pris à 1. Pour la température initiale, le choix est très difficile. Dans [72], certaines simulations donnent des températures initiales optimales entre 1 et 1,5 pour certains problèmes, mais proche de 15 pour un autre.

Algorithme 3.5 Le Thermal Perceptron

$N_{err} = N + 1$, $t = 0$, $T = T_0$, $\nu = \nu_0$ et initialisation de w au hasard
Tant que $(t < t_{max})$ et $(N_{err} \neq 0)$ faire
    Boucle sur les exemples ξ^μ (présentation aléatoire)
        $A^\mu = w.\xi^\mu$
        $s^\mu = \phi(A^\mu)$
        $\varepsilon = s^\mu - t^\mu$
        Si $(\varepsilon \neq 0)$ alors $w \longleftarrow w + \nu \varepsilon \xi^\mu e^{-|A^\mu|/T}$
    Fin Boucle $[\xi^\mu]$
    Si $(Err(w) < N_{err})$ alors $N_{err} = Err(w)$ et $w_m = w$
    $\nu \longleftarrow \nu - \nu_0/t_{max}$
    $T \longleftarrow T - T_0/t_{max}$
Fin Tant que

Nous verrons dans le chapitre 5 que le choix de cette température initiale est extrêmement délicat, et qu'une mauvaise valeur dégrade beaucoup les performances. Mais en moyenne nous verrons que cet algorithme est plus efficace que les précédents, et assez rapide, malgré le calcul des exponentielles.

3.5.3 Deux algorithmes de construction

Nous allons présenter un algorithme de la première catégorie l'Upstart, et un de la seconde, le Sequential Learning. Parmi les algorithmes qui apprennent une unité par minimisation du nombre d'erreurs, l'Upstart est l'un des plus efficaces en termes de nombre de neurones construits. Cet algorithme sera utilisé dans les simulations présentées dans le chapitre 5. Le Sequential Learning utilise une stratégie *naturelle* et très différente de celle de l'Upstart. Nous décrirons cette stratégie de construction car elle est à la base d'algorithmes que nous présenterons plus loin.

L'algorithme Upstart

L'algorithme Upstart proposé par M. Frean [71], construit un réseau de neurones binaires sous forme d'arbre binaire pas forcément équilibré. Chaque unité est connectée à ses unités filles (deux au maximum) et à l'ensemble des entrées. La profondeur de l'arbre correspond donc au nombre de couches

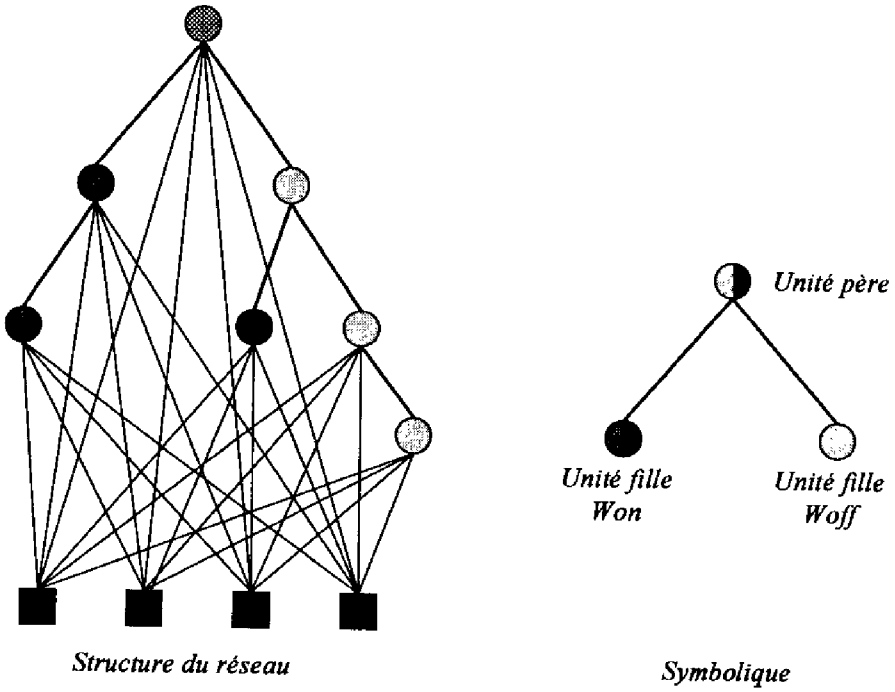

FIG. 3.11: Un réseau Upstart

du réseau. Chaque unité est apprise sur une base d'apprentissage spéciale, par une des extensions du Perceptron, afin de minimiser le nombre de vecteurs d'entrée mal classés.
L'algorithme commence par apprendre la base d'apprentissage d'origine sur un premier neurone. S'il ne commet pas d'erreurs alors la base est linéairement séparable et l'algorithme s'arrête. Sinon les exemples mal classés sont de deux types :

- ceux dont la cible était 0 et pour lesquels le réseau donne 1, ils seront appelés Won (Wrongly On)
- ceux dont la cible était 1 et pour lesquels le réseau donne 0, ils seront appelés Woff (Wrongly Off)

Ensuite deux nouvelles bases d'apprentissage sont créées, avec tous les vecteurs d'entrée de la base d'origine mais avec des cibles particulières, et apprises sur deux unités filles : une unité pour corriger les exemples Won et une autre pour corriger les Woff. Les exemples de la première base ont pour cible 1 si ce sont des Won et 0 sinon. Les exemples de la seconde base ont pour cible 1 si ce sont des Woff et 0 sinon. Lorsque ces deux unités sont apprises, la valeur des connexions les reliant à l'unité père est déterminée en fonction des connexions reliant cette unité père aux entrées, nous noterons logiquement ces poids par w_{on} et w_{off}. Supposons que les unités filles aient appris leur base d'apprentissage et qu'elles répondent correctement, alors

- pour les exemples qui étaient bien classés par l'unité père, les unités filles répondent 0. Donc quelle que soit la valeur des connexions w_{on} et w_{off}, la sortie de l'unité père ne sera pas modifiée.
- pour les exemples Won, l'unité père répond 1, l'unité fille Won 1 et l'unité fille Woff 0.
- pour les exemples Woff, l'unité père répond 0, l'unité fille Won 0 et l'unité fille Woff 1.

L'unité Won répond 1 uniquement pour les exemples Won. On peut donc se servir de cette sortie pour inverser la sortie de l'unité père qui est fausse pour ces exemples. Si on note par w_i la valeur des connexions reliant l'entrée à l'unité père, cette inversion peut être réalisée simplement en prenant

$$w_{on} = -(\max|w.\xi^\mu| + \alpha)$$

Avec α un réel positif pour avoir une marge.
De la même manière, l'unité Woff répond 1 uniquement pour les exemples Woff. On peut donc utiliser cette sortie pour inverser la sortie de l'unité père qui est fausse pour ces exemples. On prend alors

$$w_{off} = \max|w.\xi^\mu| + \alpha$$

Si les unités filles ont appris exactement leur base d'apprentissage, on peut vérifier facilement que l'unité père répond parfaitement sur toute la base. Dans le cas où les unités filles font des erreurs sur leur base d'apprentissage alors le même procédé de construction leur est appliqué. Des unités filles sont alors récursivement construites tant qu'il existe une unité qui commet encore des erreurs. Ce procédé fournit donc un arbre binaire qui peut ne pas être équilibré, car une unité peut ne commettre que des erreurs Wrongly On ou que des erreurs Wrongly Off. La figure 3.11 représente un tel réseau.
La convergence de cet algorithme est assurée si et seulement si les unités filles d'une unité quelconque commettent moins d'erreurs que le nombre d'exemples de cible 1 dans leur base d'apprentissage, car sinon leur père commettra plus d'erreurs après leur construction qu'avant. Si cette condition n'est pas vérifiée, le procédé de construction ne s'arrête pas. Cette condition de convergence est assurée dans le cas où tous les exemples de la base d'apprentissage sont en nombre fini et disposés sur un convexe. Car en utilisant le Pocket ou le Ratchet avec un nombre d'itérations suffisamment grand on pourra toujours séparer un exemple des autres et donc l'unité commettra moins d'erreurs que le nombre de 1 dans sa base. Si l'algorithme ne trouve pas un tel hyperplan en un temps raisonnable, on peut toujours séparer un exemple des autres avec l'hyperplan tangent au convexe au point considéré. Dans le cas de problèmes booléens, cette condition est vérifiée car les vecteurs d'entrée se trouvent tous sur une hypersphère (voit chapitre 4). L'hyperplan tangent à l'hypersphère en un point correspondant à un exemple est appelé neurone *grand-mère*. Mais dans le cas analogique, il faut nécessairement faire une projection sur un convexe (des exemples de telles projections sont utilisés dans [127]).
La structure arborescente obtenue est assez particulière, plus difficile à gérer qu'une struture à couches normale et plus lente. Mais l'auteur propose une méthode pour transformer l'arbre en une seule couche cachée comprenant autant d'unités cachées qu'il y avait d'unités dans l'arbre (ce qui fait une unité supplémentaire si on compte la sortie).

Le Sequential Learning

Cette stratégie de construction [125] est assez naturelle, car intuitivement elle consiste à apprendre la base d'apprentissage par une modélisation des frontières entre les exemples de cible 1 et les exemples de cible 0. Une seule couche cachée est générée. Chaque unité est construite de manière à ce que des exemples de même cible soit séparés des autres et qu'ils se trouvent du bon côté de l'hyperplan : si ce sont des exemples de cible 1 ils doivent se trouver du côté positif de l'hyperplan et sinon du côté négatif. Après la construction de chaque neurone, les exemples exclus sont retirés de la base d'apprentissage, et la construction de cette couche se poursuit jusqu'à ce que la base restante soit linéairement séparable. Ce procédé est représenté sur la figure 3.12.
Avec une telle stratégie, l'auteur de l'algorithme montre que la base obtenue pour apprendre le neurone de sortie, c'est-à-dire la base des représentations internes, est linéairement séparable. Sa démonstration

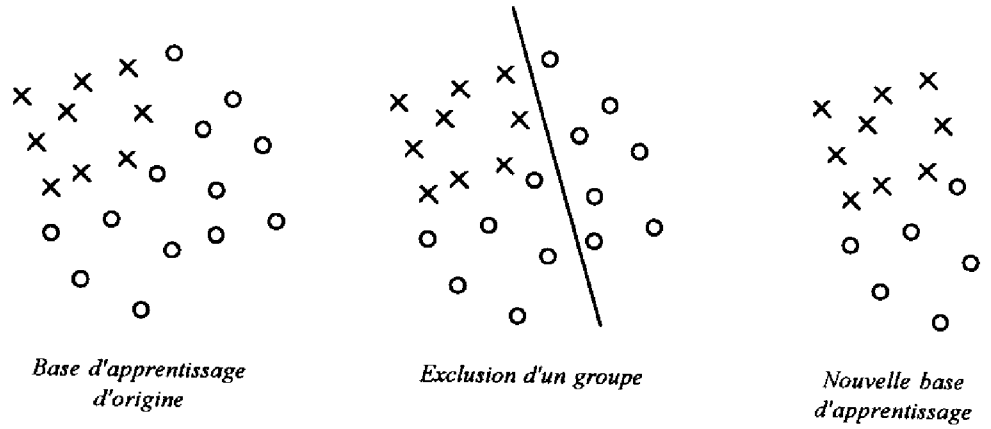

FIG. 3.12: Principe de la stratégie d'exclusion

est constructive : elle donne une solution explicite. Mais la croissance exponentielle de la valeur des connexions limite son utilisation. Aussi, il propose d'apprendre le neurone de sortie avec le Perceptron. Nous ne détaillerons pas sa démonstration car nous en ferons une similaire dans le chapitre 5 (dans le cadre du développement d'une version multi-sorties de cette stratégie). Un exemple d'application de cette stratégie est détaillé sur la figure 3.13. On peut vérifier que la base d'apprentissage du neurone de sortie, représentée comme un ensemble de sommets d'un hypercube en dimension 3 (car 3 neurones ont été construits) est LS.

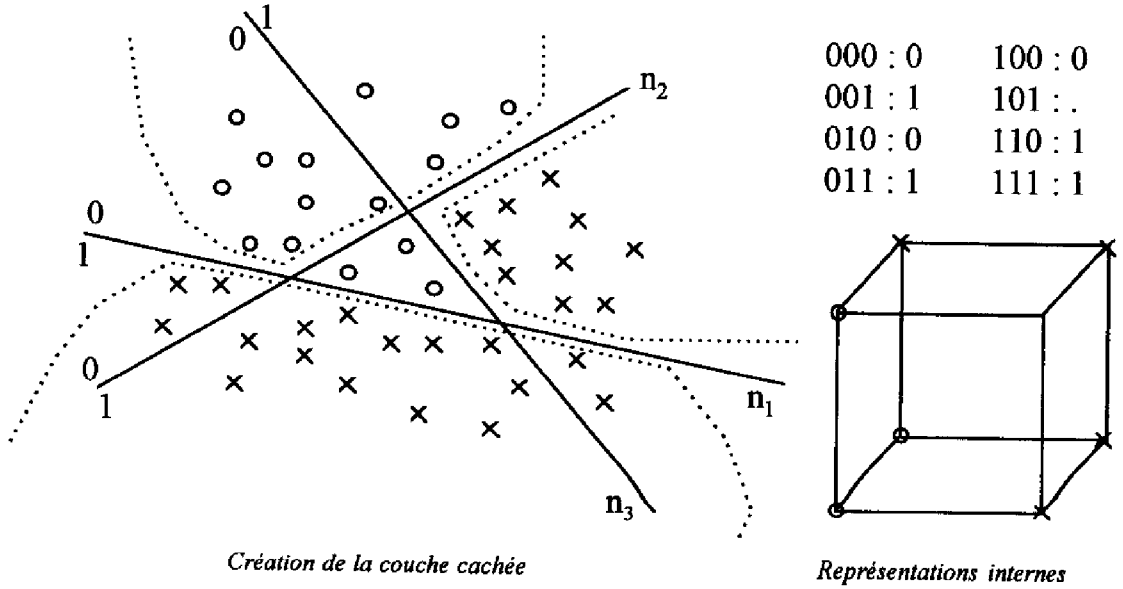

FIG. 3.13: Séparabilité linéaire des représentations internes

La convergence de cette méthode est assurée si et seulement si un algorithme peut trouver un hyperplan qui exclut un ensemble d'exemples de même cible. Un tel algorithme n'existait pas. L'auteur propose une méthode utilisant le Perceptron avec de multiples bases d'apprentissage pour tester si

des ensembles de vecteurs d'entrée de même cible sont LS. Mais cette solution est très longue et complètement rédhibitoire pour de grosses bases d'apprentissage.
Dans l'algorithme SWL [145], l'auteur utilise cette stratégie d'exclusion, et règle le problème précédent par l'utilisation de neurones à deux seuils donc actifs dans une zone délimitée par deux hyperplans parallèles. La construction de chaque neurone est réalisée en recherchant un ensemble d'exemples coplanaires par programmation linéaire. L'algorithme Target Switch n'utilise pas la même stratégie de construction mais a besoin de trouver un hyperplan qui exclut des exemples. Pour cela, il utilise un algorithme de minimisation du nombre d'erreurs tel que le Pocket et il modifie le seuil de l'hyperplan obtenu pour n'avoir que des exemples de même cible d'un côté.
Dans le chapitre 5, nous allons présenter une nouvelle méthode d'exclusion d'exemples. Son utilisation dans le Sequential Learning est très efficace et permet l'apprentissage de bases importantes en peu de temps de calcul.

3.6 Conclusion

Après avoir expliqué les fondements biologiques et les premiers modèles qui en découlèrent, nous avons présenté un état de l'art non exhaustif des méthodes d'apprentissage des réseaux de neurones à couches. Nous avons premièrement détaillé successivement le Perceptron, l'Adaline et la rétropropagation du gradient. Ce dernier algorithme est sans conteste le plus largement étudié et utilisé. Une multitude d'articles traite de cette méthode ou de ses diverses extensions.
Mais comme on l'a expliqué, la rétropropagation du gradient présente de nombreux problèmes. Les problèmes de convergence ont été largement étudiés et des algorithmes d'optimisation beaucoup plus puissants ont été proposés. Mais l'apprentissage de grosses bases d'apprentissage complexes demande toujours beaucoup de temps de calcul. De plus, le problème majeur est le dimensionnement *a priori* du réseau, qui impose de procéder par une série de plusieurs apprentissages avec des structures différentes pour obtenir une solution satisfaisante. Afin de résoudre ces divers problèmes, les méthodes de simplification de réseau et de construction se sont largement développées. Ces méthodes peuvent être regroupées sous le terme *d'apprentissage par modifications structurelles*. Les méthodes de simplification de réseau ont été simplement abordées.
Les algorithmes de construction de réseaux de neurones binaires sont particulièrement intéressants. Afin d'apprendre une unité de manière efficace sur des problèmes non linéairement séparables, plusieurs extensions du Perceptron ont été proposées. Nous avons détaillé les plus utilisées. Toutes les extensions proposées optimisent le même critère : la minimisation du nombre d'erreurs. Nous avons enfin décrit deux algorithmes de construction : l'Upstart car c'est un des plus utilisés, les résultats obtenus sont parmi les meilleurs et il sera utilisé plus loin ; et le Sequential Learning car cette stratégie va nous servir de base au développement de plusieurs algorithmes de construction dans le chapitre 5.
Comme nous l'avons vu, l'Upstart utilise une des méthodes de minimisation du nombre d'erreurs pour apprendre chaque unité. Nous présenterons dans le chapitre 5 une nouvelle méthode de ce type, le BCP Min. Elle sera comparée très favorablement aux algorithmes présentés dans ce chapitre. Le Sequential Learning nécessite une méthode de maximisation du nombre d'exclus. Nous présenterons aussi une telle méthode : le BCP Max, qui est *a priori* la première méthode efficace pour maximiser le nombre d'exclus. La base de ces deux nouvelles méthodes est un algorithme qui converge très rapidement vers une solution dans le cas linéairement séparable. Dans le cas non linéairement séparable leur comportement diffère car le critère optimisé n'est pas le même.
Les algorithmes de construction de réseaux de neurones binaires sont tous restreints à une sortie binaire. Bien que certaines possibilités d'extensions aient été envisagées dans certains articles, aucun développement théorique n'a été trouvé dans la littérature. Ce n'est que très récemment qu'une équipe

américaine [149] a développé des extensions à plusieurs sorties de plusieurs algorithmes : Tower, Upstart, Tiling... Bien que les expérimentations présentées ne soient pas très extensives, l'extension de l'Upstart donne les meilleurs résultats. Ces extensions permettent ainsi la construction de réseaux de neurones binaires pour la classification multi-classes.
Une sévère limitation de la grande majorité des algorithmes utilisant la stratégie de minimisation du nombre d'erreurs est que la convergence est assurée seulement si les vecteurs d'entrée se trouvent sur un convexe. La raison en a été détaillée pour le cas de l'Upsart. Pour le traitement de données analogiques, cette condition impose donc une projection des vecteurs d'entrée sur un convexe ou un codage binaire des données. Ce problème n'est pas présent avec le Sequential Learning. Mais ce dernier n'a pas eu le succès escompté par manque d'une méthode efficace pour apprendre une unité en maximisant le nombre de vecteurs d'entrée exclus.
Dans le chapitre 5, nous présenterons un ensemble d'algorithmes efficaces résolvant ces problèmes. Basés sur le même principe, plusieurs algorithmes d'apprentissage d'une unité binaire ont été développés. Les simulations ont montré une grande efficacité, tant pour la convergence dans le cas de problèmes LS, que pour l'optimisation d'un critère dans le cas non LS (minimisation du nombre d'erreurs ou maximisation du nombre de vecteurs d'entrée exclus). De plus la convergence d'une des versions a été démontrée dans le cas LS. Cette version a un grand intérêt car elle converge vers la solution optimale au sens du pouvoir de généralisation. Un des aboutissements en est deux algorithmes de construction, basés sur la stratégie du Sequential Learning, rapides, avec de bons pouvoirs de généralisation et très généraux : les entrées peuvent être quelconques et les sorties binaires avec les différentes classes codées comme on le désire. Des bases d'apprentissage très importantes peuvent être apprises en de relativement faibles temps de calcul.
Avant de décrire ces nouvelles méthodes, nous allons présenter une étude plus théorique un peu particulière, mais assez fondamentale dans le domaine des réseaux de neurones binaires. Cette étude est une analyse de l'hypercube afin de caractériser les fonctions booléennes linéairement séparables. Ce problème est fondamental car ce sont les fonctions booléennes linéairement séparables qui définissent les positions pertinentes de l'hyperplan dans le cas d'entrées booléennes. Ces positions pertinentes sont en nombre fini. Le cas d'entrées booléennes est important car même dans le cas de données analogiques, après la premières couche de neurones qui réalise une découpe de l'espace d'entrée en zones polyédrales, le nouveau problème est booléen. Plusieurs résultats ont été établis, dont un qui servira à la conception d'une amélioration de deux algorithmes de construction présentés dans le chapitre 5.

Newton, pardonne-moi. Tu as trouvé tout ce qui était possible en ton temps de trouver pour un homme doué de l'intelligence et de la créativité les plus puissantes (...). Les idées dont tu es l'auteur dominent encore notre façon de penser la physique.

ALBERT EINSTEIN

Chapitre 4

Une étude de l'hypercube

4.1 Introduction

L'hypercube, que l'on peut s'imaginer comme la généralisation du carré et du cube à n dimensions, est un objet mathématique qui a passionné beaucoup de chercheurs depuis fort longtemps. Le carré et le cube furent l'objet d'études dès l'Antiquité : le carré pour sa simplicité et le célèbre théorème de Pythagore, et le cube qui a fasciné les Grecs avec les quatre autres solides platoniciens [6] (tetraèdre, octaèdre, isocaèdre et dodécaèdre réguliers).
Depuis une trentaine d'années, l'hypercube a été énormément étudié, et une vaste littérature traite de cet objet. De multiples noms lui ont été donnés : hypercube, n-cube, cosmic-cube, parallélotope... Avec une définition très simple, l'hypercube a une très grande richesse mathématique. Ces recherches se sont généralement focalisées sur les propriétés de graphes, algébriques et combinatoires (par exemple [137, 175]). Beaucoup de ces recherches ont eu des applications en informatique, car ses propriétés de graphes sont fondamentales dans le domaine des réseaux informatiques. Une très grande proportion des publications sur l'hypercube traite de ce sujet, et notamment du problème de plongement d'un graphe dans un hypercube [23, 49, 106]. Des résultats ont aussi été obtenus dans le domaine des codes correcteurs par exemple [138].
Dans le domaine des réseaux de neurones, l'hypercube est étudié de manière géométrique : c'est l'étude de la position d'un hyperplan coupant l'hypercube et séparant ainsi ses sommets en deux parties linéairement séparables. Cet hyperplan est la représentation d'un neurone binaire. Dès le début des réseaux de neurones la résolution des fonctions booléennes par des unités à seuil a suscité un vif intérêt [56, 124]. L'hypercube est implicitement étudié de manière essentiellement algébrique et diverses propriétés des fonctions linéairement séparables sont démontrées. Durant toute l'histoire du connexionisme, divers études, méthodes et algorithmes basés sur une étude de l'hypercube ont été proposés, aussi bien dans le domaine des réseaux récurrents [9, 10] que dans celui des réseaux à couches [107, 172, 180].

4.2 Motivations

Pour un réseau de neurones à couches constitué d'unités binaires, la première couche réalise une décomposition de l'espace d'entrée en diverses zones polyédrales (pouvant être infinies) et implémente une fonction de $\mathbb{R}^n$ dans $\{0,1\}^q$. Les autres couches, s'il y en a, réalisent une fonction booléenne de

$\{0,1\}^q$ dans $\{0,1\}^p$, avec p étant le nombre de neurones de sortie.
L'idée de départ de cette étude est le développement d'un algorithme de construction en deux étapes : la décomposition de l'espace d'entrée en zones, puis la résolution de la fonction booléenne résultante de la représentation interne.
Or les diverses méthodes de construction de réseaux de neurones binaires sont applicables aussi bien sur des problèmes binaires qu'analogiques (mais généralement avec projection sur un convexe), alors que le problème binaire est tout à fait particulier. L'idée était donc de concevoir un algorithme de résolution de fonctions booléennes basé sur une étude géométrique de l'hypercube, spécialisé et plus efficace que les autres méthodes.
Un hyperplan coupant l'hypercube définit deux parties linéairement séparables. Mais il y a une infinité d'hyperplans qui pourraient séparer ces deux parties. Elles définissent donc un ensemble d'hyperplans qui ont exactement le même effet, d'un point de vue neuronal. Donc dans une recherche, un seul de cet ensemble d'hyperplans peut être considéré. L'étude des parties linéairement séparables est donc très intéressante car ce sont elles qui définissent les positions pertinentes des hyperplans. Et ces positions pertinentes sont en nombre fini bien inférieur au nombre de fonctions booléennes en dimension n. Une bonne définition de ces positions pourrait alors servir à des algorithmes de recherche sur des espaces discrets et non continus, ce qui devrait fournir une plus grande efficacité.
Comme nous allons le voir, cette étude de l'hypercube n'a pas abouti à la conception d'une telle méthode. Mais plusieurs résultats ont été trouvés. Un résultat particulier sur la structuration des sommets dans l'hypercube sera utilisé dans le chapitre suivant pour améliorer un algorithme de construction [156]. Plusieurs propriétés des parties linéairement séparables sont démontrées [161] dont une équivalente à une propriété étudiée dans les années soixante, mais exprimée dans le cadre d'une vision topologique de l'hypercube.

4.3 Définitions

L'hypercube en dimension n (ou n-cube) noté H^n, est défini comme l'ensemble des vecteurs binaires en dimension n. Si $\mathbb{N}_n$ est l'ensemble des entiers $\{1,\ldots,n\}$, alors

$$H^n = \{(p_1,\ldots,p_n) \in \mathbb{R}^n \,/\, \forall i \in \mathbb{N}_n\, p_i \in \{0,1\}\}$$

Les éléments de H^n sont les sommets de l'hypercube, et il y en a 2^n. Dans la suite, une fonction booléenne sera considérée comme une partie de H^n, une bijection entre l'ensemble des parties de H^n et l'ensemble des fonctions booléennes de $\{0,1\}^n$ dans $\{0,1\}$ pouvant être facilement définie. Si f est une telle fonction, on lui associe la partie de l'hypercube constituée des sommets de l'hypercube qui sont des vecteurs $x \in \{0,1\}^n$ tels que $f(x) = 1$. Cette application est évidemment une bijection. Le nombre de ces fonctions booléennes ou le nombre de parties de l'hypercube est 2^{2^n}.
D'un point de vue géométrique, on peut montrer que les sommets de l'hypercube se situent tous sur une hypersphère de centre $(1/2,\ldots,1/2)$ et de rayon $\sqrt{n}/2$. En effet si $s = (s_1,\ldots,s_n) \in H^n$ alors

$$(s_1 - \frac{1}{2})^2 + \ldots + (s_n - \frac{1}{2})^2 = \frac{n}{4} = (\frac{\sqrt{n}}{2})^2$$

Soit $B_c = (e_1,\ldots,e_n)$ la base canonique de $\mathbb{R}^n$. Un vecteur $x \in \mathbb{R}^n$ défini par $x = \sum_{i=1}^n x_i e_i$ sera noté de la même manière que le point de l'espace euclidien (O, B_c) qui a les mêmes coordonnées, le contexte permettant de savoir si on parle d'un point ou d'un vecteur. On notera $(x_1,\ldots,x_n)$ et on parlera indifféremment de points ou de vecteurs de $\mathbb{R}^n$. Si $B \subset B_c$, le complémentaire de B dans B_c sera noté $\overline{B}$.
Si $B = (e_{i_1},\ldots,e_{i_r}) \subset B_c$ avec $r \in \mathbb{N}_n$, on définit l'application de $\mathbb{R}^n$ dans $\mathbb{R}^r$ associant à un vecteur $p = (p_1,\ldots,p_n)$ le vecteur $(p_{i_1},\ldots,p_{i_r})$ noté $p_{/B}$. On appellera cette application la B-projection. Pour

un ensemble de points $P \subset \mathbb{R}^n$ la B-projection de P est alors définie par $P_{/B} = \{p_{/B} / p \in P\}$. Par convention $\forall P \subset \mathbb{R}^n \; P_{/\emptyset} = \emptyset$.
Nous allons tout d'abord définir la notion de souscube.

Définition 4.1 *Un ensemble de sommets $P \subset H^n$ est un souscube de dimension r dans l'hypercube de dimension n (que l'on appellera un r_n-souscube)* $\Longleftrightarrow$ Card $(P) = 2^r$ *et* $\exists B \subset B_c$ *avec* Card $(B) = r$, $\exists s \in H^{n-r}$ *tel que* $P_{/B} = H^r$ *and* $P_{/\overline{B}} = \{s\}$.

L'ensemble B est *la base parallèle* du souscube, et $\overline{B}$ *la base perpendiculaire*, s étant *la position* du souscube. Celui-ci sera alors noté $P = H_r^n(B, s)$. Par convention $H_n^n(B, \ldots) = H^n$ (l'unique n_n-souscube est l'hypercube entier) et $\forall s \in H^n \; H_0^n(\emptyset, s) = \{s\}$ (les 0_n-souscubes sont les sommets du n-cube).
Voici un exemple de 2_5-souscube. Si $B = (e_1, e_3)$, $\overline{B} = (e_2, e_4, e_5)$ et $s = (0, 0, 1)$ alors $H_2^5(B, s) = \{(p_1, 0, p_2, 0, 1) / (p_1, p_2) \in \{0, 1\}^2\}$. L'hypercube est ainsi composé d'une multitude de souscubes, des sommets (dimension 0) à l'hypercube tout entier (dimension n). Les souscubes de dimension $n-1$ sont généralement appelés facettes.
On a alors le théorème suivant :

Théorème 4.1 *Le nombre de r_n-souscubes est $2^{n-r}\binom{n}{r}$, et le nombre total de souscubes est 3^n.*

Démonstration 4.1 *Un souscube est entièrement déterminé par sa base parallèle et sa position. Le choix de la base parallèle offre $\binom{n}{r}$ possibilité, et le choix de la position 2^{n-r}, d'où le premier résultat. Le second est obtenu en sommant sur $r \in \{0, 1, \ldots, n\}$* □.

On peut montrer que le centre de tout r_n-souscube (barycentre des sommets) se trouve sur une hypersphère de centre $(1/2, \ldots, 1/2)$ et de rayon $\sqrt{n-r}/2$. En effet si $c = (c_1, \ldots, c_n) \in H^n$ est le centre d'un r_n-souscube $P = H_r^n(B, s)$. Nous avons alors $c_{/B} = (1/2, \ldots, 1/2)$ et $c_{/\overline{B}} = s$. Donc si $B = (e_{i_1}, \ldots, e_{i_r})$ nous avons alors

$$(s_1 - \frac{1}{2})^2 + \ldots + (s_n - \frac{1}{2})^2 = (s_{i_1} - \frac{1}{2})^2 + \ldots + (s_{i_r} - \frac{1}{2})^2 = \frac{n-r}{4} = (\frac{\sqrt{n-r}}{2})^2$$

Pour un sommet $p = (p_1, \ldots, p_n) \in H^n$ le sommet opposé $Opp(p)$ est défini par $Opp(p) = (\widetilde{p_1}, \ldots, \widetilde{p_n})$, avec $\widetilde{p_i}$ la négation de p_i exprimée par $\widetilde{p_i} = 1 - p_i$ pour tout i dans $\mathbb{N}_n$. De la même manière, l'opposé d'un ensemble de sommets est défini par $\forall P \subset H^n \; Opp(P) = \{Opp(p) / p \in P\}$. On peut alors montrer facilement que

Théorème 4.2 $Opp(H_r^n(B, s)) = H_r^n(B, Opp(s))$ *(s et $Opp(s)$ étant des éléments de H^{n-r}).*

Nous allons maintenant introduire des notions relatives à des couples de sommets. La distance utilisée entre deux sommets u et v est la distance de Hamming notée $d(u, v)$, soit le nombre de bits qui diffèrent dans les réprésentations binaires de ces deux sommets. Cette distance est aussi le carré de la distance euclidienne.

Définition 4.2 *P est le souscube propre des deux sommets (u, v) noté $\mathcal{P}(u, v)$* $\Longleftrightarrow$ $P = H_r^n(B, s)$ *tel que $u_{/B} = Opp(v_{/B})$ et $u_{/\overline{B}} = v_{/\overline{B}} = s$.*

Deux sommets u et v appartiennent à leur souscube propre et la distance entre ces deux sommets est la dimension de ce souscube. Le couple (u, v) est aussi appelé une diagonale du souscube propre. Un r_n-souscube possède donc 2^{r-1} diagonales. Par exemple si $s_1 = (0, 1, 1, 0, 0)$ et $s_2 = (0, 0, 1, 1, 0)$ leur souscube propre est défini par sa base $B = (e_2, e_4)$ et sa position $s = (0, 1, 0)$.

Pour un sommet $u \in H^n$ on définit l'ensemble de ses voisins $\mathcal{N}(u)$ par

$$\mathcal{N}(u) = \{v \in H^n / d(u,v) = 1\}$$

Chaque sommet u a exactement n voisins, que l'on peut obtenir simplement en inversant un bit et un seul dans la représentation binaire de u.
Soient u_1 et u_2 deux sommets tels que $d(u_1,u_2) = a$. L'ensemble $\mathcal{N}_{u_2}(u_1)$ est défini comme l'ensemble des voisins de u_1 qui sont plus proches de u_2 que u_1, c'est-à-dire

$$\mathcal{N}_{u_2}(u_1) = \{u \in H^n / d(u_1,u) = 1 \text{ et } d(u_2,u) = a-1\}$$

Nous avons alors

$$\text{Card}\,(\mathcal{N}_{u_2}(u_1)) = a$$

car les éléments de $\mathcal{N}_{u_2}(u_1)$ peuvent être obtenus en inversant un des a bits de u_1 qui sont différents dans la représentation binaire de u_2. Cet ensemble représente donc les voisins de u_1 dans le souscube propre du couple (u_1,u_2). On peut donc montrer facilement

$$\mathcal{N}_{u_2}(u_1) = \mathcal{N}(u_1) \cap \mathcal{P}(u_1,u_2) \tag{4.1}$$

Par définition de la distance de Hamming, on a la propriété suivante

$$d(Opp(u),Opp(v)) = \textstyle\sum_i |\widetilde{u}_i - \widetilde{v}_i| = \sum_i |(1-u_i)-(1-v_i)| = \sum_i |v_i - u_i| = d(u,v) \tag{4.2}$$

Le graphe (H^n,V) est défini de manière naturelle avec V l'ensemble des couples de sommets voisins. Un chemin entre deux sommets s_1 et s_2 est une suite $S = (u_1,\ldots,u_{l+1})$ de sommets telle que $u_1 = s_1$, $u_{l+1} = s_2$ et $\forall\, i \in \{1,\ldots,l\}$ $d(u_i,u_{i+1}) = 1$. La longueur du chemin est $\mathcal{L}(S) = l$. L'ensemble des chemins entre s_1 et s_2 est noté $\mathcal{W}(s_1,s_2)$. Pour un sous-ensemble $P \subset H^n$ le graphe (P,V_P) est le graphe induit.
On définit alors les notions suivantes :

Définition 4.3 $\forall\, P \subset H^n$, *P est connexe* $\Longleftrightarrow$ *(P,V_P) est connexe.*

Définition 4.4 $\forall (s_1,s_2) \in (H^n)^2\ \forall\, S \in \mathcal{W}(s_1,s_2)$, *$S$ est un chemin minimal* $\Longleftrightarrow \mathcal{L}(S) = d(s_1,s_2)$

Définition 4.5 $\forall\, P \subset H^n$, *P est connexe minimal* $\Longleftrightarrow \forall (s_1,s_2) \in P^2\ \exists\, S \in \mathcal{W}(s_1,s_2)\ /\ S \subset P$ *est un chemin minimal.*

La connexité minimale représente une forme de compacité de la partie. Il est évident que si P est connexe minimal alors P est connexe.
Si la distance entre deux sommets est l, un chemin minimal peut être construit en partant d'un des sommets et en inversant succcessivement les bits qui diffèrent dans la représentation binaire des deux sommets. On peut donc en déduire que si S est un chemin minimal entre s_1 et s_2 alors $S \subset \mathcal{P}(s_1,s_2)$.
On peut montrer par récurrence sur la dimension que tout r_n-souscube est minimal connexe. Le nombre de chemins minimaux entre deux sommets d'une diagonale est $r!$.

Le complémentaire P dans H^n est noté $\overline{P}$. Cette partie correspond à la fonction booléenne qui est la négation de la fonction booléenne correspondante à P. Il ne faut pas confondre cette définition avec la définition de l'opposé d'une partie.
On définit alors une partie Non Certainement Linéairement Séparable (NC-LS) par

Définition 4.6 $\forall P \subset H^n$, P *est NC-LS* $\Longleftrightarrow \exists (s_1, s_2) \in P^2 \; \exists (u_1, u_2) \in \overline{P}^2$ *tel que* $\mathcal{P}(s_1, s_2) = \mathcal{P}(u_1, u_2)$.

Une fonction Certainement Linéairement Séparable est le contraire, c'est-à-dire une fonction qui n'est pas NC-LS. Nous verrons par la suite pourquoi ce nom a été choisi.
Pour tout vecteur $w \in \mathbb{R}^{n+1}$, on définit l'application affine f_w de $\mathbb{R}^n$ dans $\mathbb{R}$ par

$$\forall x \in \mathbb{R}^n \quad f_w(x) = w_0 + w_1 x_1 + \ldots + w_n x_n$$

Une partie linéairement séparable (LS) est alors

Définition 4.7 $\forall P \subset H^n$, P *est LS* $\Longleftrightarrow \exists w \in \mathbb{R}^{n+1} \; / \forall p \in P \; f_w(p) > 0 \quad et \quad \forall p \in \overline{P} \; f_w(p) < 0$

4.4 Stratification de l'hypercube

Pour un sommet $u \in H^m$ (un hypercube ou un souscube), soit $\varepsilon_j^m(u)$ l'ensemble des sommets $v \in H^m$ obtenus en inversant exactement j bits de u. On peut montrer facilement que Card $(\varepsilon_j^m(u)) = \binom{m}{j}$, $\varepsilon_0^m(u) = \{u\}$ et $\varepsilon_m^m(u) = \{Opp(u)\}$.
Soit $\mathcal{C}(P)$ le centre de gravité des points de P. Le théorème de stratification peut alors s'exprimer comme suit :

Théorème 4.3 *Soit un* r_n*-souscube* $P = H_r^n(B, s)$, *avec* $s = (s_1, \ldots, s_{n-r}) \in H^{n-r}$. *Le vecteur* $w \in \mathbb{R}^n$ *est défini par* $w = \mathcal{C}(P) - \mathcal{C}(Opp(P))$ *avec* $q = \sum_{i=1}^{n-r} s_i$ *(le nombre de bits à 1 dans s). Pour tout* $j \in \{0, \ldots, n-r\}$ *on considère l'hyperplan* $\mathcal{H}_j(P) : w.x + j - q = 0$, *et l'ensemble des* r_n*-souscubes* $\mu_j(P) = \{ H_r^n(B, v) \,/\, v \in \varepsilon_j^{n-r}(s) \}$. *Nous avons alors*

$$\forall Q \in \mu_j(P) \quad \forall x \in Q \quad x \in \mathcal{H}_j(P)$$

Démonstration 4.3 *On peut montrer que le centre des souscubes* $P = H_r^n(B, s)$ *et* $Opp(P)$ *vérifie* $\mathcal{C}(P)_{/B} = \mathcal{C}(Opp(P))_{/B} = (1/2, \ldots, 1/2)$, $\mathcal{C}(P)_{/\overline{B}} = s$ *et* $\mathcal{C}(Opp(P))_{/\overline{B}} = Opp(s)$. *On en déduit alors que* $w_{/B} = 0$ *et* $w_{/\overline{B}} = s - Opp(s)$.
Si on considère $j \in \{0, 1, \ldots, n-r\}$ *et* $Q \in \mu_j(P)$, *alors* $\exists v \in \varepsilon_j^{n-r}(s)$ *tel que* $Q = H_r^n(B, v)$. *Considérons un sommet* $x \in Q$, *nous avons alors* $x_{/\overline{B}} = v$. *On en déduit*

$$w.x = w_{/B} \cdot x_{/B} + w_{/\overline{B}} \cdot x_{/\overline{B}} = (s - Opp(s)) \cdot x_{/\overline{B}} = (s - Opp(s)) \cdot v$$

On peut supposer sans perte de généralité que v *est obtenu à partir de* s *en inversant les* j *premières composantes. On en déduit alors*

$$w.x = s.v - Opp(s).v = \sum_{i=j+1}^{n-r} s_i - \sum_{i=1}^{j} \widetilde{s_i} = \sum_{i=j+1}^{n-r} s_i - \sum_{i=1}^{j} (1 - s_i) = \sum_{i=j+1}^{n-r} s_i - j + \sum_{i=1}^{j} s_i = \sum_{i=1}^{n-r} s_i - j = q - j$$

Dans l'exemple qui suit, nous avons $n - r = 12$, $j = 5$ *et* $q = 6$. *On peut alors vérifier que* $s.v = 4$, $Opp(s).v = 3$ *et* $s.v - Opp(s).v = 1 = q - j$.

	1 $\quad$ j	$j+1$ $\quad$ $n-r$
s	0 1 1 0 0	1 1 1 1 0 0 0
$Opp(s)$	1 0 0 1 1	0 0 0 0 1 1 1
v	1 0 0 1 1	1 1 1 1 0 0 0

Ainsi $w.x + j - q = 0$, *et* $x \in \mathcal{H}_j(P)$. *Ce qui achève la démonstration* □.

Ce théorème, dont l'expression est un peu compliquée, signifie que si l'on regarde l'hypercube suivant un axe liant un r_n-souscube et son opposé, tous les sommets sont organisés en $n - r + 1$ couches de r_n-souscubes. Et tous les sommets des souscubes d'une certaine couche sont inclus dans un unique hyperplan. Pour $j = 0$ la couche contient un souscube qui est P et pour $j = n - r$ la couche contient un souscube qui est $Opp(P)$. Pour un j quelconque la couche contient $\binom{n-r}{j}$ souscubes.
Pour $j \in \{0, n - r - 1\}$, on définit l'hyperplan $\mathcal{H}'_j(P)$ dont l'équation est $w.x + (j - q + \frac{1}{2}) = 0$. Cet hyperplan est l'hyperplan intermédiaire entre $\mathcal{H}_j(P)$ et $\mathcal{H}_{j+1}(P)$.
Cette stratification du n-cube est représentée de manière symbolique sur la figure 4.1 relativement à un 3_7-souscube.

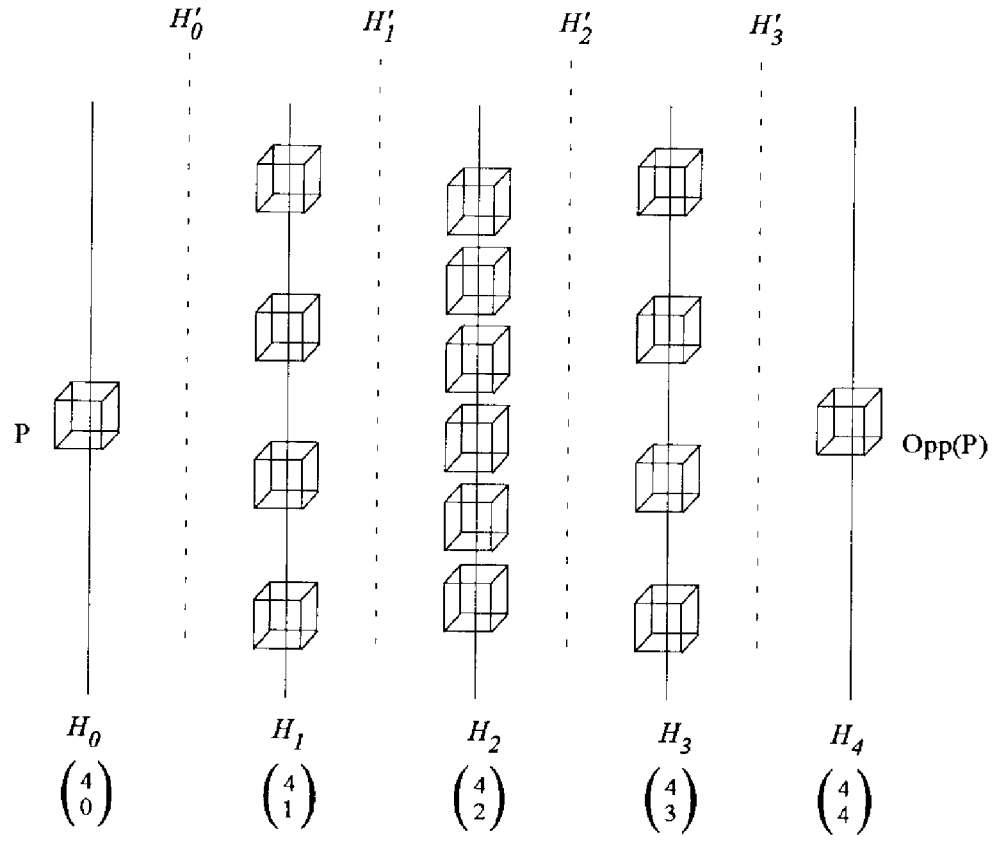

FIG. 4.1: Stratification de l'hypercube pour $n = 7$ et $r = 3$

Théorème 4.4 *Si on considère* P *un* r_n*-souscube* $H^n_r(B, s)$, *le vecteur* $w \in \mathbb{R}^n$ *tel que* $w_{/B} = 0$ *et* $w_{/\overline{B}} = s - Opp(s) = (2s_1 - 1, \ldots, 2s_{n-r} - 1)$ *avec* $s = (s_1, \ldots, s_{n-r}) \in H^{n-r}$. *Alors si* $q = \sum_{i=1}^{n-r} s_i$, *(le nombre de* 1 *dans* s*), l'hyperplan* $\mathcal{H}'_0(P)$: $f(x) = 0$ *avec* $f(x) = w.x + (-q + \frac{1}{2})$ *vérifie :*

$$\forall p \in H^n \quad p \in P \Longrightarrow f(p) > 0 \quad \text{et} \quad p \notin P \Longrightarrow f(p) < 0$$

L'hyperplan $\mathcal{H}'_0(P)$ *sépare les sommets de* P *du reste des sommets de l'hypercube.*

Démonstration 4.4 *La preuve est une conséquence directe du théorème précédent (c'est en fait un cas particulier). Pour les sommets p inclus dans P, c'est-à-dire* $j = 0$, $w.p = q$ *alors* $f(p) = 1/2 > 0$ *et pour les autres , c'est-à-dire* $j \geq 1$, $w.p = q - j$ *donc nous obtenons* $f(p) = q - j - q + 1/2 = -j + 1/2 \leq -1/2 < 0$ □.

Les célèbres *neurones grandmères* (exprimés généralement dans un formalisme binaire en $(-1, 1)$) sont les hyperplans $\mathcal{H}_0(P)$ avec P étant les sommets de l'hypercube, les 0_n-souscubes.

On définit $\mathcal{S}_{ls}$ l'ensemble des parties linéairement séparables, $\mathcal{S}_{sub}$ l'ensemble des souscubes de H^n et $\mathcal{S}_{csub}$ l'ensemble des parties constituées de couches successives de souscubes. Ces dernières sont évidemment linéairement séparables. Nous avons alors les inclusions

$$\mathcal{S}_{sub} \underset{\neq}{\subset} \mathcal{S}_{csub} \underset{\neq}{\subset} \mathcal{S}_{ls}$$

Comme il a été signalé plus haut nous avons $\text{Card}\,(\mathcal{S}_{sub}) = 3^n$.

Théorème 4.5 $\text{Card}\,(\mathcal{S}_{csub}) = 1 + 2\,n\,3^{n-1}$

Démonstration 4.5 *Pour* $r = n$ *nous n'avons qu'une partie de ce type qui est l'hypercube entier. Pour* $r \neq n$ *le nombre de ces parties est* $n - r$ *pour chaque* r_n*-souscube (c'est le nombre d'hyperplans intermédiaires défini plus haut). Donc nous avons alors*

$$\text{Card}\,(\mathcal{S}_{csub}) = 1 + \sum_{r=0}^{n-1} (n-r) 2^{n-r} \binom{n}{r}$$

Sachant que $(n-r)\binom{n}{r} = n\binom{n-1}{r}$, *nous avons enfin*

$$\text{Card}\,(\mathcal{S}_{csub}) = 1 + \sum_{r=0}^{n-1} n 2^{n-r} \binom{n-1}{r} = 1 + 2\,n\,3^{n-1}$$

□.

Ce résultat nous permet donc de minorer le nombre de parties linéairement séparables. Mais ce minorant n'est pas très fin. Un résultat beaucoup plus précis est donné dans [142] pour $n \geq 8$. Avec les majorants donnés dans [124] valables pour $n \geq 2$, nous avons alors l'encadrement

$$2^{\frac{n(n-1)}{2}+32} < \text{Card}\,(\mathcal{S}_{ls}) \leq 2 \sum_{k=0}^{n} \binom{2^n - 1}{k} < \frac{2^{n^2+1}}{n!} < 2^{n^2}$$

D'où la relation sur le comportement asymptotique

$$\frac{1}{2} < \lim_{n \longrightarrow +\infty} \frac{\log \text{Card}\,(\mathcal{S}_{ls})}{n^2} < 1$$

Il semblerait que ce soit le résultat le plus précis obtenu sur le dénombrement des parties LS, qui est encore un problème ouvert. La dernière expression est un peu abusive, car on ne sait même pas si la limite existe.

4.5 Caractérisation des parties linéairement séparables

Théorème 4.6 $\forall P \subset H^n \quad P$ *est C-LS* $\Longrightarrow P$ *est connexe minimal*

Démonstration 4.6 *Nous allons démontrer la contraposée. Soit* $P \subset H^n$ *une partie non connexe minimale. Donc* $\exists\,(s_1, s_2) \in P^2$ *tel que* $\forall\, S \in \mathcal{W}(s_1, s_2)$, $S \subset P \Longrightarrow S$ *n'est pas un chemin minimal. Supposons que* $d(s_1, s_2) = a$. *Nous allons tout de même essayer de construire un chemin minimal entre* s_1 *et* s_2, *en commençant par le chemin incomplet* $S = (s_1, \ -\ , s_2)$.
Tout d'abord supposons que $\mathcal{N}_{s_2}(s_1) \cap P \neq \emptyset$ *et soit* p_2 *un élément de cette intersection. Supposons que* $\mathcal{N}_{s_1}(s_2) \cap P \neq \emptyset$ *et soit* p_a *un élément de cette intersection. On peut alors compléter notre chemin* S *avec*

$$S = (p_1, p_2, \ -\ , p_a, p_{a+1}) \quad \text{avec} \quad p_1 = s_1 \quad \text{et} \quad p_{a+1} = s_2$$

Cette méthode de rapprochement ne peut pas continuer jusqu'à ce que S *soit complet, sinon nous aurions un chemin minimal, ce qui contredirait notre hypothèse. Donc cette construction se termine avec le chemin incomplet*

$$S = (p_1, p_2, \ldots, p_i, \ -\ , p_j, \ldots, p_a, p_{a+1})$$

$$\text{tel que} \qquad i < j \qquad \text{et} \qquad \mathcal{N}_{s_j}(s_i) \cap P = \mathcal{N}_{s_i}(s_j) \cap P = \emptyset \tag{4.3}$$

Les constructions des deux parties de ce chemin (gauche et droite) ne s'arrêtent pas forcément en même temps, une partie peut se terminer avant l'autre. Mais il est obligatoire que les deux s'arrêtent avant qu'elles ne se rencontrent.
Dans ces conditions soit $v_1 \in \mathcal{N}_{s_j}(s_i)$ *donc d'après (4.3) nous avons* $v_1 \in \overline{P}$ *et d'après (4.1) nous obtenons* $v_1 \in \mathcal{P}(p_i, p_j)$.
Le sommet v_2 *est alors défini comme l'opposé de* v_1 *dans* $\mathcal{P}(p_i, p_j)$, *c'est-à-dire tel que* $\mathcal{P}(p_i, p_j) = \mathcal{P}(v_1, v_2)$. *Si* $\mathcal{P}(p_i, p_j) = H_r^n(B, s)$, v_2 *est exactement défini par ses projections complémentaires*

$$v_{2/B} = Opp(v_{1/B}) \quad \text{et} \quad v_{2/\overline{B}} = v_{1/\overline{B}} = s$$

Mais $p_{i/B} = Opp(p_{j/B})$ *et* $p_{i/\overline{B}} = p_{j/\overline{B}} = s$, *et* $d(p_i, v_1) = 1$, *donc en utilisant la propriété (4.2) nous obtenons* $d(p_j, v_2) = d(Opp(p_i), Opp(v_1)) = d(p_i, v_1) = 1$. *On peut en déduire* $v_2 \in \mathcal{N}_{s_i}(s_j)$ *et donc que* $v_2 \in \overline{P}$. *Ce qui nous permet de conclure que* P *est NC-LS, terminant la démonstration.* □.

Théorème 4.7 $\forall P \subset H^n \quad P$ *est LS* $\Longrightarrow P$ *est C-LS*

Démonstration 4.7 *Preuve par l'absurde. Soit* $P \subset H^n$ *une partie LS et NC-LS. Donc* $\exists w \in \mathbb{R}^{n+1}$ *tel que*

$$\forall x \in P \ f_w(x) > 0 \quad \text{et} \quad \forall x \in \overline{P} \ f_w(x) < 0 \tag{4.4}$$

Et nous avons aussi

$$\exists\ (s_1, s_2) \in P^2 \ \exists\ (u_1, u_2) \in \overline{P}^2 \quad \text{tel que} \quad \mathcal{P}(s_1, s_2) = \mathcal{P}(u_1, u_2) = H_r^n(B, s)$$

Dans ces conditions nous obtenons

$$s_{1/\overline{B}} = s_{2/\overline{B}} = u_{1/\overline{B}} = u_{2/\overline{B}} = s$$

$$s_{1/B} = Opp(s_{2/B}) \quad \text{et} \quad u_{1/B} = Opp(u_{2/B})$$

Nous avons donc

$$(s_1 + s_2)_{/\overline{B}} = (u_1 + u_2)_{/\overline{B}} = 2s$$

$$\text{et} \quad (s_1 + s_2)_{/B} = (u_1 + u_2)_{/B} = (1, \ldots, 1)$$

On peut alors en déduire

$$\begin{aligned} f_w(s_1) + f_w(s_2) &= 2w_0 + w_{/B}.(s_1 + s_2)_{/B} + w_{/\overline{B}}.(s_1 + s_2)_{/\overline{B}} \\ &= 2w_0 + w_{/B}.(u_1 + u_2)_{/B} + w_{/\overline{B}}.(u_1 + u_2)_{/\overline{B}} \\ &= f_w(u_1) + f_w(u_2) \end{aligned}$$

Or P est LS, et donc d'après (4.4)

$$f_w(s_1) > 0 \quad f_w(s_2) > 0 \qquad f_w(u_1) < 0 \quad f_w(u_2) < 0$$

Ce qui est absurde. Le théorème est démontré □.

Soient $\mathcal{S}_{cls}$, $\mathcal{S}_{cm}$ et $\mathcal{S}_c$ respectivement l'ensemble des parties C-LS, connexes minimales et connexes. Les théorèmes précédents prouvent que :

$$\mathcal{S}_{ls} \subset \mathcal{S}_{cls} \underset{\neq}{\subset} \mathcal{S}_{cm} \underset{\neq}{\subset} \mathcal{S}_c$$

La figure 4.2 représente une partie P_1 connexe mais non connexe minimale en dimension 3, la figure 4.3 une partie P_2 connexe minimale mais non C-LS, et la figure 4.4 une partie P_3 C-LS et LS.

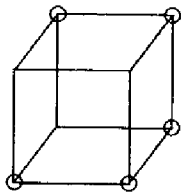

FIG. 4.2: P_1

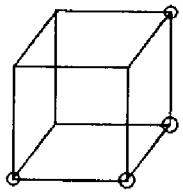

FIG. 4.3: P_2

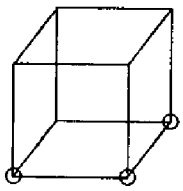

FIG. 4.4: P_3

Afin de saisir ce que la propriété d'être C-LS implique sur la notion de séparabilité, nous allons analyser ce que signifie le fait que deux couples de sommets aient le même souscube propre.

Théorème 4.8 *Si* $\forall (u, v) \in \mathbb{R}^{n2}$, $\mathcal{M}[u, v]$ *est le milieu du segment* $[u, v]$, *alors* $\forall (s_1, s_2, s_3, s_4) \in (H^n)^4$ *distincts, on a les équivalences suivantes :*

$$\begin{aligned} (P1) \quad \mathcal{P}(s_1, s_2) = \mathcal{P}(s_3, s_4) &\iff \mathcal{M}[s_1, s_2] = \mathcal{M}[s_3, s_4] \quad (P2) \\ &\iff (s_1, s_3, s_2, s_4) \text{ est un rectangle} \quad (P3) \\ &\iff]s_1, s_2[\cap]s_3, s_4[\neq \emptyset \quad (P4) \\ &\iff [s_1, s_2] \cap [s_3, s_4] \neq \emptyset \quad (P5) \end{aligned}$$

Démonstration 4.8 *L'équivalence* $(P4) \iff (P5)$ *est simplement dûe au fait que les points sont distincts et que les sommets de l'hypercube sont sur un convexe. On montre alors le théorème par la permutation circulaire suivante* $(P1) \Longrightarrow (P2) \Longrightarrow (P3) \Longrightarrow (P4) \Longrightarrow (P1)$.

$(P1) \Longrightarrow (P2)$ *: Supposons que* $\mathcal{P}(s_1, s_2) = \mathcal{P}(s_3, s_4) = H_r^n(B, s)$. *Donc*

$$2\,\mathcal{M}[s_1, s_2] = s_1 + s_2 \quad \text{et} \quad 2\,\mathcal{M}[s_3, s_4] = s_3 + s_4$$

On a alors

$$2\,\mathcal{M}[s_1, s_2]_{/\overline{B}} = s_{1/\overline{B}} + s_{2/\overline{B}} = 2\,s = s_{3/\overline{B}} + s_{4/\overline{B}} = 2\,\mathcal{M}[s_3, s_4]_{/\overline{B}}$$

De plus

$$2\,\mathcal{M}[s_1, s_2]_{/B} = s_{1/B} + Opp(s_{1/B}) = (1, \ldots, 1) = s_{3/B} + Opp(s_{3/B}) = 2\,\mathcal{M}[s_3, s_4]_{/B}$$

D'où on conclut

$$\mathcal{M}[s_1, s_2] = \mathcal{M}[s_3, s_4]$$

$(P2) \Longrightarrow (P3)$ *: Supposer* $(P2)$ *implique que* (s_1, s_3, s_2, s_4) *est un parallélogramme et que* $\forall i \in \mathbb{N}_n$ $s_{1_i} + s_{2_i} = s_{3_i} + s_{4_i}$. *Comme* $(s_{1_i}, s_{2_i}, s_{3_i}, s_{4_i}) \in \{0,1\}^4$, *il y a seulement six quadruplets possibles donnés dans le tableau qui suit*

Cas	*1*	*2*	*3*	*4*	*5*	*6*
s_{1_i}	*0*	*1*	*1*		*0*	
s_{2_i}	*0*	*1*	*0*		*1*	
s_{3_i}	*0*	*1*	*1*	*0*	*1*	*0*
s_{4_i}	*0*	*1*	*0*	*1*	*0*	*1*

A partir de ce tableau on obtient alors

$$(s_1 - s_3).(s_1 - s_4) = (s_2 - s_3).(s_2 - s_4) = (s_3 - s_1).(s_3 - s_2) = (s_4 - s_1).(s_4 - s_2) = 0$$

Donc le parallélogramme (s_1, s_3, s_2, s_4) *est un rectangle.*

$(P3) \Longrightarrow (P4)$ *: Evident.*

$(P4) \Longrightarrow (P1)$ *:* $(P4)$ *implique* $\exists\, p \in]\, s_1, s_2\, [\, \cap\,]\, s_3, s_4\, [$, *donc*

$$\exists\, (\mu, \lambda) \in]0,1[^2 \quad \text{tel que} \quad \mu s_1 + (1-\mu)s_2 = \lambda s_3 + (1-\lambda)s_4 = p$$

Comme $\forall i \in \mathbb{N}_n$ $(s_{1_i}, s_{2_i}, s_{3_i}, s_{4_i}) \in \{0,1\}^4$, *il existe seulement six quadruplets (les mêmes que ceux détaillés dans le tableau précédent) qui conduisent à des équations possibles* $0 = 0$, $1 = 1$, $\mu = \lambda$ *et* $\mu + \lambda = 1$.
En analysant ce tableau on peut en déduire que

$$s_{1_i} = s_{2_i} \Longrightarrow s_{3_i} = s_{4_i} = s_{1_i} = s_{2_i} \qquad \text{et} \qquad s_{1_i} = \widetilde{s}_{2_i} \Longrightarrow s_{3_i} = \widetilde{s}_{4_i}$$

Si on suppose que $\mathcal{P}(s_1, s_2) = H_r^n(B, s)$, *alors*

$$s_{3/\overline{B}} = s_{4/\overline{B}} \quad \text{et} \quad s_{3/B} = Opp(s_{4/B})$$

donc $\mathcal{P}(s_3, s_4) = H_r^n(B, s)$ $\square$.

Dans le cas où nous avons $(s_1, s_2) \in P$ et $(s_3, s_4) \in \overline{P}$,

la propriété $(P3)$ *est appelée une position de XOR généralisé.*

En étudiant de multiples exemples en faibles dimensions, le fait d'être C-LS semble être une caractérisation des parties LS. Nous l'avons conjecturé très longtemps et vérifié par simulations en faibles dimensions.

Toutes les parties C-LS jusqu'à la dimension 8 sont linéairement séparables.

La vérification n'a pas été faite par une génération systématique des parties de l'hypercube, car en dimension 8 le nombre de parties à générer est $2^{2^8} = 2^{256} \simeq 10^{77}$ ce qui est déjà de l'ordre de grandeur du nombre estimé de particules dans l'univers. En fait, seulement un élément de chaque *type* de fonction C-LS a été généré par un algorithme assez complexe (voir paragraphe suivant).
Si la conjecture (C-LS $\Longleftrightarrow$ LS) était vraie, d'après le théorème précédent on peut en déduire que la nonséparabilité est toujours causée par au moins deux couples de sommets en position de XOR généralisé. Comme le XOR caractérise la nonséparabilité en dimension 2, les XOR généralisés caractérisent la nonséparabilité en dimension quelconque.

4.6 Automorphismes affines et simulations

Le but des simulations était de vérifier la véracité de la conjecture en dimension la plus élevée possible. Avant de détailler la méthode et l'algorithme employé nous introduisons plusieurs notions relatives aux propriétés de symétrie de l'hypercube, qui permettent de réduire de manière très importante la quantité de calculs pour vérifier cette conjecture.

Définition 4.8 *Une isométrie affine h laissant globalement l'hypercube invariant sera appelée un automorphisme affine de H^n. On notera I_n l'ensemble de ces applications.*

Théorème 4.9 *L'ensemble I_n a une structure de groupe et* Card $(I_n) = 2^n n!$

Démonstration 4.9 *La structure de groupe est simple à montrer, mais on pourra reporter le lecteur à une étude très détaillée dans [6]. Dans cet ouvrage le dénombrement de I_n est aussi réalisé par des considérations purement algébriques. Nous allons donner une démonstration intuitive de ce théorème. Pour construire une isométrie affine il faut définir l'image de $n+1$ points affinement indépendants, autrement dit, l'image d'un point par l'application affine et l'image de n vecteurs indépendants par l'application linéaire associée. Il faut de plus que cette application soit une isométrie. On va donc définir l'image de l'origine et de ses n voisins. L'image de l'origine O peut être n'importe quel autre sommet de l'hypercube, nous avons donc 2^n possibilités, on la notera $h(O) \in H^n$. Mais pour conserver les distances il faut nécessairement que l'image des voisins de O soit des voisins de $h(O)$. Si on choisit l'image du premier voisin de O dans les voisins de $h(O)$ nous avons n possibilités. Ensuite pour le second nous avons $n-1$... Le nombre d'affectations possibles est donc $n!$. Il est aisé de vérifier qu'une telle application est une isométrie qui laisse l'hypercube globalement invariant, et que ce sont les seules. Nous avons donc bien* Card $(I_n) = 2^n n!$. *Du point de vue des fonctions booléennes, ces automorphismes affines peuvent aussi être considérés comme des permutations des variables, combinées avec des inversions.*

Définition 4.9 *Sur l'ensemble des parties de H^n on définit la relation binaire suivante*

$$P \mathcal{R} Q \quad \Longleftrightarrow \quad \exists h \in I_n \ / \ h(P) = Q$$

Il est évident de montrer que cette relation est une relation d'équivalence. Elle induit donc une partition de l'ensemble des parties de H^n. Deux éléments de la même classe d'équivalence seront dits de même *type*. D'un point de vue de graphe, quand deux parties sont de même type, alors leurs graphes induits sont isomorphes. Dans [184], l'auteur arrive à dénombrer le nombre de types de fonctions booléennes en dimension quelconque. Cet article est considéré comme *"a tour de force research paper"* par F. Harary (spécialiste des graphes [85] qui a beaucoup étudié l'hypercube [88, 86]) dans un article [87] où il étudie les fonctions booléennes considérées comme des parties de l'hypercube.

Théorème 4.10 *Si on note par $\widehat{P}$ la classe d'équivalence de P, et par $I_{(P)}$ l'ensemble des éléments de I_n qui laissent invariante la partie P, alors nous avons*

$$\text{Card}\,(I_n) = \text{Card}\,(\widehat{P})\ \text{Card}\,(I_{(P)})$$

Démonstration 4.10 *Il est facile de montrer que $I_{(P)}$ est un sous-groupe de I_n, donc* Card $(I_{(P)})$ *divise* Card (I_n). *Le quotient peut alors être interprété grâce au théorème de la théorie des groupes, dit de "la transitivité des indices", donné dans [6]. Mais on peut démontrer ce théorème de la manière suivante. Si on considère P et Q de la même classe d'équivalence soit $\widehat{P} = \widehat{Q}$, alors on peut montrer que* Card $(I_{(P)})$ = Card $(I_{(Q)})$, *en établissant une bijection entre $I_{(P)}$ et $I_{(Q)}$ grâce à l'isométrie qui transforme P en Q. Donc si on prend un élément h de I_n, soit $h \in I_{(P)}$, soit $h \notin I_{(P)}$ mais dans ce cas $h(P) \in \widehat{P}$, donc* Card $(I_{(h(P))})$ = Card $(I_{(P)})$. *En itérant ce processus jusqu'à ce qu'il n'y ait plus d'élément dans I_n nous pouvons donc en déduire que l'ensemble I_n est partitionné en* Card $(\widehat{P})$ *sous-ensembles de cardinalité* Card $(I_{(P)})$. *D'où le théorème.* □.

Ce dernier théorème n'a pas été utilisé dans les simulations, mais il permet de calculer le nombre d'éléments d'une classe d'équivalence à partir d'un seul de ses éléments, et du nombre d'isométries qui laissent cet élément invariant.

Théorème 4.11 $\forall P \subset H^n \quad \forall h \in I_n \quad P$ *est LS* $\iff$ $h(P)$ *est LS*

Démonstration 4.11 *Si P est LS cela signifie qu'il existe un hyperplan f_w qui sépare les sommets de P du reste des sommets de l'hypercube. Alors l'hyperplan $h(f_w)$ sépare les sommets de $h(P)$ du reste des sommets, car une isométrie conserve les distances* □.

Ce théorème, évident, est très important car il permet de réduire de manière très substancielle la quantité de calculs pour vérifier la conjecture, et nous a permis de la vérifier jusqu'à la dimension 8.

> *Ainsi pour une dimension donnée n, la vérification de la conjecture ne nécessite que la génération d'un seul élément de chaque type de partie C-LS et la vérification de sa séparabilité.*

La génération des parties C-LS se fait de manière récursive sur p le nombre de sommets d'une partie. Avant de présenter la méthode utilisée, nous définirons la notion de voisinage d'une partie.

Définition 4.10 *Soit $P \subset H^n$, le voisinage de P, noté $Vois(P)$ est l'ensemble des sommets qui ne sont pas dans la partie P, mais qui ont au moins un voisin dans P soit*

$$Vois(P) = \{s \in H^n / s \notin P \text{ et } \exists\, p \in P \text{ avec } s \in \mathcal{N}(p)\}$$

Le principe de l'algorithme 4.1 présenté ci-dessous, est de générer une partie Q de $n+1$ éléments par union d'une partie à n éléments P et d'un élément du voisinage de P. Donc cette partie Q est forcément connexe. Ensuite on teste si elle est minimale connexe, mais en testant uniquement les chemins qui ont le nouveau sommet s comme extrêmité. Ce n'est pas la peine de vérifier les autres car P était connexe minimale. Enfin on teste si cette partie est C-LS. Si elle l'est alors on vérifie que les parties déjà calculées ne soient pas de même type.
Le temps de calcul croît très rapidement avec la dimension. Ainsi pour $n = 7$ le calcul a nécessité une semaine sur une SPARC 10, et pour $n = 8$ il a fallu 5 mois (avec un "plantage" de machine par semaine au moins !). Toutes les parties C-LS générées sont LS. La vérification a été faite assez rapidement avec l'algorithme BCP présenté dans le chapitre suivant.
Un résultat intéressant de cette simulation est la répartition du nombre de types de parties C-LS en fonction du nombre de sommets. Ainsi les figures 4.5 et 4.6 qui suivent, représentant le nombre de type de fonctions C-LS en fonction du nombre de sommets, nous montrent plusieurs phénomènes.

Algorithme 4.1 Génération des parties C-LS en dimension n.

$C_1 = \{O\}$
Boucle de $n = 2 \longrightarrow 2^n - 1$
  $C_n = \emptyset$
  Boucle sur $P \in C_{n-1}$
    Boucle sur $v \in Vois(P)$
      $Q = P \cup \{v\}$
      Si (Q est connexe minimale) faire
        Si (Q est C-LS) faire
          deja=vrai
          Boucle sur $R \in C_n$
            Si ($\widehat{Q} == \widehat{R}$) deja=vrai
          Fin Boucle [R]
          Si (deja==faux) $C_n \longleftarrow C_n \cup Q$
        Fin si
      Fin Si
    Fin Boucle [v]
  Fin Boucle [P]
Fin Boucle [n]

- Premièrement les courbes sont symétriques autour de 2^{n-1}. Dans le cas où la conjecture est vraie, ceci est tout à fait normal et a confirmé la validité du programme de génération, et la conjecture pour les dimensions testées. En effet si on définit C_- et C_+ comme l'ensemble des parties C-LS qui ont un nombre de sommets respectivement inférieur strict et supérieur strict à 2^{n-1}, on peut aisément définir une bijection entre ces deux ensembles, car en prenant une partie LS dans l'un de ces ensembles on peut lui associer de manière unique la partie complémentaire qui est dans l'autre ensemble.

- Deuxièmement le nombre de types de fonctions C-LS pour 2^{n-1} devient comparativement au maximum très faible quand la dimension augmente. De même on peut remarquer un petit ralentissement des taux de croissance des courbes pour 2^{n-1}. Ces phénomènes doivent s'expliquer par le fait que pour ces nombres de sommets les parties C-LS ont beaucoup de symétries. Donc le nombre de ces parties est très important mais le nombre de types devient faible par rapport à des parties qui ont beaucoup moins de chances d'avoir des symétries. Mis à part ce phénomène, ces courbes ont une allure générale très proche d'une gaussienne.

Le nombre total de types de fonctions booléennes C-LS est résumé dans le tableau 4.6 pour les diverses dimensions testées, avec en comparaison le nombre de types de fonctions booléennes trouvés par D. Slepian dans [184], le nombre de fonctions linéairement séparables et le nombre total de fonctions.
Ce tableau nous montre la très grande rapidité de la croissance doublement exponentielle du nombre de parties quelconques de H^n. Le nombre de types de fonctions booléennes quelconques croît lui aussi très rapidement. Le nombre de fonctions linéairement séparables a une croissance beaucoup moins forte (inférieure à $O(2^{n^2})$), et le nombre de types de fonctions C-LS encore moins, ce qui a permis le calcul jusqu'à la dimension 8. Il convient de noter que la complexité de l'algorithme de génération des parties NC-LS semble difficile à calculer car elle doit dépendre du nombre de types de parties NC-LS et du nombre de parties connexes minimales, deux dénombrements qui ne paraissent pas aisés.

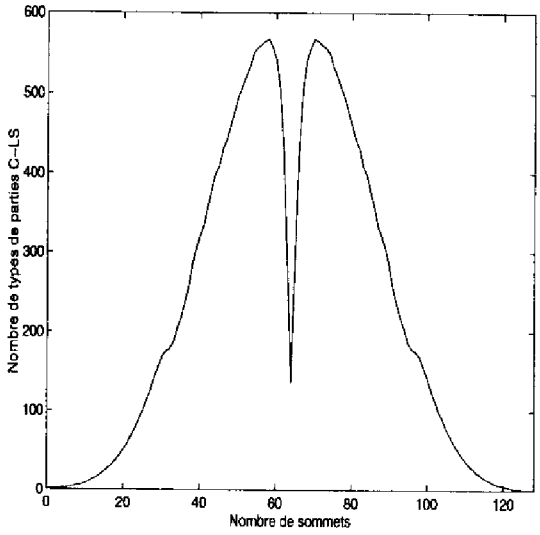

FIG. 4.5: Nombre de types de parties C-LS en fonction du nombre de sommets pour un hypercube en dimension 7.

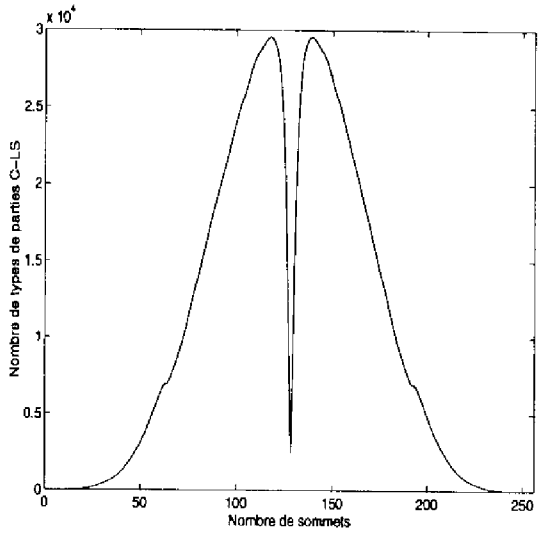

FIG. 4.6: Nombre de types de parties C-LS en fonction du nombre de sommets pour un hypercube en dimension 8.

Dimension	Nombre total de parties	Nombre de types	Card $(\mathcal{S}_{ls})$	Types C-LS
1	4	3	4	3
2	16	6	14	5
3	256	22	104	10
4	65.536	402	1.882	27
5	4.294.967.296	1.228.158	94.572	119
6	$1.8446\ 10^{19}$	400.507.806.843.728	15.028.134	1.113
7	$3.4028\ 10^{38}$	.	8.378.070.864	29.375
8	$1.1579\ 10^{77}$	.	17.561.539.552.946	2.730.166

TAB. 4.1: Nombre total de parties, nombre de types de parties quelconques, nombre de parties LS, nombre de types de parties C-LS (et LS).

Après cette simulation la conjecture nous semblait indubitable. Une démonstration théorique de l'implication C-LS $\Longrightarrow$ LS a été recherchée sans succès. Ce n'est que très récemment que nous avons pris conscience qu'une telle recherche était vaine, car malgré ses apparences trompeuses la conjoncture est fausse. En effet, d'après la propriété $(P2)$ du théorème 4.8 la propriété C-LS est en fait totalement équivalente à une propriété dénommée 2−assommabilité [142] qui s'exprime par :

$$P \text{ est } 2-\text{assommable} \iff \forall (s_1, s_2) \in P \ \ \forall (p_1, p_2) \in \overline{P} \ \ s_1 + s_2 \neq p_1 + p_2$$

Or il a été montré que cette dernière propriété est équivalente à une propriété plus anciennement étudiée, nommée la *monotonie complète*. Pendant de nombreuses années, cette dernière propriété a été considérée comme étant équivalente à la séparabilité linéaire, jusqu'à ce que Gabelman [75] trouve un contre-exemple en dimension 9. Ensuite d'autres contre-exemples ont été exhibés en dimensions plus élevées [142], mais apparement en nombre assez restreint.
La grande période de l'étude de la logique à seuil semble s'être terminée principalement sur l'étude de ce principe d'assommabilité. La seule condition nécessaire et suffisante découverte dans ce cadre est une généralisation nommée l'*assommabilité complète* : propriété qui est en fait totalement équivalente au principe de non-intersection des enveloppes convexes.
Les simulations réalisées étaient exactes car la propriété de 2−assommabilité est bien équivalente à la séparabilité linéaire jusqu'en dimension 8. Ce résultat a aussi été obtenu par Muroga [143] par énumération de toutes les parties complètement monotones jusqu'en dimension 8. Les nombres de types de parties LS jusqu'en dimension 8 donnés dans cet article correspondent exactement à ceux que nous avons trouvé.

4.7 Conclusion

Dans ce chapitre nous avons présenté une étude de l'hypercube ayant pour but la caractérisation des parties linéairement séparables. Plusieurs résultats ont été établis.
Le théorème de stratification de l'hypercube apparaît comme une généralisation des neurones grand-mères, et un cas particulier sera utilisé dans le chapitre suivant pour exclure les sommets d'un souscube des autres sommets. Avec ce théorème, nous avons donc défini un sous-ensemble particulier de parties LS.
Concernant la caractérisation des parties linéairement séparables, nous avons cru à la véracité de la conjecture pendant pratiquement toute la durée de ce travail de thèse. Ce n'est que très récemment que nous avons découvert les nouveaux éléments présentés dans le paragraphe précédent. Mais, n'étant pas influencé par ces études antérieures, nous avons développé une nouvelle approche du problème en nous intéressant aux propriétés de graphe. Certains résultats (connexité, connexité-minimale et la propriété $(P3)$) semblent être nouveaux. Par cette nouvelle approche, le dénombrement des parties C-LS, donc 2−assommables, est peut-être plus facile à envisager. Un tel dénombrement est intéressant car il fournirait certainement un majorant plus fin que le majorant classique.
Le problème de la caractérisation des parties LS reste donc posé. Peut-être que l'analyse des contre-exemples dans le cadre topologique que nous avons développé pourrait fournir de nouveaux résultats. En tous cas, il semble que le concept de souscube ait une importance capitale dans cette étude. Une voie de recherche possible est le développement d'une représentation simplifiée des parties LS par un graphe de souscubes, deux sous-cubes étant définis comme adjacents si et seulement si leur intersection est non vide et l'un n'inclut pas l'autre.
La richesse des structures mathématiques de l'hypercube en fait une source inépuisable de problèmes. Etant donnée son implication dans de multiples domaines, son étude est d'une grande importance.

Mais cette étude est difficile. Cette difficulté d'approche tient beaucoup au fait qu'il est impossible d'avoir des idées géométriques intuitives en dimensions supérieures à 3. Vu le nombre grandissant d'études et de publications touchant à cet objet complexe, l'hypercube va certainement nous révéler encore beaucoup de secrets dans les années à venir.

J'admire la beauté de cette simplicité logique en laquelle je crois,
faite d'harmonie, que nous ne pouvons qu'appréhender avec humilité,
et de façon seulement imparfaite.

ALBERT EINSTEIN

Chapitre 5

BCP : une méthode efficace d'apprentissage d'une unité

5.1 Introduction

Comme on l'a vu dans le chapitre 3, les algorithmes de construction offrent un excellent palliatif aux problèmes majeurs de la rétropropagation : minima locaux, choix *a priori* de la structure du réseau et temps de calcul très longs. Une grande partie de ces algorithmes, Upstart [71], Tiling [133], Offset [128], utilisent les versions améliorées du Perceptron pour apprendre une unité. Ces versions utilisées sont principalement : le Pocket algorithme [76], la version Ratchet de ce dernier et le Thermal Perceptron [72]. Ces améliorations permettent d'obtenir une solution satisfaisante quand le problème est non linéairement séparable (NLS) : elles essayent de placer le mieux possible l'hyperplan et de minimiser le nombre d'erreurs résiduelles. Quand le problème est linéairement séparable (LS), ces algorithmes convergent vers une solution de la même manière que le Perceptron.

Leur utilisation pose deux problèmes. Premièrement ils sont assez sensibles à la manière dont on initialise les poids et deuxièmement, ils sont assez, voire très sensibles, aux divers paramètres d'apprentissage. Ainsi le Pocket et le Ratchet sont assez sensibles au taux d'apprentissage et le Thermal Perceptron est très sensible à la température initiale. Quand celle-ci est bien initialisée, le Thermal Perceptron donne de très bons résultats. Mais comme nous allons le voir par la suite, le réglage de cette température est extrêmement difficile, il dépend complètement du type de problème.

Dans ce chapitre, nous allons présenter un nouvel algorithme d'apprentissage d'une unité binaire, basé sur des principes géométriques. Cet algorithme nommé Barycentric Correction Procedure (BCP) ou Procédure de Correction Barycentrique [155, 159], est particulièrement efficace et surclasse les précédents algorithmes à tous les niveaux, pour les problèmes LS : très grande rapidité de convergence et possibilité de convergence vers la solution optimale (au sens du pouvoir de généralisation), et pour les problèmes NLS : très bonne minimisation du nombre d'erreurs résiduelles, possibilité de trouver rapidement un ensemble d'exemples exclus pour une utilisation avec le Sequential Learning [125]. De plus le bon positionnement des hyperplans fournit finalement un bon pouvoir de généralisation aux réseaux construits.

Historiquement cet algorithme fut développé pour tester la séparabilité linéaire des parties C-LS (voir chapitre précédent) pour vérifier la conjecture C-LS $\Longleftrightarrow$ LS que nous pensions vraie. Son développement est parti de l'intuition géométrique suivante : comme les parties C-LS sont connexes minimales,

elles sont très *compactes* et de plus sur un convexe. En observant quelques exemples en faibles dimensions, la propriété d'être C-LS fait qu'elles sont encore plus compactes. Si on prend un axe liant le barycentre d'une partie avec le barycentre de son complémentaire qui est aussi une partie C-LS, on a de grandes chances de pouvoir séparer ces deux parties. Ainsi une première simulation fut réalisée en prenant comme vecteur des poids le vecteur liant ces deux barycentres, et le seuil fut calculé de manière à séparer ces deux parties. Une telle méthode donna déjà de bons résultats : une grande proportion des parties C-LS furent découpées. Mais certaines résistèrent et la possibilité, dans le cas analogique, d'avoir des distributions LS très irrégulières, imposa alors une correction itérative de la position de ces barycentres. Ainsi la première version opérationnelle de cet algorithme naquit et se révéla d'une extrême rapidité pour les problèmes LS. Par la suite, un théorème de convergence fut recherché. Basé sur l'idée que deux parties LS ont deux enveloppes convexes disjointes, une version spéciale fut développée pour converger vers le connecteur minimal (le vecteur de norme minimal liant les deux enveloppes convexes). Cette version est plus rapide que le Perceptron, mais son principal avantage est de converger vers le connecteur minimal [157]. Enfin la dernière phase du développement du BCP fut l'ajout d'un système de Pocket et de deux procédures, pour optimiser le nombre d'erreurs ou le nombre d'exclus pour les problèmes NLS, et enfin l'amélioration heuristique des quelques paramètres de l'algorithme initial.
Ce développement a ainsi abouti à un algorithme d'apprentissage d'unité à seuil très efficace, utilisable dans une grande partie des algorithmes de construction de réseaux binaires, y compris le Sequential Learning.
Après une présentation générale de l'algorithme (haut niveau de description), les diverses méthodes utilisables, heuristiques ou déterministes, sont détaillées (bas niveaux de description). La section suivante présente les simulations réalisées pour apprendre une unité. Les simulations de la section suivante concernent l'utilisation du BCP dans les algorithmes de construction. La section 5.7 décrit une amélioration du BCPSL, association du BCP et du Sequential Learning, particulièrement rapide et permettant l'apprentissage de problèmes très complexes en un temps très faible comparativement aux autres algorithmes. Cette amélioration est basée sur le théorème de stratification de l'hypercube présenté dans le chapitre précédent [156]. La dernière évolution de cette association efficace est décrite dans la section 5.8 : un algorithme de construction multi-sorties [158]. Une amélioration similaire à celle du BCPSL sera aussi présentée. Nous conclurons ce chapitre après avoir évoqué une possible implantation des réseaux fournis par ces derniers algorithmes.

5.2 L'algorithme général

5.2.1 Rappels et notations

Comme on l'a présenté dans le chapitre 3, une unité à seuil est un système à une sortie connecté à n entrées x_i. Chaque connexion est caracterisée par son poids w_i. L'unité calcule sa sortie en faisant la somme pondérée des entrées par les poids plus la quantité θ seuillée par une fonction de Heaviside ϕ.

$$s = \phi\left(\sum_{i=1}^{n} w_i x_i + \theta\right) = \phi\left(w.x + \theta\right) \tag{5.1}$$

$$\phi(A) = 1 \text{ si } A \geq 0 \qquad \text{et} \qquad \phi(A) = 0 \text{ si } A < 0 \tag{5.2}$$

En général θ est appelé le biais (*bias* en anglais), et le seuil est $-\theta$. Par abus de langage nous nommerons θ le seuil. La quantité $A = w.x + \theta$ est nommée entrée totale.
La base d'apprentissage $C = C_1 \cup C_0$ est composée de deux ensembles de vecteurs d'entrée. Un ensemble $C_1 = \{p_1, \ldots, p_{N_1}\}$ de N_1 vecteurs de cible 1 (ou *target*) et un ensemble $C_0 = \{m_1, \ldots, m_{N_0}\}$ de N_0 vecteurs de cible 0. On définit les ensembles $I_1 = \{1, \ldots, N_1\}$ et $I_0 = \{1, \ldots, N_0\}$.

Dans l'espace d'entrée $\mathbb{R}^n$, les équations 5.1 et 5.2 définissent deux demi-espaces de $\mathbb{R}^n$ séparés par l'hyperplan d'équation $w.x + \theta = 0$, un demi-espace pour lequel $A \geq 0$ que l'on nommera demi-espace positif, et l'autre demi-espace négatif où $A < 0$.
L'apprentissage d'une telle unité consiste donc à trouver, s'ils existent, un vecteur w et un seuil θ tels que l'hyperplan $w.x + \theta = 0$ sépare les points de C_1 des points de C_0, c'est-à-dire que les points de C_1 se situent dans le demi-espace positif et ceux de C_0 dans le demi-espace négatif. Si un tel couple existe on dira que cette base d'apprentissage est LS, sinon elle est NLS. Sans perte de généralité, on définira une fonction LS si les points sont du bon côté et si aucun ne se trouve sur l'hyperplan (on peut facilement montrer que les deux définitions sont équivalentes).
Ainsi on aura C_1 et C_0 LS si et seulement s'il existe un vecteur $w \in \mathbb{R}^n$ et un seuil $\theta \in \mathbb{R}$ vérifiant

$$\forall p_i \in C_1 \quad w.p_i + \theta > 0 \qquad \text{et} \qquad \forall m_i \in C_0 \quad w.m_i + \theta < 0 \tag{5.3}$$

Dans le cas où la base est NLS, le but est de trouver la meilleure solution possible en optimisant un critère, dépendant de l'algorithme utilisé : *minimisation du nombre d'exemples mal classés* ou *maximisation du nombre d'exclus de même cible*. Dans le cas où deux solutions ont la même performance, on gardera celle qui offre les meilleures caractéristiques pour le pouvoir de généralisation.

5.2.2 Description de haut niveau de l'algorithme général

Principes

Le BCP est un algorithme itératif. A chaque itération deux hyperplans seront calculés : $\mathcal{H}^t$ pour garder une convergence rapide dans le cas LS, et $\mathcal{H}^t_{poc}$ qui sert à optimiser le critère choisi. L'expression de ces hyperplans est la suivante :

$$\begin{aligned} \mathcal{H}^t \quad &: \quad w^t.x + \theta^t = 0 \\ \mathcal{H}^t_{poc} &: \varepsilon^t \; w^t.x + \theta^t_{poc} = 0 \quad \text{avec} \quad \varepsilon^t \in \{-1, 1\} \end{aligned}$$

Pour une base d'apprentissage inconnue, l'algorithme est lancé avec un nombre d'itérations limité t_{max}. Au cours de l'apprentissage la meilleure position prise par l'hyperplan $\mathcal{H}^t_{poc}$ est sauvegardée dans un troisième hyperplan $\mathcal{H}_{poc}$. Si cette base est LS, et que le BCP trouve une solution avant t_{max} itérations, l'algorithme est arrêté et la solution retenue est $\mathcal{H}^t$. Si ce n'est pas le cas (base NLS ou LS mais la solution n'est pas trouvée) la solution retenue est $\mathcal{H}_{poc}$.
En général on sauvegarde $\mathcal{H}^t_{poc}$ dans $\mathcal{H}_{poc}$ lors d'une amélioration du critère optimisé (nombre de mal classés plus faible ou nombre d'exclus plus élevé). Dans le cas d'égalité du critère, $\mathcal{H}^t_{poc}$ sera aussi sauvegardé si la quantité $\mathcal{G}^t_{poc}$ que l'on nommera marge (ou *gap*) est plus grande que celle que l'on a sauvegardée $\mathcal{G}_{poc}$. Cette quantité est définie par l'équation suivante :

$$\mathcal{G}^t_{poc} = \min_{x \in C}\{d(x, \mathcal{H}^t_{poc}) / \varepsilon^t \; w^t.x + \theta^t_{poc} \geq 0\} + \min_{x \in C}\{d(x, \mathcal{H}^t_{poc}) / \varepsilon^t \; w^t.x + \theta^t_{poc} < 0\}$$

avec $d(x, \mathcal{H}^t_{poc})$ la distance entre le point x et l'hyperplan $\mathcal{H}^t_{poc}$.
L'optimisation de la marge permet d'accroître la robustesse de la solution. Cette quantité représente la largeur de la zone qui ne contient pas de vecteurs d'entrée autour de l'hyperplan. Plus cette marge est grande, meilleur sera le pouvoir de généralisation. Il est bien entendu plus important d'optimiser le critère choisi que la marge, l'optimisation de celle-ci est donc reléguée à un niveau inférieur. En général, on a pu observer que le critère était rapidement optimisé au début de l'apprentissage, et s'il y a suffisamment d'itérations, la marge est optimisée jusqu'à la fin dès que le critère stagne.

Calcul du vecteur des poids

Soit b_1^t (resp. b_0^t) le barycentre des points de C_1 (resp. C_0) pondérés par l'ensemble de *coefficients de pondération* positifs $\alpha^t = (\alpha_1^t, \ldots, \alpha_{N_1}^t)$ (resp. $\mu^t = (\mu_1^t, \ldots, \mu_{N_0}^t)$). Leur expression est donc

$$b_1^t = \frac{\sum_{i \in I_1} \alpha_i^t p_i}{\sum_{i \in I_1} \alpha_i^t} \qquad \text{et} \qquad b_0^t = \frac{\sum_{i \in I_0} \mu_i^t m_i}{\sum_{i \in I_0} \mu_i^t} \tag{5.4}$$

Ces barycentres sont aussi appelés *combinaison convexe* car b_1^t (resp. b_0^t) est à l'intérieur de l'enveloppe convexe des points de C_1 (resp. de C_0), et les coefficients de pondération *coordonnées barycentriques*. Le vecteur w est alors calculé avec

$$w^t = b_1^t - b_0^t \tag{5.5}$$

Comme on l'a abordé dans l'introduction, le principe de base de l'algorithme est une correction itérative de la position de ces barycentres. A l'itération $t+1$, les coefficients de pondération seront augmentés d'une quantité nommée *modification de pondération* $\beta^t = (\beta_1^t, \ldots, \beta_{N_1}^t)$ et $\delta^t = (\delta_1^t, \ldots, \delta_{N_0}^t)$. Ainsi :

$$\begin{aligned} \forall i \in I_1 \qquad & \alpha_i^{t+1} = \alpha_i^t + \beta_i^t \\ \forall j \in I_0 \qquad & \mu_j^{t+1} = \mu_j^t + \delta_j^t \end{aligned} \tag{5.6}$$

On ne modifiera pas la pondération des exemples bien classés, donc pour ceux-ci les modifications de pondération seront prises à 0. Pour les exemples mal-classés les modifications de pondérations seront supérieures ou égales à 0, avec au moins une de ces quantités strictement positive pour avoir une modification de w.
On peut analyser géométriquement l'effet d'une telle méthode et avoir une idée intuitive de la manière dont converge le BCP vers une solution dans le cas LS. Augmenter la pondération des exemples mal classés implique que le barycentre de leur classe va se déplacer dans leur direction. Ainsi le vecteur w sera mieux orienté pour trouver un seuil qui permette la séparation. Cette idée sera développée plus loin sur des bases théoriques.
Les modifications de pondération seront calculées par des méthodes déterministes ou heuristiques.

Calcul des seuils

La différence dans le calcul de $\mathcal{H}^t$ et $\mathcal{H}_{poc}^t$ réside dans des techniques différentes de calcul des seuils θ^t et θ_{poc}^t, car ces hyperplans n'ont pas le même objectif. Le premier doit assurer une convergence rapide en cas de problème LS, et le second est calculé dans le but d'optimiser le critère choisi, pour une position donnée du vecteur w imposée par le calcul de $\mathcal{H}^t$. Le seuil θ^t est ainsi calculé par des procédures heuristiques, et θ_{poc}^t par des méthodes complètement déterministes.

Arrêt de l'algorithme

Avant de calculer les seuils et les modifications de pondération, on peut tester simplement si une solution peut être trouvée avec le vecteur w courant. On définit la fonction $\vartheta : \mathbb{R}^n \longrightarrow \mathbb{R}$ qui associe à un point p le seuil d'un hyperplan perpendiculaire à w et passant par le point P. L'équation de cet hyperplan doit donc vérifier $w.p + \vartheta(p) = 0$ (l'hyperplan doit aussi être orienté de manière à ce que le vecteur w soit dans le sens demi-espace négatif vers demi-espace positif). Cette fonction s'exprime alors par :

$$\vartheta(p) = -w^t.p$$

Il est important de noter que si on représente les valeurs prises par cette fonction sur un axe parallèle à w, cette fonction est croissante dans le sens opposé au sens de w. En effet si on considère deux points x_1 et x_2 tels que $x_2 - x_1 = aw$ avec $a > 0$ alors on aura $\vartheta(x_1) - \vartheta(x_2) = w(x_2 - x_1) = a||w||^2 > 0$. Pour un point p on appellera par abus de langage la valeur $\vartheta(p)$ sa *projection*. En fait cette valeur est exactement la coordonnée de la projection du point sur une droite quelconque parallèle à w, orientée dans le sens contraire de w et ayant pour origine du repère le point o de cet axe tel que $o.w = 0$. Mais une définition aussi précise ne nous est pas nécessaire, en fait l'origine du repère n'est pas importante, c'est la relation d'ordre induit par cette fonction qui est fondamentale.
On définit les ensembles

$$\vartheta_1 = \{\vartheta(p_i)/p_i \in C_1\} \qquad \vartheta_0 = \{\vartheta(m_i)/m_i \in C_0\} \qquad \text{et} \qquad \vartheta = \vartheta_1 \cup \vartheta_0$$

L'indice t a été omis dans ces définitions, mais ces ensembles dépendent évidemment de l'itération. Dans le cas

$$\max \vartheta_1 < \min \vartheta_0$$

alors ceci signifie que les ensembles ϑ_1 et ϑ_0 ne se chevauchent pas, et ainsi en prenant comme seuil

$$\theta^t = \frac{\max \vartheta_1 + \min \vartheta_0}{2} \tag{5.7}$$

le couple w^t et θ^t est une solution verifiant 5.3, et l'algorithme peut être arrêté.
En effet, si $p_i \in C_1$, alors $-\vartheta(p_i) > -\max \vartheta_1$ donc

$$w^t.p_i + \theta^t = -\vartheta(p_i) + \theta^t > -\max \vartheta_1 + \frac{\max \vartheta_1 + \min \vartheta_0}{2} = \frac{\min \vartheta_0 - \max \vartheta_1}{2} > 0$$

De la même manière pour les points m_i de C_0 nous avons $-\vartheta(m_i) < -\min \vartheta_0$ donc

$$w^t.m_i + \theta^t = -\vartheta(m_i) + \theta^t < -\min \vartheta_0 + \frac{\max \vartheta_1 + \min \vartheta_0}{2} = \frac{\max \vartheta_1 - \min \vartheta_0}{2} < 0$$

De plus même dans le cas où on ne trouve pas de solutions, le calcul de ces ensembles est nécessaire dans les procédures de calcul des seuils.

Initialisation de l'algorithme

L'initialisation de cet algorithme réside uniquement dans la valeur initiale des coefficients de pondération α^0 and μ^0. L'efficacité de l'algorithme n'est pas trop sensible à cette initialisation. En général, on peut simplement prendre tous les coefficients de pondération à 1.
Mais les expériences ont montré que pour certaines fonctions booléennes, cette initialisation uniforme peut poser des problèmes de précision numérique dans les procédures déterministes de calcul des seuils que l'on détaillera plus loin. Ainsi la méthode est d'initialiser tous les coefficients de pondération de manière aléatoire dans l'intervalle $[1, a]$, avec $a = 2$ par exemple.

Présentation de l'algorithme général

Dans l'algorithme décrit ci-dessous, la première étape est l'initialisation. Si on minimise le nombre de mal classés, le critère sera noté n^t_{poc} et sera initialisé à $N_1 + N_0$. Au cours de l'algorithme, la meilleure valeur du critère est sauvegardée dans n_{poc}. Dans le cas d'une maximisation des exclus, le critère sera Ex^t_{poc} et sera initialisé à 0. La meilleure valeur du critère est sauvegardée dans Ex_{poc}.
A l'étape 5, ($\mathcal{H}^t_{poc}$ meilleurs que $\mathcal{H}_{poc}$) signifie alors $n^t_{poc} < n_{poc}$ dans le premier cas et $Ex^t_{poc} > Ex_{poc}$ dans le second, ($\mathcal{H}^t_{poc}$ équivalent à $\mathcal{H}_{poc}$) $n^t_{poc} = n_{poc}$ ou $Ex^t_{poc} = Ex_{poc}$.

Algorithme 5.1 Algorithme général

Etape 0 : $t = 0$, initialisation de α^0, μ^0, et du critère
Etape 1 : Calcul de b_0^t, b_1^t et w^t avec (5.4) et (5.5),
Etape 2 : Calcul de ϑ_1 et ϑ_0, si $\max \vartheta_1 < \min \vartheta_0$, θ^t donné par (5.7) **Stop** et on garde $\mathcal{H}^t$,
Etape 3 : Calcul de θ^t (section 5.3.1 ou 5.3.2),
Etape 4 : Calcul θ^t_{poc}, ε^t et $\mathcal{G}^t_{poc}$ (section 5.3.3 ou 5.3.4),
Etape 5 : si ($\mathcal{H}^t_{poc}$ meilleur que $\mathcal{H}_{poc}$) ou ($\mathcal{H}^t_{poc}$ équivalent à $\mathcal{H}_{poc}$ et $\mathcal{G}^t_{poc} > \mathcal{G}_{poc}$)
alors $\mathcal{H}_{poc} \longleftarrow \mathcal{H}^t_{poc}$ et $\mathcal{G}_{poc} \longleftarrow \mathcal{G}^t_{poc}$.
Etape 6 : Calcul des modifications de pondération $(\beta_i^t)_{i\in I_1}$ et $(\delta_j^t)_{j\in I_0}$ (section 5.3.1 ou 5.3.2),
Etape 7 : Calcul de α^{t+1} et μ^{t+1} avec (5.6),
Etape 8 : $t \longleftarrow t+1$, si $t < t_{max}$ **retour à l'Etape 1** sinon **Stop** et on garde $\mathcal{H}_{poc}$.

Remarques

Une des principales originalités du BCP est la séparation du calcul du vecteur des poids w du calcul du seuil θ. En général dans les autres algorithmes, ces deux calculs sont faits en même temps en travaillant dans un espace de dimension $n+1$: on ajoute une entrée dont la valeur est toujours à 1, et le poids de la connexion de cette entrée à l'unité est θ.
Comme b_1^t est à l'intérieur de l'enveloppe convexe des points de C_1 et b_0^t à l'intérieur de l'enveloppe convexe des points de C_0, la norme du vecteur w est bornée. Les composantes de ce vecteur ne peuvent donc pas prendre des valeurs très importantes comme avec le Perceptron ou la rétropropagation.

5.2.3 Quelques résultats théoriques

Le but est de voir comment varient les positions des barycentres et l'orientation du vecteur w après la modification des poids.
Soit $E_1^t \subset C_1$ un sous-ensemble de vecteurs d'entrée mal classés de cible 1, et $E_0^t \subset C_0$ un sous-ensemble de vecteurs d'entrée mal classés de cible 0. On notera J_1^t et J_0^t les ensembles d'indices correspondants. Les modifications de pondération pour les exemples (bien ou mal classés) qui ne sont pas dans ces deux ensembles seront pris à zéro.

$$E_1^t \subset M_1^t = \{\ p \in C_1\ /\ w^t.p + \theta^t \leq 0\} \quad \text{et} \quad J_1^t = \{i \in I_1 / p_i \in E_1^t\}$$
$$E_0^t \subset M_0^t = \{\ m \in C_0\ /\ w^t.m + \theta^t \geq 0\} \quad \text{et} \quad J_0^t = \{i \in I_0 / m_i \in E_0^t\}$$

On peut prendre $E_1^t = \emptyset$ ou $E_0^t = \emptyset$ mais on doit avoir $E_1^t \cup E_0^t \neq \emptyset$. On suppose bien entendu que $M_1^t \cup M_0^t \neq \emptyset$, sinon l'algorithme aurait trouvé une solution. Pour ces exemples mal classés, on définit les *modifications de pondération réduites* et leurs sommes par :

$$\forall k \in J_1^t \quad \eta_k^t = \frac{\beta_k^t}{\sum_{i\in J_1^t} \beta_i^t + \sum_{i\in I_1} \alpha_i^t} \quad \text{et} \quad \forall k \in J_0^t \quad \lambda_k^t = \frac{\delta_k^t}{\sum_{i\in J_0^t} \delta_i^t + \sum_{i\in I_0} \mu_i^t} \tag{5.8}$$

$$\kappa^t = \sum_{i\in J_1^t} \eta_i^t \quad \text{et} \quad \nu^t = \sum_{i\in J_0^t} \lambda_i^t \tag{5.9}$$

On peut noter tout d'abord que $\sum_{i\in I_1} \beta_i^t = \sum_{i\in J_1^t} \beta_i^t$ et $\sum_{i\in I_0} \delta_i^t = \sum_{i\in J_0^t} \delta_i^t$. De plus les modifications de pondération réduites appartiennent à l'intervalle $[0, 1[$ et leurs sommes à $]0, 1[$.

Soit $b^t_{e_1}$ le barycentre des points de E^t_1 pondérés par les modifications de pondérations $(\beta^t_i)_{i\in J_1}$, et $b^t_{e_0}$ le barycentre des points de E^t_0 pondérés par les modifications $(\delta^t_i)_{i\in J_0}$. Nous avons alors

$$b^t_{e_1} = \frac{\sum_{i\in J^t_1} \beta^t_i p_i}{\sum_{i\in J^t_1} \beta^t_i} \quad \text{et} \quad b^t_{e_0} = \frac{\sum_{i\in J^t_0} \delta^t_i m_i}{\sum_{i\in J^t_0} \delta^t_i} \tag{5.10}$$

Les vecteurs e^t_1 et e^t_0 sont alors définis par

$$e^t_1 = b^t_{e_1} - b^t_1 \qquad \text{et} \qquad e^t_0 = b^t_{e_0} - b^t_0$$

On montre (voir Annexe A.1), que les variations b^t_1, b^t_0 et w^t s'expriment par

$$b^{t+1}_1 - b^t_1 = \sum_{i\in J^t_1} \eta^t_i (p_i - b^t_1) = \kappa^t e^t_1 \tag{5.11}$$

$$b^{t+1}_0 - b^t_0 = \sum_{i\in J^t_0} \lambda^t_i (m_i - b^t_0) = \nu^t e^t_0 \tag{5.12}$$

$$w^{t+1} - w^t = \sum_{i\in J^t_1} \eta^t_i (p_i - b^t_1) - \sum_{i\in J^t_0} \lambda^t_i (m_i - b^t_0) \tag{5.13}$$

$$= \kappa^t e^t_1 - \nu^t e^t_0 \tag{5.14}$$

Le barycentre b^{t+1}_1 est donc dans le segment ouvert $]b^t_1, b^t_{e_1}[$ et b^{t+1}_0 dans $]b^t_0, b^t_{e_0}[$. Donc le vecteur w^{t+1} est beaucoup mieux orienté que w^t pour pouvoir trouver un seuil θ^{t+1} tel que les exemples mal classés de $E^t_1 \cup E^t_0$ puissent être corrigés. Cette vision intuitive de la convergence est illustrée sur la figure 5.1.

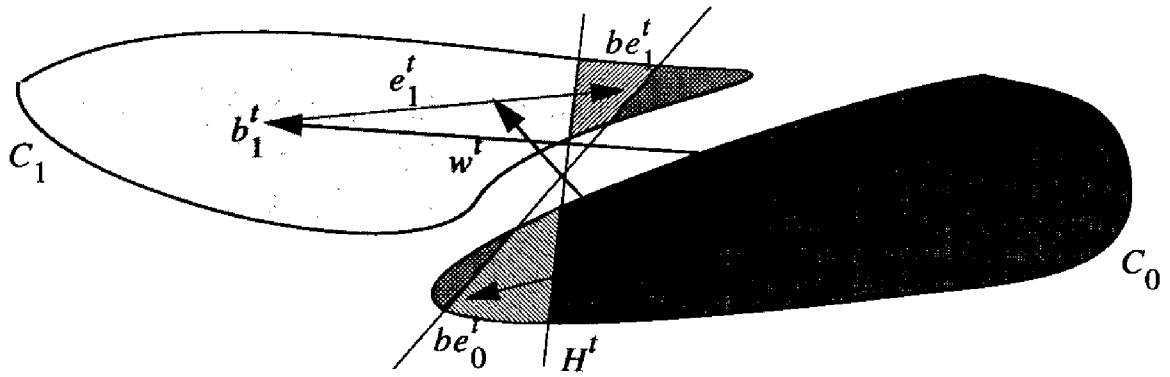

FIG. 5.1: Illustration de la convergence du BCP

Pour avoir une convergence dans le cas LS, il faut donc choisir correctement les modifications de pondération et le seuil θ^t. Le choix des sous-ensembles E^t_1 et E^t_0 ne pose pas de problème, en général on prendra toujours $E^t_1 = M^t_1$ et $E^t_0 = M^t_0$.

5.3 Les diverses méthodes

Cette section décrit de manière détaillée les diverses méthodes pour calculer premièrement le seuil θ^t et les modifications de pondération (Etape 3 et 6), et deuxièmement les paramètres de $\mathcal{H}^t_{poc}$ soit ε^t, θ^t_{poc} et $\mathcal{G}^t_{poc}$ (Etape 4).
Pour le calcul de θ^t et des modifications de pondérations, deux versions sont présentées :

- la version dont la convergence a été prouvée, et dont la propriété principale est la convergence vers le connecteur minimal quelque soit le problème. Donc s'il est LS, l'algorithme converge vers la solution optimale (tant qu'il y a des erreurs) et dans le cas NLS le vecteur w^t tend vers 0. Nous appellerons cette version BCP Prouvé. Comme nous allons le voir cette version nécessite le choix de θ^t sous contrainte, et le calcul des modifications de pondération de manière déterministe afin de minimiser une fonction.

- la version heuristique du BCP qui pour l'instant donne les meilleurs résultats. Nous nommerons cette version BCP Heuristique. Les modifications de pondération et le seuil θ^t sont calculés par des heuristiques.

Nous détaillerons ensuite les deux procédures d'optimisation du critère pour le calcul des paramètres $\mathcal{H}^t_{poc}$ (Etape 4) :

- une procédure pour minimiser le nombre d'exemples mal classés,

- une procédure de maximisation des exclus.

Comme nous le verrons, bien qu'elles peuvent être utilisées avec le BCP Prouvé, ces deux dernières procédures sont beaucoup plus efficaces avec le BCP Heuristique. Car dans le cas où il est nécessaire d'optimiser le critère, c'est-à-dire le cas NLS, le BCP Prouvé a un comportement très singulier : le vecteur w^t converge vers 0. Cette convergence est très rapide et l'algorithme n'a pas le temps d'optimiser le critère. Comme nous le verrons, ce comportement déterministe dans le cas NLS peut être utilisé pour détecter la non séparabilité d'un problème.

5.3.1 BCP Prouvé

Le connecteur minimal

Avant tout développement, il convient de définir exactement le connecteur minimal. Soit $\mathcal{E}_1$ l'enveloppe convexe des points de C_1 et $\mathcal{E}_0$ celle des points C_0. Par abus de langage, on appellera enveloppe convexe l'intérieur de l'enveloppe plus la frontière.

Définition 5.1 *Le connecteur minimal w_m est défini comme le vecteur de norme minimale qui joint un point de $\mathcal{E}_1$ à un point de $\mathcal{E}_0$. C'est-à-dire*

$$w_m = a_1 - a_0 \quad \text{tel que} \quad \|a_1 - a_0\| = \min \{ \|a'_1 - a'_0\| \ / \ a'_1 \in \mathcal{E}_1 \text{ et } a'_0 \in \mathcal{E}_0 \} \tag{5.15}$$

On démontre alors le théorème suivant.

Théorème 5.1 *Les ensembles C_1 et C_0 sont LS $\iff$ le vecteur $w_m = a_1 - a_0$ défini par 5.15 et le seuil θ_m dans l'intervalle $]-w_m.a_1^t, -w_m.a_0^t[$ vérifient la condition 5.3.*

Démonstration 5.1 *La condition suffisante est évidente. La condition nécessaire est prouvée par l'absurde. Supposons C_1 et C_0 LS, et soit w_m et θ_s un vecteur et un seuil défini comme ci-dessus. Soit $\mathcal{H}_s$ l'hyperplan d'équation $w_m.x + \theta_s = 0$. Cet hyperplan coupe le segment $]a_0a_1[$. Le point a_0 se trouve dans le demi-espace négatif défini par $\mathcal{H}_s$ et a_1 dans le positif. Soit $\mathcal{D}$ une boule ouverte, dont un diamètre est le segment $]a_0a_1[$. Supposons que la relation 5.3 n'est pas vérifiée, alors on peut supposer par exemple que $\exists p \in C_1 / w_m.p + \theta_s < 0$. Deux cas sont possibles.*
Premièrement, si le point p appartient à la boule $\mathcal{D}$, alors le point h, défini comme la projection perpendiculaire de a_0 sur la droite (a_1p), n'appartient pas au segment $[a_1p]$. Mais dans ce cas, on a $\|p - a_0\| < \|w_m\|$, ce qui est une contradiction d'après la définition de w_m.

Deuxièmement, si le point p n'appartient pas à la boule $\mathcal{D}$, alors h appartient au segment $[a_1 p]$. Les points a_1 et p se trouvent dans $\mathcal{E}_1$, ainsi que le segment $[a_1 p]$ et donc h. Mais par construction on a $\|h - a_0\| < \|w_m\|$. Ce qui est absurde.
Une démonstration similaire pourrait être faite en supposant que $\exists m \in C_0 / w_m.p + \theta_s > 0$ □.

Ce théorème est une version constructive du théorème classique (voir [192] par exemple)

$$C_1 \text{ et } C_0 \text{ sont LS} \iff \mathcal{E}_0 \cap \mathcal{E}_1 = \emptyset$$

On peut en déduire

$$C_1 \text{ et } C_0 \text{ sont LS} \Longleftrightarrow \|w_m\| > 0 \quad \text{et} \quad C_1 \text{ et } C_0 \text{ sont NLS} \Longleftrightarrow \|w_m\| = 0 \tag{5.16}$$

Généralités

Avant toute chose il convient de faire une remarque importante. Cette version du BCP a 3 applications : trouver une solution dans le cas LS, savoir si un problème est NLS ou LS, et converger vers le connecteur minimal. Bien qu'en théorie, elle peut être utilisée pour optimiser un critère dans le cas NLS, son principe même la rend totalement inefficace, du fait de sa rapidité de convergence vers le connecteur minimal qui est le vecteur nul dans ce cas.
La convergence du BCP va être prouvée dans les conditions suivantes :

- le seuil θ^t est calculé tel que l'hyperplan $\mathcal{H}^t$ coupe le segment ouvert $]b_0^t, b_1^t[$. Le seuil doit donc vérifier la condition $\theta^t \in]-w^t.b_1^t, -w^t.b_0^t[$. Mais une convergence théorique en un nombre fini d'itérations nécessite un intervalle fermé, on devra donc avoir

$$\theta^t \in K^t \quad \text{avec} \quad K^t = [-w^t.b_1^t + \gamma \|w^t\|^2, -w^t.b_0^t - \gamma \|w^t\|^2] \quad \text{et} \quad \gamma \in]0, \frac{1}{2}[\tag{5.17}$$

- les modifications de pondération seront calculées de manière à minimiser à chaque itération la variation du carré de la norme du vecteur w^t, donc pour minimiser la fonction

$$\Delta^t = \|w^{t+1}\|^2 - \|w^t\|^2 \tag{5.18}$$

On supposera que l'on a toujours $w^t \neq 0$, car dans le cas $w^t = 0$ l'algorithme est arrêté et on peut conclure que le problème est NLS. Comme nous le verrons dans la section suivante, dans le cas du BCP Heuristique, ce cas peut se produire, et il est alors réglé de manière simple pour pouvoir continuer l'algorithme normalement afin de trouver une solution satisfaisante en optimisant le critère désiré.
Dans le cas où l'on prend des modifications de pondération identiques pour tous les éléments de E_1^t et identiques pour tous les éléments de E_0^t, c'est-à-dire

$$\forall k \in J_1^t \quad \beta_k^t = \beta^t \; \eta_k^t = \eta^t \quad \text{et} \quad \forall k \in J_0^t \quad \delta_k^t = \delta^t \; \lambda_k^t = \lambda^t \tag{5.19}$$

les modifications de pondérations réduites sont alors données par

$$\kappa^t = \frac{q_1^t \beta^t}{q_1^t \beta^t + \sum_{i \in I_1} \alpha_i^t} \quad \text{et} \quad \nu^t = \frac{q_0^t \delta^t}{q_0^t \delta^t + \sum_{i \in I_0} \mu_i^t} \tag{5.20}$$

avec $q_1^t = \text{Card}(E_1^t)$ et $q_0^t = \text{Card}(E_0^t)$. Dans ces conditions, les barycentres $b_{e_1}^t$ et $b_{e_0}^t$ sont des isobarycentres, et ne dépendent donc plus des modifications de pondération.

$$b_{e_1}^t = \frac{\sum_{i \in I_1} p_i}{q_1^t} \quad \text{et} \quad b_{e_0}^t = \frac{\sum_{i \in I_0} m_i}{q_0^t}$$

On définit enfin la constante d par $d = \max \{\|c_1 - c_0\| / (c_0, c_1) \in (\mathcal{E}_0 \cup \mathcal{E}_1)^2\}$.

Théorème de convergence

Dans les définitions qui suivent on supposera que $E_1^t \neq \emptyset$ ou $E_0^t \neq \emptyset$, soit $q_1^t > 0$ ou $q_0^t > 0$. Soit $q^t = q_1^t + q_0^t$, donc $q^t > 0$. Le vecteur y^t est la concaténation des modifications de pondération des exemples contenus dans l'ensemble $E_0^t \cup E_1^t$, et le vecteur x^t est la concaténation des modifications de pondération réduites. Si on suppose que $E_1^t \neq \emptyset$ et $E_0^t \neq \emptyset$ alors

$$y^t = (\beta_{i_1}^t, \ldots, \beta_{i_{q_1^t}}^t, \delta_{j_1}^t, \ldots, \delta_{j_{q_0^t}}^t) \quad \text{et} \quad x^t = (\eta_{i_1}^t, \ldots, \eta_{i_{q_1^t}}^t, \lambda_{j_1}^t, \ldots, \lambda_{j_{q_0^t}}^t)$$
$$\text{avec} \quad \{i_1, \ldots, i_{q_1^t}\} = J_1 \quad \text{et} \quad \{j_1, \ldots, j_{q_0^t}\} = J_0$$

Deux définitions similaires pourraient être données en supposant que $E_1^t = \emptyset$ ou $E_0^t = \emptyset$.
Par définition, on a $x^t \in [0,1[^{q^t}$, et $(\kappa^t, \nu^t) \in]0,1[^2$. Quand la modification de pondération réduite d'un exemple p tend vers 1, sa modification de pondération tend vers $+\infty$. Autrement dit, le barycentre des points mal classés choisis tend vers le point p. Pour éviter de travailler sur des infinis, on doit définir un domaine fermé que l'on notera U_t défini par

$$U_t = \{x^t \in \mathbb{R}_+^{q^t} / \kappa^t \leq \xi \text{ et } \nu^t \leq \xi\} \quad \text{avec} \quad \xi \in [\frac{1}{2}, 1[\tag{5.21}$$

Avec une telle définition, on a $U_t \subset [0,\xi]^{q^t}$, et on peut montrer que la fonction $\mathcal{F}_t$ de $\mathcal{A}_t$ dans U_t, tel que $\mathcal{F}_t(y^t) = x^t$ définie par les relations 5.8 est une bijection (Voir Annexe A.2). La fonction réciproque $\mathcal{F}_t^{-1}$ peut s'exprimer simplement. Le domaine $\mathcal{A}_t$ est défini par $\mathcal{A}_t = \mathcal{F}_t^{-1}(U_t)$.
Soit le domaine $U_t' \subset U_t$, défini comme l'ensemble des vecteurs x^t de U_t tels que x^t et $y^t = \mathcal{F}_t^{-1}(x^t)$ vérifient la condition 5.19. Donc

$$U_t' = \{x^t \in U_t / \forall k \in J_1^t \ \eta_k^t = \eta^t \text{ et } \forall k \in J_0^t \ \lambda_k^t = \lambda^t\} \tag{5.22}$$

En élevant au carré la relation 5.13, on peut calculer la fonction $\Delta^t = \|w^{t+1}\|^2 - \|w^t\|^2$ comme une fonction des composantes du vecteur x^t. On notera donc cette fonction $\Delta^t(x^t)$.
Soit la fonction ω^t de U_t dans $[0, 2\xi]$ définie par

$$\omega^t(x^t) = \sum_{i \in J_1^t} \eta_i^t + \sum_{i \in J_0^t} \lambda_i^t = \kappa^t + \nu^t \tag{5.23}$$

On définit aussi la fonction G_t de $\mathbb{R}$ dans $\mathbb{R}$ avec $G^t(\psi) = \psi^2 d^2 - 2\psi\gamma\|w^t\|^2$ et finalement la fonction $\mathcal{M}^t$ de U_t dans $\mathbb{R}$ par $\mathcal{M}^t = G^t \, o \, \omega^t$.

Théorème 5.2 Théorème de convergence avec paramètre de relaxation variable.

Si U_t'' est un domaine vérifiant $U_t' \subset U_t'' \subset U_t$ et $(r_t)_{t \geq 0}$ une suite telle que

$$\forall t \ r_t \in]0,1] \quad \text{et} \quad \exists r^2 \neq 0 \ \ \forall p \geq 1 \ \sum_{t=0}^{p-1} {r_t}^2 > pr^2$$

alors un algorithme BCP, tel que tant que l'on prend des exemples mal classés dans $E_0^t \cup E_1^t$, c'est-à-dire $q^t > 0$, prend θ^t dans K^t et calcule les modifications de pondération par la formule $y^t = \mathcal{F}_t^{-1}(r_t x_m^t)$ avec

$$x_m^t \in \arg \min_{x \in U_t'} \mathcal{M}^t(x) \qquad (AL_0)$$
$$\text{ou} \quad x_m^t \in \arg \min_{x \in U_t'} \Delta^t(x) \qquad (AL_1)$$
$$\text{ou} \quad x_m^t \in \arg \min_{x \in U_t''} \Delta^t(x) \qquad (AL_2)$$

vérifie alors les propriétés

- *(P1) :* C_1 *et* C_0 *sont LS* $\Longrightarrow \exists t_0 \,/\, q^{t_0} = 0.$
- *(P2) :* C_1 *et* C_0 *sont NLS* $\Longrightarrow \|w^t\| \longrightarrow 0.$

Démonstration 5.2 *Voir Annexe A.6.*

Une implémentation de l'algorithme AL_1

Cette partie propose une implémentation simple de la version AL_1. Les modifications de pondération sont calculées de manière à minimiser la fonction $\Delta^t(x^t)$ dans le domaine U_t'. Les composantes des vecteurs x^t et y^t vérifient la condition 5.19. La fonction Δ^t peut alors être exprimée comme une fonction de κ^t et ν^t uniquement, car les barycentres $b^t_{e_1}$ et $b^t_{e_0}$ ne dépendent pas des modifications de pondération. Si on suppose que $E^t_1 \neq \emptyset$ et $E^t_0 \neq \emptyset$, on obtient :

$$g^t(\kappa^t, \nu^t) = a\,\kappa^{t2} + b\,\nu^{t2} - c\,\kappa^t - d\,\nu^t + e\,\kappa^t\nu^t$$

Les constantes a, b, c, d et e sont données en annexe, les quatres premières sont strictement positives. Il faut donc rechercher le minimum de cette fonction dans l'ensemble $P = [0, \xi]^2$. Les conditions du premier ordre donnent deux équations de droites :

$$\begin{aligned} \mathcal{D}_{\kappa^t} \quad &: \quad \frac{\partial g^t}{\partial \kappa^t} = 0 \iff 2a\kappa^t - c + e\nu^t = 0 \\ \mathcal{D}_{\nu^t} \quad &: \quad \frac{\partial g^t}{\partial \nu^t} = 0 \iff 2b\nu^t - d + e\kappa^t = 0 \end{aligned}$$

Le déterminant de ce système est alors :

$$Det = \begin{vmatrix} 2a & e \\ e & 2b \end{vmatrix} = 4ab - e^2 = 4\,\|e^t_1\|^2\,\|e^t_0\|^2 - 4\,(e^t_1.e^t_0)^2$$

L'inégalité de Cauchy-Schwartz nous montre que ce déterminant est nul si et seulement si les vecteurs e^t_1 et e^t_0 sont colinéaires, ce qui peut arriver. Maintenant si on suppose que $Det \neq 0$, le minimum absolu de g^t dans $\mathbb{R}^2$ est donné par

$$\kappa^t_{abs} = \frac{2bc - ed}{Det} \quad \text{et} \quad \nu^t_{abs} = \frac{2ad - ec}{Det}$$

Soit J l'ensemble des intersections des droites $\mathcal{D}_{\kappa^t}$ et $\mathcal{D}_{\nu^t}$ avec les droites d'équations $\kappa^t = 0$, $\kappa^t = \xi$, $\nu^t = 0$ et $\nu^t = \xi$, définissant le pavé de recherche. On a Card $(()J) \leq 8$. Soit $J' \subset J$ le sous-ensemble de ces points qui appartiennent au pavé P, alors $J' = J \cap P$, et Card $(()J') \leq 4$. Finalement, J'' est l'ensemble des points extrêmes du pavé P, soit $J'' = \{(0,0), (0,\xi), (\xi,0), (\xi,\xi)\}$.
Pour calculer le minimum de g^t dans P, trois cas sont possibles :

- si $Det \neq 0$ et $(\kappa^t_{abs}, \nu^t_{abs}) \in P$ alors $(\kappa^t, \nu^t) = (\kappa^t_{abs}, \nu^t_{abs})$.
- sinon si $J' \neq \emptyset$ alors $(\kappa^t, \nu^t) = \arg \min_{(\kappa,\nu)\in J'} g^t(\kappa, \nu)$
- sinon $(\kappa^t, \nu^t) = \arg \min_{(\kappa,\nu)\in J''} g^t(\kappa, \nu)$

Dans le premier cas, le minimum absolu est dans P, les deux conditions du premier ordre sont satisfaites. Dans le second cas une des conditions est satisfaite et au moins une contrainte saturée. Dans le troisième cas, deux contraintes sont saturées et les deux conditions non satisfaites. Cette méthode de minimisation peut alors se justifier par la nature convexe de la fontion, qui est un paraboloïde.

Après avoir choisi un paramètre de relaxation $r_t \in]0,1]$, les modifications de pondération peuvent être calculées simplement avec les relations (voir Annexe A.2)

$$\forall k \in J_1^t \quad \beta_k^t = \frac{r_t \kappa^t / q_1^t}{1 - r_t \kappa^t} \sum_{i \in I_1} \alpha_i^t \quad \text{et} \quad \forall k \in J_0^t \quad \delta_k^t = \frac{r_t \nu^t / q_0^t}{1 - r_t \nu^t} \sum_{i \in I_0} \mu_i^t$$

En pratique, les bonnes valeurs du paramètre de relaxation se situent entre 0,15 et 0,25, pour une convergence rapide vers une solution quand le problème est LS. Pour la détection de non séparabilité, il vaut mieux prendre 1.0. Pour les ensembles de travail E_1^t et E_0^t, on peut prendre simplement M_1^t et M_0^t.
Si $E_1^t = \emptyset$ ou $E_0^t = \emptyset$, le minimum de Δ^t peut être facilement calculé dans l'ensemble $[0, \xi]$. Il suffit de calculer le minimum d'une parabole dans un intervalle fermé.

Remarques

La première méthode de calcul des modifications de pondération AL_0 est la base du théorème de convergence. Le calcul est trivial. En pratique elle converge mais très lentement. La méthode AL_1, présentée ci-dessus offre un très bon rapport performance sur temps de calcul. La procédure de minimisation est de complexité $O(nN)$, car il faut calculer les barycentres des éléments de E_1^t et E_0^t, et donc l'algorithme général en $O(nN)$. La dernière méthode est la plus générale. Dans le cas où $U_t'' = U_t$ la minimisation de la fonction $\Delta_t(x^t)$ se fait par rapport à toutes les modifications de pondération réduites. Cette méthode devrait fournir une plus grande rapidité, mais les calculs sont complexes : il faut minimiser une fonction quadratique de q^t variables sous q^t+2 contraintes (toutes les modifications de pondérations réduites doivent être positives et leurs sommes κ^t et ν^t inférieures ou égales à ξ). Un compromis peut être développé entre cette méthode et la méthode AL_1, en partitionnant les ensembles E_1^t et E_0^t en un nombre limité de sous-ensembles. Dans chacun de ces sous-ensembles, les modifications de pondération sont égales. Ainsi, un domaine de recherche U_t'' est défini et vérifie $U_t' \subset U_t'' \subset U_t$.
On peut noter qu'une minimisation globale de la norme du vecteur w^t en fonction des coefficients de pondération peut être réalisée. Mais la complexité des calculs d'une telle méthode est très importante, car elle nécessite alors la minimisation d'une fonction quadratique de N variables sous $N+2$ contraintes, car la somme des coefficients de pondération pour chaque cible doit être égale à 1.
On notera enfin que dans le cas LS, on peut continuer l'algorithme même si une solution a été trouvée, en forçant des erreurs. Si la partie est LS alors nous avons $\vartheta(b_1^t) \leq \max \vartheta_1 < \min \vartheta_0 \leq \vartheta(b_0^t)$, donc en prenant

$$\theta^t \in K^t \cap (\,]\vartheta(b_1^t), \max \vartheta_1] \cup [\min \vartheta_0 \leq \vartheta(b_0^t)[\,)$$

la norme du vecteur w^t continuera à decroître strictement jusqu'au connecteur minimal.
En résumé, cette méthode tend vers le connecteur minimal tant qu'il y a des exemples mal classés.

5.3.2 BCP Heuristique

Nous allons détailler les méthodes heuristiques du calcul de θ^t (Etape 3) et des modifications de pondération (Etape 6). Cette version donne pour l'instant les meilleurs résultats, mais il en existe peut-être de meilleurs. Avant de décrire ces heuristiques, nous donnerons quelques définitions, qui seront utiles ultérieurement.

Définitions

A l'étape 3 de l'algorithme les ensembles ϑ_1 et ϑ_0 sont déjà calculés, ainsi que les extrêma $\max \vartheta_1$ et $\min \vartheta_0$ pour tester la possible séparation avec le vecteur w^t courant. Il faut alors calculer les deux autres extrêma $\min \vartheta_1$ et $\max \vartheta_0$.

On ordonne ces quatre extrêmes et on définit alors

$$\{ext_1, ext_2, ext_3, ext_4\} = \{\min \vartheta_1, \max \vartheta_1, \min \vartheta_0, \max \vartheta_0\} \qquad \text{avec} \qquad ext_1 \leq ext_2 \leq ext_3 \leq ext_4$$

Les trois ensembles suivants sont alors définis

$$P_- = [ext_1, ext_2[\cap \vartheta \quad P_+ =]ext_3, ext_4] \cap \vartheta \qquad \text{et} \qquad P_{ov} = [ext_2, ext_3] \cap \vartheta$$

Il est très important de noter que tous les éléments de P_- sont les projections de vecteurs d'entrée de même cible. Celle-ci peut être 0 ou 1, et nous la noterons t_-. De la même manière, tous les éléments de P_+ sont des projections de vecteurs d'entrée qui ont même cible, celle-ci sera notée t_+. L'ensemble P_{ov} contient les projections d'exemples de cible 0 et 1, nous l'appellerons la zone de chevauchement. On définit enfin N_-, N_+ et N_{ov} par

$$N_- = \text{Card } P_- \qquad N_+ = \text{Card } P_+ \qquad \text{et} \qquad N_{ov} = \text{Card } P_{ov} = N - N_- - N_+$$

Intuitivement, on a tendance à penser que la zone P_- contient des exemples de cible 1 et la zone P_+ des exemples de cible 0, car comme nous l'avons dit l'axe des éléments de ϑ est orienté dans le sens contraire à w^t. C'est un cas qui se produit souvent pour des distributions de points régulières, avec deux zones assez distinctes pour les deux cibles, et une zone où les deux types de points sont mélangés. Mais dans le cas général, on peut tout à fait avoir d'autres configurations.
En fait si on analyse les ordres possibles pour ces quatres extrêmes, il y en a seulement 4. On a déjà $\min \vartheta_1 \leq \vartheta(b_1^t) \leq \max \vartheta_1$ et $\min \vartheta_0 \leq \vartheta(b_0^t) \leq \max \vartheta_0$. Et par définition du vecteur w^t nous avons $\vartheta(b_1^t) < \vartheta(b_0^t)$, donc $\min \vartheta_1 < \max \vartheta_0$. Ainsi si on fait exception du cas C_0 LS détecté à l'Etape 2 de l'algorithme, on suppose alors que $\max \vartheta_1 > \min \vartheta_0$. Or pour ordonner 4 quantités il suffit de $\binom{4}{2} = 6$ relations d'ordre et nous en avons déjà 4, donc il en reste deux. Ces deux relations d'ordre nous donnent donc 4 cas de chevauchement possibles :

Cas C_1 :	$\min \vartheta_1 \leq \min \vartheta_0 \leq \max \vartheta_1 \leq \max \vartheta_0$	alors $t_- = 1$ et $t_+ = 0$
Cas C_2 :	$\min \vartheta_1 \leq \min \vartheta_0 \leq \max \vartheta_0 < \max \vartheta_1$	alors $t_- = 1$ et $t_+ = 1$
Cas C_3 :	$\min \vartheta_0 < \min \vartheta_1 \leq \max \vartheta_1 \leq \max \vartheta_0$	alors $t_- = 0$ et $t_+ = 0$
Cas C_4 :	$\min \vartheta_0 < \min \vartheta_1 < \max \vartheta_0 < \max \vartheta_1$	alors $t_- = 0$ et $t_+ = 1$

Modification des pondérations

On suppose que les coefficients de pondération initiaux ont été initialisés dans l'intervalle $[1, a]$, et que à chaque itération on considère tous les exemples mal classés, c'est-à-dire que $E_1^t = M_1^t$ et $E_0^t = M_0^t$. On prendra les modifications de pondérations égales pour tous les exemples mal classés de cible identique, et elles resteront constantes durant tout l'apprentissage.

$$\forall t \; \forall i \in J_1^t \quad \beta_i^t = \beta \qquad \text{et} \qquad \forall t \; \forall i \in J_0^t \quad \delta_i^t = \delta \tag{5.24}$$

En prenant simplement $\beta = \delta \simeq 1$ on obtient déjà de très bons résultats sur différents types de base d'apprentissage LS. Mais une convergence un peu lente apparaît pour certaines bases d'apprentissage pour lesquelles les nombres de points des deux cibles ne sont pas du même ordre de grandeur, une des classes a beaucoup plus de points que l'autre. Pour améliorer la rapidité de convergence pour ce genre de problèmes, sans ralentir la convergence pour les autres, une meilleure heuristique a été trouvée. Son principe est d'augmenter la valeur de β (resp. δ) si $N_1 >> N_0$ (resp. si $N_0 >> N_1$). Ainsi on aura alors

$$\beta = \max\{\beta_{min}, \min\{\beta_{max}, \frac{N_1}{N_0}\}\} \qquad \text{et} \qquad \delta = \max\{\delta_{min}, \min\{\delta_{max}, \frac{N_0}{N_1}\}\}$$

En pratique, on peut prendre $\beta_{min} = \delta_{min} \in [1, a]$ et $\beta_{max} = \delta_{max} \simeq 30$.

Choix du θ^t

Le seuil θ^t est pris au hasard dans l'intervalle

$$[ext_2 - \gamma, ext_3 + \gamma] \quad \text{avec} \quad \gamma = \gamma_r \, (ext_3 - ext_2) \tag{5.25}$$

Le paramètre γ_r doit être strictement plus grand que $-1/2$. En pratique $\gamma_r \in [0, 2]$, et en moyenne une bonne valeur est $\gamma_r = 0,25$ pour les problèmes analogiques et $\gamma_r = 0$ pour les problèmes booléens. Avec $\gamma_r = 0.25$, les expériences ont montré que certaines fonctions analogiques étaient plus rapidement résolues qu'avec $\gamma_r = 0$. Mais quelques fonctions booléennes LS (à peu près 1 pour 2000 en dimensions 8 et 9 seulement) étaient difficiles à séparer.
En résumé, le meilleur choix pour un problème pour lequel on n'a aucune information est d'initialiser γ_r à 0.25, et de faire décroître cette valeur jusqu'à 0 pendant l'apprentissage.
Pour conclure cette partie, on peut dire que le seuil θ^t est choisi au hasard dans la zone de chevauchement, un peu élargie quand $\gamma_r > 0$. Cette heuristique en association avec la précédente, donne une convergence extrêmement rapide pour les problèmes LS.

Remarques

Si l'on ne prend pas en compte l'étape 4 de l'algorithme général, le BCP Heuristique a une complexité en $O(nN)$, comme le Perceptron et le Thermal Perceptron.
Il convient de souligner un petit problème d'implantation de l'algorithme. Dans le cas de problèmes NLS, les deux barycentres b_1^t et b_0^t peuvent être confondus, le vecteur w^t est alors égal à zéro. Cette situation apparaît par exemple pour des problèmes très symétriques comme le XOR dès la première itération si les coefficients de pondération ont été initialisés de manière uniforme. Il peut aussi apparaître pour tout problème NLS car l'intersection des enveloppes convexes des vecteurs d'entrée des deux cibles n'est pas vide. Une solution simple à ce problème est de soustraire temporairement 1 à un des coefficients de pondération. On aura alors $w^t \neq 0$, et on pourra optimiser le critère.
Cette version heuristique est beaucoup plus rapide que la version prouvée du BCP pour trouver une solution dans le cas LS (voir les simulations). Mais le BCP Prouvé peut être utilisé pour accroître la robustesse de la solution trouvée, comme nous l'avons vu dans la section 5.3.1.
De plus, même quand le problème est NLS, et que l'hyperplan retenu est $\mathcal{H}_{poc}$, la robustesse de celui-ci peut aussi être améliorée avec le BCP Prouvé. Bien que la marge soit optimisée pendant l'algorithme, sa valeur dépend des diverses positions prises par le vecteur w^t. La meilleure position de cet hyperplan est le connecteur minimal entre l'enveloppe convexe des points contenus dans le demi-espace positif et l'enveloppe convexe des points du demi-espace négatif. Cette position optimale peut ainsi être atteinte avec le BCP Prouvé, en prenant une base d'apprentissage avec des cibles modifiées : un vecteur d'entrée est de cible 1 s'il est dans le demi-espace positif, et 0 s'il est dans le demi-espace négatif.

5.3.3 Minimisation du nombre d'exemples mal classés : l'algorithme BCP Min

Le but de cette procédure est de trouver le seuil θ_{poc}^t et $\varepsilon^t \in \{-1, 1\}$, qui fournit un minimum d'exemples mal classés par l'hyperplan $\varepsilon^t \, w^t.x + \theta_{poc}^t = 0$. Lors de la recherche, si deux hyperplans fournissent le même nombre d'erreurs, on conservera celui qui permet d'avoir la plus grande marge $\mathcal{G}_{poc}^t$. Le BCP Heuristique utilisant cette procédure de minimisation du nombre d'erreurs sera nommé BCP Min.
On définit les fonctions $Err_1(\theta)$ associant à une valeur de seuil θ le nombre d'exemples mal classés par l'hyperplan $w^t.x + \theta = 0$, et la fonction $Err_{-1}(\theta)$ le nombre de mal classés par l'hyperplan $-w^t.x - \theta = 0$. Si on suppose qu'aucun point ne se trouve exactement sur un de ces hyperplans (qui sont d'ailleurs confondus), nous avons alors la propriété évidente $Err_1(\theta) + Err_{-1}(\theta) = N$. La fonction

$Err_1(\theta)$ est définie exactement par :

$$Err_1(\theta) = \text{Card}\,(p_i \in C_1 / w^t.p_i + \theta \leq 0) + \text{Card}\,(m_i \in C_0 / w^t.m_i + \theta \geq 0)$$

Le principe de cette procédure est simple et basé sur l'idée suivante. Si on considère deux valeurs de seuil θ_1 et θ_2 n'appartenant pas à l'ensemble ϑ alors si on pose $T = [\theta_1, \theta_2]$, on obtient la relation suivante :

$$Err_1(\theta_2) = Err_1(\theta_1) - \text{Card}\,\{p \in C_1 / \vartheta(p) \in T\} + \text{Card}\,\{m \in C_0 / \vartheta(m) \in T\} \qquad (5.26)$$

En effet lorsque l'on prend l'hyperplan $w^t.x + \theta_1 = 0$ on obtient $Err_1(\theta_1)$ erreurs. Si on déplace cet hyperplan parallèlement à lui-même pour obtenir $w^t.x + \theta_2 = 0$, tous les exemples de cible 1 situés entre ces deux hyperplans et qui étaient mal classés vont devenir bien classés. Et tous les exemples de cible 0 situés entre ces deux hyperplans qui étaient bien classés deviennent mal classés. D'où la relation 5.26.
Maintenant supposons que $T' =]\theta_3, \theta_4[$ avec $(\theta_3, \theta_4) \in \vartheta^2$, et qu'il n'y ait aucun élément de ϑ dans T'. Si $\varepsilon^t = 1$ et $\theta^t_{poc} \in T'$, on montre alors que la marge est obtenue simplement par :

$$\mathcal{G}^t_{poc} = \frac{\theta_4 - \theta_3}{||w^t||} \qquad (5.27)$$

En effet, considérons les deux ensembles :

$$C^+ = \{p \in C / w^t.x + \theta^t_{poc} \geq 0\} \qquad \text{et} \qquad C^- = \{p \in C / w^t.x + \theta^t_{poc} < 0\}$$

Nous avons $w^t.p + \theta^t_{poc} = -\vartheta(p) + \theta^t_{poc}$. La marge s'exprime alors par :

$$\mathcal{G}^t_{poc} = \min_{p \in C^+} d(p, H^t_{poc}) + \min_{p \in C^-} d(p, H^t_{poc}) = \min_{p \in C^+} \frac{-\vartheta(p) + \theta^t_{poc}}{||w^t||} + \min_{p \in C^-} \frac{\vartheta(p) - \theta^t_{poc}}{||w^t||}$$

Or $\theta^t_{poc} \in T'$ donc nous avons $\theta^t_{poc} - \theta_3 > 0$ et $\theta^t_{poc} - \theta_4 < 0$. Ainsi $\exists p_3 \in C^+$ tel que $\vartheta(p_3) = \theta_3$ et $\exists p_4 \in C^-$ tel que $\vartheta(p_4) = \theta_4$. De plus comme aucun élément de ϑ ne se trouve dans T', et que $\forall p \in C^+ \ \vartheta(p) \leq \theta^t_{poc}$ et $\forall p \in C^- \ \vartheta(p) < \theta^t_{poc}$, on peut en déduire que $\forall p \in C^+ \ \vartheta(p) \leq \vartheta(p_3)$ et $\forall p \in C^- \ \vartheta(p) \geq \vartheta(p_4)$. Dans l'expression de la marge, les deux minima sont donc atteints par p_3 et p_4, d'où on conclut que :

$$\mathcal{G}^t_{poc} = \frac{-\vartheta(p_3) + \theta^t_{poc}}{||w^t||} + \frac{\vartheta(p_4) - \theta^t_{poc}}{||w^t||} = \frac{\theta_3 + \theta^t_{poc}}{||w^t||} + \frac{-\theta_4 - \theta^t_{poc}}{||w^t||} = \frac{\theta_3 - \theta_4}{||w^t||}$$

On définit l'ensemble $\vartheta' = P_{ov} \cup \{\max(P_-), \min(P_+)\}$, et la suite ϑ'_i de ses éléments rangés par ordre croissant, avec $\text{Card}\,(\vartheta') = N'$, avec N' qui peut être inférieur strictement à $\text{Card}\,(P_{ov}) + 2$.
Le fonctionnement de l'algorithme 5.2 est le suivant. La méthode consiste à faire varier un seuil θ sur toutes les valeurs pertinentes, (celles-ci étant des valeurs croissantes, milieux de deux éléments consécutifs de la suite ϑ'), et à calculer ainsi de proche en proche le nombre d'erreurs commises par l'hyperplan $w^t.x + \theta = 0$. A l'initialisation le seuil est placé au premier de ces milieux qui est $\theta = (\vartheta'_1 + \vartheta'_2)/2$ (mais nous avons en fait $\vartheta'_1 = \max(P_-)$ et $\vartheta'_2 = \min(P_{ov})$), et le nombre courant d'erreurs $n_e = Err_1(\theta)$ est mis à $N_1 - N_-$ si $t_- = 1$ et a $N_1 + N_-$ si $t_- = 0$, car P_- contient la projection des vecteurs d'entrée de même cible t_-. Le nombre courant d'erreurs est mis à jour grâce à la relation 5.26, et la marge courante $\mathcal{G}$ est calculée pour chaque position avec la relation 5.27. On peut utiliser cette relation car le seuil est le milieu de deux éléments consécutifs de la suite ϑ'. Pour chacune des positions, on teste aussi le nombre d'erreurs commises par l'hyperplan $-w^t.x - \theta = 0$,

Algorithme 5.2 Algorithme de minimisation du nombre d'exemples mal classés

Etape a : Prétraitement
- **1 :** Calcul de ext_1, ext_2, ext_3, ext_4, t_- et t_+
- **2 :** Calcul de $\max(P_-)$, $\min(P_+)$ et N_-
- **3 :** Calcul de la suite $(\vartheta'_i)_{i\in\{1,\ldots,N'\}}$

Etape b : Initialisation de la boucle
- **1 :** $\theta = (\vartheta'_1 + \vartheta'_2)/2$ et $\mathcal{G} = (\vartheta'_2 - \vartheta'_1)/||w^t||$
- **2 :** si $(t_- = 1)$ alors $n_e = N_1 - N_-$ sinon $n_e = N_1 + N_-$
- **3 :** si $(N - n_e \geq n_e)$ alors $n^t_{poc} = n_e$, $\theta^t_{poc} = \theta$, $\mathcal{G}^t_{poc} = \mathcal{G}$ et $\varepsilon^t = 1$
  sinon $n^t_{poc} = N - n_e$, $\theta^t_{poc} = -\theta$, $\mathcal{G}^t_{poc} = \mathcal{G}$ et $\varepsilon^t = -1$

Etape c : Pour $k = 3$ à N'
- **1 :** $n_e \longleftarrow n_e - \text{Card}\,\{p \in C_1/\vartheta(p) = \vartheta'_{k-1}\} + \text{Card}\,\{m \in C_0/\vartheta(m) = \vartheta'_{k-1}\}$
- **2 :** $\theta = (\vartheta'_k + \vartheta'_{k-1})/2$ et $\mathcal{G} = (\vartheta'_k - \vartheta'_{k-1})/||w^t||$
- **3 :** si $(n_e < n^t_{poc})$ ou $(n_e = n^t_{poc}$ et $\mathcal{G} > \mathcal{G}^t_{poc})$ alors
  $n^t_{poc} = n_e$, $\theta^t_{poc} = \theta$, $\mathcal{G}^t_{poc} = \mathcal{G}$ et $\varepsilon^t = 1$
  si $(N - n_e < n^t_{poc})$ ou $(N - n_e = n^t_{poc}$ et $\mathcal{G} > \mathcal{G}^t_{poc})$ alors
  $n^t_{poc} = N - n_e$, $\theta^t_{poc} = -\theta$, $\mathcal{G}^t_{poc} = \mathcal{G}$ et $\varepsilon^t = -1$

donné simplement par $Err_{-1}(\theta) = N - n_e$, car avec cette méthode on est sûr qu'aucun vecteur d'entrée ne se trouve sur l'hyperplan. C'est aussi pour la même raison que l'on peut utiliser la relation 5.26. L'étape **c.1** est l'implémentation de la relation 5.26, et l'étape **c.3** minimise n^t_{poc} et maximise $\mathcal{G}^t_{poc}$ dans les cas d'égalité entre n_e et n^t_{poc}. A la fin de la boucle, la meilleure position est donnée par θ^t_{poc} et ε^t, le nombre d'erreurs par n^t_{poc} et la marge par $\mathcal{G}^t_{poc}$.

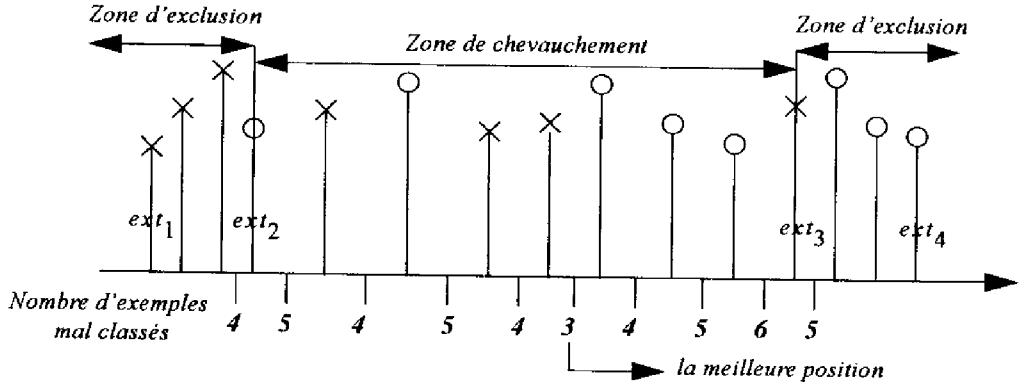

FIG. 5.2: Choix de θ^t_{poc} dans le BCP Min

Un des grands principes de cette procédure dont le principe est illustré sur la figure 5.2, est que la boucle ne se fait que sur les exemples de la zone de chevauchement. Il est inutile de la faire sur les éléments de P_- et P_+, car ces derniers contiennent la projection de points de même cible. La conséquence est que le nombre d'erreurs dans ces deux zones est une fonction monotone stricte. Donc si on considère uniquement ces deux zones, le minimum d'erreur est à une des extrémités. Avec l'algorithme, on teste les deux extrémités intérieures uniquement, mais c'est volontaire car il serait absurde de placer l'hyperplan complètement à l'extérieur du nuage, car dans ce cas il ne sert à rien,

même si cette position minimise le nombre d'erreurs.
C'est un des avantages de cet algorithme par rapport au Pocket par exemple. Dans une situation où la position qui minimise le nombre d'erreurs est complètement à l'extérieur, le Pocket va conserver cette solution. Et dans une telle situation les algorithmes de construction ne convergent pas. Le BCP Min lui va volontairement trouver une solution sous-optimale, mais qui permettra la poursuite de l'algorithme. Une telle situation est par exemple quatre points de cible 1 aux sommets d'un carré et un point de cible 0 au centre. Si on utilise l'Upstart avec le Pocket, il ne convergera pas sur ce problème, alors que l'Upstart avec le BCP Min convergera très rapidement.
D'un point de vue complexité algorithmique, le calcul de la suite ϑ' nécessite un tri réalisé en $O(NlogN)$, alors que la boucle se fait en $O(N)$. Dû au calcul de l'ensemble ϑ, le BCP Heuristique a une complexité de $O(nN)$. Donc le BCP Min a une complexité algorithmique de $O(nN + NlogN)$.
Mais cette complexité est calculée au pire des cas, quand tous les vecteurs d'entrée de ϑ sont dans la zone de chevauchement. En pratique, le BCP est beaucoup plus rapide que la version Ratchet du Pocket. De plus, dès que le nombre de vecteurs d'entrée est assez grand, le Pocket est plus lent. Par rapport au Thermal Perceptron, le BCP Min est plus lent, mais seulement de 47% dans le cas de problèmes hautement NLS comme des fonctions booléennees aléatoires contenant 2^{12} exemples. Comme nous allons le voir dans les simulations, cet algorithme fournit une solution assez bonne en peu d'itérations.
Le BCP Min minimise le nombre d'erreurs et maximise la marge à deux niveaux : pour un w^t fixé, la précédente procédure trouve la solution optimale (si on considère les solutions extérieures comme absurdes), et dans l'algorithme général pour différentes valeurs de w^t. De plus le calcul du seuil comme le milieu de deux éléments de ϑ en association avec la maximisation de la marge fait que l'hyperplan est bien positionné, et en final doit fournir un bon pouvoir de généralisation. Nous le verrons dans les simulations.

5.3.4 Maximisation du nombre d'exclus : l'algorithme BCP Max

Le but de cette procédure est de trouver $\mathcal{H}^t_{poc}$ qui exclut un maximum de vecteurs d'entrée de cible 1 ou 0, à w^t fixé. Ce nombre est noté Ex^t_{poc}. Ces exemples exclus doivent être bien classés, c'est-à-dire que si l'on exclut des exemples de cible 1, ils doivent se trouver dans le demi-espace positif et dans le demi-espace négatif si ce sont des exemples de cible 0.

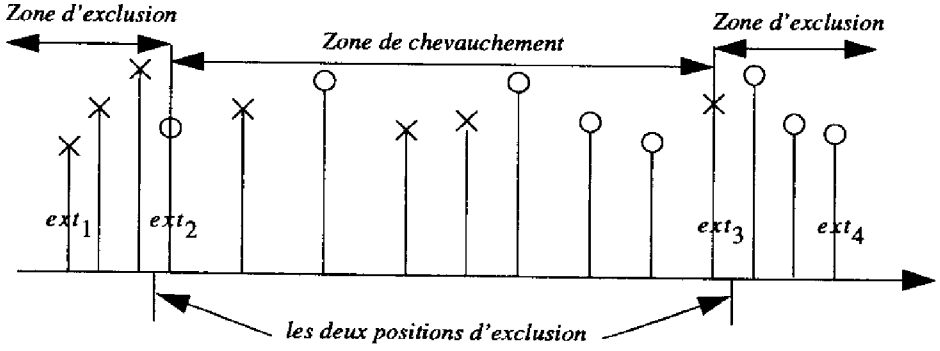

FIG. 5.3: Choix de θ^t_{poc} dans le BCP Max

Le principe de cet algorithme 5.3, illustré sur la figure 5.3, est extrêmement simple. Si $N_- > N_+$ (resp. $N_- < N_+$) les exemples exclus seront ceux dont la projection se trouve dans P_- (resp. P_+). Dans

le cas où $N_- = N_+$, on prendra la position avec la marge la plus grande. Le calcul de cette marge nécessite le calcul des deux autres extrêma qui sont $\max(P_-)$ et $\min(P_+)$.

Algorithme 5.3 Algorithme de maximisation du nombre d'exemples exclus

Etape a : Calcul de ext_1, ext_2, ext_3, ext_4, t_- et t_+
Etape b : Calcul de $\max(P_-)$, $\min(P_+)$, N_- et N_+
Etape c : $\mathcal{G}^- = (ext_2 - \max(P_-))/||w^t||$ et $\mathcal{G}^+ = (\min(P_+) - ext_3)/||w^t||$
Etape d : Si $(N_- > N_+)$ ou $(N_- = N_+ > 0$ et $\mathcal{G}^- \geq \mathcal{G}^+)$ alors
$\theta^t_{poc} = (ext_2 + \max(P_-))/2$, $\mathcal{G}^t_{poc} = \mathcal{G}^-$ et $Ex^t_{poc} = N_-$
Si $(t_- = 1)$ $\varepsilon^t = 1$
Si $(t_- = 0)$ $\varepsilon^t = -1$ et $\theta^t_{poc} \longleftarrow -\theta^t_{poc}$
Etape e : Si $(N_- > N_+)$ ou $(N_- = N_+ > 0$ et $\mathcal{G}^- < \mathcal{G}^+)$ alors
$\theta^t_{poc} = (\min(P_+) + ext_3)/2$, $\mathcal{G}^t_{poc} = \mathcal{G}^+$ et $Ex^t_{poc} = N_+$
Si $(t_+ = 0)$ $\varepsilon^t = 1$
Si $(t_+ = 1)$ $\varepsilon^t = -1$ et $\theta^t_{poc} \longleftarrow -\theta^t_{poc}$

Le BCP Heuristique utilisant cette procédure sera appelé *BCP Max*. Comme dans la procédure précédente, le seuil est toujours calculé comme le milieu de deux éléments consécutifs de ϑ, ce qui accroît la robustesse, en association avec la maximisation de la marge.
Dans le cas particulier où $N_- = N_+ = 0$, c'est-à-dire que les éléments sont dans la zone de chevauchement, alors $ext_1 = ext_2$ et $ext_3 = ext_4$. Dans une telle situation aucun point ne peut être exclu. Une solution à ce problème est simplement d'ajouter 1 temporairement au coefficient de pondération d'un exemple qui ne se trouve pas sur la droite $(b^t_1 b^t_0)$. On peut alors calculer un vecteur w^t (temporaire) qui permet d'exclure un ensemble d'exemples.
Cette procédure est simple et sa complexité est de $O(N)$, car elle nécessite le calcul de 2 extrêma en plus des 4 déjà calculés. Ainsi en final, le BCP Max a une complexité en $O(nN)$, comme le Perceptron. Empiriquement, il est même plus rapide que le Thermal Perceptron (voir les simulations qui suivent).

5.4 Simulations pour apprendre une unité sur des problèmes LS

5.4.1 Fonctions booléennes complètes LS

Cette simulation a été réalisée pour comparer la rapidité de convergence du Perceptron simple, du BCP Prouvé et du BCP Heuristique (sans procédure d'optimisation d'un critère), sur des fonctions booléennes complètes LS. Complète signifie qu'en dimension n, les 2^n vecteurs binaires sont dans la base d'apprentissage avec une cible définie.
Pour construire de telles bases d'apprentissage, on choisit un hyperplan au hasard qui coupe l'hypercube, et la cible de chaque sommet est alors 1 si le sommet est dans le demi-espace positif et 0 sinon.
Cinq types de simulations ont été faites. Trois avec le Perceptron utilisant des valeurs différentes du taux d'apprentissage (0.01, 0.05 et 0.2), une avec le BCP Prouvé et la dernière avec le BCP Heuristique simple. Ces cinq simulations furent lancées sans limitation du nombre d'itérations.
Pour les cinq simulations, la moyenne du nombre d'itérations nécessaires pour atteindre une solution fut calculée sur 10000 tirages. Sauf pour le Perceptron pour les dimensions supérieures à 10 et pour le BCP Prouvé pour les dimensions supérieures à 12, pour des raisons évidentes de temps de calcul.
Les résultats obtenus sont représentés sur la figure 5.4, en fonction de la dimension. Les lignes pleines représentent les moyennes du nombre d'itérations nécessaire pour chaque algorithme, et les lignes en

pointillés la moyenne plus l'écart-type.
Cette figure nous montre la grande efficacité du BCP Heuristique. Le BCP Prouvé est en moyenne un ordre de grandeur plus rapide que le Perceptron. Le BCP Heuristique est déjà 500 fois plus rapide que le Perceptron en dimension 10 pour la même complexité algorithmique. Mais l'allure de ces courbes est totalement différente. Ainsi en dimension 16, le BCP Heuristique ne prend en moyenne que 14,6 itérations avec un écart-type de 4,4. En extrapolant les courbes, on peut estimer le nombre d'itérations requises par le Perceptron à 10^6 (5.10^4 pour le BCP Prouvé), ce qui fait une accélération de 70000 par rapport au Perceptron.

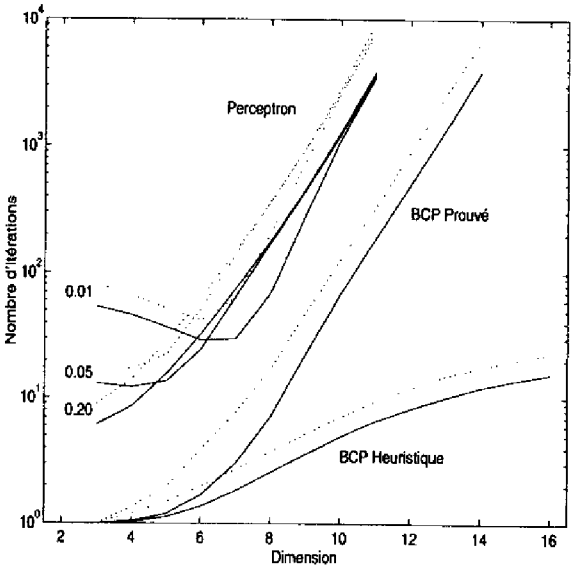

FIG. 5.4: Fonctions booléennes complètes LS.

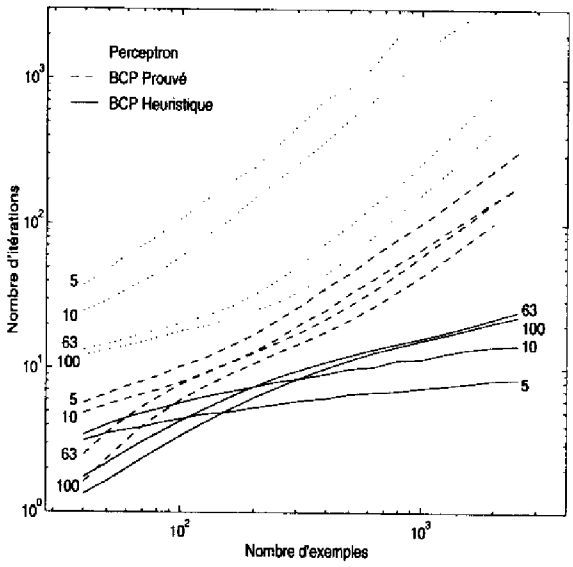

FIG. 5.5: Problèmes analogiques LS.

5.4.2 Problèmes analogiques

Dans cette simulation, la comparaison est faite sur des problèmes analogiques. Un hyperplan est choisi au hasard de manière à couper le cube $[0.2, 0.8]^n$. Chaque vecteur d'entrée tiré au hasard dans le cube unité est alors de cible 1 s'il se trouve du côté positif de l'hyperplan et 0 sinon.
Trois types de simulations ont été réalisés, sans limitation du nombre d'itérations, avec quatre valeurs différentes de dimensions de l'espace (5, 10, 63 et 100) : le Perceptron avec le paramètre d'apprentissage mis à 0,1, le BCP Prouvé et le BCP Heuristique. Ces simulations ont été faites avec 20 valeurs différentes du nombre d'exemples dans la base d'apprentissage, réparties uniformément sur une échelle logarithmique.
La valeur moyenne du nombre d'itérations nécessaires pour atteindre une solution est tracée en fonction du nombre d'exemples dans la base, pour chaque algorithme et chacune des dimensions testées, sur la figure 5.5. Pour chaque courbe, la dimension est indiquée sur le côté.
Ce graphique nous montre que le BCP Prouvé est au minimum 2 fois plus rapide que le Perceptron, et le BCP Heuristique au minimum un ordre de grandeur plus rapide. Mais là encore, l'allure des courbes nous montre la grande efficacité du BCP Heuristique. Ainsi en dimension 5 avec 2500 vecteurs d'entrée, il faut en moyenne moins de 10 itérations au BCP Heuristique, alors qu'on peut extrapoler le nombre d'itérations nécessaires au Perceptron à plus de 10^4. Ce qui fait une accélération de plus de 1000.

5.4.3 Justification de la convergence du BCP Heuristique

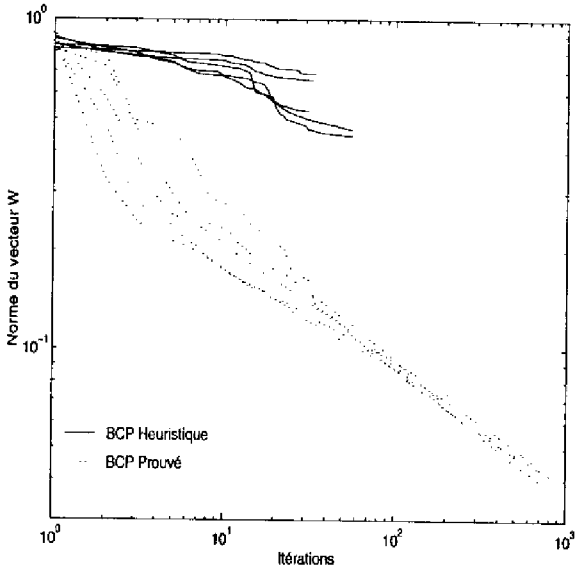

FIG. 5.6: Evolution de la norme du vecteur w^t pour des problèmes LS.

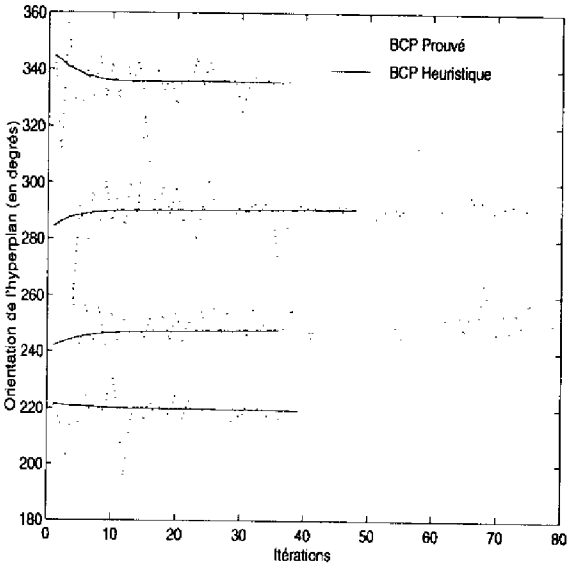

FIG. 5.7: Evolution de l'orientation de l'hyperplan pour des problèmes LS.

Les deux simulations qui suivent ont pour but de justifier la convergence du BCP Heuristique, car pour cette version aucun théorème de convergence n'a été trouvé.

La première simulation fut réalisée sur des fonctions booléennes LS complètes en dimension 11. Cinq bases d'apprentissage parmi les plus difficiles à séparer furent sélectionnées (quand le BCP Heuristique mettait plus de 30 itérations pour converger).

La figure 5.6 représente l'évolution de la norme du vecteur w^t lors de l'apprentissage de ces bases. Pour le BCP Prouvé, cette norme est strictement décroissante tant qu'il reste des exemples mal classés. Ceci est normal, car l'algorithme a été conçu afin de minimiser la norme du vecteur w^t, pour qu'il converge vers le connecteur minimal. Mais on peut noter que pour le BCP Heuristique, la norme de w^t est *en moyenne* aussi décroissante. Le taux de décroissance n'est pas aussi important, et quelques fluctuations peuvent apparaître. Néanmoins, l'expérience a montré qu'en moyenne, $||w^t||$ decroît.

Dans la seconde expérience, on analyse l'orientation de l'hyperplan pendant l'apprentissage. Les problèmes étudiés sont des problèmes analogiques LS en dimension 2 avec 1000 exemples. Quatre bases d'apprentissage difficiles à résoudre furent sélectionnées, avec la même méthode que dans l'expérience précédente. L'angle que fait le vecteur w^t avec l'axe des abscisses a été calculé pendant l'apprentissage de ces problèmes, et est représenté sur la figure 5.7 en fonction des itérations. On peut noter que pour le BCP Prouvé, il y a beaucoup de fluctuations, certainement dûes à la procédure de minimisation de la fonction Δ^t. Au contraire, le BCP Heuristique converge rapidement vers la bonne orientation sans fluctuation.

Ces deux simulations nous permettent de comprendre pourquoi le BCP Heuristique est si rapide à converger, ce qui peut se résumer comme suit :

La rapidité de convergence du BCP Heuristique semble provenir de la décroissance moyenne de la norme du vecteur w^t, et surtout d'une convergence de l'orientation de l'hyperplan sans fluctuations. Alors que la convergence du BCP Prouvé est dûe à la décroissance stricte de la norme de w^t, celle du BCP Heuristique est plutôt dûe à une meilleure évolution de l'orientation de l'hyperplan.

5.5 Simulations pour apprendre une unité sur des problèmes NLS

Le but des deux premières simulations est de montrer les performances du BCP Prouvé pour la détection de la non séparabilité d'une base d'apprentissage. On étudie donc la rapidité de décroissance de la norme du vecteur w^t vers 0, sur des bases NLS.
La troisième simulation compare l'évolution de la norme du vecteur w^t en utilisant le BCP Prouvé et le BCP Heuristique, ce qui amène à conclure que l'utilisation du BCP Prouvé pour optimiser un des deux critères n'est pas efficace.
Les deux suivantes présentent les performances de BCP Max pour exclure des vecteurs d'entrée, en vue d'une utilisation avec le Sequential Learning.
Les trois dernières simulations concernent le BCP Min et les performances obtenues pour minimiser le nombre d'erreurs, en comparaison avec le Pocket, le Ratchet et le Thermal Perceptron.
Une comparaison des temps de calcul termine cette section.

5.5.1 BCP Prouvé sur des fonctions booléennes aléatoires

Dans cette première simulation, les bases d'apprentissage sont constituées de fonctions booléennes complètes pour lesquelles les cibles sont tirées au hasard avec des probabilités égales de cibles 1 et de cibles 0. A l'exception de quelques bases en faibles dimensions, ces bases d'apprentissage sont hautement NLS. Cette simulation a été faite pour des dimensions allant de 3 à 17.
Le BCP Prouvé fut arrêté dans trois cas :

- le BCP Prouvé détecte une base LS,
- la norme du vecteur w^t était inférieure à 10^{-8}. Il est alors raisonnable d'estimer que la base est NLS et que le BCP Prouvé l'a détecté.
- le nombre d'itérations atteint la limite fixée à 1000. Dans ce cas le problème peut être LS, mais il y de grandes chances qu'il soit NLS et que le BCP Prouvé ne l'ait pas détecté. Ces problèmes seront nommés NLS durs (NLS_d). Ce cas exceptionnel est apparu seulement pour les dimensions 4 et 5, et au vu de la figure 5.4, si ces bases avaient été LS, il est vraisemblable que l'algorithme l'aurait détecté en moins de 1000 itérations.

Pour les dimensions allant de 3 à 13, on a réalisé un nombre d'essais suffisant pour avoir 20000 bases NLS détectées par l'algorithme, moins pour les dimensions supérieures. Le tableau suivant récapitule le nombre d'essais réalisés, le nombre de fonctions NLS détectées, le nombre de fonctions LS trouvées et le nombre de fonctions NLS_d, pour chaque dimension.

Dimension	3	4	5	6 ... 13	14	15	16	17
Essais	33465	22010	20028	20000	5000	1000	1000	500
NLS	20000	20000	20000	20000	5000	1000	1000	500
LS	13465	621	1	0	0	0	0	0
NLS_d	0	1389	27	0	0	0	0	0

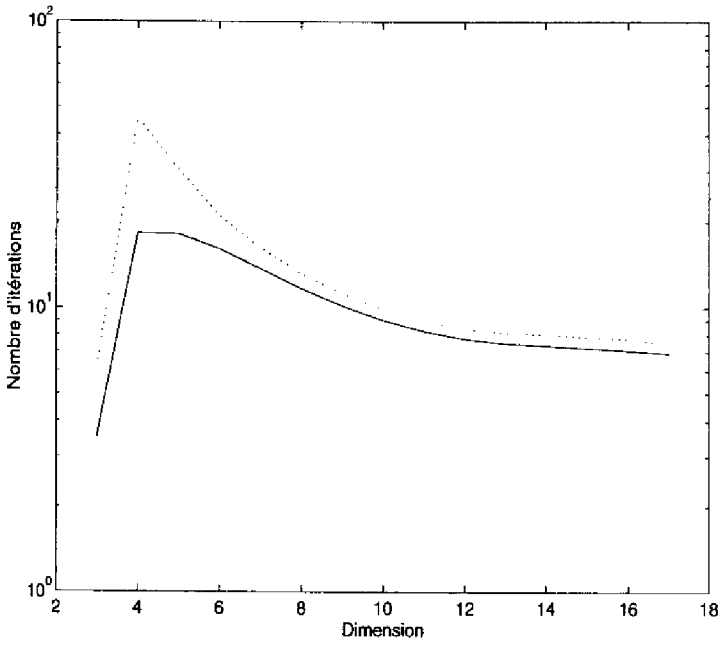

FIG. 5.8: Rapidité de convergence de la norme du vecteur w^t vers 0 pour des fonctions booléennes complètes NLS.

La figure 5.8 représente, en fonction de la dimension, le nombre moyen (ligne pleine) d'itérations nécessaires pour avoir $\|w^t\| < 10^{-8}$ sur les essais pour lesquels la fonction a été détectée NLS. La moyenne plus l'écart-type est tracée en pointillés. Les parties LS et NLS_d n'ont donc pas été prises en compte dans ce graphique.
On peut noter le faible nombre d'itérations nécessaire pour arriver à une très faible valeur de la norme de w^t, ce qui démontre l'efficacité de la méthode. Les cas exceptionnels en dimensions 4 et 5 semblent apparaître quand la dimension affine de l'intersection entre les enveloppes convexes des points de cible 1 et des points de cible 0 est inférieure à n. Dans ce cas, on a pu observer que la norme de w^t décroît mais la décroissance est lente. De tels cas ne se produisent pas souvent.

5.5.2 BCP Prouvé sur des distributions gaussiennes avec chevauchement

Dans cette seconde simulation, les bases d'apprentissage sont constituées de deux distributions gaussiennes qui se chevauchent. S'il y a suffisamment d'exemples, ce type de base est très NLS. Chaque distribution est constituée de $N/2$ vecteurs d'entrée, une d'exemples de cible 1 et l'autre d'exemples de cible 0, les centres étant les points de coordonnées $(-1, 0, \ldots)$ et $(1, 0, \ldots)$. La matrice de covariance est la matrice unité. Les essais furent réalisés pour $N \in \{100, 300, 500, 1000, 2000\}$ et pour des dimensions allant de 3 à 13.
Sur la figure 5.9, la moyenne du nombre d'itérations nécessaire pour avoir $\|w^t\| < 10^{-8}$ est tracée (lignes pleines) en fonction de la dimension pour chaque valeur de N. Pour chaque dimension et chaque N, la moyenne fut calculée sur 1000 tirages. La moyenne plus l'écart-type est représentée en pointillés.
Quelques fonctions LS furent détectées et non prises en compte dans cette figure, seulement pour $N = 100$ et de hautes dimensions : 1 pour $n = 9$, 2 pour $n = 11$, 3 pour $n = 12$ et 5 pour $n = 13$.

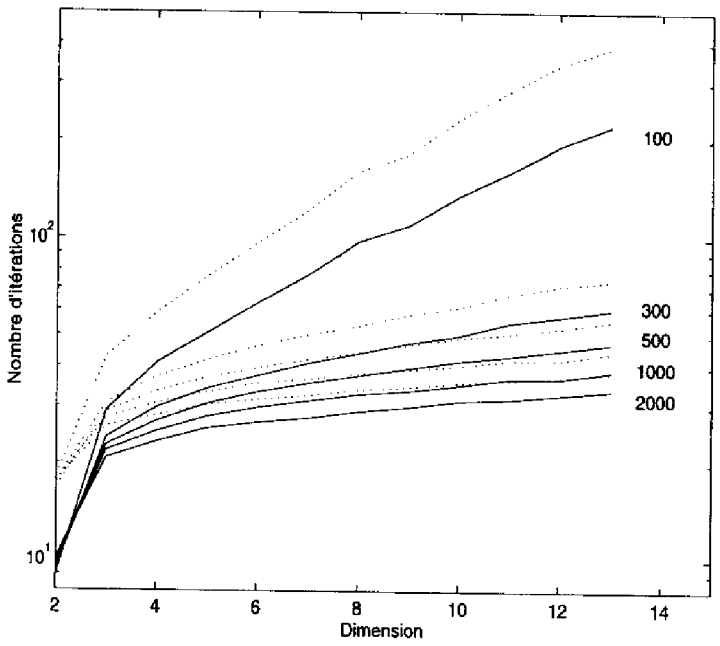

FIG. 5.9: Rapidité de convergence de la norme du vecteur w^t vers 0 pour deux distributions gaussiennes qui se chevauchent.

Le BCP Prouvé détecta donc toutes les parties NLS avec un nombre assez faible d'itérations. Aucune partie NLS_d ne fut détectée.

5.5.3 Comparaison de l'évolution de $||w^t||$ pour des problèmes NLS

Les bases d'apprentissage utilisées sont constituées de deux distributions gaussiennes en dimension 6 avec 400 exemples de cible 1 et 400 de cible 0. La figure 5.5.3 représente l'évolution de la norme du vecteur w^t pendant l'apprentissage pour le BCP Prouvé et le BCP Heuristique.
Pour le BCP Prouvé, $||w^t||$ converge rapidement vers 0. Il a fallu seulement de 40 à 45 itérations pour avoir $||w^t|| < 10^{-10}$ pour les cinq bases d'apprentissage testées.
Pour le BCP Heuristique, la norme du vecteur w^t décroît faiblement au début puis ne décroît plus, avec beaucoup de fluctuations. Ce sont ces fluctuations qui permettent aux deux procédures présentées précédemment d'optimiser le critère désiré. Avec le BCP Prouvé, la convergence vers 0 est trop rapide pour avoir le temps d'optimiser le critère.

5.5.4 BCP Max sur des fonctions booléennes aléatoires

Cette simulation et la suivante concernent le BCP Max, et l'étude du nombre de vecteurs d'entrée exclus trouvés par l'algorithme. Comme on l'a déjà signalé, cette version est destinée à être utilisée avec le Sequential Learning proposé par Marchand [125]. Dans son article, Marchand propose un algorithme pour trouver un ensemble d'exemples exclus, mais cette méthode est très lourde en temps de calcul. Le Perceptron doit être utilisé sur de multiples bases d'apprentissage pour avoir un ensemble d'exemples exclus. Mis à part le BCP Max, la méthode proposée par Marchand semble être la seule. Mais elle ne peut être comparée à BCP Max.

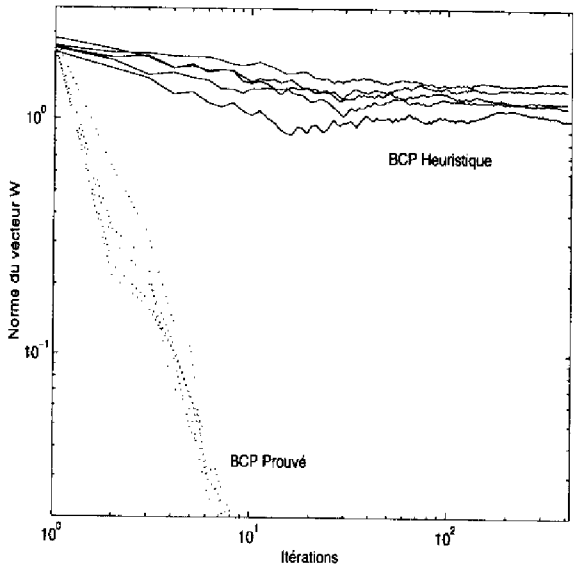

FIG. 5.10: Comparaison de l'évolution de $||w^t||$ avec BCP Heuristique et BCP Prouvé sur des problèmes NLS.

La première simulation a été menée avec des fonctions booléennes aléatoires en dimension 8. Pour un nombre fixe d'itérations, le BCP Max a été lancé sur plusieurs bases d'apprentissage tirées au hasard. La moyenne du nombre d'exemples exclus est représentée sur la figure 5.11 en fonction du nombre d'itérations.

Vingt valeurs différentes du nombre d'itérations ont été considérées, uniformément réparties sur une échelle logarithmique. Chaque point de la courbe est une moyenne de 1500 tirages. Les lignes en pointillés représentent la moyenne plus ou moins l'écart-type du nombre d'exemples exclus.

Cette figure nous montre que pour ce problème, il n'est pas nécessaire de prendre plus de 500 itérations, car le nombre d'exclus augmente peu (ceci est plus marqué sur un graphique à échelles normales, non logarithmiques).

5.5.5 BCP Max sur deux distributions gaussiennes

Deux bases d'apprentissage seulement ont été considérées dans cette simulation. Elles consistent en deux distributions gaussiennes qui se chevauchent en dimension 2, constituées de 200 exemples de cible 1 et 200 exemples de cible 0.

La valeur moyenne du nombre d'exclus a été tracée en fonction du nombre d'itérations sur la figure 5.12, pour les deux bases d'apprentissage. Dans cette simulation, pour un nombre d'itérations fixé et une base d'apprentissage donnée, la moyenne est calculée sur différentes utilisations du BCP Max, car dans ce dernier le choix du seuil θ^t est aléatoire, donc deux utilisations peuvent donner des résultats différents. Les lignes en pointillés représentent la moyenne du nombre d'exclus plus ou moins l'écart-type.

Cette figure nous montre que 200 itérations sont suffisantes pour avoir une bonne position.

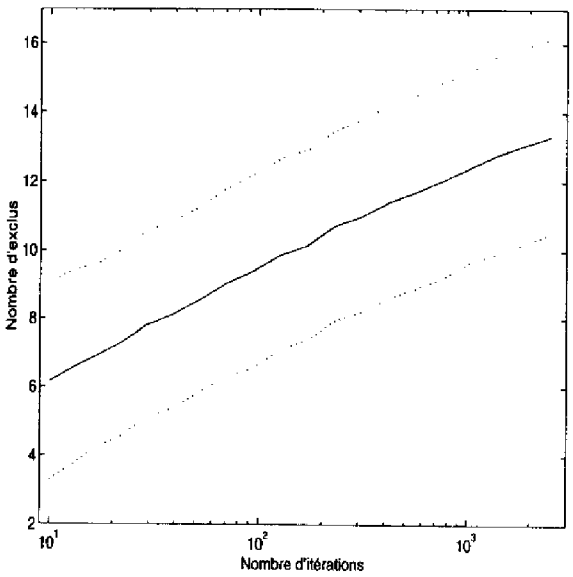

FIG. 5.11: Nombre d'exemples exclus par BCP Max en fonction du nombre d'itérations pour des fonctions booléennes aléatoires en dimension 8.

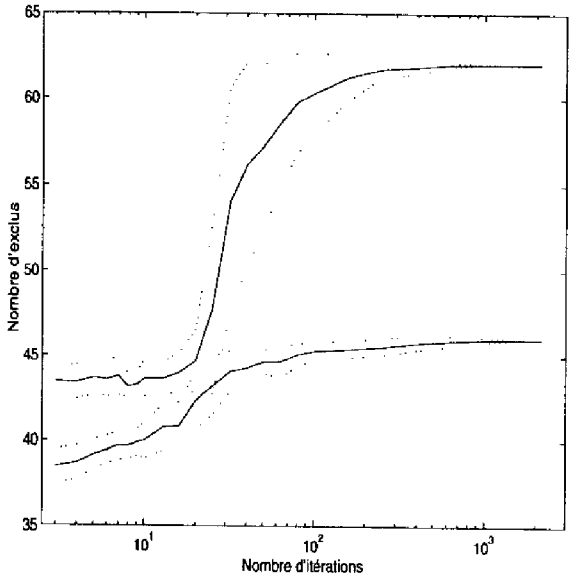

FIG. 5.12: Nombre d'exemples exclus par BCP Max en fonction du nombre d'itérations pour des distributions gaussiennes qui se chevauchent en dimension 2.

5.5.6 BCP Min sur des fonctions booléennes aléatoires

Dans cette simulation et les deux qui suivent, nous comparons les performances du BCP Min avec d'autres algorithmes pour la minimisation du nombre d'exemples mal classés.
La première simulation est réalisée avec des fonctions booléennes aléatoires en dimension 8. On compare le nombre d'erreurs résiduelles après un nombre fixe d'itérations pour les différents algorithmes : le Pocket, le Ratchet, le Thermal Perceptron avec trois températures initiales différentes et le BCP Min. L'utilisation du Perceptron simple, du Pocket et du Ratchet pose un problème délicat qui est le choix du paramètre d'apprentissage. Dans le cas du Thermal Perceptron ce paramètre d'apprentissage est toujours mis à 1 au début et decroît linéairement pendant l'algorithme. Mais avec celui-ci le problème majeur réside dans le choix de la température initiale. Comme on l'a déjà abordé dans l'introduction, la performance de ces algorithmes dépend plus ou moins de ces divers paramètres, et particulièrement le Thermal Perceptron. Donc pour faire une comparaison honnête, nous avons décidé de rechercher la température initiale optimale pour le Thermal Perceptron. Il convient de signaler que cette recherche est très longue et que, comme nous allons le voir, la température optimale dépend complètement de la base d'apprentissage. Il semble que cette température dépende même du nombre d'itérations. Avec la même base d'apprentissage, le Thermal Perceptron fut lancé 20 fois avec des températures initiales différentes réparties uniformément sur une échelle logarithmique, avec un nombre d'itérations fixé à 500. La moyenne du nombre d'exemples mal classés, calculée sur 2000 tirages différents est représentée sur la figure 5.13 en fonction de la température initiale. La température initiale optimale est environ égale à $T_{opt} \simeq 1,24$.
Les deux autres températures initiales utilisées dans la simulation qui suit sont $T_0 = 4$ et $T_1 = 0,3$. Durant l'algorithme, la température decroît linéairement de la température initiale jusqu'à 0. Pour le Pocket et le Ratchet le paramètre d'apprentissage utilisé est $0,1$.
Les six algorithmes sont lancés sur la même base d'apprentissage et relancés depuis le début pour chaque valeur du nombre d'itérations (20 valeurs). La figure 5.14 représente la moyenne du nombre

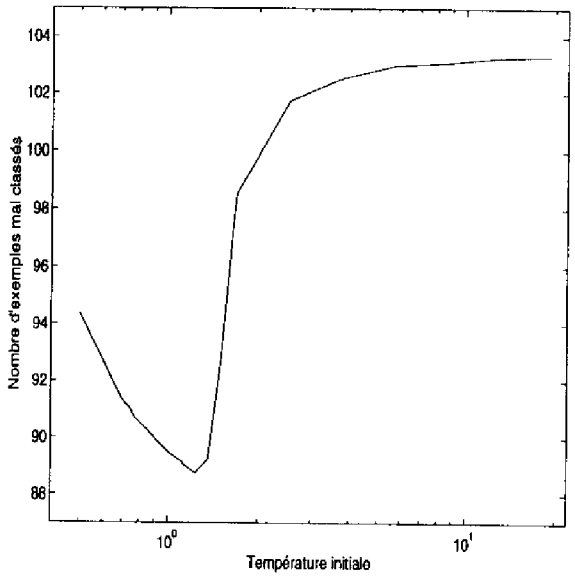

FIG. 5.13: Recherche de la température initiale optimale pour le Thermal Perceptron avec des fonctions booléennes aléatoires en dimension 8.

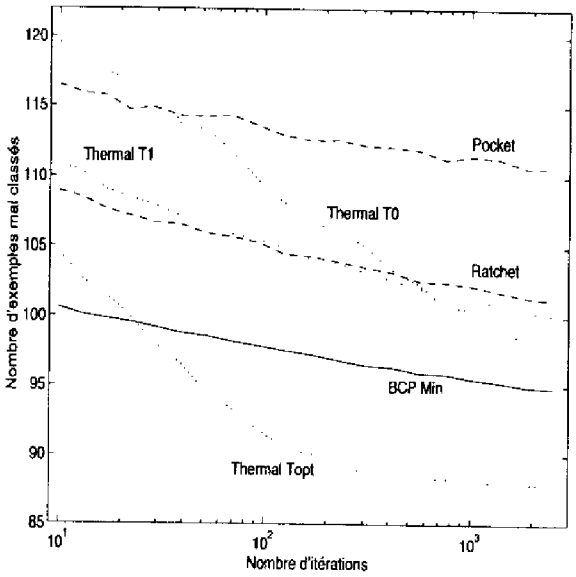

FIG. 5.14: Minimisation du nombre d'exemples mal classés pour des fonctions booléennes aléatoires en dimension 8.

d'exemples mal classés en fonction du nombre d'itérations pour les divers algorithmes. Les moyennes sont calculées sur 500 tirages différents.

Le BCP donne de meilleurs résultats que le Ratchet qui est plus efficace que le Pocket. Le Thermal Perceptron avec la température optimale est meilleur que le BCP, mais donne de plus mauvais résultats avec les deux autres températures.

5.5.7 BCP Min sur des problèmes analogiques

Dans cette simulation, les bases d'apprentissage sont analogiques, et constituées de 500 vecteurs d'entrée tirés au hasard dans le cube unité, avec les cibles prises au hasard équitablement réparties entre 0 et 1. Pour le Thermal Perceptron, la température initiale optimale a été recherchée avec 500 itérations. Les résultats représentés sur la figure 5.15 nous indiquent une température initiale optimale de $T_{opt} \simeq 0,56$.

Comme le Pocket donne toujours de plus mauvais résultats que le Ratchet (ce qui est logique d'après leurs fonctionnements), la comparaison est faite avec le Ratchet et Thermal avec deux températures initiales, l'optimale et une autre prise égale à $T_2 = 1,0$, et le BCP Min. Le paramètre d'apprentissage du Pocket est pris égal à 0,1. Comme dans la simulation précédente, les quatre algorithmes sont relancés depuis le début sur la même base d'apprentissage pour les différentes valeurs du nombre d'itérations. La figure 5.16 représente la moyenne du nombre d'exemples mal classés en fonction du nombre d'itérations. Les moyennes sont calculées sur 700 bases d'apprentissage différentes.

Pour cette simulation, le BCP est plus performant que le Thermal avec la température initiale T_2 et que le Ratchet. Avec seulement 10 itérations, il fournit le même résultat que le Thermal ou le Ratchet avec plus de 1500 itérations. En utilisant la température optimale, le Thermal donne de meilleurs résultats, mais il arrive au niveau de performance du BCP avec plus de 1500 itérations. Le BCP donne donc les meilleurs résultats pour ce problème.

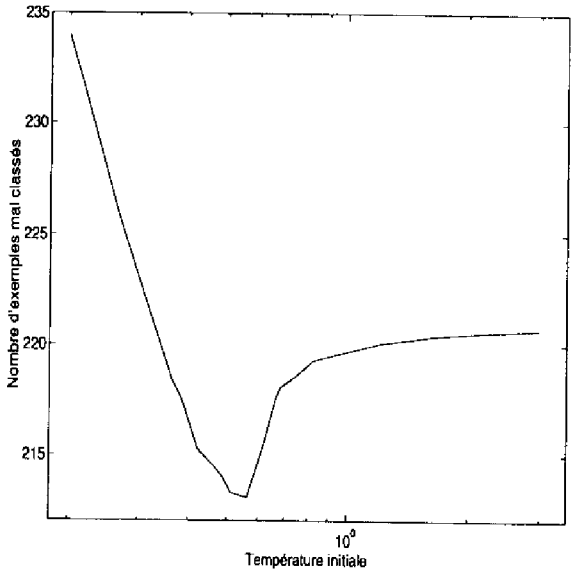

FIG. 5.15: Recherche de la température initiale optimale avec des problèmes analogiques aléatoires en dimension 4.

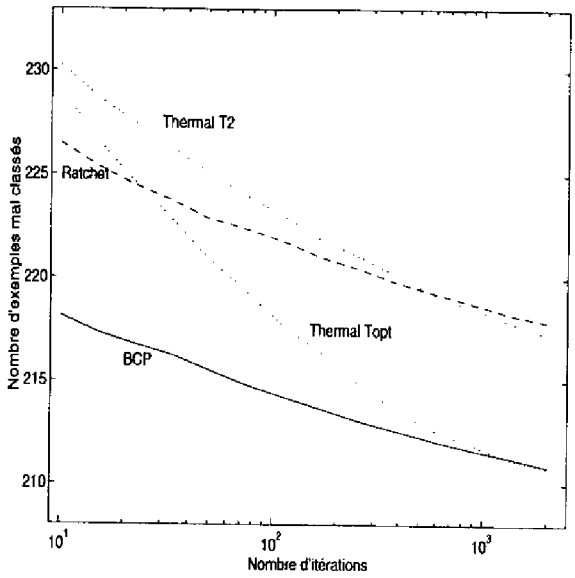

FIG. 5.16: Minimisation du nombre d'exemples mal classés pour des problèmes analogiques aléatoires en dimension 4.

5.5.8 BCP Min sur le problème deux spirales

La figure 5.18 présente les résultats d'une simulation similaire à la précédente, réalisée sur un problème deux spirales. La base d'apprentissage est constituée de deux spirales de 300 exemples chacune, une d'exemples de cible 1 et l'autre d'exemples de cible 0. Ces deux spirales entrelacées sont symétriques par rapport à l'origine du repère et sont constituées de 50 points par π, ce qui fait en fait 3 tours complets pour chacune. La moyenne du nombre d'exemples mal classés est calculée sur 1000 essais pour chaque algorithme et chaque valeur du nombre d'itérations.

Comme dans les deux simulations précédentes, la température initiale optimale a été recherchée avec 500 itérations. Les résultats sont représentés sur la figure 5.17 et nous indique une température optimale à $T_{opt} = 12$. Cette figure nous montre l'extrême difficulté du réglage de cette température pour ce problème, dûe à la finesse de la zone optimale. La seconde température utilisée dans la simulation a été prise à $T_2 = 100$.

Comme on peut le voir sur la figure 5.18, le Ratchet donne de meilleurs résultats que le Thermal avec T_2, et après 350 itérations le Thermal avec sa température optimale devient meilleur. Cette simulation nous montre la grande efficacité du BCP. Un excellent résultat est obtenu dès 10 itérations, atteint par le Thermal Perceptron avec la température initiale optimale avec plus de 1400 itérations. En conclusion, pour ce problème, le BCP Min est le plus performant même quand le Thermal est complètement optimisé. Le BCP Min n'a besoin d'aucun réglage, alors que le réglage du Thermal nécessite de très longs calculs.

Pour conclure cette série de simulations, ces trois dernières nous ont montré que la température initiale optimale dépend beaucoup du type de problème ($T_{opt} = 1.24$, 0.56 and 12), et que quand celle-ci n'est pas utilisée, les performances de ce dernier sont beaucoup plus faibles que celles atteintes par le BCP Min.

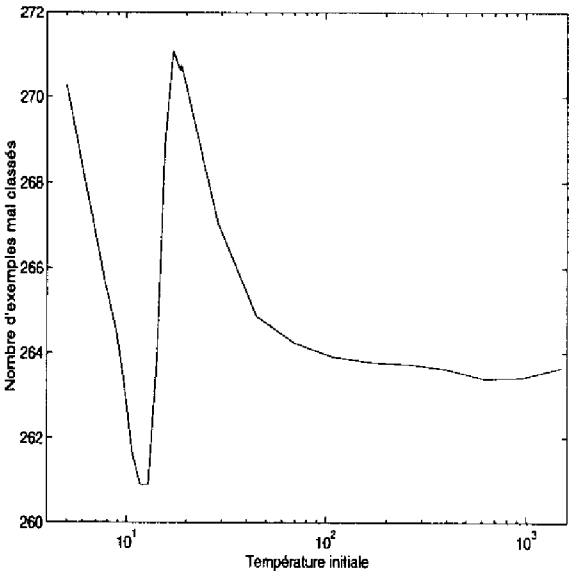

FIG. 5.17: Recherche de la température optimale pour le problème deux spirales.

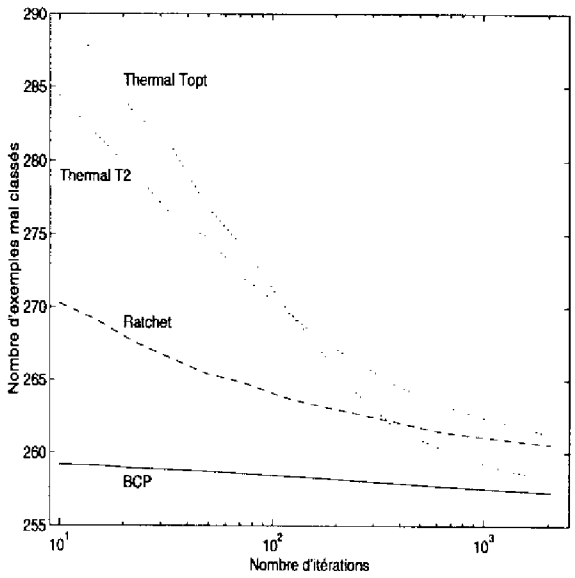

FIG. 5.18: Minimisation du nombre d'exemples mal classés pour le problème deux spirales.

5.5.9 Comparaison des temps de calcul

Dans cette section, nous comparons le temps de calcul des divers algorithmes utilisés précédemment sur plusieurs types de bases d'apprentissage. Les résultats sont résumés dans le tableau 5.1. Pour chaque type de base et chaque algorithme, le résultat présenté est le temps de calcul par itération divisé par ce même temps pour le BCP Min sur la même base. Ces valeurs sont des moyennes de plusieurs essais sur la même base, sur 500 bases différentes de chaque type. Le nombre d'essais réalisés pour chaque base dépend de l'algorithme et du type de base utilisée. Par exemple, le BCP Min et BCP Max étant beaucoup plus rapides (en nombre d'itérations) que les autres algorithmes pour les problèmes LS, le nombre d'essais réalisés est beaucoup plus important pour ces deux algorithmes, afin d'avoir des temps globaux comparables et assez importants pour minimiser les erreurs dues à la faible précision de la base de temps (20ms). Toutes ces simulations ont été réalisées sur le même système d'exploitation monotâche, et les programmes écrits dans le même langage et compilés avec les mêmes options.
On peut noter premièrement que le BCP Min est le plus rapide, un peu plus rapide que le Thermal Perceptron, ce qui doit être dû au calcul de l'exponentielle. Deuxièmement, le temps de calcul du Ratchet croît de manière très importante avec le nombre de vecteurs d'entrée. Le BCP Min aussi, mais de manière moins significative. Troisièmement, dans le cas de problèmes LS, et en fait d'une manière générale quand le nombre d'exemples dans la zone de chevauchement est faible, le temps de calcul du BCP Min est sensiblement le même que pour le BCP Max, et même plus faible que le Thermal Perceptron. Les temps de calcul du BCP Prouvé n'ont pas été mis dans ce tableau, car ils sont très proches de ceux de BCP Max.

5.6 Le BCP et les algorithmes de construction : simulations

L'objectif de cette section est une comparaison des algorithmes de construction utilisant le Thermal, le Ratchet, le BCP Min et le BCP Max, en termes de nombre d'unités construites et de pouvoir de

TAB. 5.1: Comparaison des temps de calcul

Problèmes	BCP Min.	BCP Exc.	Thermal	Ratchet	Pocket
Fonctions booléennes aléatoires n=8	1	0,641	0,677	0,594	0,672
Fonctions booléennes aléatoires n=10	1	0,598	0,679	1,660	0,633
Fonctions booléennes aléatoires n=12	1	0,561	0,679	3,390	0,620
Fonctions booléennes LS n=10	1	0,991	1,210	8,970	1,190
Prob. analog. NLS 1000 exemples n=3	1	0,454	0,528	1,579	0,441
2 distrib. gaus. 1000 exemples n=6	1	0,539	0,588	3,211	0,549

généralisation. Comme algorithme "conventionnel", c'est-à-dire qui utilise une stratégie de minimisation du nombre d'erreurs, nous avons utilisé l'Upstart [71], car il semble que ce soit le plus efficace [128]. Le Sequential Learning a bien sûr été utilisé. La comparaison est donc faite avec Upstart + Thermal, Upstart + Ratchet, Upstart + BCP Min et Sequential Learning + BCP Max.

5.6.1 Fonctions booléennes aléatoires en dimension 6

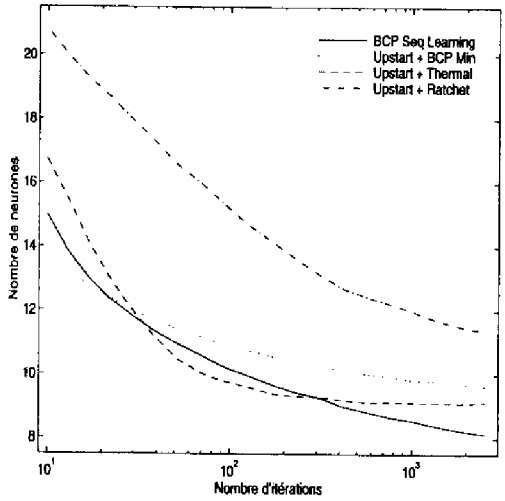

FIG. 5.19: Nombre d'unités construites pour des fonctions booléennes aléatoires en dimension 6.

Le nombre d'unités construites par les différents algorithmes est étudié pour plusieurs valeurs du nombre d'itérations pour apprendre une unité. Avec l'algorithme Upstart, si le nombre d'erreurs résiduelles d'une unité fille est plus grand ou égal au nombre d'exemples de cible 1 dans la base apprise, alors un exemple de cible 1 est séparé des autres avec un neurone grand-mère. Cette situation apparaît pour les nombres d'itérations faibles, et peut rendre impossible la convergence, c'est pour cette raison qu'il est résolu avec un neurone grand-mère. La température initiale pour le Thermal Perceptron a été prise à 1, ce qui est proche de l'optimal calculé pour la dimension 8.
La figure 5.19 représente la moyenne sur 2000 essais du nombre d'unités construites en fonction du nombre d'itérations. Le Ratchet a donné de faibles résultats. Pour un nombre faible d'itérations, le BCP donne les meilleurs résultats, avec le Sequential Learning et avec l'Upstart. Entre 20 et 300 itérations, Upstart + Themal donne les meilleurs résultats, mais après 300 itérations le BCP Sequential Learning

(BCPSL) donne le plus faible nombre d'unités. Il convient enfin de noter que le nombre d'unités construites considéré ici avec l'Upstart est le nombre d'unités pour la structure de réseaux en arbre binaire. Ce nombre d'unité est d'une unité supérieur au nombre d'unité utilisé dans la structure à une couche cachée. Donc en prenant cette remarque en compte, si on compare à structure équivalente, le BCPSL donne toujours le meilleur résultat.

5.6.2 Problème symétrie miroir

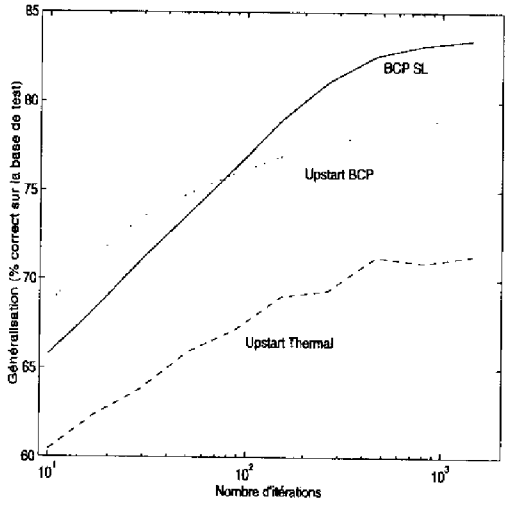

FIG. 5.20: Pouvoir de généralisation pour le problème symétrie miroir.

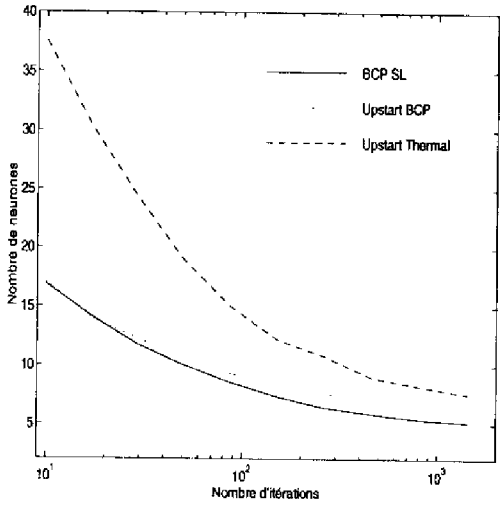

FIG. 5.21: Nombre d'unités construites pour le problème symétrie miroir.

Les bases d'apprentissage considérées ici sont constituées de 150 vecteurs binaires en dimension 20. La base de test est de 1500 vecteurs. Un vecteur est de cible 1 si sa représentation binaire est symétrique par rapport au centre, et de cible 0 sinon. Chaque vecteur est constitué en fixant d'abord sa cible, équitablement répartie entre 0 et 1. Ensuite la moitié de ce vecteur est générée aléatoirement, et la seconde est générée aléatoirement si c'est un exemple de cible 0, ou construite à partir de la première si c'est un exemple de cible 1, pour avoir un vecteur symétrique. Pour chaque nombre d'itérations, les algorithmes sont utilisés sur les mêmes bases d'apprentissage et les réseaux construits testés sur les mêmes bases de tests. Les résultats sont des moyennes sur 500 essais.
La figure 5.20 nous montre le pourcentage d'exemples bien classés dans la base de test en fonction du nombre d'itérations, et la figure 5.21 le nombre d'unités construites.
Avec seulement 200 itérations, le BCPSL donne de très bons résultats en généralisation et le plus faible nombre d'unités. Avec le Sequential Learning ou avec l'Upstart le BCP est plus performant que le Thermal, au niveau du nombre d'unités construites et de la capacité de généralisation. Avec plus de 1400 itérations, la capacité de généralisation des réseaux construits avec le BCPSL est de 13% supérieure à celle des réseaux construits avec l'Upstart + Thermal.

5.6.3 Problème de détection de décalage

Dans ce problème, les bases d'apprentissage se composent de 100 vecteurs en dimension 30, et les bases de test de 1000 vecteurs. Pour générer un vecteur, la cible est tirée au hasard, ainsi que les 15 premiers bits, qui sont dupliqués dans la seconde partie du mot binaire (sans symétrie). Si la cible est 1, les 15 derniers bits subissent une rotation à droite, et dans le cas d'une cible 0 une rotation à gauche.

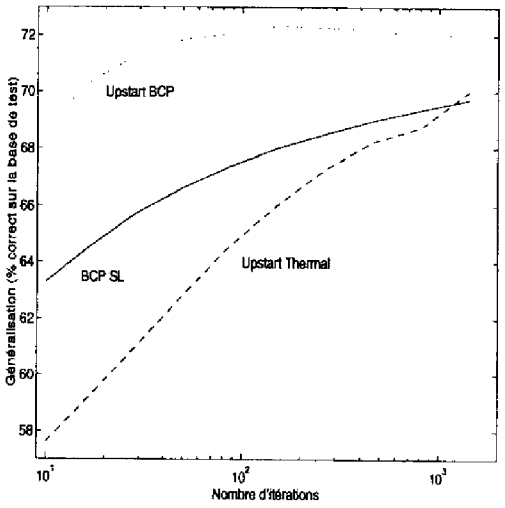

FIG. 5.22: Pouvoir de généralisation pour le problème de détection de décalage.

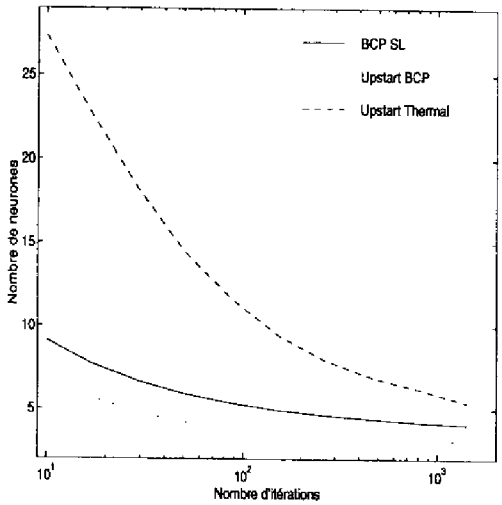

FIG. 5.23: Nombre d'unités construites pour le problème de détection de décalage.

Cette simulation est similaire à la précédente, les résultats étant des moyennes sur 2500 tirages. Les capacités de généralisation (pourcentage de bien classés dans la base de test) sont représentées sur la figure 5.22 en fonction du nombre d'itérations, et le nombre d'unités construites sur la figure 5.23 . L'Upstart plus le BCP donne de très bons résultats avec seulement 100 à 200 itérations. Ce résultat n'est pas atteint par l'Upstart + Thermal, même avec plus de 1400 itérations.

5.6.4 Problème de parité en dimension 8

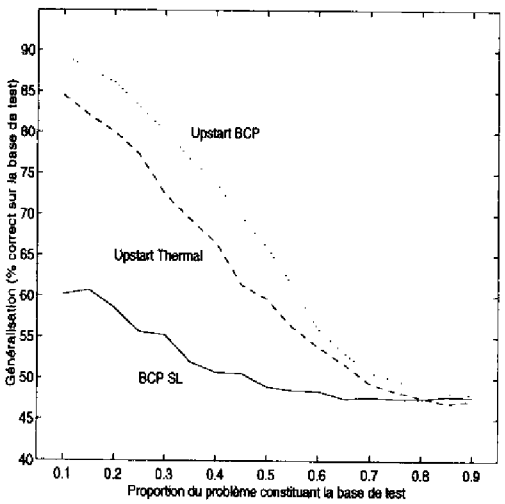

FIG. 5.24: Pouvoir de généralisation pour le problème de parité en dimension 8.

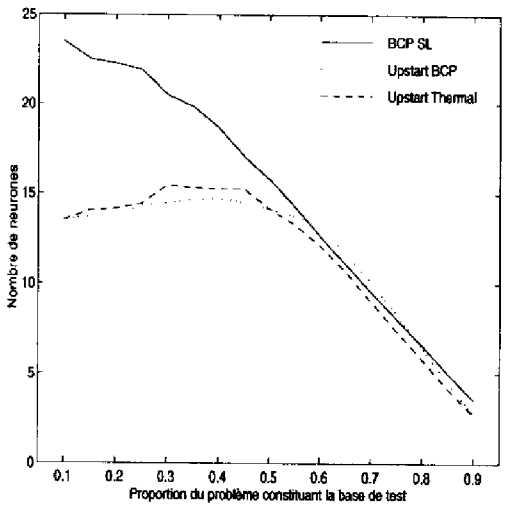

FIG. 5.25: Nombre d'unités construites pour le problème de parité en dimension 8.

Dans cette dernière simulation, le nombre d'itérations est fixé à 1000 pour les trois algorithmes. Le problème parité en dimension 8 est généré. Ce problème est séparé en deux parties d'une manière

aléatoire, une base d'apprentissage et une base de test. La proportion de ce partage pouvant aller de 10% à 90%.
La figure 5.24 représente le pourcentage de bien classés dans la base de test en fonction de la proportion de la base totale qui constitue la base de test. Comme dans les précédentes simulations, les algorithmes sont lancés sur les mêmes bases d'apprentissage et testés sur les mêmes bases de test. Les moyennes sont calculées sur 500 tirages, pour chaque proportion (17 valeurs différentes). La figure 5.25 représente le nombre d'unités construites en fonction de la proportion de partage.
On peut noter premièrement que le BCPSL n'est pas efficace sur ce problème. Dans cette simulation, le plus intéressant est que l'utilisation du BCP à la place du Thermal accroît de manière significative les capacités de généralisation des réseaux, même à nombre de neurones égal. Par exemple, pour une proportion de 50% de vecteurs d'entrée dans la base de test, l'Upstart + Thermal donne 60,7% d'exemples bien classés dans la base de test, et Upstart + BCP 67,3%, pour un nombre d'unités respectivement égal à 14 et 13,7.

5.7 Amélioration du BCPSL : l'algorithme BCPSLhd

5.7.1 BCPSL et BCPSLml : leurs limitations

Comme on l'a déjà signalé, le BCPSL est l'association de la stratégie Sequential Learning, décrite dans le chapitre 3 et du BCP Max pour apprendre les unités de la première couche. De plus les hyperplans correspondant à ces unités sont bien orientés. C'est-à-dire que si les exemples exclus sont de cible 1 (resp. 0) la sortie de l'unité correspondante est 1 (resp. 0), ou en d'autres termes les exemples exclus se trouvent dans le demi-espace positif (resp. négatif) défini par l'hyperplan. Le nouveau problème alors généré par la représentation interne de cette couche cachée est linéairement séparable. Donc il peut être appris simplement avec le BCP Heuristique, sans procédure d'optimisation d'un critère et sans limitation du nombre d'itérations. C'est cette implémentation que l'on nommera BCPSL. La convergence de cette méthode de construction est assurée par le fait que BCP Max peut toujours trouver un ensemble d'exemples exclus, en un nombre faible d'itérations. Donc un nombre fini d'unités seront construites dans la première couche cachée. Ce nombre est majoré par le nombre de vecteurs d'entrée.
Dans le cas où pour apprendre le neurone de sortie, la nouvelle base d'apprentissage est très compliquée et que le BCP Heuristique ne converge pas très vite, une stratégie identique à celle utilisée pour construire la première couche peut être appliquée pour résoudre ce problème. Ainsi une seconde couche cachée est construite avec le BCPSL. On calculera alors le nouveau problème généré par cette seconde couche cachée, qui va contenir beaucoup moins de neurones que la première. Ce nouveau problème est alors appris de manière identique. Si une solution est trouvée en un nombre d'itérations inférieur à la limite, la construction du réseau se termine, sinon elle continue avec une autre couche. Cette méthode de construction sera nommée BCPSLml (BCPSL Multi Layer). Elle peut être appliquée sur de complexes problèmes de classification.
Pour les problèmes extrêmement complexes, deux limitations majeures interviennent dans l'application de ces deux versions. En effet, si le nombre de neurones de la couche cachée m est très important, le temps de calcul du nouveau mapping peut être très long, car il faut calculer toutes les représentations internes et enlever toutes les redondances de la base ainsi générée. Et l'apprentissage de cette nouvelle base peut alors être lent, même si peu d'itérations sont nécessaires. C'est pour cela que le BCPSLml a été développé. Mais un autre problème très important limite aussi l'application de ces deux méthodes, c'est la très grande occupation mémoire. En effet, si m est important, alors le nombre d'exemples de la nouvelle base sera important, ainsi cette base sera constituée de beaucoup de vecteurs en très grande dimension. Même si ceux-ci sont des vecteurs booléens, l'occupation mémoire est très grande. Lors de

simulations sur des problèmes très complexes, cette limite a été atteinte.
Afin de résoudre ces problèmes, de temps de calcul et d'occupation mémoire, une version spéciale du BCPSL a été développée, par un procédé original de construction d'une seconde couche cachée d'au plus $m/2$ neurones de manière directe, et le calcul des connexions liant cette couche au neurone de sortie est trivial. Le calcul est instantané et l'occupation mémoire très faible. Il n'est pas nécessaire de calculer les représentations internes. Cette méthode, basée sur le théorème de stratification de l'hypercube présenté dans le chapitre 4, a uniquement besoin de la cible des groupes d'exemples successivement exclus lors de la construction de la première couche.

5.7.2 La méthode

On notera ici la base d'apprentissage sous la forme de N paires (ξ^μ, o^μ) où ξ^μ est le vecteur d'entrée et $o^\mu \in \{0,1\}$ sa cible. Soit m le nombre d'unités construites dans la première couche cachée. On a donc m groupes d'exemples qui ont été successivement exclus. Soit $\psi_k \in C$ avec $k \in \mathbb{N}_m$ l'ensemble des exemples du k^{th} groupe. Pour tout $k \in \mathbb{N}_m$ $t_k \in \{0,1\}$ est la cible des exemples du k^{th} groupe, et par extension on dira que c'est la cible de ce groupe. Donc

$$\forall \mu \in \{1,\ldots,N\} \;\; \xi^\mu \in \psi_k \Longrightarrow t_k = o^\mu$$

Soit S^μ la représentation interne de l'exemple ξ^μ. C'est la sortie de la première couche cachée pour cet exemple, c'est donc un vecteur de $\{0,1\}^m$. Ce vecteur peut donc être interprété comme un sommet de l'hypercube en dimension m, donc $S^\mu \in H^m$.
Du fait de la stratégie de construction, nous avons alors

$$\forall \mu \in \{1,\ldots,N\} \qquad \xi^\mu \in \psi_k \implies S^\mu = (\tilde{t_1}, \tilde{t_2}, \ldots, \tilde{t_{k-1}}, t_k, *, \ldots, *) \tag{5.28}$$

avec $*$ pouvant être 1 ou 0, cela dépend de l'exemple. Pour tout $k \in \mathbb{N}_m$, on définit alors le sous-ensemble de la base canonique

$$B_a^k = (e_{k+1}, \ldots, e_m) \subset B_c$$

le sommet

$$s_k = (\tilde{t_1}, \ldots, \tilde{t_{k-1}}, t_k) \in H^k$$

et le $(m-k)_m$-souscube

$$P_k = H_{m-k}^m(B_a^k, s_k)$$

Il est alors évident, d'après 5.28

$$\forall \mu \in \{1,\ldots,N\} \;\; \xi^\mu \in \psi_k \implies S^\mu \in P_k$$

La représentation interne de tout exemple du k^{th} groupe est donc un sommet du $(m-k)_m$-souscube P_k. En appliquant le théorème 4.4, ces sommets peuvent être séparés du reste des sommets de l'hypercube par l'hyperplan $\mathcal{H}_0'(P_k)$. Si on construit une unité sur la seconde couche cachée qui implémente cet hyperplan, sa sortie sera 1 pour tous les sommets de P_k, c'est-à-dire pour tous les exemples du k^{th} groupe, et 0 pour tous les autres sommets de l'hypercube. Si on note par w_j^k avec $j \in \{0,1,\ldots,m\}$ les poids de cette unité, avec w_0^k le seuil, leur expression est simple et donnée par le théorème 4.4. Ainsi on aura

$$w_0^k = -\tilde{t_1} - \tilde{t_2} - \ldots - \tilde{t_{k-1}} - t_k + \frac{1}{2} \tag{5.29}$$

$$\forall j \in \{1,\ldots,k-1\} \qquad w_j^k = 2\tilde{t_j} - 1 \tag{5.30}$$

$$w_k^k = 2t_k - 1 \tag{5.31}$$

$$\forall j \in \{k+1,\ldots,m\} \qquad w_j^k = 0 \tag{5.32}$$

On peut noter que ces poids sont très simples. Mis à part le seuil leur valeur est 1, -1 ou 0. En fait une unité k peut n'être connectée qu'avec les k premiers neurones de la première couche cachée. De plus dans le cas d'unités classiques pour lesquelles la sortie est 1 si l'entrée totale est supérieure ou égale à 0, le 1/2 qui figure dans l'expression du seuil peut être enlevé. Dans ce cas le seuil sera alors un entier relatif dans l'intervalle $\{-k, \ldots, +k\}$.

La construction de la seconde couche cachée va être réalisée par exclusion des souscubes de groupes de même cible. Mais deux solutions sont possibles, on exclut soit les souscubes des groupes de cible 1, soit les souscubes des groupes de cible 1. C'est ce choix qui permet d'avoir toujours un nombre d'unités construites sur cette seconde couche inférieur ou égal à $m/2$. Soit m_1 le nombre de groupes de cible 1, et m_0 de cible 0 donc $m_1 + m_0 = m$. Si $m_1 > m_0$ on va alors découper les souscubes de groupes de cible 0, et si $m_1 \leq m_0$ on va découper les souscubes de groupes de cible 1. Donc on aura toujours un nombre de souscubes découpés inférieur ou égal à $m/2$.

Enfin, la construction du neurone de sortie est trivial.

Dans le cas où l'on découpe les souscubes de groupes de cible 1, le neurone de sortie doit répondre 1 si et seulement si au moins une des unités de la seconde couche répond 1, car cela veut dire que la représentation interne de l'exemple est un sommet de l'hypercube appartenant à un souscube d'un groupe de cible 1. Ce neurone de sortie doit donc résoudre un OU logique, ce qui est réalisé très simplement en prenant toutes les connexions à 1 et le seuil à $-1/2$. En fait ce seuil peut être pris à -1 dans le cas d'une unité classique.

Dans le second cas, la sortie du neurone doit être 1 si et seulement si toutes les unités de la première couche répondent 0. Donc cette unité doit réaliser un NOR. Ceci peut être fait de manière très simple en prenant toutes les connexions à -1 et le seuil à 1/2, ou 1 pour une unité conventionnelle (c'est la négation du OU, donc les poids et le seuil sont opposés).

Ainsi avec cette méthode de découpe successive de souscubes de dimension décroissante, la seconde couche cachée contenant au plus $m/2$ unités, et le neurone de sortie sont construits par un calcul direct très simple. La valeur des connexions est aussi très simple. Cette version du BCPSL a été nommée BCPSLhd (*hypercube dichotomy*).

5.7.3 Simulations

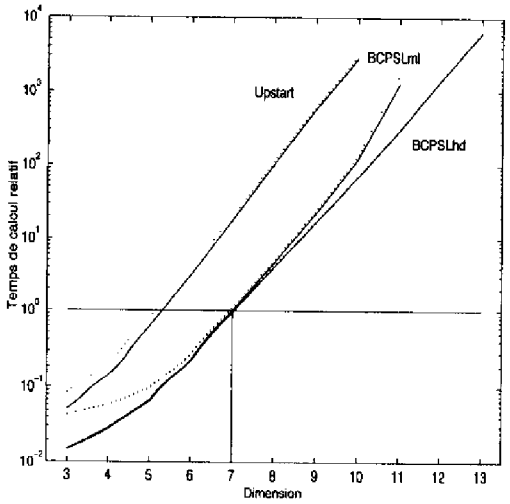

FIG. 5.26: Fonctions booléennes aléatoires

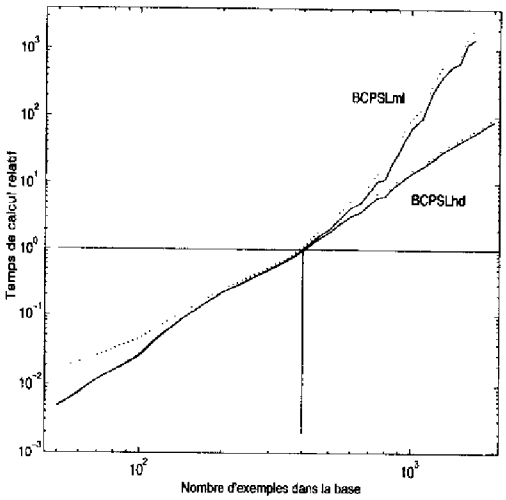

FIG. 5.27: Problème deux spirales

Les deux simulations suivantes ont pour but de comparer les temps de calcul du BCPSLml et du BCPSLhd. Les résultats sont présentés relativement à un temps de référence, qui est la moyenne du temps de calcul d'un problème de référence sur un grand nombre d'essais avec le BCPSLml. Pour les deux simulations, le nombre d'itérations t_{max} a été fixé à 80 pour la première couche, et à 200 pour les autres couches dans le cas du BCPSLml.

La première simulation a été faite sur des fonctions booléennes aléatoires en dimensions allant de 3 à 11 avec le BCPSLml et de 3 à 13 avec le BCPSLhd. Le problème de référence est défini en dimension 7. La figure 5.26 représente le temps de calcul moyen en fonction de la dimension. Le nombre d'essais réalisés est supérieur à 1000 pour les dimensions inférieures à 7 et moins pour les dimensions supérieures (décroissant jusqu'à 20 en dimension 13 avec le BCPSLhd et 11 avec le BCPSLml).

Par comparaison, des simulations ont été réalisées avec l'Upstart, utilisant le Thermal Perceptron. Le nombre d'itérations pour le Thermal Perceptron a été fixé à 600. Cette valeur a été choisie de manière à avoir en moyenne le même nombre d'unités construites pour le problème de référence traité avec le BCPSLhd. Ce nombre moyen d'unités est d'à peu près 30 pour l'Upstart et le BCPSLhd, et de 20 pour le BCPSLml. La température initiale a été prise *a priori* à 100. Elle décroît linéairement de 100 à 0 et le paramètre d'apprentissage de 1 à 0. Si pour une unité fille, le nombre d'erreurs ne diminue pas après 600 itérations, le Thermal Perceptron est relancé avec 1200 itérations, et si le nombre d'erreurs ne décroît toujours pas, un exemple est exclu avec un neurone grand-mère. Premièrement, on peut noter que les algorithmes *BCPSLml* et *BCPSLhd* sont plus rapides que l'Upstart d'un ordre de grandeur, et cela avec un nombre relativement faible d'itérations utilisé par le Thermal Perceptron. En fait pour les dimensions supérieures à 8, l'Upstart a fourni des réseaux avec beaucoup plus de neurones que les deux autres algorithmes. Le Thermal Perceptron aurait nécessité beaucoup plus d'itérations pour fournir un nombre de neurones comparable, donc il aurait été encore plus lent.

Pour les dimensions inférieures à 8, *BCPSLml* et *BCPSLhd* ont les mêmes temps de calcul, car le calcul des représentations internes n'est pas très compliqué et le temps de calcul est négligeable par rapport à la construction de la première couche. De plus l'apprentissage du neurone de sortie est très simple. Pour les grandes dimensions, une différence apparaît entre ces deux algorithmes, car le calcul des représentations internes devient complexe, ce qui ralentit le BCPSLml, alors que le BCPSLhd garde le même comportement. Pour chaque courbe, la ligne pointillée représente la valeur moyenne plus l'écart-type. Pour ce problème le temps de référence était de $t_{ref} = 0.81s$ pour une implémentation sur un ordinateur de type PC.

La seconde simulation a été réalisée sur le problème deux spirales, avec une complexité croissante. En dimension 2, la base d'apprentissage consiste en 2 ensembles de $N/2$ points dont les coordonnées sont données par l'équation paramétrique suivante

$$x_t = \varepsilon\ \rho_t \cos\phi_t \quad \text{et} \quad y_t = \varepsilon\ \rho_t \sin\phi_t \qquad \text{avec} \qquad \rho_t = \phi_t \quad \text{et} \quad \phi_{t+1} = \phi_t + \Delta\phi$$

Le paramètre $\Delta\phi$ est choisi de manière à avoir 50 points dans un demi-cercle pour chaque spirale, donc $\Delta\phi = \pi/50$. La génération des points commence avec $\phi_1 = \Delta\phi$, jusqu'à $N/2$ exemples. Les points de cible 1 sont générés avec $\varepsilon = 1$, et ceux de cible 0 avec $\varepsilon = -1$. Ces deux spirales sont symétriques par rapport au point de coordonnées $(0,0)$. Le nombre de points N varie de 100 à 2000 (seulement 1700 pour BCPSLml), donc la variation totale de l'angle $\phi_{N/2}$ est égale à 20π pour chaque spirale. Le problème de référence est défini pour $N = 400$. Le temps de calcul moyen du BCPSLml et du BCPSLhd en fonction du nombre total de points N est représenté sur la figure 5.27. 200 essais ont été réalisés pour $N \leq 1000$ et entre 50 et 10 pour $N > 1000$. Les résultats sont qualitativement similaires à ceux de la précédente simulation, et prouvent la grande efficacité du BCPSLhd. Le temps de référence est égal à $t_{ref} = 1.44s$.

5.8 MOBCPSL : un algorithme de construction multisorties

5.8.1 Principes

Le principe de cet algorithme de construction MOBCPSL (pour *Multiple Output BCPSL*) est similaire à celui du BCPSL. C'est une généralisation à plus d'un neurone de sortie, donc à plus de deux classes. Cet algorithme peut apprendre toute base d'apprentissage de $\mathbb{R}^n$ dans $\{0,1\}^p$, le codage des classes pouvant être quelconque. Il convient de remarquer qu'à notre connaissance, c'est le premier algorithme de construction aussi général (entrées booléennes ou analogiques quelconques et sorties binaires multiples) fournissant des réseaux binaires.

La construction de la première couche cachée est toujours une stratégie d'exclusion d'exemples de même classe. Chaque unité est apprise sur une base d'apprentissage particulière, avec une version modifiée du BCP Max, de manière à exclure uniquement des exemples d'une seule classe. On appellera cette version le BCP Max à cible fixe, plus précisément *BCP Max* CF_1 si la cible des exclus doit être 1, et *BCP Max* CF_0 si la cible des exclus doit être 0. La construction de cette première couche réalise donc une découpe de l'espace d'entrée en zones polyédrales, et dans chacune d'elles il n'y a que des exemples de même classe, c'est-à-dire de même vecteur de sortie.

Avec une telle construction, on démontre que l'apprentissage de chaque neurone de sortie consiste à apprendre une base LS, donc aisément résolue avec le BCP Heuristique. Cette démonstration est tout à fait similaire à celle faite par Marchand [125]. Mais c'est un cas un peu particulier ne rentrant pas dans les hypothèses du théorème de Marchand, et donc nécessitant une démonstration.

Pour les bases d'apprentissage très complexes, cet algorithme souffre des mêmes maux que le BCPSL ou le BCPSLml : temps de calcul des représentations internes très important, et occupation mémoire très grande. Une amélioration similaire au BCPSLhd est alors proposée. Son principe repose aussi sur la découpe de souscubes, et est pratiquement aussi simple.

Après avoir présenté le BCP Max à cible fixe, nous détaillerons l'algorihme MOBCPSL. Ensuite nous présenterons l'amélioration de cet algorithme, nommée MOBCPSLhd. Nous terminerons par quelques résultats de simulations, en comparant le MOBCPSL et le MOBCPSLhd a l'algorithme *Cascade Correlation*. Temps de calcul et pouvoir de généralisation seront comparés pour divers problèmes de classification.

5.8.2 BCP Max à cible fixe

Dans cet algorithme, le BCP Heuristique est toujours utilisé de la même manière, c'est la procédure d'exclusion qui est modifiée, pour que l'hyperplan n'exclue que des exemples d'une cible donnée t_f. L'algorithme est le suivant.

Comme on peut le constater, cet algorithme est pratiquement le même que l'algorithme 5.3, avec quelques tests supplémentaires au début pour éviter les calculs superflus, et l'exclusion d'exemples uniquement si la cible correspond à la cible t_f. Lorsqu'il y a la possibilité, la marge est aussi optimisée.

Quelle que soit la cible, il y a toujours 1 cas sur les 4 cas possibles de positionnement des extrêma qui rend impossible l'exclusion d'exemples d'une cible donnée t_f. Il est clair que le BCP Max à cible fixe peut ne pas pouvoir exclure d'exemples, même avec un grand nombre d'itérations. En effet, si on considère la base d'apprentissage constituée d'un point de cible 1 au centre d'un cercle de points de cible 0, et que l'on veuille exclure le point de cible 1, ce sera impossible. Mais si on lance une fois BCP Max CF_0 et une fois BCP Max CF_1, on est sûr d'avoir un ensemble d'exclus, de cible 1 ou de cible 0. C'est sur ce principe que l'on peut justifier la convergence de l'algorithme que l'on va décrire maintenant.

Algorithme 5.4 Algorithme de maximisation du nombre d'exemples exclus d'une cible donnée t_f

Etape a : Calcul de ext_1, ext_2, ext_3, ext_4, t_- et t_+
Etape b : Si $t_- \neq t_f$ et $t_+ \neq t_f$ alors $Ex^t_{poc} = 0$ et **Stop**
Etape c : Si $(t_- = t_f)$ alors calcul de $\max(P_-)$, N_- et $\mathcal{G}^- = (ext_2 - \max(P_-))/||w^t||$
Etape d : Si $(t_+ = t_f)$ alors calcul de $\min(P_+)$, N_+ et $\mathcal{G}^+ = (\min(P_+) - ext_3)/||w^t||$
Etape e : Si [$t_- = t_f$] et [$(N_- > N_+)$ ou $(N_- = N_+ > 0$ et $\mathcal{G}^- \geq \mathcal{G}^+)$] alors
$\theta^t_{poc} = (ext_2 + \max(P_-))/2$, $\mathcal{G}^t_{poc} = \mathcal{G}^-$ et $Ex^t_{poc} = N_-$
Si $(t_- = 1)$ $\varepsilon^t = 1$
Si $(t_- = 0)$ $\varepsilon^t = -1$ et $\theta^t_{poc} \longleftarrow -\theta^t_{poc}$
Etape f : Si [$t_+ = t_f$] et [$(N_- > N_+)$ ou $(N_- = N_+ > 0$ et $\mathcal{G}^- < \mathcal{G}^+)$] alors
$\theta^t_{poc} = (\min(P_+) + ext_3)/2$, $\mathcal{G}^t_{poc} = \mathcal{G}^+$ et $Ex^t_{poc} = N_+$
Si $(t_+ = 0)$ $\varepsilon^t = 1$
Si $(t_+ = 1)$ $\varepsilon^t = -1$ et $\theta^t_{poc} \longleftarrow -\theta^t_{poc}$

5.8.3 L'algorithme

Construction de la première couche

La base d'apprentissage est toujours un ensemble de N paires (ξ^μ, o^μ) où $\xi^\mu \in \mathbb{R}^n$ est le vecteur d'entrée, mais o^μ est sa classe, c'est-à-dire un vecteur de $\{0,1\}^p$, correspondant au vecteur des cibles de cet exemple pour les p sorties. L'ensemble des N exemples est toujours noté C.
On va définir d'abord l'ensemble des classes Cl par :

$$Cl = \{o^\mu / \mu \in \{1, \ldots, N\}\}$$

On a alors Card $(Cl) = q$, le nombre de classes différentes généralement très inférieur à N. Tout élément c^k de Cl avec $\forall k \in \{1, \ldots, q\}$ est dans $\{0,1\}^p$.
Pour un sous-ensemble d'exemples $C' \subset C$, l'ensemble partiel des classes est alors :

$$Cl_{(C')} = \{o^\mu / \mu \in \{1, \ldots, N\} \text{ et } \xi^\mu \in C'\}$$

On définit alors l'indicatrice des classes $\mathcal{I}^\mu_k$ par :

$$\forall k \in \{1, \ldots, q\} \quad \forall \mu \in \{1, \ldots, N\} \quad \text{alors} \quad \begin{cases} \mathcal{I}^\mu_k = 1 & \text{si } o^\mu = c^k \\ \mathcal{I}^\mu_k = 0 & \text{si } o^\mu \neq c^k \end{cases}$$

Pour un sous-ensemble d'exemples $C' \subset C$, on définit alors la base partielle relative à la classe k pour $k \in \{1, \ldots, q\}$ par

$$\mathcal{B}^k(C') = \{(\xi^\mu, \mathcal{I}^\mu_k) / \xi^\mu \in C'\}$$

C'est-à-dire qu'à tout exemple de C' on associe la cible 1 si cet exemple est de la classe c^k et 0 sinon. Ce sont ces bases d'apprentissage qui vont nous servir à apprendre les unités de la première couche.
Le principe de l'algorithme 5.5 est de partir de l'ensemble des vecteurs d'entrée d'origine, et de créer successivement des unités avec des hyperplans qui excluent un maximum d'exemples de même classe. Il est très important de noter que les exemples exclus seront toujours du côté positif de l'hyperplan qui les exclut. Pour un sous-ensemble C' de C, on va rechercher l'hyperplan qui exclut un maximum d'exemples de même classe, en tentant d'exclure successivement des exemples des classes encore présentes. Et on prendra la position qui en exclut un maximum. Pour cela, le BCP Max CF_1 est

Algorithme 5.5 MOBCPSL : Construction de la couche cachée

a. $C' = C$, $Cl_{(C')} = Cl$ et $j = 0$
b. Tant que Card $(Cl_{(C')}) \geq 2$ faire
  1. $Ex_m = 0$, $\mathcal{G}_m = 0$ et $j \longleftarrow j + 1$
  2. Boucle sur $c^k \in Cl_{(C')}$
    i. $(w, \theta, Ex, \mathcal{G}) \longleftarrow$ BCP Max CF_1 avec la base $\mathcal{B}^k(C')$
    ii. Si $(Ex > Ex_m)$ ou $(Ex = Ex_m$ et $\mathcal{G} > \mathcal{G}_m)$ alors
      $(w_m, \theta_m, Ex_m, \mathcal{G}_m) \longleftarrow (w, \theta, Ex, \mathcal{G})$ et $c_m = c^k$
  Fin Boucle $[c^k]$
  3. $\psi_j = \{x \in C'/w_m.x + \theta_m > 0\}$
  4. $C' \longleftarrow C'/\psi_j$
  5. $T_j = c_m$
  6. Création neurone j avec w_m et θ_m
  7. Mise à jour $Cl_{(C')}$
  Fin Tant que
d. $T_{j+1} = \{c^k/\{c^k\} = Cl_{(C')}\}$
e. $\psi_{j+1} = C'$

utilisé avec $\mathcal{B}^k(C')$ comme base d'apprentissage, pour toutes les classes c^k encore présentes dans le sous-ensemble C'. Dans la mesure du possible, la marge sera encore optimisée.

Le sous-ensemble C' est le sous-ensemble courant d'exemples non encore exclus, et $Cl_{(C')}$ est l'ensemble des classes de ce sous-ensemble d'exemples, l'ensemble des classes présentes. Le compteur j est le nombre courant d'unités. L'initialisation est évidente. Le processus de création de neurones continue jusqu'à ce que les exemples qui restent dans C' soient de même classe, donc tant que Card $(Cl_{(C')}) \geq 2$. Ensuite les étapes **b.1** et **b.2** servent à maximiser le nombre d'exclus de même classe. Pour chaque classe présente c^k, on lance le BCP Max CF_1 sur la base constituée des exemples de C' associés de la cible 1 si ce sont des exemples de la classe c^k et 0 sinon. Le résultat de cet algorithme d'exclusion est le quadruplet $(w, \theta, Ex, \mathcal{G})$, c'est-à-dire les paramètres de l'hyperplan w et θ, le nombre d'exclus et la marge. Si ce résultat est meilleur que le meilleur résultat rencontré jusque là et sauvegardé dans $(w_m, \theta_m, Ex_m, \mathcal{G}_m)$, on le conserve. De plus, on garde aussi dans c_m la classe des exclus. Une position est meilleure si le nombre d'exclus est supérieur, ou en cas d'égalité si la marge est plus grande.

Après cette boucle, l'étape **b.3** consiste à grouper dans ψ_j les exemples exclus. Ce groupe est alors enlevé de l'ensemble des exemples courants C' à l'étape **b.4**. A l'étape **b.4**, on sauvegarde la classe c_m des exclus dans une matrice (car les classes sont des vecteurs). On créé le nouveau neurone à l'étape suivante avec les paramètres w_m et θ_m, et on met à jour l'ensemble des classes actives. Les deux étapes finales servent à récupérer les informations du groupe d'exemples qu'il reste à la fin : la classe de ce groupe et les exemples eux-mêmes (en fait cette dernière étape n'est pas nécessaire, mais nous l'avons indiqué pour une meilleure compréhension).

Pour une implémentation réelle de l'algorithme, la gestion des classes actives peut se faire simplement avec un compteur par classe qui indique le nombre d'exemples de chaque classe qui sont encore dans la base. Et lorsque l'on calcule les exemples à retirer de la base, on décrémente le compteur de la classe des exclus. Si ce compteur atteint zéro, cela veut dire que cette classe n'est plus présente, et on peut décrémenter un compteur du nombre de classes présentes utilisable pour arrêter la boucle.

Apprentissage des neurones de sortie

Nous allons tout d'abord étudier les représentations internes obtenues sur la première couche cachée. Supposons que nous avons m neurones construits dans la couche cachée avec la méthode précédente. Nous avons donc exclu successivement m groupes ψ_j de classe t_j. Le dernier groupe restant ψ_{m+1} étant de classe t_{m+1}. Du fait que lors de l'exclusion, l'hyperplan est toujours positionné de manière à ce que les exclus soient du côté positif, la représentation interne des exemples est donc :

$$\forall \mu \in \{1,\ldots,N\} \quad \xi^\mu \in \psi_j \text{ avec } j \in \{1,\ldots,m\} \implies S^\mu = (0,\ldots,0,1,*,\ldots,*) \quad (5.33)$$

$$\forall \mu \in \{1,\ldots,N\} \quad \xi^\mu \in \psi_{m+1} \implies S^\mu = (0,0,\ldots,0) \quad (5.34)$$

Dans le vecteur de la relation 5.33, le 1 est à la j^{eme} position, et les $*$ peuvent être des 0 ou des 1, dépendant de la position de l'exemple.
Pour $k \in \{1,\ldots,m,m+1\}$, on définit $S_{(\psi_k)}$ l'ensemble des représentations internes différentes pour lesquelles les vecteurs d'entrée correspondants sont dans le groupe ψ_k

$$S_{(\psi_k)} = \{S^\mu / \xi^\mu \in \psi_k\}$$

L'union de ces ensembles S pour tous les groupes constituera donc les vecteurs d'entrée des bases d'apprentissage pour apprendre les neurones de sortie :

$$S = S_{(\psi_1)} \cup \ldots \cup S_{(\psi_m)} \cup S_{(\psi_{m+1})}$$

Il est évident que cette union est une union disjointe, et qu'en général nous aurons Card $(S) = M$, avec M inférieur à N.
On va démontrer alors que toute base d'apprentissage constituée des représentations internes de S auxquelles on associe une cible quelconque mais identique pour tous les éléments qui sont des représentations internes d'exemples d'un même groupe ψ_k, est LS.

Théorème 5.3 *Pour tout vecteur $u = (u_1,\ldots,u_m,u_{m+1}) \in \{0,1\}^{m+1}$, le vecteur des cibles des représentations internes des groupes, la base d'apprentissage $\mathcal{B}_u(S)$ définie par*

$$\mathcal{B}_u(S) = \{(v,u_i)/v \in S \text{ et } v \in S_{(\psi_i)}\}$$

est une base linéairement séparable.

Démonstration 5.3 *Il faut donc trouver un vecteur des poids $w = (w_1,\ldots,w_m) \in \mathbb{R}^m$ et un seuil w_0 tel que*

$$\forall i \in \{1,\ldots,m,m+1\} \quad \forall v \in S_{(\psi_i)} \quad \phi(w.v + w_o) = u_i$$

avec ϕ la fonction de Heaviside. Le vecteur w et le seuil w_0 doivent donc vérifier :

$$\forall i \in \{1,\ldots,m,m+1\} \quad \forall v \in S_{(\psi_i)} \quad \begin{cases} w.v + w_0 > 0 & \text{si} \quad u_i = 1 \\ w.v + w_0 < 0 & \text{si} \quad u_i = 0 \end{cases}$$

Les différents vecteurs v sont donnés par les expressions 5.33 et 5.34.
Pour $i = m+1$, nous devons donc avoir $\phi(w_0) = u_{m+1}$, donc si $u_{m+1} = 1$ il suffit de prendre $w_0 > 0$, par exemple $w_0 = 1$, et dans le cas $u_{m+1} = 0$, il faut avoir $w_0 < 0$, par exemple $w_0 = -1$. Donc l'expression générale de w_0 est :

$$w_0 = (2u_{m+1} - 1)\, 2^0$$

Pour $i = m$, nous devons avoir $\phi(w_m + w_0) = u_m$. Donc si $u_m = 1$, il faut que $w_m + w_0 > 0$. Ceci est vérifié quelle que soit la valeur de w_0 si $w_m > |w_0|$. On peut donc prendre $w_m = |w_0| + 1 = 2$. De

la même manière, dans le cas où $u_m = 0$, on doit avoir $w_m < |w_0|$, donc on peut prendre simplement $w_m = -(|w_0| + 1) = -2$. L'expression générale du poids w_m est donc :

$$w_m = (2u_m - 1)(|w_0| + 1) = (2u_m - 1)\,2^1$$

Pour $i = m - 1$, nous devons avoir $\phi(w_{m-1} + w_m * + w_0) = u_{m-1}$, avec $*$ qui peut être 0 ou 1. Donc dans le cas où $u_{m-1} = 1$, on peut prendre $w_{m-1} = |w_m| + |w_0| + 1 = 2|w_m|$, et si $u_{m-1} = 0$, on peut prendre $w_{m-1} = -(|w_m| + |w_0| + 1) = -2|w_m|$. L'expression générale est donc :

$$w_{m-1} = (2u_{m-1} - 1)(|w_m| + |w_0| + 1) = (2u_{m-1} - 1)2|w_m| = (2u_{m-1} - 1)\,2^2$$

On peut alors montrer par une récurrence simple que la valeur de la connexion w_i pour $i \in \{1, \ldots, m\}$ est donnée par :

$$w_i = (2u_i - 1)(|w_{i+1}| + \ldots + |w_m| + |w_0| + 1) = (2u_{m-1} - 1)2|w_{i+1}| = (2u_{m-1} - 1)\,2^{m+1-i}$$

On peut donc en conclure que quelque soit le vecteur $u = (u_1, \ldots, u_m, u_{m+1}) \in \{0,1\}^{m+1}$ la base $\mathcal{B}_u(S)$ est linéairement séparable, et une solution est donnée par

$$\begin{array}{rrcl} & w_0 & = & (2u_{m+1} - 1)\,2^0 \\ \forall i \in \{1, \ldots, m\} & w_i & = & (2u_i - 1)\,2^{m+1-i} \end{array}$$

□.

Cette démonstration est un peu différente de la démonstration de Marchand [125], car les points exclus sont toujours du côté positif de l'hyperplan qui les exclut, et la cible de ces exclus peut être quelconque. La solution proposée dans cette démonstration n'est pas utilisable, à cause de la croissance exponentielle de la valeur des connexions.
Avec ce théorème, on en déduit que l'apprentissage des neurones de sortie est l'apprentissage de bases LS. Ces bases sont alors définies à partir des vecteurs des classes et de la classe de chacun des groupes d'exclus. On rappelle que les classes des groupes successivement exclus sont rangées dans la matrice $T = (T_1, \ldots, T_m, T_{m+1}) = (t_{il})$ de dimension $p \times (m+1)$, car les éléments de T sont des classes donc des vecteurs binaires de p composantes. Ainsi pour apprendre le neurone de sortie l avec $l \in \{1, \ldots, q\}$, il faut associer à chaque représentation interne correspondant à un exemple d'un groupe de classe c, la cible 1 si la composante l de c est un 1, et 0 sinon.
Donc pour apprendre le neurone de sortie l, il suffit d'apprendre la base $\mathcal{B}_{u^l}(S)$ avec le vecteur des cibles $u^l = (u^l_1, \ldots, u^l_m, u^l_{m+1})$ défini par :

$$\forall i \in \{1, \ldots, m, m+1\} \qquad \text{alors} \quad \begin{cases} u^l_i = 1 \iff t_{li} = 1 \\ u^l_i = 0 \iff t_{li} = 0 \end{cases}$$

On peut remarquer que le vecteur u^l est simplement la $l^{ème}$ ligne de la matrice T.
Pour chaque neurone de sortie, la base d'apprentissage associée est alors apprise simplement avec le BCP Heuristique.

5.8.4 Amélioration du MOBCPSL : l'algorithme MOBCPSLhd

Le principe est le même que dans le BCPSLhd : au lieu de calculer les représentations internes sur la première couche cachée pour réaliser l'apprentissage des neurones de sortie, une seconde couche cachée est construite par calcul direct. Chaque neurone de cette seconde couche réalise l'exclusion

d'un sous-cube qui contient toutes les représentations internes des exemples contenus dans un groupe d'exclus. Comme la dimension des sous-cubes est décroissante, cette couche réalise une dichotomie de l'hypercube. Cette version est donc nommée MOBCPSLhd (hypercube dichotomy).
Mais on ne peut pas appliquer la même stratégie que dans le BCPSLhd pour minimiser le nombre de neurones sur cette seconde couche. Car dans le cas multisorties, le nombre de classes est supérieur à 2. Donc si on n'exclut que les sous-cubes contenant les représentations internes des exemples d'une seule classe, toutes les autres classes auront une sortie identique sur cette seconde couche cachée, c'est-à-dire $(0,\ldots,0)$. Il est nécessaire d'associer un neurone pour chacun des groupes d'exclus, plus un neurone pour les exemples restant après l'exclusion du dernier groupe. On va donc créer un neurone pour exclure chaque sous-cube défini par les relations 5.33 et 5.34.
Alors que la première couche cachée contient m neurones, la seconde va ainsi contenir $m+1$ neurones. On note w_j^k, avec $k \in \{1,\ldots,m,m+1\}$ et $j \in \{1,\ldots,m\}$, la valeur de la connexion liant un neurone k de la seconde couche cachée avec le neurone j de la première couche, et w_0^k le seuil.
Pour $k \in \{1,\ldots,m\}$, nous avons alors :

$$w_0^k = -1 + \frac{1}{2} \tag{5.35}$$

$$\forall j \in \{1,\ldots,k-1\} \quad w_j^k = -1 \tag{5.36}$$

$$w_k^k = 1 \tag{5.37}$$

$$\forall j \in \{k+1,\ldots,m\} \quad w_j^k = 0 \tag{5.38}$$

Pour le dernier neurone $k = m+1$, les poids sont définis par :

$$w_0^{m+1} = \frac{1}{2} \tag{5.39}$$

$$\forall j \in \{1,\ldots,m\} \quad w_j^{m+1} = -1 \tag{5.40}$$

Ces relations sont similaires aux relations de 5.29 à 5.32, mais particularisées car les exemples exclus sont toujours du côté positif de l'hyperplan, et les réprésentations internes sont particulières 5.33 et 5.34.
Si on met les poids w_j^k sous la forme d'une matrice $W = (w_{kj})$, de dimension $(m+1) \times m$, nous avons alors :

$$W = \left[\begin{array}{cccc} 1 & 0 & \ldots & 0 \\ -1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ -1 & & -1 & 1 \\ \hline -1 & \ldots & \ldots & -1 \end{array}\right]$$

Cette découpe de l'hypercube est une dichotomie complète : tout sommet est dans un des sous-cubes découpés. Donc quelle que soit la représentation interne sur la première couche cachée, il y aura un et un seul neurone de la seconde couche dont la sortie sera 1, tous les autres auront une sortie égale à 0.

La construction des neurones de sortie est alors très simple. A chaque neurone de la seconde couche cachée on peut associer la classe des exemples du groupe dont les représentations internes sont des sommets du sous-cube qui a été exclu par ce neurone. On rappelle que ces classes sont regroupées en une matrice T, de dimension $p \times (m+1)$. La sortie du neurone de sortie l doit être 1 pour un sous-ensemble de classes. Donc il suffit de faire un OU logique de la sortie de tous les neurones de la seconde couche dont la classe est dans ce sous-ensemble.

Ainsi si on note z_{jl} le poids de la connexion reliant le neurone de sortie l avec le neurone de la seconde couche cachée j, avec $l \in \{1, \ldots, p\}$ et $j \in \{1, \ldots, m, m+1\}$, nous avons alors :

$$t_{jl} = 1 \quad \Longrightarrow \quad z_{jl} = 1 \tag{5.41}$$

$$t_{jl} = 0 \quad \Longrightarrow \quad z_{jj} = 0 \tag{5.42}$$

et le seuil de ce neurone est :

$$z_{0l} = \frac{1}{2} \tag{5.43}$$

Si on note $Z = (z_{jl})$ la matrice des poids des connexions liant la seconde couche cachée aux neurones de sortie, nous avons donc $Z = T$.

Comme on peut le voir, le calcul des connexions liant la première couche cachée à la seconde, et la seconde à la couche de sortie est direct. Il faut seulement avoir les classes des divers groupes d'exclus lors de la construction de la première couche, et le codage des classes. De plus, toutes les connexions ont pour valeur 1, -1 ou 0. Dans les expressions 5.35, 5.39 et 5.43, le $1/2$ peut être remplacé par 0 dans le cas d'unités à seuil classiques dont la sortie est 1 si l'entrée est 0.
Il convient enfin de remarquer que le calcul des sorties de la seconde couche cachée avec un calcul classique en utilisant les poids que l'on vient de définir a une complexité en $O(m^2)$. Mais la fonction particulière de cette couche rend possible une diminution substancielle de ces calculs. En effet, nous verrons dans le paragraphe suivant, traitant d'une possible implantation VLSI, que ces sorties peuvent être calculées de manière récursive en $O(m)$. En conclusion, nous pouvons dire que l'ajout de cette seconde couche cachée n'a pratiquement aucun inconvénient et qu'elle a des avantages importants pour l'apprentissage des bases complexes.

5.8.5 Simulations

N'ayant pas trouvé dans la littérature un algorithme de construction de réseaux de neurones binaires pouvant traiter des problèmes de classification multi-classes analogiques, sans codage binaire ou projection sur un convexe, nous allons comparer les performances du MOBCPSL et du MOBCPSLhd à Cascade Correlation. Les deux bases testées sont des problèmes de référence assez souvent utilisés pour tester les algorithmes de classification neuronaux ou non. Ils proviennent de l'Université d'Irvirne en Californie, où est gérée une base de données très importante contenant la plupart des problèmes de référence utilisés et beaucoup d'autres [144].

La base *Glass*

Cette base de données résume une analyse chimique de 6 types de verre. Elle est constituée de 214 résultats d'analyse. Chaque résultat se compose de 9 données analogiques caractéristiques de la composition chimique et de la catégorie à laquelle le verre appartient. La répartition de ces analyses par type de verre est la suivante : 70, 76, 17, 13, 9 et 29. La première des caractéristiques chimiques est l'indice de réfraction. Les 8 autres sont les proportions de sodium, magnésium, aluminium, silicium, potassium, calcium, barium et fer.
Cette base de données a été séparée en deux : la base d'apprentissage contenant 75 exemples (35%) et la base de test 139. Les 6 classes ont été codées sur 6 neurones. Donc chaque neurone de sortie est actif pour une classe. Nous avons fait des simulations pour les deux versions du MOBCPSL avec trois nombres d'itérations différents (40, 100 et 200) et pour Cascade Correlation. Ce dernier a été utilisé avec 200 itérations pour l'apprentissage des neurones (à sigmoïde).

Algorithmes		MOBCPSL			MOBCPSLhd			CasCor
Nb itérations		40	100	200	40	100	200	200
Généralisation (% sur base de test)	$\overline{g}$	86,0	86,1	86,0	86,0	86,1	85,9	86.0
	σ_g	1,48	1,51	1,49	1,49	1,48	1,47	1,46
Tps de calcul (en seconde)	$\overline{t}$	0,71	1,42	2,55	0,71	1,43	2,57	9,96
	σ_t	0,139	0,279	0,554	0,139	0,276	0,54	1,66
Nb neurones	$\overline{n_u}$	13,3	12,1	11,6	27,3	25,2	24,2	6,74
	σ_{n_u}	2,01	1,71	1,76	3,67	3,35	3,27	1,25

TAB. 5.2: Résultats de simulations sur la base *Glass*.

Le tableau 5.2 rassemble les résultats obtenus en faisant des moyennes sur 500 tirages différents. Après apprentissage, chaque réseau est testé sur la base de test. Ce tableau contient la moyenne et l'écart-type de trois quantités : le pourcentage d'exemples bien classés dans la base de test (exactement le nombre de bits de sortie correct), le temps de calcul et le nombre de neurones cachés.
En analysant ce tableau, on peut faire quatre remarques. Premièrement les résultats en généralisation sont presque semblables pour les trois algorithmes, et parfois légèrement supérieurs pour le MOBCPSL ou le MOBCPSLhd. De plus, avec Cascade Correlation chaque base d'apprentissage n'est pas apprise parfaitement, il y a toujours quelques erreurs résiduelles. Deuxièmement, on peut remarquer qu'avec le MOBCPSL ou le MOBCPSLhd, il n'est pas nécessaire de prendre beaucoup d'itérations pour avoir une bonne généralisation. Troisièmement, le temps de calcul est très supérieur pour Cascade Correlation. Ainsi pour des pouvoirs de généralisation identiques, nous avons des accélérations d'environ 14 avec 40 itérations. Quatrièmement, le nombre de neurones utilisés par Cascade Correlation est très inférieur au nombre de neurones obtenu avec les deux autres algorithmes. Mais ces nombres sont difficilement comparables car les neurones ne sont pas du même type et ils sont beaucoup plus connectés avec Cascade Correlation. Finalement on peut remarquer que le nombre de neurones important utilisé par le MOBCPSLhd n'a aucun impact sur le pouvoir de généralisation. Par rapport au MOBCPSL, ceci est tout à fait normal car la construction de la seconde couche cachée est simplement une autre manière de calculer les fonctions booléennes que doivent réaliser les neurones de sortie.
Des simulations avec Cascade Correlation ont été réalisées avec plus et moins d'itérations mais sans amélioration significative des résultats.

La base *Vowel*

Des simulations similaires aux précédentes ont été réalisées sur la base *Vowel*. Cette base de données est constituée de signatures de 11 prononciations différentes (prononciations anglophones) de voyelles. La prononciation de 8 personnes a été enregistrée (chacune 6 fois chaque phonème). Des méthodes d'extraction de caractéristiques par traitement du signal ont été utilisées sur chaque enregistrement pour fournir 10 données numériques par enregistrement. La base de données contient donc 528 vecteurs (de 10 composantes analogiques) appartenant à l'une des 11 classes. Pour l'apprentissage des réseaux, nous avons codé les 11 classes sur quatre neurones de sortie. Comme dans l'expérience précédente, 35% de la base (185 exemples) est tirée au hasard pour constituer la base d'apprentissage, le reste sert de base de test.
Le tableau 5.3 rassemble les résultats obtenus en moyenne sur 500 tirages différents. Les résultats sont qualitativement similaires aux précédents. Le pouvoir de généralisation est légérement meilleur (1%) pour MOBCPSL et MOBCPSLhd. L'accélération est plus faible que dans la simulation précédente

Algorithmes		MOBCPSL			MOBCPSLhd			CasCor
Nb itérations		40	100	200	40	100	200	200
Généralisation (% sur base de test)	$\overline{g}$	83,8	83,5	83,3	83,8	83,2	83,0	82,6
	σ_g	1,86	1,97	1,96	2,16	1,99	2,32	1,70
Tps de calcul (en seconde)	$\overline{t}$	7,38	14,8	25,5	6,46	14,3	25,0	29,0
	σ_t	1,79	3,89	6,91	1,66	3,71	9,80	7,62
Nb neurones	$\overline{n_u}$	27,8	24,6	22,5	56,9	50,5	47,1	12,4
	σ_{n_u}	4,47	4,55	4,25	9,04	9,04	9,16	2,24

TAB. 5.3: Résultats de simulations sur la base *Vowel*.

(4,5 maximum). En revanche, la base de données devenant importante et assez complexe, le calcul des représentations internes prend du temps. Ainsi on peut observer qu'avec un nombre d'itérations identique le MOBCPSLhd est un peu plus rapide que le MOBCPSL. C'est un de ses intérêts.

5.9 Vers une implémentation VLSI

5.9.1 Implémentation des réseaux fournis par le MOBCPSLhd

Principes

Pour une implémentation VLSI, un des avantages fondamentaux de l'algorithme MOBCPSLhd par rapport au MOBCPSL est le calcul des neurones de sortie qui peut s'exprimer sous la forme d'expressions logiques. De plus, la seconde couche cachée est toujours la même quelque soit le réseau construit. Ainsi nous allons voir que les difficultés d'intégration d'un tel réseau résident dans l'intégration de la première couche cachée. La solution que nous proposons est basée sur le travail de M. Amine Bermak [20, 21] pour l'intégration de la première couche cachée sur une architecture systolique. Les avantages d'une telle solution résident dans un bon rapport vitesse - surface. La réalisation de la seconde couche cachée est extrêmement simple et nécessite peu de composants. De plus, la solution proposée pour le calcul des neurones de sortie est aussi relativement simple. La figure 5.28 présente le schéma général de l'intégration du réseau.

Architecture systolique

Une architecture systolique est agencée en forme de réseau composé d'un grand nombre de processeurs élémentaires identiques et localement connectés. Chacun d'eux reçoit des informations provenant des cellules voisines, effectue un calcul simple puis transmet les résultats à d'autres cellules voisines un temps de cycle plus tard. Les caractéristiques principales des architectures systoliques se résument par les trois propriétés suivantes : communications locales, simplicité et régularité des unités de calcul et parallélisme important qui permet des calculs intensifs.
Le système adopté pour intégrer la première couche est principalement constitué d'une unité arithmétique et d'une unité de contrôle. L'unité arithmétique est une architecture systolique rectangulaire où chaque ligne va implanter la fonction d'un neurone.
L'architecture systolique est représentée sur la figure 5.29.
Chaque processeur élémentaire reçoit les données x_{in} de la cellule du dessus, les signaux sh_{in} et S_{in} de la cellule de gauche et fournit les résultats sh_{out} et S_{out} à la cellule de droite et x_{out} à la cellule

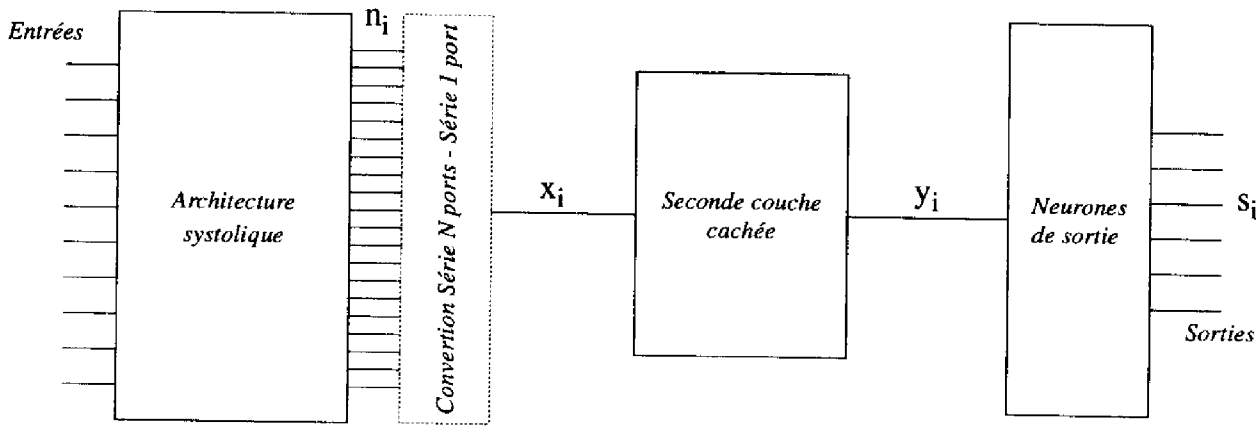

FIG. 5.28: Schéma général

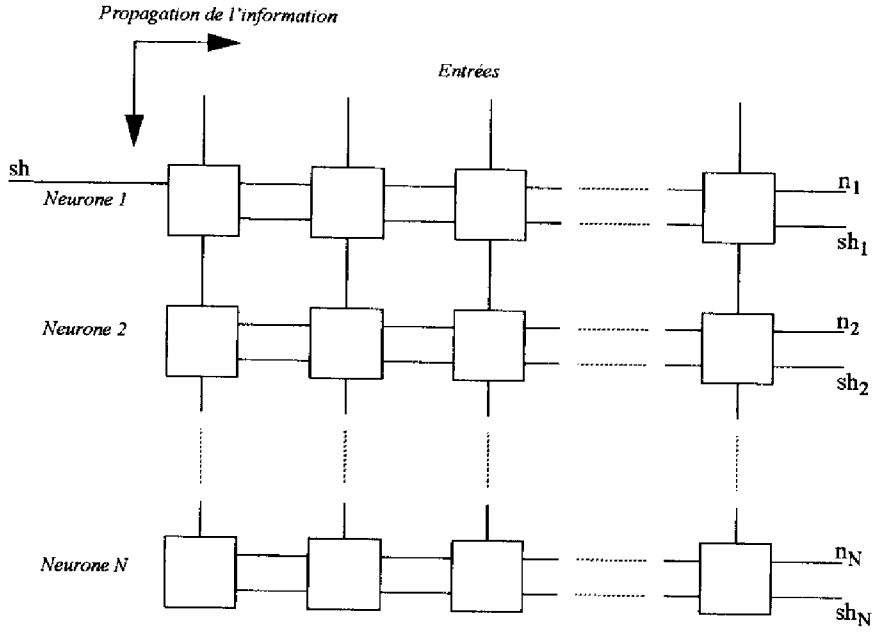

FIG. 5.29: Architecture systolique

du dessous. Les calculs réalisés par chaque processeur élémentaire sont les mêmes, donnés par les équations :

$$x_{out} = x_{in} \qquad sh_{out} = sh_{in} \quad \text{et} \quad S_{out} = w_{ij}\, x_{in} + S_{in}$$

Chaque processeur est donc muni d'un multiplieur, d'un additionneur et d'une mémoire locale où est stockée une partie des connexions du réseau. Ces processeurs travaillent au niveau bit. Nous n'entrerons pas dans les détails du fonctionnement [20, 21] de cette architecture, qui nécessite une étude détaillée du séquencement des processeurs et des divers flux de données.
En final, les entrées totales des neurones sont générées en sortie, du bit de poids faible au bit de poids fort. En complément à deux, le dernier bit généré est le bit de signe. Le signal *sh* qui est un signal de contrôle généré par l'unité de contrôle est transmis latéralement dans l'architecture et indique, quand il arrive en sortie d'une ligne, que la ligne a fini de calculer l'entrée totale du neurone correspondant. Le bit disponible en sortie à cet instant est donc le bit de signe. Un cycle d'horloge suivant, c'est le neurone suivant qui a fini le calcul et qui délivre le bit de signe de son entrée totale, et ainsi de suite jusqu'au dernier neurone.
La précision des calculs est réglable et est contrôlée par l'unité de contrôle qui intègre entres autre un registre de précision. Le séquencement de l'architecture est contrôlé par le *sh*.

Seconde couche cachée

On note x_i la sortie des neurones de la première couche cachée. On suppose que le nombre de neurones construits (par l'algorithme) sur cette couche est $N-1$. Le nombre de neurones sur la seconde couche cachée est donc N. Leur sortie sera notée y_i. La sortie y_i du neurone i doit être égale à 1 si et seulement si la représentation interne sur la première couche est $S^\mu = (0, \ldots, 0, 1, *, \ldots, *)$, avec le 1 en i^{eme} position pour tout i dans $\{1, \ldots, N-1\}$ (équation 5.34). Pour le dernier neurone d'indice N, sa sortie doit être à 1 pour la représentation interne $S^\mu = (0, \ldots, 0)$ (équation 5.34). On peut alors écrire l'expression logique des y_i en fonction des x_i. Si on note le OU logique additivement et le ET multiplicativement, nous avons :

$$\begin{aligned}
y_1 &= x_1 \\
y_2 &= \overline{x}_1 \,.\, x_2 \\
y_3 &= \overline{x}_1 \,.\, \overline{x}_2 \,.\, x_3 \\
y_4 &= \overline{x}_1 \,.\, \overline{x}_2 \,.\, \overline{x}_3 \,.\, x_4 \\
&\vdots \\
y_{N-1} &= \overline{x}_1 \,.\, \overline{x}_2 \,.\, \overline{x}_3 \,.\, \ldots \,.\, \overline{x}_{N-2} \,.\, x_{N-1} \\
y_N &= \overline{x}_1 \,.\, \overline{x}_2 \,.\, \overline{x}_3 \,.\, \ldots \,.\, \overline{x}_{N-2} \,.\, \overline{x}_{N-1}
\end{aligned}$$

Afin d'éviter un cas particulier dans le calcul de y_N, il faut ajouter un neurone d'indice N sur la première couche cachée avec une sortie toujours à 1. Dans ce cas nous aurons :

$$y_N = \overline{x}_1 \,.\, \overline{x}_2 \,.\, \overline{x}_3 \,.\, \ldots \,.\, \overline{x}_{N-2} \,.\, \overline{x}_{N-1} \,.\, x_N$$

Ceci peut se faire simplement au niveau de l'architecture systolique en ajoutant un neurone dont la sortie est toujours positive. On pourrait aussi connecter directement n_N à la masse sur le schéma 5.30 mais il est nécessaire de générer un signal sh_N artificiel en prenant le signal sh_{N-1} et en le retardant d'un cycle d'horloge avec une bascule. On supposera que ce problème est résolu d'une de ces deux manières et que les signaux x_N et sh_N sont générés. Ce dernier est important car il signalera en sortie que le réseau a fini de calculer.

Les dernières équations logiques peuvent ainsi être réécrites de manière récursive. Si on définit la suite w_i avec $i \in \{1, \ldots, N\}$ par :

$$\begin{aligned} w_1 &= 1 \\ \forall i \in \{2, \ldots, N\} \quad w_i &= \overline{x}_1 . \overline{x}_2 . \overline{x}_3 . \ldots . \overline{x}_{i-1} \end{aligned}$$

alors on peut la mettre sous la forme récursive suivante :

$$\begin{aligned} w_1 &= 1 \\ \forall i \in \{2, \ldots, N\} \quad w_i &= w_{i-1} . \overline{x}_{i-1} \end{aligned} \tag{5.44}$$

Dans ces conditions, le calcul des y_i est très simple :

$$\forall i \in \{1, \ldots, N\} \quad y_i = w_i . x_i \tag{5.45}$$

L'implémentation de ce calcul est proposée sur la figure 5.30.

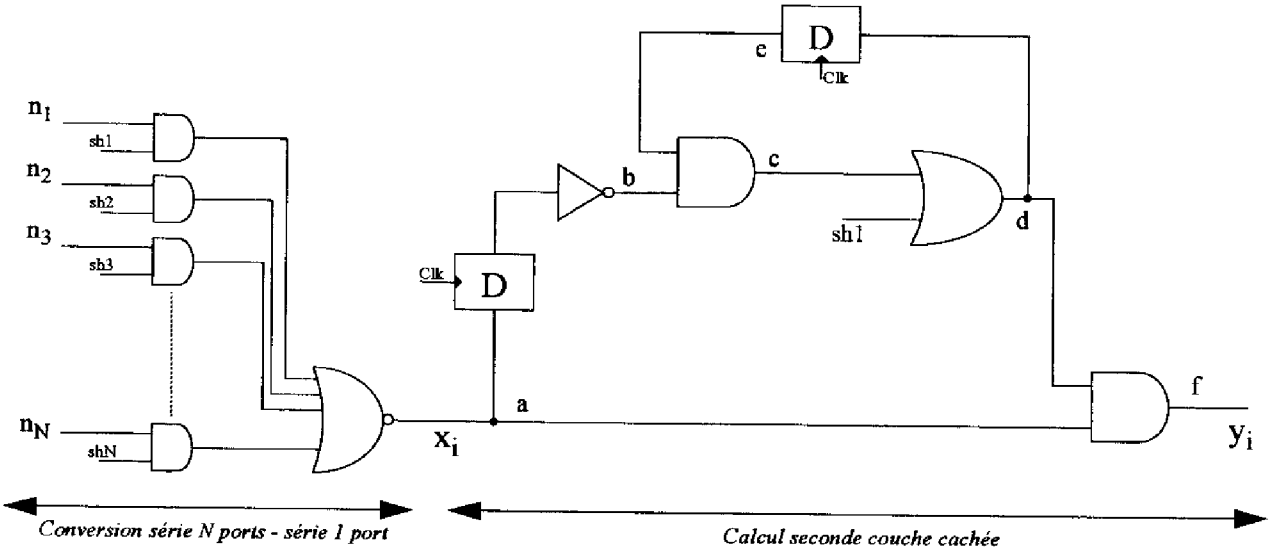

FIG. 5.30: Schéma électronique pour le calcul seconde couche cachée

La partie gauche du schéma sert à convertir les sorties *série N-ports* de l'architecture systolique en *série 1-port*. Pour cela on fait un ET entre la sortie n_i et le signal sh_i qui indique quand le neurone correspondant vient de terminer le calcul. Lors d'un front montant de sh_i, n_i aura le bit de signe de l'entrée totale du neurone i. Au cycle d'horloge suivant, ce sera le neurone $i+1$ qui donnera son bit de signe. Donc pour avoir tous ces signaux en série il suffit de faire un NOR sur ces signaux, car un 1 en complément à 2 signifie que le résultat est négatif. Il faut donc inverser pour avoir 0 dans le cas négatif et 1 dans le cas positif. Donc au point a du schéma, nous aurons successivement (en N cycles d'horloge) toutes les sorties des neurones de la première couche cachée dès que le signal sh_1 aura été activé, avec la sortie x_N du dernier neurone toujours à 1.
La partie droite du schéma permet de réaliser le calcul des neurones de la seconde couche cachée de manière récursive par implémentation des relations 5.44 et 5.45. Ce schéma est composé de deux parties. La partie haute est le calcul récursif de w_i et la partie basse le calcul de y_i. La porte OU de sortie d sert à l'initialisation de la relation de récurrence. Le calcul commence dès que le premier

neurone de l'architecture systolique donne le bit de signe de son résultat, donc sur un front de sh_1. Le dernier résultat est donné N cycles d'horloge après, donc sur le front de sh_N. Le fonctionnement de ce système est décrit dans le tableau d'états 5.9.1, où les temps t_i correspondent aux fronts montants des sh_i.

Neurones de sortie

Le calcul des neurones de sortie est simple : il suffit de faire un OU inclusif de la sortie d'un sous-ensemble de neurones de la seconde couche cachée. Plus précisément, la sortie d'un neurone de sortie d'indice j avec $j \in \{1, \ldots, p\}$ est un OU de la sortie des neurones d'indice k de la seconde couche, avec $k \in \{1, \ldots, N\}$ tels que $t_{jk} = 1$. Donc d'une manière générale on peut écrire la sortie s_j avec la fonction logique :

$$s_j = t_{j1}.y_1 + t_{j2}.y_2 + \ldots + t_{jN}.y_N \tag{5.46}$$

Or ce calcul peut aussi se faire de manière récursive par associativité du OU. Ce qui est tout à fait compatible avec l'arrivée des données en série du module précédent. Sur le schéma de la figure 5.31, chaque module calcule de manière récursive un neurone de sortie, sur un procédé similaire à celui du schéma 5.30. Quand l'architecture systolique délivre le bit de signe de son dernier neurone, la seconde couche délivre la sortie du dernier neurone et tous les modules du schéma de la figure 5.31 délivrent la sortie globale du réseau.
Le stockage de la matrice T qui est donc une matrice de dimension $p \times N$ peut se faire de manière interne ou externe. Un stockage interne peut être réalisé par p registres à décalage de dimension N comme dans le stockage des poids synaptiques des architectures proposées dans [79] et [126]. Si cette solution est trop gourmande en surface, un stockage externe sera préférable. Dans ce cas il faut fournir à ce système chaque colonne de la matrice T à chaque cycle d'horloge à partir du signal sur sh_1. Il faut aussi avoir autant de ports d'entrée pour la matrice T que de ports de sortie pour les neurones de sortie du réseau en parallèle. Mais dans les problèmes de classification multi-classes, le nombre de classes est rarement très important et comme l'algorithme MOBCPSLhd permet de coder les classes comme on le désire, si on a p neurones de sortie on peut faire une classification en 2^p classes. Donc avec peu de neurones on peut très rapidement coder beaucoup de classes. Donc le nombre de ports qu'il sera nécessaire d'ajouter ne sera pas très important. Dans le cas du stockage interne, le chargement de la matrice T pourra s'effectuer avec les ports de sortie du réseau. Il faut ajouter un aiguillage qui permette d'utiliser ces ports comme des entrées des registres à décalage ou comme les sorties des neurones lors du calcul du réseau.

Temps	a	b	c	d	e	f
t_1	x_1	?	?	1	?	x_1
t_2	x_2	$\overline{x}_1$	$\overline{x}_1$	$\overline{x}_1$	1	$\overline{x}_1.x_2$
t_3	x_3	$\overline{x}_2$	$\overline{x}_1.\overline{x}_2$	$\overline{x}_1.\overline{x}_2$	$\overline{x}_1$	$\overline{x}_1.\overline{x}_2.x_3$
$\vdots$			$\vdots$			
t_i	x_i	$\overline{x}_{i-1}$	w_i	w_i	w_{i-1}	y_i
$\vdots$			$\vdots$			
t_N	x_N	$\overline{x}_{N-1}$	w_N	w_N	w_{N-1}	y_N

TAB. 5.4: Détails des états pour le calcul de la seconde couche

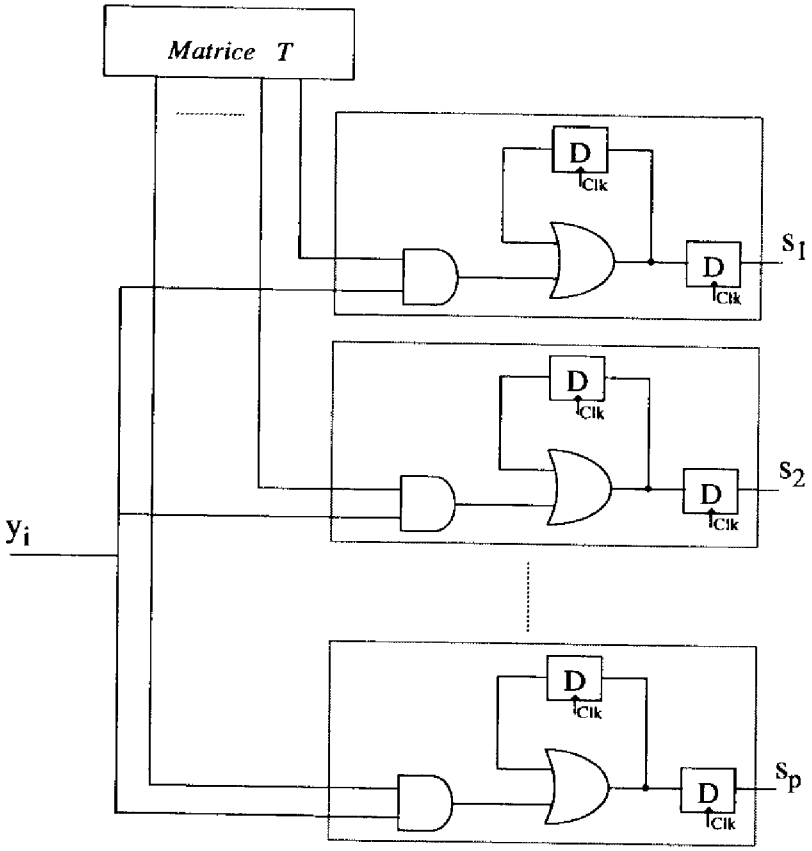

FIG. 5.31: Calcul des sorties

5.9.2 Implémentation de l'algorithme d'apprentissage

L'implémentation de l'algorithme d'apprentissage est extrêmement compliquée, voire impossible sans construire un véritable micro-processeur. Les calculs à réaliser sont complexes et suivent un algorithme avec des phases très différentes les unes des autres. La solution est d'utiliser un micro-contrôleur ou un micro-processeur standard sur une carte neuronale qui pourrait venir se greffer sur un PC. Si le nombre de classes est important, une parallélisation sur une machine MIMD (réseau de Transputers par exemple) peut être réalisée au niveau de la construction de la première couche. Pour la construction de chaque neurone, on doit tester l'exclusion d'exemples de chacune des classes. Mais ces diverses tentatives d'exclusion peuvent être effectuées en parallèle. C'est *a priori* la seule possibilité de parallélisme, car la construction de chaque neurone se fait séquentiellement, et le BCP Max CF_1 est aussi une procédure séquentielle.

5.9.3 Conclusion sur l'implémentation

L'algorithme MOBCPSLhd fournit donc des réseaux avec plus de neurones que le MOBCPSL. Mais ils sont plus facilement intégrables. Mise à part la première couche cachée qui est une couche de neurones classiques qui nécessitent le calcul de l'entrée totale avec des multiplieurs et des additionneurs, la seconde couche cachée et les neurones de sortie sont très facilement implémentables. Cette facilité est dûe au fait que la sortie de ces neurones peut s'exprimer sous forme logique avec des expressions

simples. De plus, ces expressions peuvent se mettre sous une forme récursive donc très adaptée à un arrivage de la sortie des neurones de la première couche en série. L'architecture systolique est donc bien adaptée pour le calcul de cette première couche. Avec une telle association, il n'y a aucune perte de temps dans le calcul de la seconde couche cachée et des neurones de sortie. Sachant que le nombre de classes n'est pas très important, le nombre de neurones de sortie est relativement faible donc la surface nécessaire à l'implémentation de la seconde couche et de la couche de sortie est très réduite.
D'autres architectures ont été étudiées, notamment le calcul de la seconde couche cachée par une architecture systolique linéaire qui s'adapte parfaitement à l'architecture systolique matricielle qui calcule la première couche. Les processeurs élémentaires sont très simples. Mais cette solution nécessite globalement beaucoup plus de portes que celle retenue. Pour la couche de sortie une seconde solution a aussi été envisagée. Elle consiste à attendre que tous les neurones de la seconde couche aient calculé leur sortie et à calculer en parallèle toutes les sorties. Cette solution nécessite seulement des portes OU à la place des petits modules du schéma de la figure 5.31, mais la complexité du câblage est plus importante et le nombre de portes doit aussi être plus grand. Pour chaque neurone de sortie la porte OU doit avoir en entrée les réponses de N portes ET, où chacune a en entrée la sortie d'un neurone de la couche précédente et le bit de la matrice T correspondants à ce dernier neurone et au neurone de sortie calculé. Pour chaque neurone c'est un calcul direct de la relation 5.46. Toutes les cellules de la matrice T doivent donc être connectées aux différentes portes ET, et tous les neurones de la couche précédente aux divers neurones de sortie.
L'algorithme d'apprentissage est en revanche beaucoup trop complexe pour être intégré sans construire un coeur de micro-processeur, donc autant utiliser des produits déjà existant.
Les solutions proposées pour l'intégration des réseaux ne constituent bien sûr qu'une pré-étude, qui demande d'être approfondie et d'être testée sur simulateur électronique.
Finalement, une remarque peut être faite sur l'implémentation des réseaux pour le cas d'un problème de classification en 2 classes. Pour ce type de problème, il est préférable d'utiliser le BCP Max pour apprendre une unité, et d'inverser l'orientation de l'hyperplan si c'est nécessaire, de manière à avoir toujours les exclus du côté positif.

5.10 Conclusion

Dans ce chapitre, nous avons présenté une nouvelle méthode d'apprentissage d'une unité à seuil pour la construction de réseaux de neurones binaires. Les algorithmes de construction de tels réseaux sont importants, car ces réseaux sont les plus faciles à intégrer sur silicium.
Plusieurs versions ont été dérivées de l'algorithme général, ainsi que plusieurs algorithmes de construction basés sur la stratégie Sequential Learning de Marchand [125]. L'ensemble de ces algorithmes est représenté sur la figure 5.32.
Pour l'apprentissage d'une unité binaire, l'avantage primordial du BCP Prouvé est la convergence vers le connecteur minimal. Donc dans le cas LS, la solution trouvée est la plus robuste, et dans le cas NLS le vecteur des poids converge vers zéro. Donc cette propriété peut être utilisée pour détecter la non-séparabilité. Pour trouver une solution dans le cas LS, le BCP Heuristique est beaucoup plus efficace.
Pour pouvoir être utilisé dans des algorithmes de construction, il faut que dans le cas NLS, l'algorithme donne une solution qui optimise un critère, qui peut être soit la minimisation du nombre d'erreurs, soit la maximisation du nombre d'exclus. Donc le BCP Heuristique a été muni de deux extensions, chacune dédiée à un de ces deux critères à optimiser.
Le BCP Min, association du BCP Heuristique avec la procédure de minimisation du nombre d'erreurs, est utilisable par tout algorithme de construction qui utilise un algorithme similaire, c'est-à-dire le Pocket, le Ratchet, le Thermal... Les simulations présentées dans ce chapitre montrent la grande

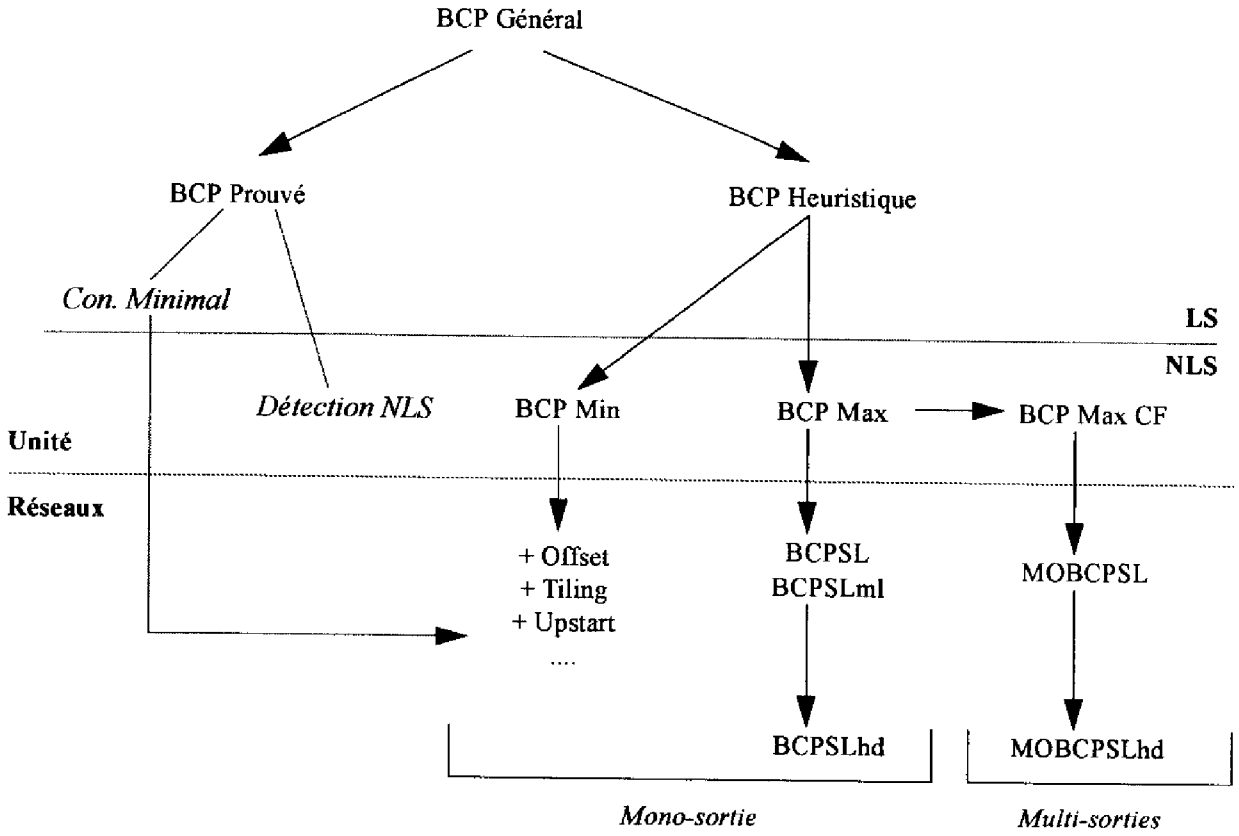

FIG. 5.32: Le grande famille du BCP

efficacité du BCP Min, comparativement à ses concurrents. Il donne généralement les meilleurs résultats sans problème de réglage de paramètres, ni problème d'initialisation. D'un point de vue temps de calcul, le BCP Min est un peu plus lent que le Thermal, mais pour des bases très NLS avec beaucoup d'exemples. Pour des bases LS ou presque, les temps de calcul sont comparables. Il est aussi beaucoup plus rapide que le Ratchet. De plus, dans le cas où la solution optimale est sans intérêt (hyperplan extérieur), le BCP Min donne une solution sous-optimale, contrairement aux autres algorithmes. Ce qui permet de ne pas bloquer les algorithmes de construction. De plus, le procédé d'optimisation de la marge quand c'est possible, à un double niveau, fournit un bon placement des hyperplans, et ainsi un bon pouvoir de généralisation aux réseaux construits. Cet algorithme, en association avec l'Upstart, a montré de multiples avantages par rapport aux autres algorithmes.

Le BCP Max, association du BCP Heuristique avec la procédure d'exclusion, est dédié à la stratégie de construction Sequential Learning. C'est *a priori* le premier algorithme de ce genre, mise à part la solution d'explorer de multiples combinaisons d'exemples et de les tester avec le Perceptron comme cela a été proposé par Marchand, ce qui est très long et non utilisable dès que les bases d'apprentissage contiennent beaucoup de vecteurs d'entrée (même si en théorie la solution optimale peut être trouvée). L'association du BCP Max et de la stratégie Sequential Learning a fourni deux algorithmes de construction très performants : le BCPSL et le BCPSLml. Ils peuvent apprendre des problèmes complexes en peu de temps : le BCP Min est très rapide, et du fait de la stratégie d'exclusion d'exemples de la base d'apprentissage active, au fur et à mesure de la construction de la première couche cachée,

c'est de plus en plus rapide. Ils peuvent aussi traiter toutes sortes de bases d'apprentissage : booléennes ou analogiques, sans avoir besoin de projection sur un convexe ou de codage binaire dans le cas analogique, ce qui est le cas de la majorité des autres algorithmes. Pour les bases d'apprentissage très complexes, le problème de temps de calcul et d'occupation mémoire pour générer les représentations internes et la base d'apprentissage du neurone de sortie, a été résolu par la construction d'une seconde couche cachée, dont le calcul direct est basé sur la structuration des sommets de l'hypercube. L'algorithme BCPSLhd a été testé sur des problèmes très compliqués, et a fourni des réseaux rapidement, possédant aussi de bons pouvoirs de généralisation, dûs à l'optimisation de la marge.
La dernière évolution de l'association du BCP Max et de la stratégie Sequential Learning a été la conception d'un algorithme de construction multi-sorties, le MOBCPSL. Pour cela une version particulière du BCP Max a été dérivée : le BCP Max à cible fixe. Toujours basée sur la découpe de sous-cubes d'un hypercube, une amélioration identique au BCPSLhd a été développée, et a fourni le MOBCPSLhd. Ce dernier algorithme ainsi que le MOBCPSL, sont *a priori* les premiers algorithmes de construction de réseaux binaires aussi généraux : entrées quelconques et sorties binaires multiples, avec les classes codées de manière quelconque. N'ayant pas de concurrent fournissant des réseaux binaires, cet algorithme a été comparé favorablement à Cascade Correlation, d'un point de vue temps de calcul et pouvoir de généralisation, sur divers problèmes de classification multi-classes.

Le pouvoir de généralisation d'un réseau de neurones binaires peut encore être amélioré grâce au BCP Prouvé. Après l'apprentissage d'une unité, la position optimale de l'hyperplan peut être trouvée, en conservant la partition réalisée par cette unité (c'est-à-dire qu'aucun exemple ne change de côté). Il suffit d'utiliser le BCP Prouvé pour rechercher le connecteur minimal entre les exemples situés du côté positif et les exemples situés du côté négatif. Même si la marge est optimisée pendant les divers algorithmes, sa valeur dépend tout de même des diverses positions prises par l'hyperplan, donc elle n'est en général pas optimale. Cette possibilité n'a pas été utilisée dans les simulations présentées dans ce chapitre.

On peut conclure sur une voie de recherche possible afin d'accroître encore les performances en généralisation pour des bases de données bruitées. Une caractéristique des algorithmes de construction de réseaux binaires, est l'apprentissage parfait des bases. Donc dans le cas de bases d'apprentissage bruitées, le pouvoir de généralisation peut être diminué par un surapprentissage du bruit. Toujours dans le cadre de la stratégie Sequential Learning, une solution envisageable à ce problème est d'utiliser en premier lieu la stratégie d'exclusion, et à la fin une stratégie de minimisation du nombre d'erreurs. Cette union est d'ailleurs assez naturelle : on classe d'abord les zones où les points de même classe sont regroupés, et ensuite dans la zone de chevauchement des classes, on essaye de placer le mieux possible des unités, en évitant le surapprentissage, et en acceptant certaines erreurs qui si on les évitait contraindraient trop l'algorithme.

Il convient enfin de faire une remarque sur le pouvoir de généralisation, et plus particulièrement sur l'idée générale que *plus le nombre de neurones est faible, tout en réalisant la tâche de manière satisfaisante, plus le pouvoir de généralisation est important.* D'une manière générale, nous n'allons pas contredire ce fait établi et largement accepté dans la communauté connexioniste. Mais nous pensons que l'on peut y apporter quelques nuances. Ainsi, les algorithmes que nous avons développés montrent qu'un bon pouvoir de généralisation dépend aussi de deux facteurs : le positionnement des hyperplans (dans le cas de neurones binaires) et le rôle des neurones en plus de celui d'offrir plus de degrés de liberté aux algorithmes. En effet, le bon pouvoir de généralisation obtenu par les algorithmes MOBCPSL et MOBCPSLhd par rapport à Cascade Correlation est certainement dû à un bon placement des hyperplans. Les expériences menées en comparaison avec le Thermal Perceptron et le Ratchet l'ont

aussi prouvé. D'autre part, le fait de doubler le nombre de neurones cachés dans les réseaux fournis par le MOBCPSLhd n'altère absolument pas le pouvoir de généralisation, car ces neurones sont ajoutés dans un but particulier et constituent une implémentation différente d'une fonction qui pourrait être réalisée avec les neurones de sortie uniquement. D'un point de vue plus théorique, on peut comprendre cette nuance grâce à la théorie de la décision bayésienne présentée dans le chapitre 1. En effet, si pour un problème de classification la frontière bayésienne entre les classes est courbe et que l'on utilise des neurones binaires pour l'approcher, c'est-à-dire des hyperplans, même en les plaçant de manière optimale il en faudra nécessairement une infinité pour atteindre l'optimal bayésien.

Thèse : *La loi ultime du monde est le hasard et tout déterminisme partiel qu'on peut y trouver est un effet de la loi des grands nombres.*
Anti-thèse : *La loi ultime du monde est entièrement déterministe et tout phénomène aléatoire qu'on peut y observer est un effet du chaos déterministe.*

JACQUES HARTHONG, *Colloque de Cerisy*

Chapitre 6

Retour au diagnostic : la méthodologie STANDING

STAtistiques et Neurones artificiels pour un système de DIagNostic Générique

6.1 Introduction

Comme nous l'avons expliqué dans le chapitre 2, après essais sur véhicule, le premier système a montré la pertinence de certaines méthodes employées et la limitation d'autres méthodes. Le gestionnaire de hiérarchie, basé sur une approche combinatoire, a montré quelques limitations dans le diagnostic de certaines fonctions. L'utilisation des réseaux de neurones a prouvé les avantages du diagnostic multidimensionnel, mais a soulevé d'importants problèmes théoriques, dont les deux principaux sont : le choix des variables d'entrée des réseaux de neurones (sur ce premier système ce choix se faisait intuitivement, sans méthodologie) et la génération des bases d'apprentissage (elles étaient générées manuellement, à la souris avec un outil graphique, et les réseaux étaient limités à 2 dimensions).
Toute la pertinence de la réponse d'un réseau de neurones dépend de la base d'apprentissage. Pour avoir une réponse fiable, il faut une base d'apprentissage précise, c'est-à-dire que la frontière entre les points de bon fonctionnement et les points de mauvais fonctionnement doit être bien définie. Le gros problème pour les voitures comme pour une multitude d'autres systèmes in-situ, est que l'on peut toujours acquérir des millions et des millions d'échantillons de points de bon fonctionnement, et ainsi couvrir assez bien la région de bon fonctionnement, mais il est impossible de couvrir correctement la région de mauvais fonctionnement, de manière à englober la région de bon fonctionnement, pour que les frontières soient bien définies. En général les points de mauvais fonctionnement que l'on peut obtenir en injectant des pannes artificielles sont très peu nombreux, et souvent triviaux. Il faut arriver à trouver une méthode pour définir les frontières de *l'anormalité* à partir de la *normalité*, et créer ainsi une base d'apprentissage pertinente et équilibrée qui donnerait aux réseaux de neurones une réponse fiable. Une base d'apprentissage équilibrée signifie que le nombre de points de bon fonctionnement doit être du même ordre de grandeur que le nombre de points de mauvais fonctionnement ; et une base d'apprentissage pertinente signifie que le nombre de points ne doit pas être démesuré mais suffisant pour définir la frontière. Comme nous allons le voir, cet objectif a été atteint par l'utilisation couplée

d'une modélisation statistique des nuages de points, par une estimation de densité de probabilité, et des traitements géométriques. Les bases d'apprentissage obtenues sont ensuite apprises par une version spéciale du BCPSL qui fournit des réseaux de neurones binaires, et pallie ainsi les divers problèmes que pose l'utilisation des réseaux de neurones continus (temps d'apprentissage, lenteur en reconnaissance...). De plus, la modélisation statistique offre aussi la possibilité d'utiliser des réseaux de neurones gaussiens, qui peuvent être utilisés de la même manière que les réseaux binaires, et qui peuvent servir au diagnostic préventif. La construction de ces deux types de réseaux est complètement automatique à partir de la modélisation statistique, et sans limitation sur la dimension de l'espace étudié. La modélisation statistique est elle *pratiquement* automatique.
La constitution d'un réseau de neurones nécessite en premier lieu de définir les variables d'entrées. Le diagnostic multidimensionnel réalisé par les réseaux de neurones est intéressant lorsque les signaux mis en jeu ont une certaine corrélation entre eux. C'est-à-dire que la zone de bon fonctionnement dans le sous-espace défini par ces signaux n'a pas une forme quelconque, mais est définie par des règles plus ou moins complexes qui peuvent être par exemple des règles de fonctionnement internes aux calculateurs (les cartographies), ou des liens dûs à la nature physique des signaux (la tension aux bornes d'une résistance et l'intensité du courant qui la traverse). La recherche de corrélation entre divers signaux sans modélisation est un problème d'ordre statistique, et la méthode qui semblait la plus adaptée est l'*Analyse en Composantes Principales* (ACP). Ce prétraitement fournit de nouvelles variables, et ce sont ces dernières qui seront les entrées des divers réseaux de neurones.
L'identification des pannes à partir d'un ensemble de défauts élémentaires (simples ou multidimensionnels), réalisée sur le premier système par le Gestionnaire de Hiérarchie, est effectuée par un organe d'identification. Ce dernier inclut un filtrage des défauts élémentaires, et un traitement totalement différent de l'approche combinatoire du Gestionnaire de Hiérarchie. Comme nous allons le voir, l'identification se réalise dans des sous-espaces synthétiques, issus d'une ACP de vecteurs de probabilités d'apparition des défauts élémentaires. Cette méthode d'identification semble beaucoup plus robuste que la méthode développée dans la maquette.

Les trois problèmes majeurs rencontrés lors du développement du premier système ont donc été résolus. La méthodologie qui a été développée [162, 160] utilise ainsi des techniques de divers domaines : l'analyse statistique avec l'ACP et l'estimation de densité, des traitements géométriques, des réseaux de neurones, des méthodes de diagnostic simples... C'est une méthode de diagnostic de type reconnaissance de formes, pouvant faire du diagnostic de pannes et du diagnostic préventif. Elle a été développée pour faire du diagnostic de pannes automobiles, mais est assez générique pour avoir, *a priori*, de multiples applications.

6.2 La méthodologie STANDING

6.2.1 Principes

Comme dans la maquette, le diagnostic est effectué en deux étapes. Une première étape de *détection de défauts élémentaires*, et une seconde étape *d'identification du défaut père*, c'est-à-dire l'origine de la panne, la cause. Les défauts élémentaires seront de deux types : les défauts élémentaires simples monodimensionnels et les défauts élémentaires multidimensionnels. Ces deux types de défauts seront alors fusionnés grâce à *l'organe d'identification*, qui remplace le module de filtrage et le gestionnaire de hiérarchie développés dans le premier système.

L'intérêt des réseaux de neurones est de pouvoir faire du diagnostic multidimensionnel sur des signaux qui ont une certaine corrélation. La recherche des corrélations entre ces signaux est effectuée grâce à

l'*Analyse en Composantes Principales* de la base de données de bon fonctionnement. Cette méthode statistique réalise une compression d'informations et élimine les redondances. Elle fournit de nouvelles variables de travail nommées composantes principales, ou composantes synthétiques, en nombre généralement inférieur au nombre de signaux d'entrée possédant pratiquement la même quantité d'informations. Le choix de plusieurs composantes détermine un nouvel espace où le nuage de points de bon fonctionnement et ses structures seront modélisés.
La modélisation d'un nuage de points dans un espace multidimensionnel nécessite la recherche de deux caractéristiques : la distribution des points et les structures qui en découlent. Ceci est réalisé par une *estimation de la densité de probabilité* du nuage de points suivie d'une classification automatique des points en groupes.
La modélisation du nuage étant réalisée, les réseaux peuvent alors être construits grâce à deux *méthodes de génération complètement automatiques*, exploitant les résultats de la modélisation statistique effectuée. La première fournit des *réseaux de neurones gaussiens*. Après un traitement géométrique, la seconde fournit des *réseaux de neurones binaires*. Ces réseaux ainsi construits permettent de savoir si un point dans ce nouvel espace se trouve dans la zone de bon fonctionnement ou non.

6.2.2 Contraintes

Toute la pertinence du système de diagnostic dépend de la *base de données de bon fonctionnement*. Celle-ci doit donc être *importante et extrêmement exhaustive*. Elle doit être la plus représentative possible de la vie du système diagnostiqué en bon fonctionnement. Pour notre application automobile, la base doit donc contenir des informations sur le véhicule dans toutes les situations envisageables : conduite (ralenti, haut régime, ...), routes (ville, autoroute, nationale, ...), environnement (conditions météorologiques, charge de la voiture, ...) etc.

Cette contrainte est assez forte, c'est pour cette raison que la *sensibilité du système doit être réglable à tous les niveaux* : détection des défauts simples et détection avec réseaux de neurones. Par exemple, il est difficile de faire l'acquisition de données dans toute la gamme de températures extérieures acceptables par une voiture. Ce manque de données doit donc pouvoir être comblé par un réglage de sensibilité.

Le système étudié peut être extrêmement bruité, ce qui est le cas d'une voiture. Le système d'acquisition peut donc enregistrer des données absurdes ne correspondant pas à un état de bon fonctionnement. Dans les espaces de travail, on nommera ces points des *points extrêmes*. Lors de la constitution des réseaux de neurones, la présence de ces points extrêmes peut être très néfaste à la qualité du système de détection. Ils doivent donc être éliminés lors d'une étape de *filtrage* de la base de bon fonctionnement.

Les contraintes de rapidité imposées par une possible utilisation temps réel sont résolues par la nature même du système. Toute la mise au point, qui nécessite une puissance de calcul relativement importante, se fait en temps différé, et le système de diagnostic implanté est *parfaitement adapté à une application temps réel*. Cette implantation peut se faire sur une machine monoprocesseur, mais si les contraintes de temps ne sont pas respectées, elle peut se paralléliser presque à tous les niveaux, et particulièrement au niveau de la détection de défauts multidimensionnels, c'est une des caractérisiques intrinsèques des réseaux de neurones. Cette parallélisation pourrait se faire sur une machine multiprocesseurs ou avec des circuits spécialisés.

Enfin, pour pouvoir réaliser un système de diagnostic de pannes performant et identifier parfaitement

la cause d'une panne de manière symbolique (par exemple conclure qu'une bougie est défectueuse et ne pas en rester sur la fonction allumage défectueuse), la méthodologie est soumise à une contrainte très importante : la *nécessité d'avoir des liens de cause à effet.* Pour réaliser de la simple *détection de défauts* (savoir si il y a un défaut ou non sans en connaître l'origine) la base de données de bon fonctionnement suffit. Pour faire une préidentification, que l'on appellera aussi *rapprochement fonctionnel* (c'est-à-dire savoir que c'est la fonction allumage ou la fonction injection qui est défaillante), l'utilisation de l'ACP avec la base de bon fonctionnement devrait suffir, c'est un des avantages primordiaux de l'ACP. Mais pour faire de l'*identification*, il est nécessaire d'avoir une base de données de mauvais fonctionnement pour chaque défaut père que l'on désire détecter. Ces bases de données de mauvais fonctionnement peuvent s'obtenir de deux manières, soit par *injection de pannes artificielles*, soit par *simulation de pannes.* Ces dernières sont réalisées dans le cas où il est impossible de créer une panne artificielle, c'est-à-dire agir physiquement sur le système. Pour cela, il est nécessaire d'avoir le plus d'informations possible, fournies par un expert, sur les signaux qui seront modifiés. Ces données de mauvais fonctionnement sont la base du paramétrage de l'organe d'identification.
De plus, *une composante synthétique peut séparer la zone de bon fonctionnement de certains défauts et la confondre avec les autres.* Ainsi, les données de mauvais fonctionnement permettent de choisir le mieux possible les composantes synthétiques sensibles aux défauts pères que l'on veut détecter. A chaque sous-espace synthétique choisi est associé un réseau de neurones. Les défauts pères, séparés de la zone de bon fonctionnement délimitée par ce réseau dans ce sous-espace particulier, seront détectés. Ce processus de détection, nommé *détection par exclusion*, permet de cibler de différentes manières les défauts élémentaires. Ces réseaux répondent *défaut élémentaire en dehors de la zone de bon fonctionnement.*

L'utilisation des réseaux de neurones offre une autre possibilité de détection de défauts élémentaires, dite *détection par inclusion.* En effet, dans le cas où un défaut père est parfaitement séparé de la région de bon fonctionnement ainsi que de tous les autres défauts, un réseau de neurones peut alors être construit pour identifier parfaitement ce défaut. Ce réseau répond *défaut élémentaire à l'intérieur d'une zone qui inclut uniquement le défaut père.* L'utilisation de ces derniers réseaux est à éviter dans la mesure du possible, car basée principalement sur le mauvais fonctionnement. Or celui-ci est généré par des pannes articielles ou simulées, et peut ne pas être représentatif des pannes réelles.

6.2.3 Choix des composantes et implantation du système

L'implantation du système de diagnostic est représentée sur la figure 2. Toutes les potentialités de la méthodologie sont représentées sur ce schéma, avec en pointillés les modules non indispensables pour faire un diagnostic de pannes, mais qui peuvent apporter de précieux renseignements.
La partie basse du schéma est pratiquement identique au schéma fonctionnel de la maquette présentée sur la figure 2.13 dans le chapitre 2, sauf que l'organe d'identification remplace le module de filtrage et le gestionnaire de hiérarchie. Les divers signaux issus du module de prétraitement sont donc diagnostiqués séparément dans le module de diagnostics simples monodimensionnels. Ce module produit en sortie un vecteur de défauts élémentaires qu'il délivre à l'organe d'identification. Les composantes de ce vecteur sont binaires : présence / absence du défaut, 1 ou 0.
Mais ces signaux sont aussi traités en parallèle par les divers réseaux de neurones. Ces signaux ne sont pas fournis directement au module de détection par réseaux de neurones : un changement de variables est effectué qui transforme les signaux en composantes synthétiques. Ce changement de variables comprend trois étapes : un centrage, une normalisation et un changement de base, c'est-à-dire une multiplication matricielle (changement de variables linéaires). Tous les paramètres de ce changement de variables sont issus de l'ACP : le centre de gravité du nuage, la norme et la matrice de changement

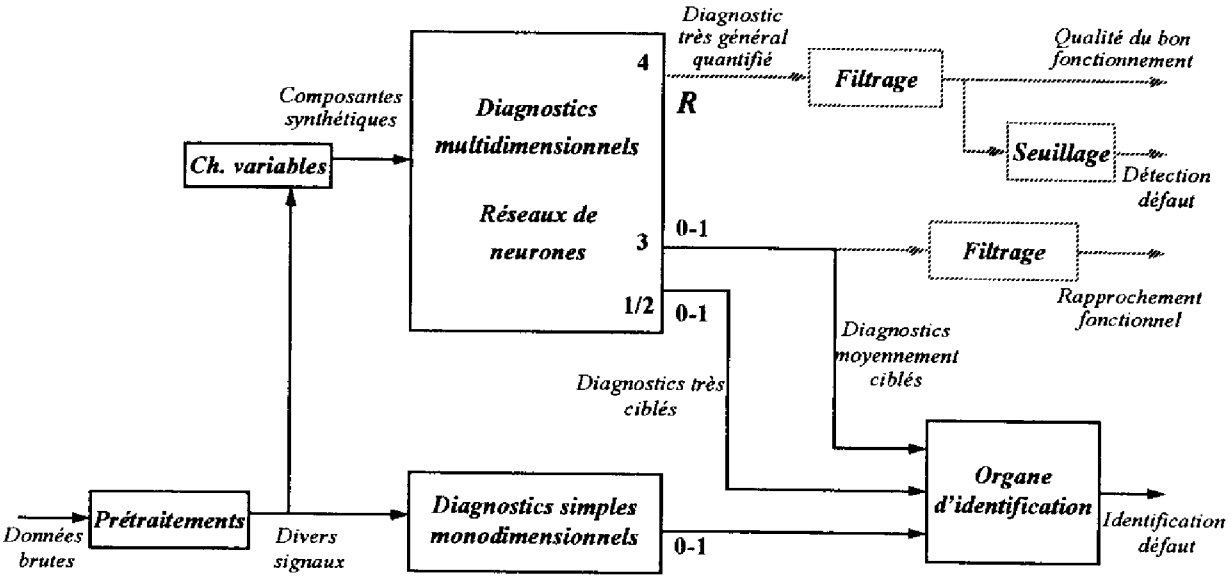

FIG. 6.1: Schéma fonctionnel du système de diagnostic

de base qui est en fait une matrice de transfert composée des vecteurs propres du produit de la matrice de covariance des signaux centrés par la métrique choisie (voir la section suivante).
Suivant les composantes synthétiques fournies en entrée, les réseaux de neurones peuvent être de quatre types, représentant une hiérarchie dans la précision du diagnostic, du plus précis au moins précis. Parmi ces quatre types de réseaux, trois fourniront un vecteur de défauts élémentaires binaires à l'organe d'identification.

- Type 1 : *Détection très ciblée par exclusion.* Un tel réseau est conçu quand une ou plusieurs composantes sont sensibles à un et un seul défaut père. Le réseau réalise alors une détection par exclusion qui est en fait l'identification du défaut père. Ce cas parfait sera très rare. Dans un cas limite où un seul défaut père est très éloigné de la zone de bon fonctionnement, et que tous les autres sont autour du bon fonctionnement sans être parfaitement dessus, on pourra alors arbitrairement élargir la zone de bon fonctionnement grâce au réglage de sensibilité et ainsi créer un tel réseau.

- Type 2 : *Détection très ciblée par inclusion.* Un réseau de neurones est construit à partir des informations de mauvais fonctionnement et est sensible à un père et un seul. Comme on l'a dit précédemment, il faudra éviter le plus possible l'usage de ce type de détection. On pourra l'utiliser lorsque la zone d'activation du défaut père est parfaitement distincte du bon fonctionnement et aussi de tous les autres défauts pères. Il faudra en plus être certain que cette zone est la même pour tous les cas de pannes similaires. Dans le cas où cette zone d'activation est extrêmement bien séparée avec une marge importante, cette marge pourra être utilisée pour rendre ce détecteur plus robuste à des écarts éventuels des pannes réelles par rapport au modèle qui a servi à constituer le système : on élargira au maximum la zone de détection.

- Type 3 : *Détection moyennement ciblée* (implicitement par exclusion). C'est ce type de réseaux qui sera le plus utilisé, car le plus robuste dans son principe. Ces réseaux seront des organes de détection par exclusion sensibles à plusieurs défauts pères. Même si ces réseaux ne sont

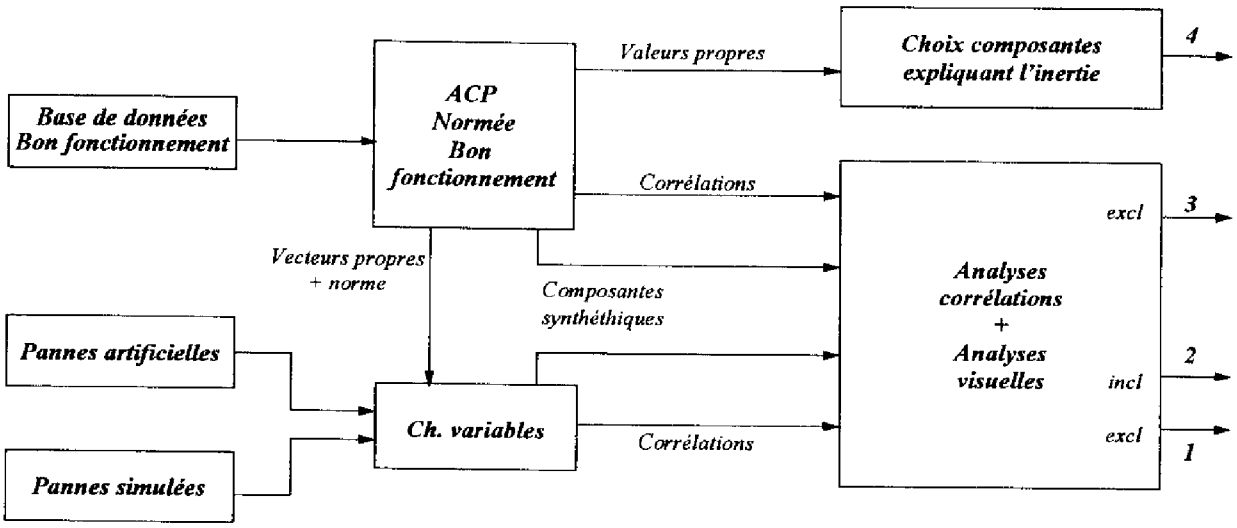

FIG. 6.2: Choix des composantes synthétiques suivant le type de réseau

pas caractéristiques d'un défaut père, ils vont participer activement à l'identification, par un accroissement de la discrimination. Ainsi, dans le cas où le système d'identification ne parvient pas à discriminer deux défauts pères, il suffira en général d'ajouter un de ces réseaux sensibles à un seul de ces défauts pères, même s'il est sensible à d'autres défauts. Un second avantage de ces réseaux est dû à l'ACP. Comme l'ACP regroupe en composantes synthétiques des signaux fortement corrélés entre eux, on pourra concevoir des réseaux sensibles à des défauts pères reliés à une même fonctionnalité : par exemple l'allumage, l'injection, ou même une sous-fonction, dans certains cas. Ces défauts élémentaires moyennement ciblés seront alors très intéressants lors d'un échec d'identification. C'est ce que l'on nomme *le rapprochement fonctionnel.* La constitution de ces réseaux ne nécessite aucune information de mauvais fonctionnement.

- Type 4 : *Détection très générale.* Les principales applications de ces réseaux, sensibles à un maximum de défauts pères, sont d'une part le diagnostic préventif, et d'autre part la détection générale, sans objectif d'identification. La sortie de ces réseaux n'est pas binaire, elle est quantifiée, et représente la manière dont est positionné le vecteur d'état courant par rapport au nuage de bon fonctionnement : s'il est bien au centre, où se trouve la majorité des points, ou s'il est proche de la frontière, ou carrément à l'extérieur. Ainsi, cette sortie pourra par exemple être négative à l'extérieur, et positive à l'intérieur avec plus ou moins d'amplitude suivant sa position par rapport aux frontières. Un tel réseau pourra donc détecter s'il y a un défaut ou non en seuillant sa sortie (après un filtrage), ou pourra être utilisé pour faire du diagnostic préventif lorsqu'il n'y a pas de défauts, en *quantifiant la qualité du bon fonctionnement.*

Pour ce dernier type de réseaux, le choix des composantes ne pose pas de problème. On prendra 3, 4 ou 5 composantes qui expliquent le plus d'inertie du nuage de bon fonctionnement, donc les composantes qui correspondent aux axes des plus grandes valeurs propres de la matrice de covariance diagonalisée (voir paragraphe 6.3.2). De plus, on pourra compléter ce diagnostic général avec un second réseau, en prenant les composantes de plus grande inertie mais après une ACP réalisée avec une métrique particulière qui amoindrit l'importance des axes choisis dans le premier réseau. Avec ces deux réseaux,

on aura un maximum de sensibilité sur tous les signaux, même ceux qui ont une relativement faible variabilité et qui sont généralement effacés par l'ACP, même normée.

La conception de chaque type de réseau impose un choix particulier des composantes synthétiques fournies en entrée. Le choix de ces composantes, sauf pour le type 4, est un problème délicat. Et une méthode automatique sûre n'a pas été trouvée. Comme on peut le voir sur la figure 6.2, ce choix fait appel à une analyse visuelle guidée par des informations telles que les corrélations entre variables synthétiques et signaux, pour le bon fonctionnement et le mauvais fonctionnement.

6.3 Les méthodes statistiques

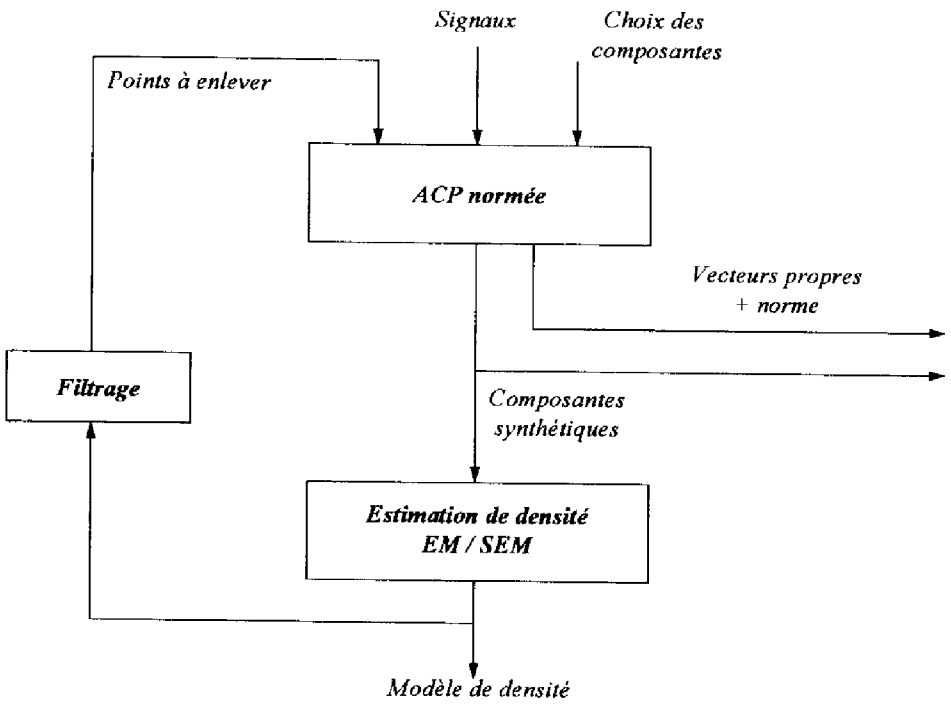

FIG. 6.3: Les méthodes statistiques employées et leur enchaînement

6.3.1 Généralités

L'enchaînement des diverses méthodes statistiques employées dans cette méthodologie est représenté sur la figure 6.3. Comme on peut le voir, l'estimation de densité sert de base au filtrage des *points extrêmes*, les points éliminés sont les points qui se trouvent dans des zones de très faible densité de probabilité. Après une phase de filtrage, il est nécessaire de refaire une ACP, car l'élimination de points extrêmes, même s'ils ne sont pas nombreux, peut modifier significativement les résultats de l'ACP, en attirant les axes principaux vers ces points extrêmes. On boucle, jusqu'à ce qu'il n'y ait plus de points extrêmes, ce qui peut se détecter grâce à la distribution des densités de probabilité pour tous les points (voir paragraphe 6.3.4).

6.3.2 Analyse en Composantes Principales

L'Analyse en Composantes Principales [176] fait partie des méthodes d'analyse statistique dites méthodes factorielles. Cette méthode d'analyse est basée sur un procédé de calcul largement utilisé en mécanique du solide. L'étude du mouvement d'un solide soumis à des forces diverses, nécessite de connaître ses paramètres d'inertie, et plus particulièrement son ellipsoïde d'inertie, avec ses axes principaux. La recherche des axes d'inertie d'un solide est exactement une ACP d'une distribution continue de points avec la métrique usuelle. On peut représenter ceci sur le schéma qui suit.

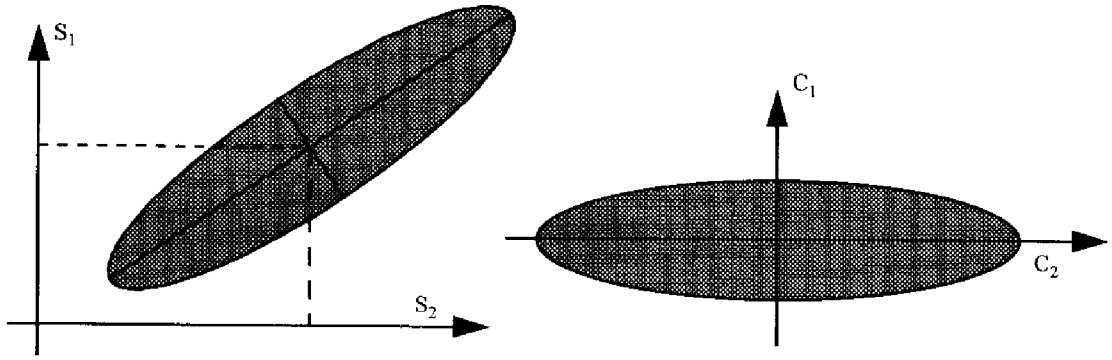

FIG. 6.4: Vision mécanique de l'ACP : recherche des axes d'inertie

Après un centrage du nuage de points, on va rechercher les axes suivant lesquels les points sont distribués. Ces axes sont construits de manière à former une nouvelle base orthonormale, qui formera un nouveau repère avec le centre de gravité du nuage comme centre. D'un point de vue statistique, cette méthode permet de compresser l'information contenue dans les données et d'éliminer les redondances entre les paramètres mesurés.
Les données sont représentées par une matrice X de dimensions $n \times p$, où n est le nombre de vecteurs d'états et p le nombre de paramètres (signaux). Le terme général $X(i,j)$ de X est la valeur prise par le $j^{ème}$ paramètre à la $i^{ème}$ étape. L'ensemble des vecteurs d'état forme un nuage de points $N_x = \{x_i / i = 1 \ldots n\}$ dans l'espace euclidien E de dimension p, avec x_i un point représentant le $i^{ème}$ vecteur ligne de X.
Le principe de l'ACP consiste à chercher des sous-espaces S de dimension s variant de 1 à p qui contiennent le maximum de cette information. Cette dernière est mesurée par l'inertie du nuage relativement à son centre de gravité. En prenant le centre de gravité comme origine du repère, ce qui revient à centrer les données, cette inertie est définie par :

$$I[N_x] = \sum w_i ||x_i||^2_M \text{ avec } \sum w_i = 1$$

avec w_i le poids de chaque individu, généralement identique pour tous dans notre cas. Lorque la norme est calculée relativement à la métrique usuelle, c'est-à-dire lorsque M est l'identité, cette information est la somme des variances des paramètres.
L'inertie de la projection $\widetilde{N_x}$ de N_x sur un sous-espace S, l'inertie du nuage expliquée par S, mesure l'information contenue dans ce sous-espace. Maximiser cette information revient à minimiser l'inertie non expliquée, c'est-à-dire :

$$I[N_x] - I[\widetilde{N_x}] = \sum w_i ||x_i - \widetilde{x_i}||^2_M$$

avec $\widetilde{x_i}$ la projection de x_i sur S.

En notant V la matrice de covariance, on montre que chaque sous-espace S est engendré par les s premiers vecteurs propres c_j de VM, soit

$$VMc_j = \lambda_j c_j \quad \text{avec} \quad \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_p \geq 0.$$

La recherche de ces éléments propres revient à diagonaliser la matrice VM. Les vecteurs [resp. valeurs] propres sont dits vecteurs [resp. moments] principaux de l'A.C.P du triplet (X, M, D) où D est la matrice diagonale des poids w_i.
A chaque vecteur principal c_j est associée une Composante Principale C^j . Elle représente les coordonnées des projections des points sur l'axe engendré par ce vecteur et est définie par

$$C^j = XMc_j \tag{6.1}$$

A chaque Composante Principale est associé un sous-groupe de paramètres via les corrélations linéaires. Cette dernière représente le cosinus de l'angle entre chaque composante principale et chaque paramètre et est définie par

$$\rho(X^k, C^j) = \frac{Cov(X^k, C^j)}{\sqrt{var(X^k)var(C^j)}} \tag{6.2}$$

Chaque paramètre est associé à la composante avec laquelle il est fortement corrélé. Pour visualiser les structures du nuage de points induites par un groupe de variables bien corrélées entre elles relativement à un autre groupe ayant la même propriété, on représente sur un plan la composante principale associée à l'un par rapport à celle associée à l'autre.
Les données traitées sont généralement hétérogènes. Pour éliminer l'effet d'échelle, des réductions, ou normes, sont prévues. Ces réductions reviennent à des transformations linéaires, à des choix de métriques particulières, ou des normalisations avant l'ACP. Elles permettent de faire différentes analyses et de faire apparaître des informations complémentaires.
L'aspect réduction dimensionnelle ou compression d'information intervient en pratique en se fixant un seuil sur le pourcentage d'inertie expliquée par les diverses composantes synthétiques. Dans nos expériences nous avons conservé toutes les composantes synthétiques qui expliquaient plus de 0.1% de l'inertie totale.

6.3.3 Estimation de densité de probabilité : algorithmes EM et SEM

Généralités

L'estimation de la densité de probabilité du nuage de points est fondamentale, et a deux objectifs : le filtrage des données pour éliminer les points extrêmes, et la modélisation du nuage pour la constitution des réseaux de neurones. Les méthodes d'estimation de densité sont nombreuses, et de deux types : les non paramétriques, et les paramétriques. Les premières estiment les densités en chaque point du nuage en fonction de la position des autres, et les secondes calculent les paramètres d'un modèle de densité pour s'adapter le mieux posiible aux données.
Une première étude a été réalisée avec une méthode non paramétrique, consistant en résumé à estimer la densité de probabilité (pdf) avec un calcul d'histogrammes. Mais cette méthode a vite montré ses limitates : elle est très gourmande en moyens de calcul même en dimension 2 (impossible au-delà) et peu précise. Nous nous sommes alors tournés vers une famille de méthodes paramétriques plus puissantes, basées sur une modélisation de la pdf en somme de distributions gaussiennes généralisées. Dans chaque sous-espace principal, les points sont généralement regroupés en différentes structures non homogènes. Il est donc naturel de supposer que l'échantillon provient d'un mélange d'un certain

nombre de populations dont on cherche à estimer les paramètres des lois de probabilité. On suppose donc que la densité de probabilité est une somme pondérée de q gaussiennes, c'est-à-dire :

$$p(x) = \sum_{k=1}^{q} \alpha_k \; p_k(x) \quad \text{avec} \quad \sum_{k=1}^{q} \alpha_k = 1$$

$$\text{et} \quad p_k(x) = \frac{1}{\sqrt{2\pi}^d \sqrt{\det V_k}} \exp[-\frac{1}{2} (x - \mu_k)^T \; V_k^{-1} \; (x - \mu_k)]$$

Le paramètre μ_k est le centre de la gaussienne, V_k sa matrice de covariance, et α_k son poids dans la somme pondérée. La somme de ces derniers doit nécessairement être égale à un pour que p soit une densité de probabilité (la somme intégrale sur tout l'espace devant être égale à un). La dimension de l'espace de travail est notée d.
Le choix d'une somme de gaussiennes comme modèle est assez classique, la gaussienne ayant des propriétés remarquables, et aussi car il est prouvé qu'une densité de probabilité peut être approchée avec une précision aussi grande que l'on veut, par une somme de gaussiennes (somme pouvant être infinie). Ce modèle est appelé *mélange de gaussiennes.*

L'algorithme EM

La méthode EM (Expectation Maximisation) [164, 54] consiste à estimer de manière itérative les paramètres relatifs à toutes les gaussiennes, c'est-à-dire les triplets $\{(\alpha_k, \mu_k, V_k), k = 1, q\}$ par le maximum de vraisemblance.
Après l'initialisation de ces éléments, une étape de l'algorithme EM est définie ainsi :

$$\alpha_k^t = \frac{1}{n} \sum_{i=1}^{n} \frac{\alpha_k^{t-1} p_k^{t-1}(x_i)}{p^{t-1}(x_i)} \tag{6.3}$$

$$\mu_k^t = \frac{\sum_{i=1}^{n} x_i \frac{\alpha_k^{t-1} p_k^{t-1}(x_i)}{p^{t-1}(x_i)}}{\sum_{i=1}^{n} \frac{\alpha_k^{t-1} p_k^{t-1}(x_i)}{p^{t-1}(x_i)}} \tag{6.4}$$

$$V_k^t = \frac{\sum_{i=1}^{n} (x_i - \mu_k^{t-1})(x_i - \mu_k^{t-1})^T \frac{\alpha_k^{t-1} p_k^{t-1}(x_i)}{p^{t-1}(x_i)}}{\sum_{i=1}^{n} \frac{\alpha_k^{t-1} p_k^{t-1}(x_i)}{p^{t-1}(x_i)}} \tag{6.5}$$

Cet algorithme est très sensible à l'initialisation des paramètres des gaussiennes. Nous avons essayé de l'initialiser le mieux possible, grâce au calcul d'une partition optimale en q classes avec l'algorithme k-mean. Les éléments α_k^0 , μ_k^0 et V_k^0 sont initialisés respectivement au poids, au centre de gravité et à la matrice des variances intraclasses de la $k^{ème}$ classe de la partition optimale.
Cet algorithme s'est montré extrêmement puissant. Si le nombre de gaussiennes est suffisant, la densité de probabilité est modélisée de manière très précise. Un des problèmes de cet algorithme est le choix a priori du nombre de composantes. De plus, malgré une relativement bonne initialisation avec une partition optimale, il arrive que l'EM tombe dans un maximum local de la vraisemblance. Ces deux problèmes peuvent être en partie résolus grâce à l'algorithme SEM.

L'algorithme SEM

L'algorithme SEM (Stochastique EM) [38] intègre un procédé de recuit simulé dans l'algorithme EM. Sa particularité consiste à incorporer une étape stochastique entre les différentes étapes de l'algorithme

EM. Il vise à éliminer les gaussiennes qui ont des probabilités d'apparitions faibles. On peut alors, en théorie, commencer l'algorithme avec un majorant du nombre de composantes nécessaires, et l'algorithme se chargera d'éliminer les composantes non indispensables. De plus, l'introduction d'une sorte de bruitage de la maximisation de la vraisemblance permet d'éviter la stabilisation dans un maximum local, c'est le principe du recuit simulé.
Les étapes de cet algorithme sont les suivantes :

- Initialisation : On fixe le nombre maximal de composantes. Pour chaque point x_i , on choisit au hasard les probabilités d'appartenance aux composantes :

$$0 < \tau_{ik}^0 < 1 \qquad \sum_{k=1}^{q} \tau_{ik}^0 = 1$$

- Etape S : Pour tout x_i , on tire la variable aléatoire multinomiale $e^t(x_i) = (e_k^t(x_i), k = 1, ..., q)$ d'ordre 1 et de paramètres $(\tau_{ik}^t, k = 1, ..., q)$. Cette variable induit une partition de l'échantillon, la $k^{éme}$ classe est définie par :

$$P_k^t = \{x_i / e_k^t(x_i) = 1\}$$

  Si le cardinal d'une classe est inférieur au seuil fixé, alors on diminue le nombre de composantes et on réinitialise l'algorithme.

- Etape M : Les éléments α_k^{t+1} , μ_k^{t+1} et V_k^{t+1} sont respectivement le poids, le centre de gravité et la matrice des variances intraclasses de la $k^{ème}$ classe P_k^t .

- Etape E : Pour chaque point, on calcule les probabilités d'appartenance a posteriori à chaque classe :

$$\tau_{ik}^{t+1} = \frac{\alpha_k^{t+1} p_k^{t+1}(x_i)}{\sum \alpha_k^{t+1} p_k^{t+1}(x_i)}.$$

Malgré ces avantages théoriques par rapport à l'EM, cet algorithme a des inconvénients non négligeables. Premièrement, le choix du seuil d'élimination d'une composante n'est pas évident. Deuxièmement, comme l'algorithme est sensé éliminer les composantes redondantes, partant de deux configurations avec un nombre de composantes différentes, il devrait converger vers le même nombre de composantes finales. Ce qui n'est pratiquement jamais le cas. Sauf dans des cas triviaux, où les distributions de points sont issues de générateurs gaussiens plus ou moins bruités.

6.3.4 Filtrage des points extrêmes

Après avoir estimé la densité de probabilité par p_i calculée pour chaque point du nuage à partir du modèle, on étudie la distribution de la variable aléatoire y_i définie par $y_i = Log(p_i)$.
La distribution de probabilité de la variable aléatoire y_i est très asymétrique : la pdf maximale est atteinte au centre du nuage de points et autour de ce centre il y a pratiquement tous les points qui ont des pdf assez fortes. En fait il y a peu de points qui ont des pdf faibles, ce sont les points proches du bord de la zone de bon fonctionnement. Les points à enlever sont ceux qui ont des pdf extrêmement faibles, complètement à l'extérieur des zones d'influence des diverses gaussiennes. La queue de la distribution y_i est donc très étroite en direction des valeurs positives (cette queue est de largeur nulle dans le cas d'une distribution p_i gaussienne), et une queue beaucoup plus large en direction des valeurs négatives. Le filtrage consiste donc à définir un seuil sur la queue de distribution en direction des valeurs négatives, et à éliminer tous les points pour lesquels
la valeur de y_i est inférieure à ce seuil.

Dans nos expérimentations, ce seuil a généralement été fixé à

$$Seuil = E(y_i) - \alpha \, \sigma_{y_i}$$

avec $\alpha \simeq 10$. Avec une telle valeur, d'après la loi de Tchebychev-Bienaymé, la probabilité qu'un point soit retiré est inférieure à 1% (en fait elle devrait être plus faible car la distribution n'est pas du tout symétrique). Cette première heuristique semble très bien adaptée dans le cas d'une distribution gaussienne, et relativement bien dans le cas d'une distribution assez symétrique. Pour nos distributions, cette heuristique a donné *en général* de bons résultats, mais pas systématiquement, une confirmation visuelle a été nécessaire, et dans certains cas le paramètre α a du être modifié.
Nous avons donc essayé de rechercher une meilleure méthode heuristique de détermination automatique de ce seuil. Une solution semble être de définir le seuil en fonction de la valeur de l'écart-type de la variable y_i, mais aussi de la valeur du moment réduit d'ordre 4 (c'est-à-dire le moment d'ordre 4 divisé par le carré de l'écart-type), qui est caractéristique des queues de distribution (car dans le calcul du moment d'ordre 4, ce sont les termes de plus grandes valeurs qui prennent toute l'importance dans le calcul de l'espérance, donc ce sont les points extrêmes qui caractérisent ce moment).
Ainsi une heuristique plus adaptée à nos distributions semble être

$$Seuil = E(y_i) - (\alpha + \beta \, \gamma_4) \, \sigma_{y_i}$$

où γ_4 est le moment réduit d'ordre 4. D'après les quelques expérimentations menées avec cette heuristique, avec $\alpha \simeq 10$ et $\beta \simeq 0.1$, il semble que le seuil soit mieux fixé, car l'heuristique tient compte de l'asymétrie de la distribution (voir les figures de 6.13 à 6.15 de la section 6.7 qui présentent les résultats de plusieurs phases de filtrage).

6.3.5 Alternance des deux algorithmes EM et SEM

En général, la plus grande partie de l'apprentissage du mélange de gaussiennes est réalisée par l'algorithme EM, initialisé avec un assez grand nombre de composantes (entre 10 et 20). Mais l'utilisation de l'algorithme SEM de manière passagère s'est révélée extrêmement utile en deux endroits :

- L'algorithme EM a un autre problème : lorsqu'un petit groupe de points est complètement en dehors de nuage de bon fonctionnement mais en nombre légèrement supérieur à 8 ou 10 (insignifiant par rapport au nombre total de points, qui était dans nos expériences de l'ordre de 15000), l'algorithme EM a généralement tendance à associer une gaussienne sur ce petit groupe. Le problème qui en découle est que leur pdf n'est plus très faible, car le modèle les a inclus, alors que ce sont tout de même des points extrêmes. La conséquence est que le filtrage ne peut pas les retirer sans éliminer une grande quantité d'autres points qui eux ne sont pas des points extrêmes. Pour régler ce problème, l'utilisation de l'algorithme SEM est primordiale. En fixant le seuil d'élimination d'une composante entre 0, 2% et 0, 5% du nombre total de points du nuage, ces gaussiennes "récalcitrantes" sont automatiquement retirées du mélange.

- Le contrôle de la convergence de l'algorithme EM doit normalement se faire sur la valeur de la vraisemblance du modèle V, ou son logarithme L la log-vraisemblance, car c'est ce critère qui est maximisé par l'algorithme. Ces quantités sont définies par

$$V = \prod_{i=1}^{N} p(x_i) \qquad L = \log V = \sum_{i=1}^{N} \log p(x_i)$$

  Mais un contrôle par un de ces critères pose de gros problèmes numériques, car ils se stabilisent beaucoup trop vite. Les informations sur l'évolution de la convergence deviennent très rapidement

les dernières décimales même en double précision, et donc inexploitables. Nous avons donc utilisé en parallèle un autre critère pour contrôler la convergence. Ce critère est la norme de la variation du vecteur des poids des gaussiennes entre deux itérations soit,

$$||\Delta\alpha^t|| = ||\alpha^t - \alpha^{t-1}||$$

L'étude de ce critère s'est avérée très intéressante, car assez représentative de la manière dont converge l'EM. Ainsi, l'algorithme EM semble alterner entre des phases que l'on nommera de *restructuration* et durant lesquelles le vecteur α^t a de fortes variations, et des phases de *réadaptation*. Il semblerait que pendant les phases de restructuration, tous les paramètres des gaussiennes (particulièrement les centres et les poids) bougent de manière très sensible pour mieux adapter les différentes gaussiennes aux divers groupes de points du nuage, et que pendant les phases de réadaptation, l'algorithme affine les valeurs de ces paramètres sous une forte contrainte de maximisation de la vraisemblance dans cette nouvelle configuration. De plus l'utilisation de ce critère ne pose pas de problèmes numériques, car à chaque itération nous avons toujours $\sum_{k=1}^{q} \alpha_k^t = 1$. Au cours du processus de convergence, ce paramètre décroît en moyenne vers 0, et l'algorithme est arrêté lorsqu'il prend une valeur très faible, de l'ordre de 10^{-10}. Dans toutes les expériences menées, la log-vraisemblance était stabilisée bien avant que $||\Delta\alpha^t||$ soit proche de 0, ce qui montre son intérêt.

Pour améliorer la convergence, l'algorithme SEM a donc été utilisé au début des phases de réadaptation, juste après les phases de restructuration. Il semble que cette utilisation accélère le processus de manière significative, en *stimulant* l'apparition de nouvelles phases de restructuration. La valeur de C^t après une phase de restructuration, est généralement plus faible qu'avant cette phase, d'où l'intérêt de stimuler l'apparition de ces phases.

Nous avons donc utilisé l'algorithme EM, et l'algorithme SEM temporairement. Mais il convient de distinguer deux cas d'apprentissage :

- lorsque le mélange est appris pour faire un filtrage, la convergence parfaite jusqu'à ce que $||\Delta\alpha^t|| < 10^{-10}$ n'était pas nécessaire. L'algorithme SEM ne doit pas trop être utilisé pour stimuler une restructuration car ceci peut être la cause de l'adaptation d'une gaussienne sur un groupe de points extrêmes. Donc dans ce cas, l'algorithme SEM est utilisé deux ou trois fois, sans trop se soucier de l'instant, uniquement pour retirer les gaussiennes "récalcitrantes". En pratique la convergence jusqu'à $||\Delta\alpha^t|| < 10^{-5}$ était suffisante pour avoir un bon filtrage.
- dans le cas où l'on sait, d'après le dernier filtrage, qu'il n'y a plus de points extrêmes francs, complètement à l'extérieur du nuage, une convergence très précise est alors nécessaire. Nous avons donc utilisé l'algorithme SEM, en vue de stimuler les restructurations, et l'algorithme EM (toujours utilisé en final) a été arrêté quand $||\Delta\alpha^t|| < 10^{-10}$.

Cette technique d'alternance des deux algorithmes a été supervisée par nos soins. Il conviendrait d'essayer de l'automatiser (voir les résultats d'une estimation de densité avec cette méthode, figures 6.16 et 6.17 de la section 6.7).

6.3.6 Remarques sur l'initialisation de l'association EM - SEM

Comme nous l'avons signalé, les algorithmes EM et SEM sont très sensibles aux conditions initiales, c'est-à-dire à la manière d'initialiser tous les paramètres du mélange. Une mauvaise initialisation peut conduire à des difficultés de convergence et/ou à la convergence vers un mauvais maximum local de la vraisemblance. Au lieu d'initialiser les paramètres des gaussiennes au hasard, nous les avons initialisé

sur la base d'une partition obtenue par l'algorithme *k-mean* [65, 57]. Cette technique est largement utilisée (par exemple [37]). Nous avons donc initialisé les poids du mélange à partir des proportions de points dans chaque classe, les matrices de covariance et les vecteurs moyens à ceux des classes calculés séparément.

Mais deux problèmes nous ont incité à réflechir à l'initialisation du *k-mean*. Celui-ci est *a priori* initialisé avec une partition tirée aléatoirement. Avec une telle initialisation, les temps de calcul peuvent être assez longs sur les grosses bases de données que nous avons utilisées (15000 vecteurs). De plus, cet algorithme peut aussi tomber dans un minimum local peu intéressant. Bien que ce dernier problème ne soit pas très grave, puisque le *k-mean* est simplement utilisé pour initialiser l'EM, le premier problème est plus gênant. Pour le résoudre, nous avons développé une méthode de partitionnement de la base simple et rapide.

Intuitivement, le but est de fournir une partition telle que les points soient bien répartis dans les classes et que celles-ci forment un pavage régulier de l'espace. L'idée de la méthode développée est de partir de q points que nous avons nommés les *pôles*. Ceux-ci doivent être bien répartis dans la zone occupée par les points. On va générer une classe par pôle. Pour cela, il suffit d'associer à un point la classe du pôle le plus proche. Donc tout le problème est dans la génération de ces pôles. Cette génération commence par le calcul d'une distance de référence. Pour cela on calcule le centre de gravité du nuage de points, la moyenne $\overline{d}$ et l'écart-type σ_d de la distance des points à ce centre de gravité. On définit alors le premier pôle comme le centre de gravité. Les autres pôles sont alors obtenus successivement en recherchant un point du nuage qui est à une distance de tous les pôles déjà calculés supérieure à une distance minimale. Cette distance est initialisée au début de la boucle à $\overline{d} + 2\sigma_d$. Au cours de la boucle, si aucun point n'est trouvé, cela signifie que cette distance est trop grande. Dans ce cas elle est diminuée de 5% et on boucle à nouveau sur les points pour trouver un pôle.

Cette méthode de partition est rapide. Bien que d'un point de vue classification les résultats ne soient pas très intéressants, elle permet d'accélérer de manière importante le *k-mean* par rapport à une initialisation avec une partition aléatoire. Ce qui était notre but.

Il y a enfin un problème d'initialisation assez délicat que nous n'avons pas eu le temps d'aborder. Il s'agit de déterminer automatiquement un majorant du nombre de composantes nécessaires. Une idée, que nous n'avons pas eu le temps de tester, est peut-être d'utiliser une version paramétrée de l'algorithme des *clusters principaux* [116]. Une des remarquables propriétés de cet algorithme développé par Mirkin [136] est qu'il n'est pas nécessaire de donner le nombre de classes *a priori*, ce nombre est un résultat de l'algorithme. La version paramétrée permet de moduler le nombre de composantes obtenues en modifiant la valeur d'un simple paramètre. Ainsi en réglant ce paramètre de manière à avoir beaucoup de composantes, cette version pourrait nous donner automatiquement un majorant du nombre de composantes à prendre. Les composantes superflues étant ensuite éliminées par l'algorithme SEM.

Une autre solution à ce problème est peut-être d'utiliser le critère de qualité des approximations de partitions optimales fournies par le *k-mean* proposé dans [115]. Ce critère de qualité est issu d'une propriété des partitions optimales [117]. Il permet de choisir de manière heuristique le nombre de composantes qui assure une bonne qualité de la partition. Cette propriété des partitions optimales est démontrée grâce à une présentation factorielle du problème de classification automatique basé sur le critère de minimisation de la variance intra-classes. Ce résultat est une généralisation à une métrique quelconque d'un résultat déjà obtenu pour des métriques particulières [122]. Ce résultat est théoriquement très intéressant car il présente le problème de classification sous la forme d'une ACP sous contraintes. Il réunit donc sous un même formalisme, deux écoles pour l'instant assez distantes. Mais ce n'est qu'un premier pas. Intuitivement ce n'est pas très surprenant, car les deux problèmes sont basés sur un critère commun : la minimisation d'une inertie. Mais l'un doit prendre en compte une notion de partition, qui est un problème combinatoire, et l'autre une notion de changement de

base optimale, ce qui revient à un problème de diagonalisation.
Bien que le problème soit de trouver un majorant du nombre de composantes, ce qui n'est pas très restrictif, il n'est pas sûr que les méthodes possibles précédemment citées soient un moyen pertinent, car comme nous le verrons dans les sections 6.4.3 et 6.7, les algorithmes basés sur le critère d'inertie ont un grave problème dans le cas de distributions de points non approximativement sphériques. Ce problème est intrinsèque à ces méthodes car il est lié au critère même utilisé.

6.4 Génération automatique des réseaux de neurones

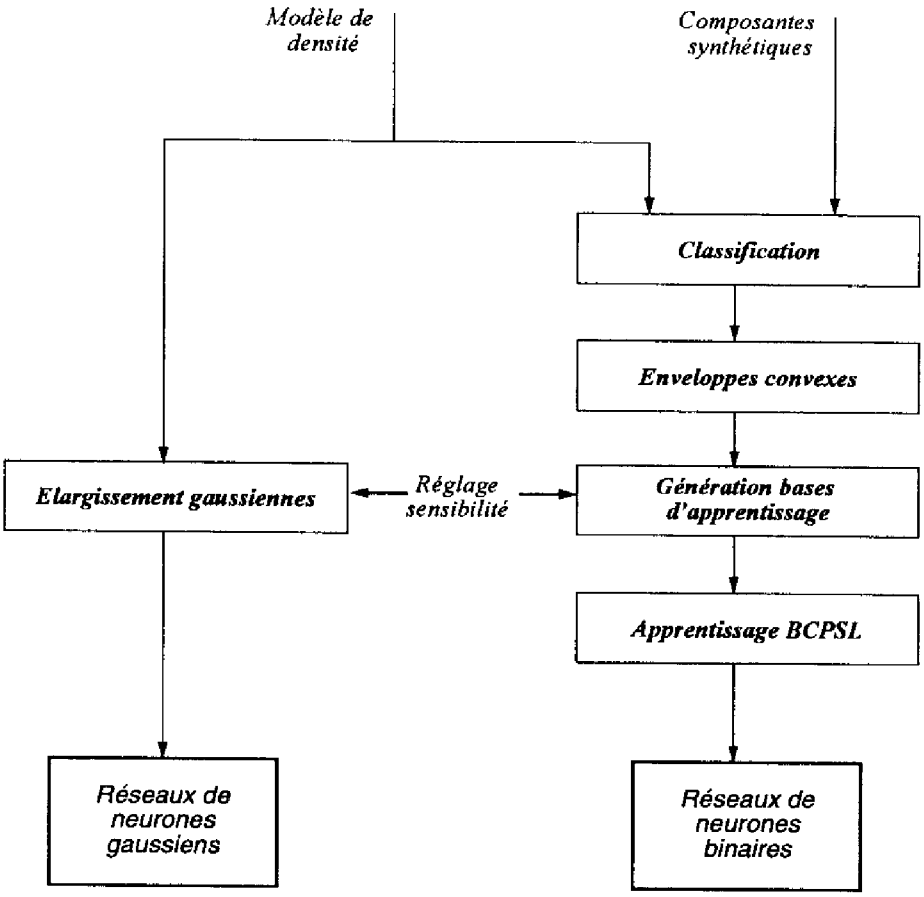

FIG. 6.5: Génération des réseaux de neurones

6.4.1 Généralités

Deux types de réseaux de neurones peuvent être utilisés pour détecter les défauts élémentaires : les *réseaux de neurones gaussiens* et les *réseaux de neurones binaires*. Comme nous allons le voir, la génération des réseaux gaussiens ne pose pas de problèmes, mais la génération des réseaux de neurones binaires nécessite plusieurs étapes de traitement des données. Pour ces deux types de réseaux, la sensibilité est totalement réglable. Pour les réseaux binaires, la sensibilité peut être réglée de manière isotrope, alors que pour les réseaux gaussiens ce réglage n'est pas isotrope.

La figure 6.5 présente le schéma général de la génération des réseaux.
Les réseaux gaussiens comportent beaucoup moins d'unités que les réseaux binaires. Mais les calculs effectués par les unités binaires sont beaucoup plus simples que ceux des unités gaussiennes. Les deux types de réseaux peuvent être utilisés, mais dans certains cas il sera préférable d'utiliser l'un ou l'autre. En dimension 2 et 3, les réseaux binaires seront en général plus rapides dans le cadre d'une implantation sur une machine monoprocesseur. Mais pour des dimensions plus élevées, l'usage des réseaux gaussiens paraît nécessaire, car le nombre d'unités des réseaux binaires devient très élevé. En revanche, dans des cas où une sensibilité isotrope sera nécessaire, il est indispensable d'utiliser les réseaux binaires.
D'autre part, dans le cadre d'une utilisation pour du diagnostic préventif, les réseaux gaussiens seront utilisés pour deux raisons : le nombre d'entrées du réseau est important, et la quantification du bon fonctionnement est plus facile avec les réseaux gaussiens qu'avec les réseaux binaires.
Au niveau des potentialités de parallélisme, les réseaux binaires sont plus faciles à paralléliser à un haut degré que les réseaux gaussiens. Pour les réseaux binaires, il suffit d'associer un processeur à chaque neurone, mais pour les réseaux gaussiens, un fort parallélisme nécessite de paralléliser les multiplications matricielles inhérentes au modèle.
On rappelle que ces réseaux fonctionnent avec les composantes synthétiques en entrée, et en sortie ils répondent 0 ou 1, pour absence de défaut / présence de défaut.
Dans le cas des réseaux de type 2 (détection très ciblée par inclusion), la sortie du réseau est inversée, pour répondre défaut dans la zone d'activation du réseau et pas défaut à l'extérieur.

6.4.2 Les réseaux de neurones gaussiens

Ils sont directement créés à partir du mélange de gaussiennes. Comme le montre la figure 6.6, ils comportent une seule couche cachée, où chaque unité correspond à une gaussienne du mélange. L'unité k calcule la densité locale $p_k(x)$, et l'unité de sortie calcule la somme de ces densités pondérée par les poids α_k, associés aux connections liant les unités cachées à l'unité de sortie. Celle-ci calcule donc la densité $p = \sum_{k=1}^{q} \alpha_k p_k(x)$, qui est seuillée : si la pdf est inférieure au seuil, la sortie globale du réseau est 1, sinon elle est égale à 0.

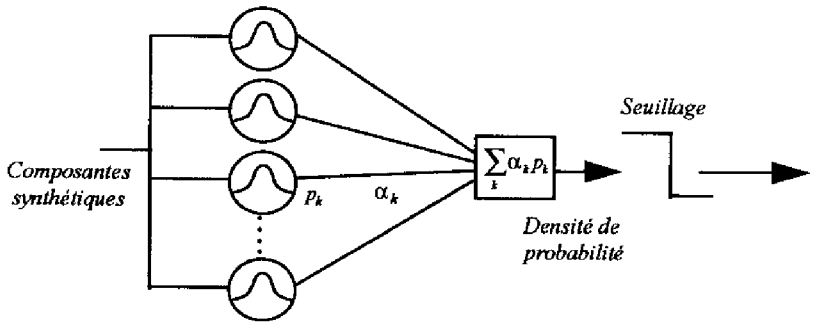

FIG. 6.6: Réseaux de neurones gaussiens

Le seuil est déterminé en fonction de la distribution des pdf du nuage de points qui a servi à l'apprentissage. On pourra par exemple prendre la pdf minimale du bon fonctionnement divisée par 2, car il ne faut pas oublier que les exponentielles des gaussiennes tendent très rapidement vers 0 lorsqu'on s'éloigne du centre.

Le réglage de la sensibilité peut se faire de deux manières. En modifiant le seuil de détection ou par un élargissement de toutes les gaussiennes du mélange. Le fait que la pdf tende très rapidement vers 0 quand on s'éloigne de la distribution rend la première méthode délicate dans certains cas, si on veut une faible sensibilité, le seuil devient très faible et peut rapidement dépasser la précision numérique de la machine utilisée. La seconde méthode, qui peut d'ailleurs être combinée avec la première, consiste à élargir toutes les gaussiennes en multipliant toutes les valeurs propres de la matrice de covariance de chaque gaussienne par le même coefficient positif (voir les figures de 6.27 à 6.29 de la section 6.7 pour des exemples de réseaux gaussiens pour la détection).

Pour une utilisation de ces réseaux dans le cadre d'un diagnostic préventif, la quantification du bon fonctionnement est très simple : il suffit de prendre la pdf, de ne pas seuiller la sortie du réseau (voir les figures 6.25 et 6.26 de la section 6.7).

6.4.3 Les réseaux de neurones binaires

Généralités

Le problème de génération des réseaux de neurones classiques (binaires ou continus) fut un problème difficile à résoudre. Car l'apprentissage de ces réseaux, par les algorithmes de construction ou la rétropropagation [174] nécessite une bonne qualité de la base d'apprentissage : elle doit être pertinente et bien renseignée. Elle doit contenir des informations sur toutes les zones où l'on veut que la réponse du réseau soit fiable. Or notre objectif est de construire un réseau de neurones qui réponde 0 à l'intérieur d'une zone assez bien délimitée mais généralement pas convexe ni connexe, et 1 partout ailleurs.

De telles contraintes nécessitent un traitement géométrique assez complexe pour générer des bases d'apprentissage pertinentes, et l'usage d'un algorithme de construction particulier : le BCPSL avec exclusion d'exemples de cible 1 uniquement, donc avec le BCP Max CF_1 pour apprendre chaque unité. L' emploi de neurones continus fut a priori impossible, car l'apprentissage avec la rétropropagation du gradient n'assure pas le respect des contraintes. De plus cet apprentissage est très lent et le calcul des réseaux plus lent que les binaires.

Avec l'usage du BCPSL, la contrainte imposée sur la base d'apprentissage est très allégée : elle doit contenir des informations uniquement sur la frontière. Plus la frontière est bien définie, meilleure est la réponse du réseau.

Le problème est donc de générer une base d'apprentissage modélisant la frontière entre le bon et le mauvais fonctionnement, à partir d'un nuage de points dont on connaît la distribution.

La méthodologie développée est basée sur les étapes suivantes. Premièrement le nuage de points est partitionné en groupes par un algorithme de classification, ce qui permet de résoudre en partie les problèmes de non convexité et de non connexité. Ensuite chaque groupe de points est traité séparément. Un réseau est construit pour chaque groupe. La réponse d'un de ces réseaux doit être 1 si on se trouve à l'extérieur du groupe et 0 à l'intérieur. Pour chaque groupe, la création de la base d'apprentissage nécessite d'abord le calcul de l'enveloppe convexe des points, et ensuite la génération de deux ensembles de points (un de mauvais fonctionnement et un de bon fonctionnement) à une distance réglable. Après leur apprentissage, ces réseaux sont alors associés dans une architecture globale, en combinant leur sortie par un ET (la sortie est 1 si et seulement si la sortie de tous les réseaux est 1), comme le montre le schéma global de la figure 6.7.

Pour faire du diagnostic préventif, l'utilisation des réseaux binaires est à déconseiller, car la quantification du bon fonctionnement est un problème difficile : il faut calculer la distance d'un point aux frontières d'un ensemble constitué de polyèdres qui peuvent être non convexes et non connexes.

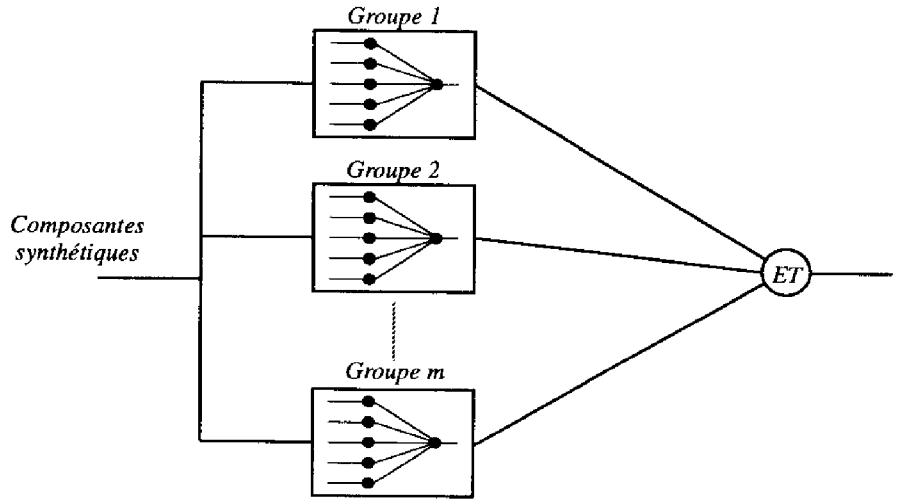

FIG. 6.7: Réseaux de neurones binaires

Classification

Dans une première phase d'étude nous avons utilisé l'algorithme *k-mean* basé sur une maximisation de l'inertie interclasse [65, 57]. Bien que cette méthode fournisse une partition sans recouvrement, ce qui est un avantage, elle a été abandonnée car elle a intrinsèquement tendance à donner des groupes sphériques. Dans beaucoup de cas de distributions non organisées approximativement en groupes sphériques, les résultats ne sont pas pertinents. Le critère utilisé n'est donc pas toujours le plus adapté. Dans nos expérimentations, nous avons obtenu des distributions organisées en groupes très longitudinaux, très filiformes et absolument pas sphériques. Ces structures semblent provenir de la cartographie interne du calculateur d'injection. Ceci nous a amené à chercher d'autres méthodes plus adaptées.

Un critère beaucoup plus intéressant à minimiser est la somme des hypervolumes de chaque groupe. Il existe quelques algorithmes basés sur ce critère. Ils sont complexes et très lents. En pratique leur utilisation est limitée à de faibles dimensions (inférieures à trois).

En visualisant les iso-pdf de la densité de probabilité fournie par les algorithmes détaillés dans la section précédente, nous avons pensé utiliser directement le mélange de gaussiennes pour partitionner le nuage de points. Ainsi, à la $k^{éme}$ gaussienne du mélange est associée une classe P_k d'éléments de l'échantillon par le procédé :

$$P_k = \{x_i / p_k(xi) = \max_{l=1}^{q} p_l(x_i)\}$$

Une caractéristique de cette méthode de classification est la possibilité de chevauchement de certains groupes. En effet, lorsque les données sont organisées en groupes compacts contenant la majorité des éléments plus quelques éléments non structurés et éparpillés de façon plus ou moins aléatoire, chaque groupe structuré est parfaitement identifié par une gaussienne, et les éléments épars sont partagés par les autres gaussiennes. Les résultats fournis par cette méthode se sont avérés extrêmement pertinents (voir figure 6.18 de la section 6.7). Le résultat de cette classification, obtenu de manière automatique, est comparable à celui que pourrait donner une expertise humaine, et assez proche d'une minimisation de l'hypervolume, pour divers types de distributions. Une comparaison de cette méthode et de l'algorithme k-mean est illustrée dans la section 6.7 (figures 6.19 et 6.20).

Calcul d'enveloppes convexes

Pour chaque groupe, la génération de la base d'apprentissage nécessite de connaître uniquement les points de bon fonctionnement qui définissent la frontière de ce groupe. Ces points peuvent être définis comme les éléments de l'enveloppe convexe du groupe, car celle-ci permet de délimiter la zone occupée par le groupe. Ce calcul permet une très grande réduction du nombre de points à utiliser. La frontière est ainsi définie par un minimum de points.
L'*enveloppe convexe* d'un ensemble de points P est *la frontière topologique du convexe de volume minimal contenant tous les éléments de* P. Dans le cas où P est un ensemble discret de points, cette frontière est un polyèdre, et les éléments de P sont soit à l'intérieur soit sur la surface du polyédre. On le notera $C = Conv(P)$. L'ensemble des sommets de ce polyèdre noté $Extr(C)$ (points extrémaux) est un sous-ensemble de P. Nous avons donc $Extr(Conv(P)) \subset P$. La figure 6.8 représente l'enveloppe convexe d'un ensemble de points en dimension 2, c'est un polygône.

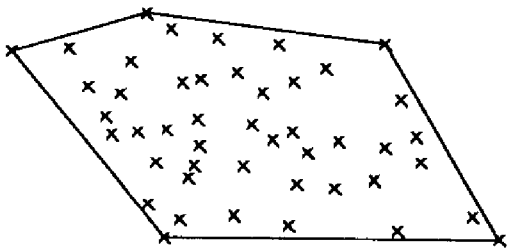

FIG. 6.8: Visualisation de l'enveloppe convexe d'un ensemble de points

Le calcul d'enveloppes convexes est un problème complexe largement étudié ces dix dernières années. De nombreux algorithmes ont été proposés en dimensions 2 et 3 (par exemple [63, 108, 81]) et aussi en dimension quelconque (par exemple [64, 41])
Dans une première phase d'étude, nous avons développé un algorithme de calcul d'enveloppes convexes basé sur le procédé de Graham [81]. Mais cet algorithme est limité à la dimension 2. Une équipe américaine du Geometry Center a développé un logiciel très performant basé sur de récentes recherches [11]. Il se nomme *Quick Hull* ou *qhull* et est disponible gratuitement par le réseau Internet. Son utilisation nous a épargné le développement d'un logiciel qui aurait nécessité beaucoup de temps.
Ce logiciel est très complet. Il calcule les points de l'enveloppe convexe, ainsi que les éléments des facettes du polyèdre constituant l'enveloppe. Mais il peut fournir beaucoup d'autres données, dont plusieurs nous étaient nécessaires. En particulier : les hypersurfaces et les normales aux facettes. De plus, ce logiciel intègre un contrôle des problèmes de précision numérique très critiques dans ce domaine, et il peut calculer des approximations de l'enveloppe convexe, ce qui est très utile lorsqu'il y a beaucoup de points.

Génération des bases d'apprentissage

Le problème est de générer, à partir de l'enveloppe convexe de chaque groupe, deux ensembles de points autour de cette enveloppe : un ensemble de bon fonctionnement et un de mauvais fonctionnement. Cette génération, qui apparaît comme un élargissement de l'enveloppe convexe, est soumise à deux contraintes très importantes. Les points doivent être répartis le plus uniformément possible et la distance séparant ces ensembles de l'enveloppe d'origine, ainsi que la distance séparant ces deux enveloppes doivent être uniformes tout autour de l'enveloppe et réglables : c'est le moyen de régler la sensibilité du réseau final.

Pour chaque groupe P_k avec $k \in \{1, \ldots, q\}$, on notera $C_k = Conv(P_k)$ et $S_k = Extr(C_k)$. La dimension de l'espace de travail est d. Une facette f (plus précisément une $d-1$ - facette) de C_k est aussi un polyèdre convexe de dimension afffine $d-1$, et l'ensemble de ses sommets $Extr(f) \subset P$ est de cardinal supérieur ou égal à d. Dans le cas où le nombre de sommets d'une facette est égal à d, elle est appelée *un simplexe*. Mais ce n'est pas toujours le cas. L'ensemble des facettes de C_k sera noté $Fac(C_k)$.
Le logiciel *qhull* calcule l'enveloppe convexe sous la forme d'un ensemble de simplexes, et utilise un procédé de fusion des simplexes coplanaires (ou approximativement coplanaires).
La même méthode est utilisée pour générer les deux ensembles de points, mais avec une distance à l'enveloppe convexe différente. Cette méthode procède en deux étapes :

- Génération d'un premier ensemble de points D_k à partir des éléments de S_k tel que la distance de ces points à C_k est constante, soit $\forall p \in D_k \quad d(p, C_k) = \min_{x \in C_k} d(p, x) = \mathcal{C}\, d_r$. La distance d_r est une distance de référence dont le calcul sera détaillé plus loin.

  Soit p un sommet de C_k et $\mathcal{A}(p)$ l'ensemble des facettes adjacentes à p, c'est-à-dire que p est un point extrême de ces facettes, soit

  $$\mathcal{A}(p) = \{f \in Fac(C_k) / p \in Extr(f)\}$$

  Pour une facette quelconque f, la normale à f notée n_f est le vecteur unitaire perpendiculaire à cette facette et orienté vers l'extérieur du polyèdre. Pour chaque facette de $\mathcal{A}(p)$ on calcule alors le point p_f défini par

  $$p_f = p + \mathcal{C}\, d_r\, n_f$$

  Pour chaque facette adjacente à p, le point p_f est à l'extérieur du polyèdre à une distance $\mathcal{C}\, d_r$ et tel que le vecteur $p_f - p$ soit perpendiculaire à la facette.

  Afin d'éviter des problèmes d'élargissement de la zone d'activation du réseau aux points très anguleux du polyèdre, un autre point p_c est calculé. Il est positionné à une distance $\mathcal{C}\, d_r$ sur l'axe liant le sommet p au barycentre des points p_f,

  $$p_c = p + \mathcal{C}\, d_r \frac{b_p - p}{||b_p - p||} \qquad \text{avec} \qquad b_p = \frac{\sum_{f \in \mathcal{A}(p)} p_f}{\text{Card}\,(\mathcal{A}(p))}$$

  Les points $\{p_f / f \in \mathcal{A}(p)\} \cup \{p_c\}$ se trouvent donc sur une hypersphère de rayon $\mathcal{C}\, d_r$ et de centre p. L'ensemble D_k est alors l'union des points p_f et p_c pour toute facette f dans $\mathcal{A}(p)$ et tout p dans S_k, soit

  $$D_k = \cup_{p \in S_k} \quad \{p_f / f \in \mathcal{A}(p)\} \cup \{p_c\}$$

  Avec un tel procédé, illustré par la figure 6.9 en dimension 2, tous les points de D_k sont des sommets de l'enveloppe convexe de D_k, c'est-à-dire $Extr(Conv(D_k)) = D_k$. Le polyèdre $E_k = Conv(D_k)$ est donc construit en déplacant vers l'extérieur toute facette de C_k parallèlement à elle-même et en *arrondissant* aux sommets de C_k. Ce nouveau polyèdre a beaucoup plus de sommets et de facettes que C_k.

- Génération d'un ensemble de points R_k dit points de *régularisation* sur les facettes de E_k. Ils sont ajoutés pour avoir une distribution de points assez uniforme sur toute la surface de E_k. On notera $\mathcal{S}(f)$ la surface (en fait l'hypersurface) d'une facette f.

  On calcule une surface de référence qui est la moyenne des surfaces des facettes de E_k, soit

  $$\mathcal{S}_r = \frac{\sum_{f \in Fac(E_k)} \mathcal{S}(f)}{\text{Card}\,(Fac(E_k))}$$

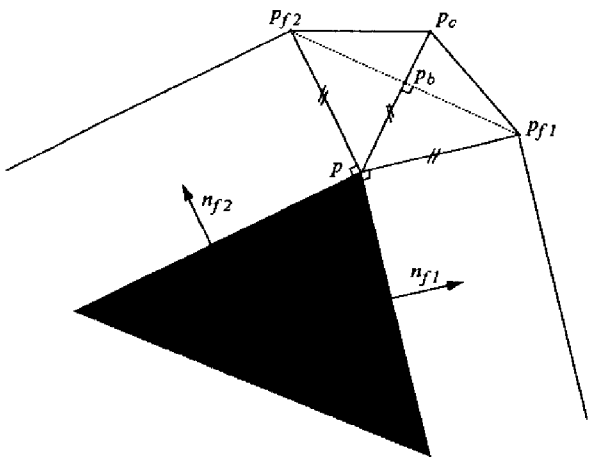

FIG. 6.9: Génération des points de l'ensemble D_k

Le nombre de points ajoutés sur une facette f est la partie entière de la quantité $\frac{\mathcal{S}(f)}{\mathcal{S}_r C_r}$.

Plus la surface d'une facette est importante, plus le nombre de points ajoutés sur cette facette sera grand. Le coefficient C_r est nommé coefficient de régularisation. Il est identique pour tous les groupes et permet de régler de manière uniforme le nombre de points ajoutés. Chaque point ajouté est un barycentre des sommets de la facette pondérés par des coefficients positifs. Le choix de ces coefficients avec un générateur aléatoire uniforme entre 0 et 1 n'a pas donné de très bons résultats. La densité de points était plus élevée vers le centre de la facette. En revanche le produit de plusieurs générateurs de même type a donné de meilleurs résultats (nous en avons utilisé 4), car la densité de la variable aléatoire ainsi obtenue est plus importante vers les valeurs proches de zéro. La répartition des points est ainsi plus uniforme sur la surface de la facette. Bien que le cas particulier des facettes simpliciales ne semble pas trop complexe, une solution exacte à ce problème pour des facettes quelconques paraît difficile.

Le coefficient C est le coefficient de réglage de sensibilité, qui sera différent suivant que l'on calcule les points de bon fonctionnement ou les points de mauvais fonctionnement.
Pour la génération des deux ensembles, la même distance de référence d_r est utilisée et est calculée par le procédé suivant :

- Calcul du barycentre $b_k(f)$ des sommets d'une facette f d'un groupe C_k :

$$b_k(f) = \frac{\sum_{p \in Extr(f)} p}{\text{Card}\,(Extr(f))}$$

- Estimation du barycentre b_k de l'enveloppe convexe de C_k par le barycentre des $b_k(f)$ pondéré par les $\mathcal{S}(f)$:

$$b_k = \frac{\sum_{f \in Fac(C_k)} \mathcal{S}(f)\, b_k(f)}{\sum_{f \in Fac(C_k)} \mathcal{S}(f)}$$

- d_r est alors défini par la moyenne des distances entre les $b_k(f)$ et les b_k sur tous les groupes :

$$d_r = \frac{\sum_{k=1}^{q} \sum_{f \in Fac(C_k)} d(b_k, b_k(f))}{\sum_{k=1}^{q} \text{Card}\,(Fac(C_k))}$$

La base d'apprentissage sera donc constituée de l'ensemble de points de bon fonctionnement $D_k^0 \cup R_k^0$ générés avec $C = C_0$ (ils auront 0 comme cible), et de l'ensemble des points de mauvais fonctionnement $D_k^1 \cup R_k^1$ générés avec $C = C_1$ (ils auront 1 comme cible). Il faut nécessairement que $C_1 > C_0$.
La génération de ces bases d'apprentissage est donc uniquement paramétrée par les trois coefficients C_1, C_0 et C_r. Ils permettent de régler la sensibilité du réseau et la précision de sa réponse.

- La *sensibilité* dépend des deux premiers paramètres, car ceux-ci définissent une zone interdite entre C_k et les points de bon fonctionnement générés. Aucun hyperplan du réseau de neurones ne passera dans cette zone. Plus C_1 et C_0 sont faibles plus cette zone interdite sera faible et plus le réseau sera sensible. Dans le cas limite où ces deux paramètres sont pratiquement nuls, la limite de la zone d'activation du réseau d'un groupe sera exactement l'enveloppe convexe de ce groupe. Donc la limite de la zone de bon fonctionnement du réseau global sera la superposition des zones polyèdrales définies par les enveloppes convexes des groupes.

- la *précision* de la réponse d'un réseau dépend des trois paramètres. Elle est très grande quand $C_1 \simeq C_0$ et C_r est très faible. En pratique $C_r = 0.4$ donne une bonne uniformisation des points. La valeur prise par les deux autres paramètres dépend de la sensibilité désirée, mais on peut dire qu'un écart relatif de 20% donne en pratique une bonne précision de la réponse.

La figure 6.10 illustre la génération des points de bon et de mauvais fonctionnement en dimension 2. Ils sont respectivement représentés par les symboles "o" et "+". Les points des ensembles D_k^0 et D_k^1 sont les points liés à l'unique sommet de C_k représenté sur la figure, les autres étant des points de régularisation.

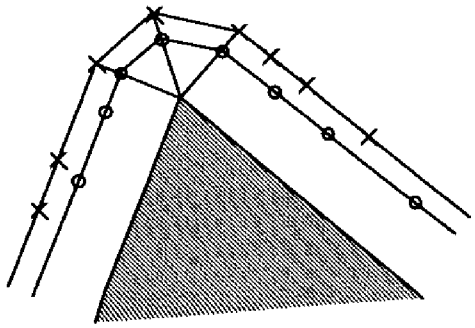

FIG. 6.10: Génération des points de bon et mauvais fonctionnement

Bien que cette méthode de génération de base d'apprentissage semble complexe, elle est effectuée très rapidement. Le nombre de sommets des enveloppes convexes des groupes est en général très inférieur au nombre de points des groupes (par exemple pour des groupes de 1000 à 2000 points en dimension 2 ou 3, nous avons obtenu très souvent un nombre de sommets allant de 5 à 100). Cette très importante

réduction du nombre de points compense donc largement l'augmentation dûe à la génération des ensembles $D_k^0 \cup R_k^0$ et $D_k^1 \cup R_k^1$. En revanche pour des dimensions supérieures ou égales à 5, le nombre de facettes des enveloppes convexes devient très vite important, donc le nombre de points de la base d'apprentissage aussi. En pratique nous pouvons dire que la limite de cette méthode se situe environ à la dimension 5 ou 6.
Les temps de calcul les plus importants sont les calculs des trois enveloppes convexes (calcul de C_k et calcul de E_k pour les deux ensembles). Mais même avec des groupes de plusieurs milliers de points en dimension allant jusqu'à 4 ou 5, le logiciel *qhull* a toujours donné ses résultats en moins d'une quinzaine de secondes (illustration de la méthode sur les figures 6.21 et 6.22 de la section 6.7).

Apprentissage BCPSL CF_1

Le BCPSL est utilisé pour apprendre les bases d'apprentissage de chaque groupe mais avec une contrainte en plus : la cible des exclus doit obligatoirement être 1, car aucun hyperplan ne doit couper la zone de bon fonctionnement. Sinon il y aurait un risque d'avoir des zones de mauvais fonctionnement à l'intérieur. Donc le BCPSL est utilisé avec le BCP Max CF_1. Cette contrainte est très forte sur l'algorithme, et la convergence n'est plus assurée. Comme les points de cible 0 se trouvent sur un convexe qui est à l'intérieur du convexe sur lequel se trouvent les points de cible 1, en théorie un ensemble d'exclus de cible 1 peut toujours être trouvé. Mais il peut y avoir des configurations des deux barycentres qui rendent difficile l'exclusion de points de cible 1. Dans nos expériences plusieurs bases d'apprentissage ont posé des problèmes, à partir de la dimension 4 (et quelques unes en dimension 3). Une solution à ce problème est d'utiliser une seconde méthode de construction en collaboration avec BCPSL, lorsque celui-ci a des problèmes de convergence.

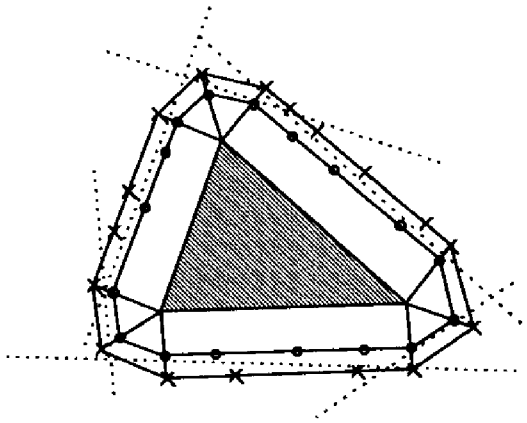

FIG. 6.11: Construction du réseau de neurones pour un groupe de points

Cette seconde méthode est purement géométrique. Comme l'illustre la figure 6.11, la construction du réseau de neurones consiste à placer un ensemble d'hyperplans tel que tous les points de cible 0 se trouvent du côté négatif de l'hyperplan et un maximum de points de cible 1 du côté positif. Ces hyperplans peuvent être construits géométriquement à partir de chacune des facettes. Quand il converge, l'avantage du BCPSL par rapport à un tel procédé est l'optimisation du nombre d'hyperplans utilisés. Mais cette méthode peut être utilisée pour relancer le BCPSL lorsqu'il n'arrive plus à exclure des exemples. La convergence est alors assurée. C'est une technique similaire à l'utilisation des neurones *grandmères* pour assurer la convergence des algorithmes comme Upstart. La réponse globale d'un

réseau est visualisée sur la figure 6.24 de la section 6.7, avec l'ensemble des bases d'apprentissage des divers groupes 6.23.

6.5 Organe d'identification

6.5.1 Généralités

Les modules présentés dans les paragraphes précédents permettent de détecter divers défauts grâce aux réseaux de neurones gaussiens ou binaires. Certains défauts pères seront directement identifiés par les réseaux du type 2 (détection par inclusion). Ce cas de figure est très rare. D'une manière générale, les réseaux de neurones donneront un ensemble de défauts élémentaires plus ou moins ciblés.
La détection de défauts élémentaires avec les méthodes simples, utilisées dans la première maquette développée, fournit des informations qu'il est indispensable d'utiliser pour l'identification des défauts pères. En particulier, les défauts élémentaires prédiagnostiqués soit par les calculateurs, soit par le système d'acquisition, soit par le module de prétraitements. En sortie du générateur de signaux, ces informations sont des données booléennes non prises en compte dans l'ACP. Ce sont par exemple : l'absence d'étincelles, divers tests de tensions ou de masse, etc... Le module de détection simple génère à partir de ces données, et des données quantitatives, un ensemble de défauts élémentaires.
Ces derniers sont alors concaténés avec les divers défauts générés par les réseaux, en *un vecteur de défauts élémentaires*, fourni au module d'identification. *L'objectif de ce dernier est d'identifier un défaut père à partir de ce vecteur.* La technique mise en oeuvre est un peu une synthèse des avantages du système Filtrage-Gestionnaire Hiérarchie développé dans la maquette (sans ses inconvénients), et des derniers développements présentés dans ce rapport basés sur l'ACP et la détection de points dans des zones de forme quelconque dans des sous-espaces synthétiques.
Comme l'illustre la figure 6.12, la phase d'identification est basée sur le calcul d'un vecteur de probabilités d'apparition nommé *défautgramme*. Le calcul des défautgrammes de chaque panne (artificielle ou simulée) est réalisé en temps différé. Ces défautgrammes constitueront des *signatures* de chaque père. Une ACP est réalisée sur ces défautgrammes de pannes et analysée. Cette ACP permet alors de définir un changement de variables à réaliser et les meilleurs composantes à prendre en compte pour identifier chaque père sans ambiguïté. Comme nous allons le voir, les divers modules identifieurs seront très simples.

6.5.2 Calcul des défautgrammes

Un défautgramme est le *vecteur des probabilités d'apparition des défauts élémentaires dans une fenêtre glissante*. Le calcul d'un défautgramme se fait donc à chaque trame, mais dépend de plusieurs trames précédentes.
Soit $e^t = (e_1^t, \ldots)$ le vecteur des défauts élémentaires à l'instant t. Le nombre de ces défauts sera noté n_e. Nous avons donc $e^t \in \{0,1\}^{n_e}$, avec 1 signifiant défaut et 0 bon fonctionnement. Dans nos notations le temps t est en fait le numéro de la trame. On définit alors le défautgramme à l'instant t noté $g^t = (g_1^t, \ldots)$ par :

$$\forall i \in \{1, \ldots, n_e\} \qquad g_i^t = \frac{\sum_{j=t-T+1}^{t} e_i^j}{T}$$

Avec T la largeur de la fenêtre glissante exprimée en nombre de trames. Nous avons donc $g^t \in [0,1]^{n_e}$. De plus afin de filtrer encore les divers bruits des systèmes, si la somme des probabilités est inférieure à un seuil, le défautgramme est mis entièrement à zéro. En fait, ce n'est pas indiqué sur le schéma, mais dans une telle situation, on considère qu'il s'agit de bon fonctionnement et le processus d'identification

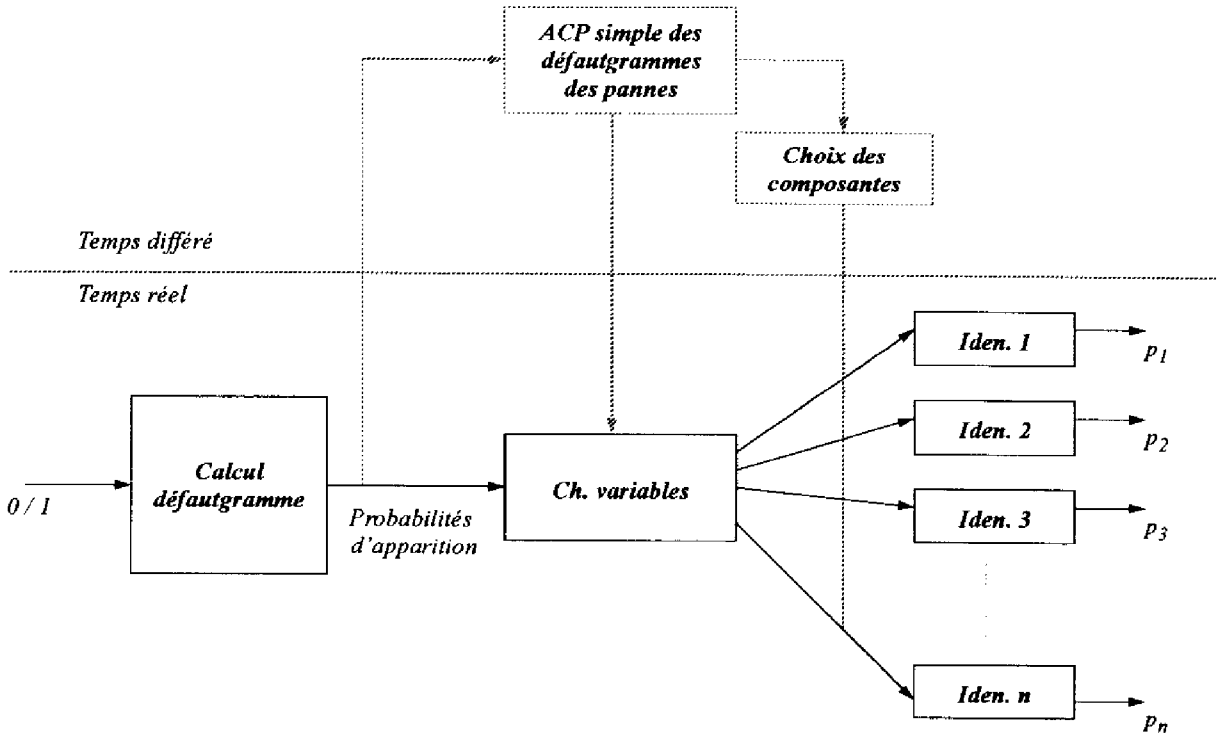

FIG. 6.12: Schéma fonctionnel de l'organe d'identification

se termine après le calcul du défautgramme. La valeur de ce seuil, qui doit être inférieure à 1, dépend de la largeur de la fenêtre, et du nombre de défauts élémentaires. En pratique nous avons pris $T = 50$, et pour une trentaine de défauts élémentaires le seuil à 0,1. Ce calcul de défautgrammes est similaire au filtrage utilisé dans la maquette mais sans le seuillage.

6.5.3 ACP des défautgrammes et identifieurs

L'utilisation de ces défautgrammes est très intéressante car ils sont constitués de données analogiques, donc aptes à être traitées par une ACP. Dans ce nouvel espace, l'ensemble de bon fonctionnement est réduit à un point, qui ne sera jamais considéré dans l'analyse. Contrairement aux modules de détection modélisant l'espace de bon fonctionnement, cet organe d'identification va travailler uniquement sur le mauvais fonctionnement.

Chaque défaut père occupe une certaine zone suivant une direction particulière. Pour mettre au point le module d'identification, il est nécessaire de localiser ces zones à partir des défautgrammes obtenus avec les pannes artificielles ou simulées. Cette localisation pourrait être faite dans l'espace entier de dimension n_e, mais cela rendrait le système très lent. Pour réduire la dimension de l'espace de travail et par conséquent les temps de calcul, sans perte d'information, l'ACP est utilisée sur toute la base de données des défautgrammes des pannes. Les quelques expériences menées montrent que cette analyse fournit des sous-espaces de dimension faible (2 ou 3) où certains défauts pères sont parfaitement dissociés entre eux et d'autres. Plus généralement, pour chaque défaut père, nous avons toujours trouvé un sous-espace dans lequel il était complètement séparé des autres, et formant une zone compacte (voir les figures de 6.30 à 6.34 de la section 6.7). Cette dernière est facilement modélisable

par une seule gaussienne ou un réseau binaire simple qui constituera l'identifieur.
En résumé, ce module est basé sur la procédure suivante :

- calcul des défautgrammes des pannes,
- ACP simple de la base ainsi générée,
- pour chaque défaut père, recherche du sous-espace le plus discriminant,
- calcul de la gaussienne ou du réseau modélisant la zone occupée par le défaut père,

Lorsque deux défauts pères très semblables ne pourront pas être séparés dans tous les sous-espaces synthétiques, il suffit alors d'ajouter au système de détection de défauts élémentaires, un réseau de neurones sensible à un et un seul de ces deux défauts. Ce nouveau réseau augmentera les différences entre les vecteurs de défauts élémentaires pour ces deux pères, et ils seront discriminés plus facilement. Dans un tel cas, il faut relancer toute la procédure décrite ci-dessus.
Cette modélisation de chaque nuage correspondant à une panne par une gaussienne permet alors d'appliquer directement la théorie de la décision bayésienne dans chaque plan : chaque gaussienne est bien sûr normalisée et correspond à la densité conditionnelle de chaque panne. Les probabilités *a priori* pourront être choisies égales entre elles au début, mais on peut imaginer qu'avec l'expérience ces probabilités puissent évoluer, traduisant ainsi les apparitions plus fréquentes de certaines pannes. La discrimination est alors très facile (voir chapitre 1). De plus la théorie bayésienne permet aussi de faire du rejet d'ambiguïté et d'obtenir un résultat en terme de probabilités, que l'utilisateur peut utiliser. De plus la possibilité de faire du rejet de distance peut permettre de détecter l'apparition de nouvelles pannes : ainsi quand plusieurs défautgrammes auront été rejetés en distance et qu'ils sont dans une zone assez bien définie (ce qui peut se chiffrer en calculant l'inertie du nuage), une nouvelle panne pourra être définie et intégrée au système. Il suffira alors de demander la cause symbolique à l'utilisateur.
Dans ce système d'identification, les problèmes de hiérarchie entre défauts pères sont gérés de manière implicite.

6.6 Conclusion et perspectives

Avant de conclure ce chapitre, il convient de faire une remarque sur le réglage de sensibilité des réseaux de neurones. On peut se demander s'il vaut mieux une marge de sécurité constante tout autour du nuage ou une dilatation simple du nuage (homothétie) qui fournit ainsi une marge de securité non constante, mais qui conserve la forme du nuage. En étudiant la position des points de mauvais fonctionnement des diverses pannes, nous avons observé qu'il n'y avait pas de position ou de direction privilégiées, liées à la forme du nuage. Ce qui semble logique, car une panne est par nature un fonctionnement anormal, et le fonctionnement des systèmes n'a donc plus les mêmes caractéristiques. Dans ces conditions, il nous a paru moins risqué de créer une marge de sécurité constante tout autour du nuage. Dans le cas des réseaux de neurones binaires, à cause des formes convexes variées des groupes (longitudinales ou sphériques) fournis par la classification par le mélange de gaussiennes, la création d'une marge de sécurité uniforme est impossible avec de simples homothéties. Nous n'avons pas eu le temps de bien étudier ce problème dans le cas des réseaux gaussiens, mais une marge de sécurité plus constante que celle obtenue avec la méthode proposée ne semble pas trop difficile à obtenir.

Ce chapitre synthétise les nouvelles méthodes utilisées pour créer des systèmes de diagnostic de pannes et de diagnostic préventif en ne considérant aucune information symbolique : tout le système est basé sur des données de bon et de mauvais fonctionnement. Les données de bon fonctionnement permettent

de faire du diagnostic préventif, et du diagnostic de pannes moyennement ciblé : du rapprochement fonctionnel. Les données de mauvais fonctionnement servent à mettre au point le système d'identification des diverses pannes susceptibles de se produire.
Cette méthodologie a de nombreux avantages :

- abstraction des systèmes diagnostiqués,
- généricité, vaste champ d'application,
- rapidité du système implanté pouvant être amélioré dans le cadre d'applications industrielles par ses hautes potentialités de parallélisme,
- sensibilités entièrement réglables,
- robustesse aux bruits divers,
- diagnostics multidimensionnels très précis,
- développement réalisable sur un PC.

L'objectif, relativement audacieux, a nécessité l'utilisation de méthodes assez complexes dans leurs descriptions. Mais leur complète automatisation rendra cette complexité transparente au développeur d'une application. On peut aussi citer deux problèmes qui restent à résoudre :

- dans le cas de la présence de deux défauts pères simultanément, et si ce cas est absent dans la base de pannes, la réponse de l'organe d'identification est imprévisible. Elle peut être bonne (les deux défauts sont identifiés), incomplète (un seul est identifié) ou mauvaise (aucun n'est identifié),

- pour les défauts qui se définissent non comme un état défaillant, mais comme une série d'états valables à chaque instant, mais dont la succession est un défaut.

Pour le second problème, l'introduction d'une procédure spéciale au niveau du module de prétraitement, permettra de détecter ce genre de défaut. Ce genre de traitement est d'ailleurs déjà réalisé pour certains signaux : commandes d'allumage, signal PMH (Point Mort Haut) qui est le signal de synchronisation de tous les systèmes,... Plusieurs caractéristiques de ces signaux sont calculées qui permettent de détecter des défauts élémentaires sur ces signaux (voir chapitre 2).
Plusieurs possibilités d'amélioration sont envisageables, en particulier l'utilisation de la méthode des *projections révélatrices*, et l'Analyse Factorielle Discriminante (AFD). Les principes de ces deux méthodes sont semblables à celui de l'ACP, avec des contraintes. La première tentera de trouver des sous-espaces particuliers de bon fonctionnement où des structures peuvent être trouvées sans être sensibles aux points extrêmes. Elle peut donc apporter une autre forme de filtrage et des informations complémentaires. Une méthode particulière de calcul de projections révélatrices est intéressante car elle fonctionne comme une ACP avec une métrique particulière calculée à partir des données [170, 35, 36]. Cette méthode ainsi que les études développées dans [119, 118], montrent l'importance de la métrique en ACP : suivant sa forme on peut faire ressortir diverses informations, dilater des axes principaux particuliers, ne pas être influencé par les points extrêmes... L'AFD est une méthode qui peut être utilisée sur toutes les données de mauvais fonctionnement : les données qualitatives et les données quantitatives, et peut de la même manière compléter les informations en prenant en compte les données qualitatives. D'autres méthodes d'analyse de données pourraient aussi être envisagées, notamment l'analyse en composantes indépendantes (ACI) [45] et l'analyse en composantes curvilignes (ACC) [52, 53], développée récemment.
La méthode employée dans l'organe d'identification permet aussi l'intégration facile d'une notion d'appartenance floue au domaine de mauvais fonctionnement. En effet, le vecteur d'entrée au lieu d'être booléen, pourrait être un vecteur de possibilité d'appartenance à l'espace de défaut. Pour engendrer ce

vecteur, il suffit de définir des règles floues dans le module de détection simple, et de ne pas seuiller les réseaux de neurones mais d'en déduire une possibilité de défaut. Ceci pourrait être fait par exemple en tronquant les gaussiennes et en réajustant les amplitudes au lieu de seuiller. Pour les réseaux binaires ceci peut se faire très simplement en transformant les unités à seuils par des unités linéaires sur un intervalle centré sur zéro et seuillé ailleurs, en remplacant pour chaque réseau élémentaire la sortie qui est en fait un OU logique par son équivalent flou (maximum des sorties des neurones) et de la même manière pour la sortie du réseau global qui est un ET (minimum des sorties des réseaux élémentaires). Bien que la génération des bases d'apprentissage des réseaux binaires en dimension quelconque et la construction des réseaux aient été complètement automatisées, une automatisation complète de toute la méthodologie n'a pas pu être réalisée par manque de temps. Mais certaines phases nous semblent très difficiles, notamment le choix des bonnes composantes pour la détection par réseaux comme pour l'organe d'identification. L'automatisation complète d'autres phases, comme l'apprentissage par l'EM et le filtrage nous semble envisageable. De la même manière, le choix automatique des seuils de détection dans le module de détection simple nous semble facilement réalisable par estimation de densités de probabilités monodimensionnelles, donc très rapide.

Pour conclure, nous pouvons dire que les méthodes présentes offrent de puissants moyens de détection et de diagnostic au sein d'une méthodologie basée sur le concept suivant :

> *"Les systèmes diagnostiqués ne sont pas modélisés, on ne fait qu'observer leurs entrées-sorties dans la plus grande zone de bon fonctionnement possible, et dans des zones particulières de mauvais fonctionnement que l'on désire identifier parfaitement".*

De nombreuses améliorations peuvent être apportées et certains points nécessitent des recherches complémentaires. Mais sur la base des travaux effectués le développement d'un *Générateur de Systèmes de Diagnostic* peut déjà être envisagé.

6.7 Résultats de simulations

Toutes les étapes de traitement détaillées dans ce chapitre sont présentées successivement. La base de bon fonctionnement utilisée comportait 15000 trames dans divers environnements de conduite, et la base de données de mauvais fonctionnement est constituée de 14 défauts pères obtenus par pannes artificielles.

- Figures de 6.13 à 6.15 : ACP et filtrage des données.
- Figures de 6.16 à 6.17 : Estimation d'une densité avec un mélange de gaussiennes.
- Figures de 6.18 à 6.20 : Classification avec le mélange de gaussiennes et comparaison de cette méthode avec l'algorithme k-mean sur un problème de classification.
- Figures de 6.21 à 6.22 : Génération des bases d'apprentissage pour un groupe (dimensions 2 et 3).
- Figures de 6.23 à 6.24 : Ensemble des bases d'apprentissage (dimension 2) et réponse globale du réseau construit avec ces bases.
- Figures de 6.25 à 6.26 : Illustration des possibilités pour le diagnostic préventif.
- Figures de 6.27 à 6.29 : Constitution de réseaux gaussiens et visualisation de la réponse dans les plans principaux avec les points de la base de données de mauvais fonctionnement.
- Figures de 6.30 à 6.34 : Identification des défauts pères dans les plans principaux issus de l'analyse en composantes principales des défautgrammes de pannes artificielles.

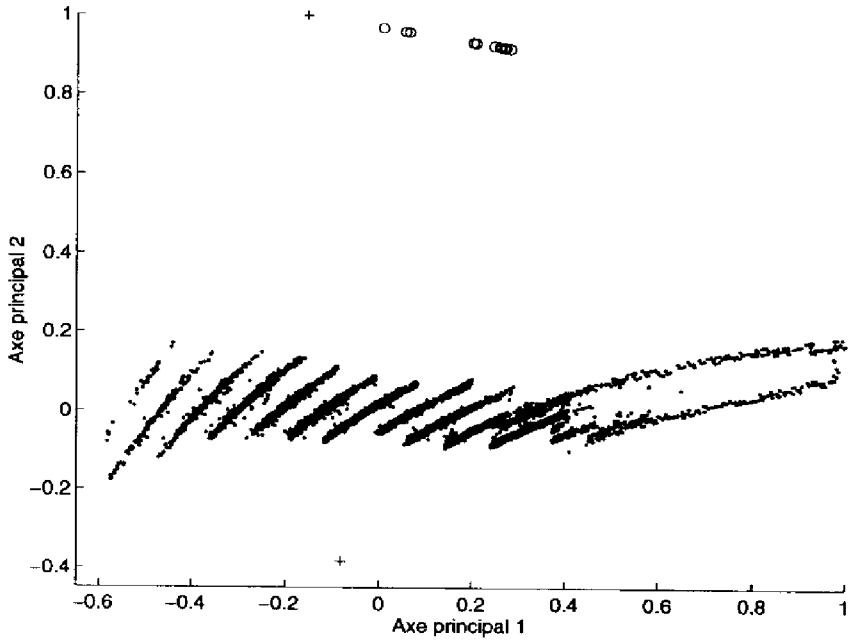

FIG. 6.13: Représentation des étapes de filtrage 1 et 2 dans le plan principal 1-2 de l'ACP. Le premier filtrage a enlevé les points représentés par des + et le second les points représentés par des *o*.

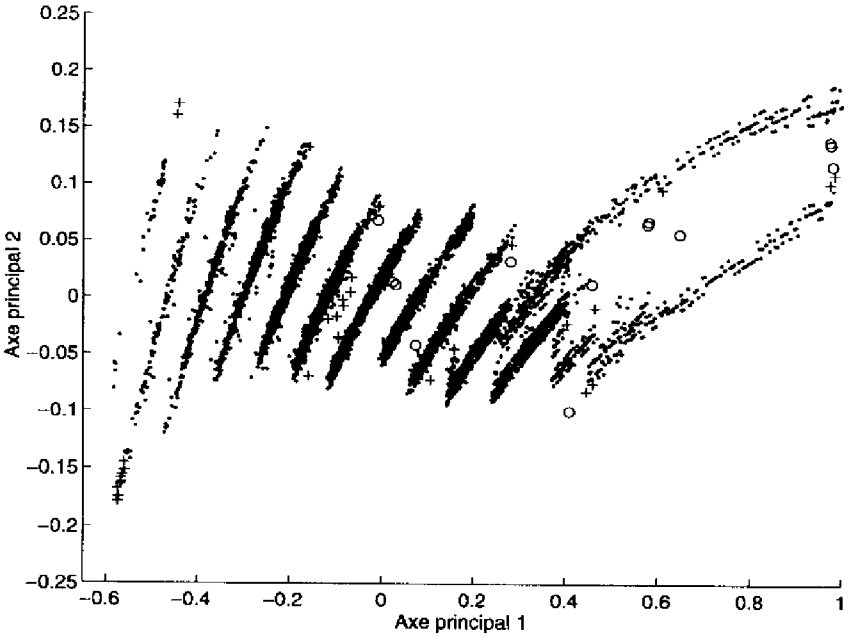

FIG. 6.14: Représentation des étapes de filtrage 3 et 4 dans le plan principal 1-2 de l'ACP. Le troisième filtrage a enlevé les points représentés par des *o* et le quatrième les points représentés par des +.

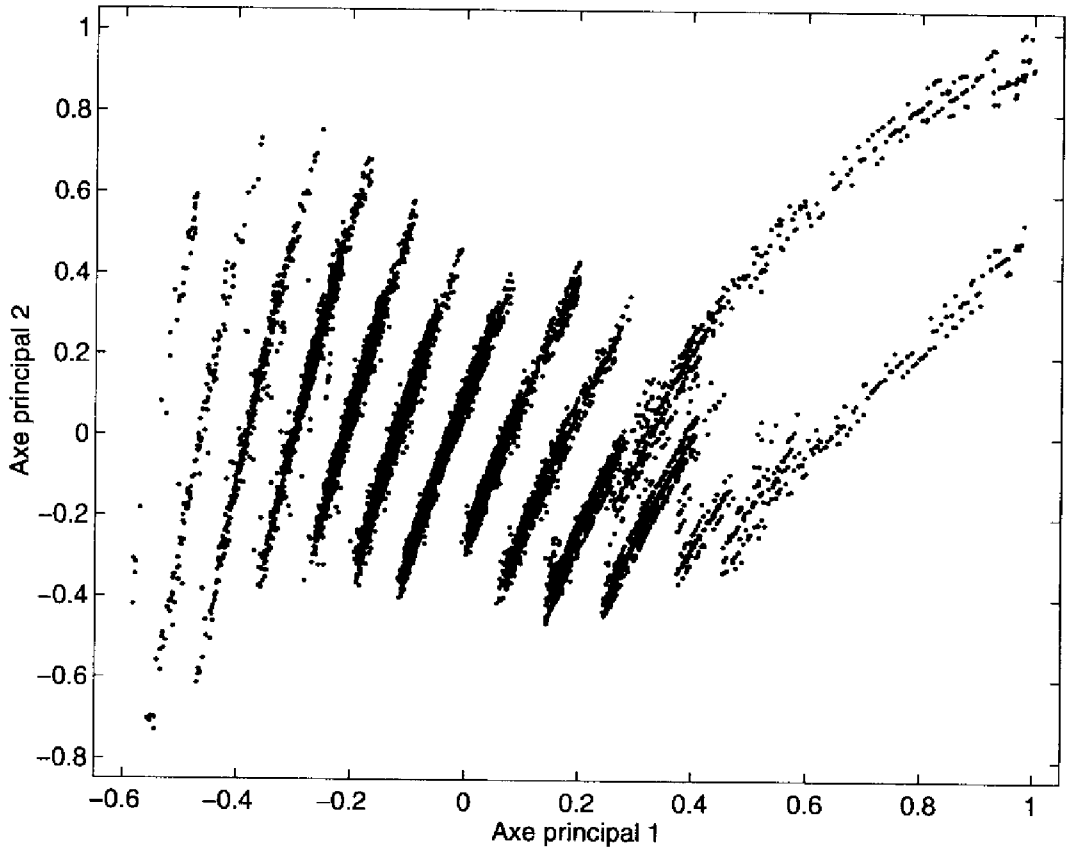

FIG. 6.15: ACP finale des données filtrées.

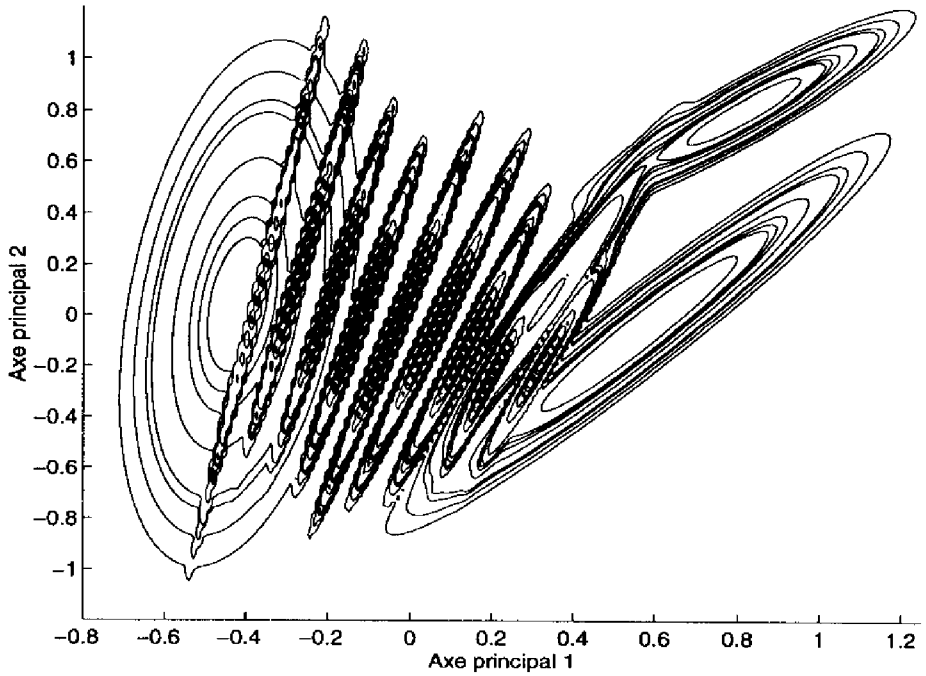

FIG. 6.16: Visualisation de l'estimation de densité par un mélange de gaussiennes. Les contours sont des iso-pdf.

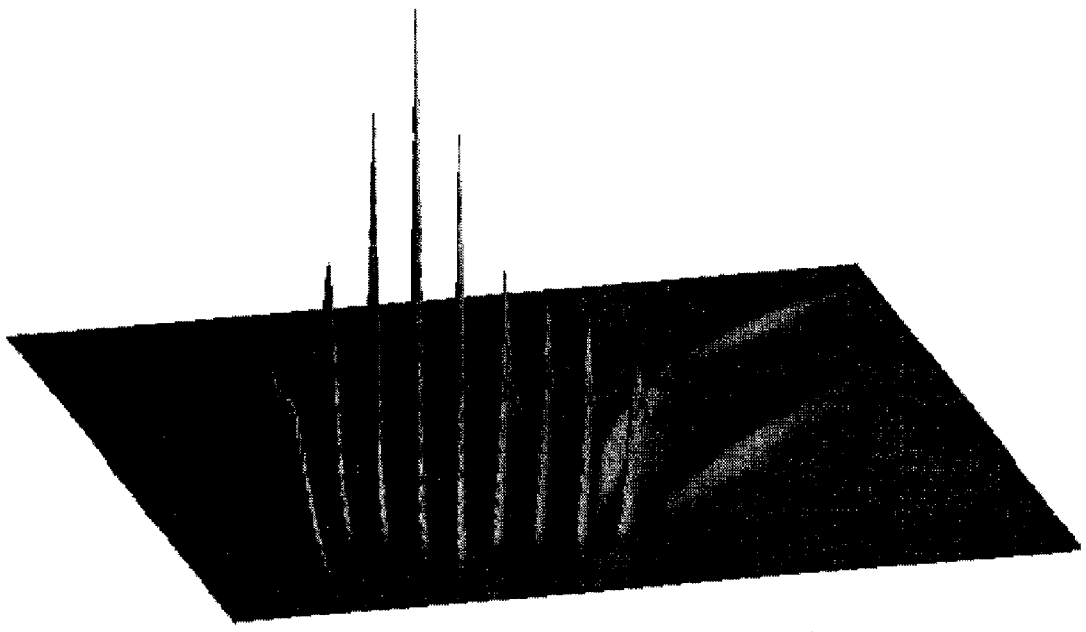

FIG. 6.17: Visualisation en trois dimensions de l'estimation de densité.

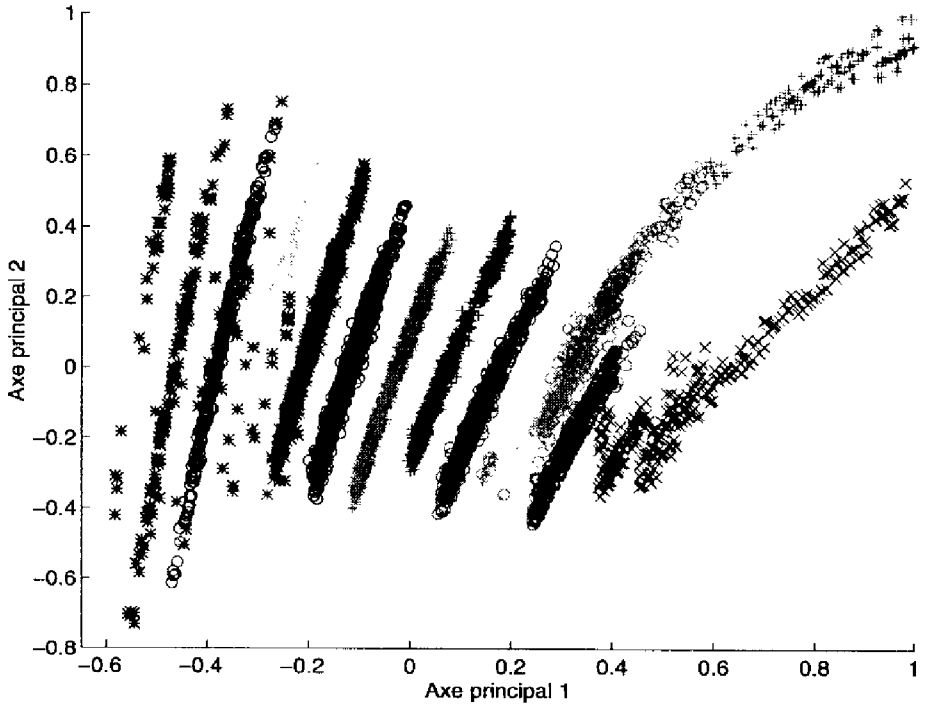

FIG. 6.18: Classification pertinente des données avec un mélange de gaussiennes.

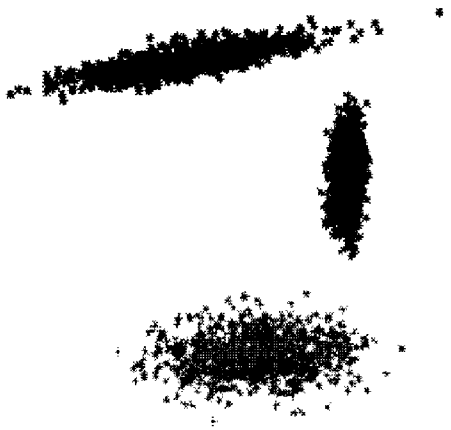

FIG. 6.19: Résultat d'une classification avec un mélange de gaussiennes.

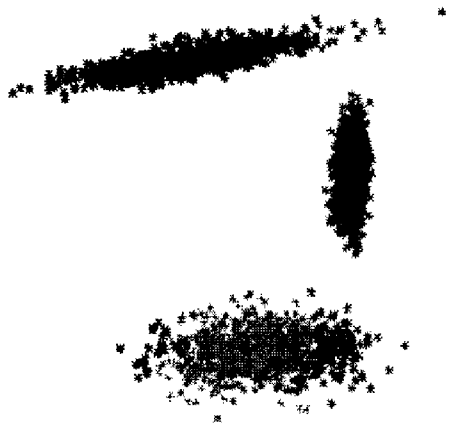

FIG. 6.20: Le même problème traité avec l'algorithme k-mean.

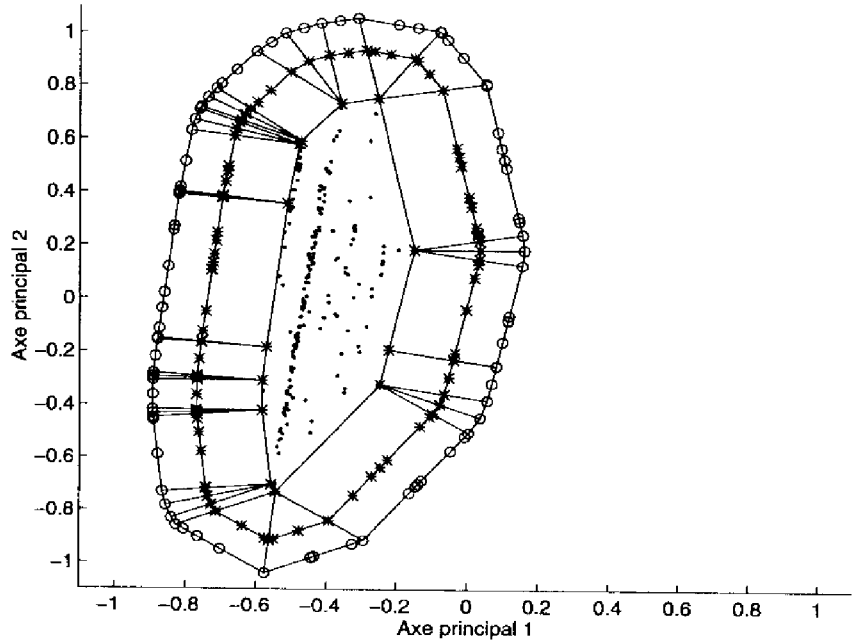

FIG. 6.21: Génération d'une base d'apprentissage pour un groupe en 2 dimensions. La ligne intérieure est l'enveloppe convexe des points du groupe, la ligne extérieure (resp. intermédiaire) est l'enveloppe des points générés de mauvais fonctionnement (resp. bon fonctionnement).

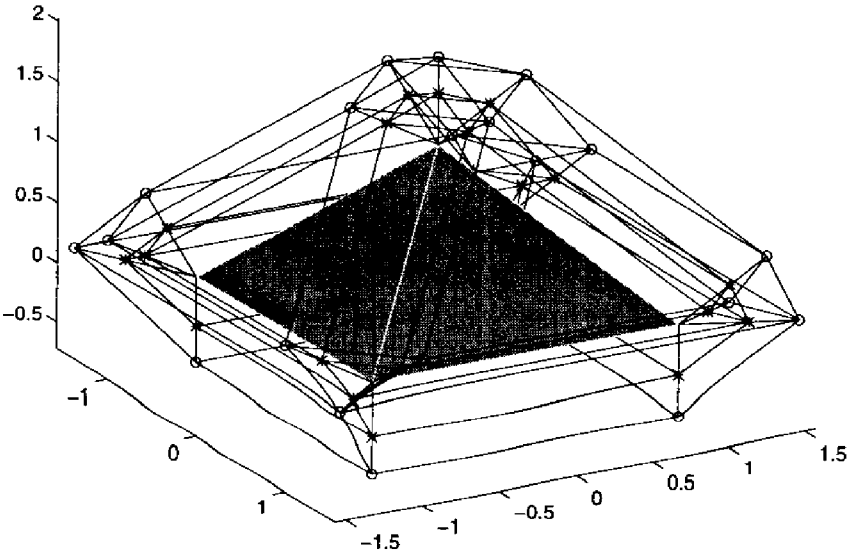

FIG. 6.22: Génération d'une base d'apprentissage d'un groupe simple en 3 dimensions. Pour ne pas compliquer le schéma les points de régularisation n'ont pas été représentés.

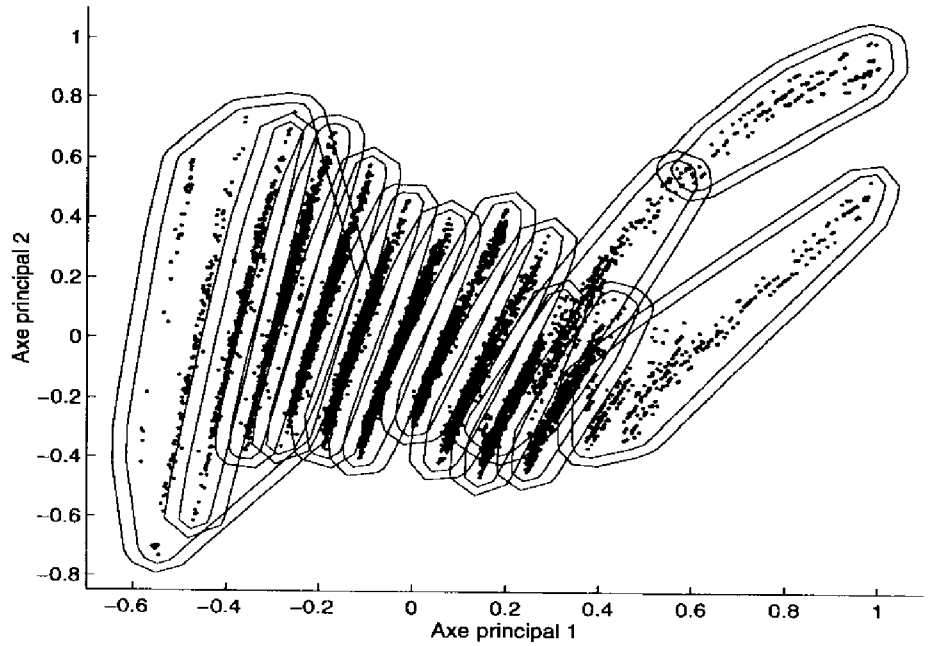

FIG. 6.23: Représentation des bases d'apprentissage pour tous les groupes. Les enveloppes convexes des ensembles de points générés de bon et de mauvais fonctionnement sont respectivement représentés par les polygones intérieurs et extérieurs.

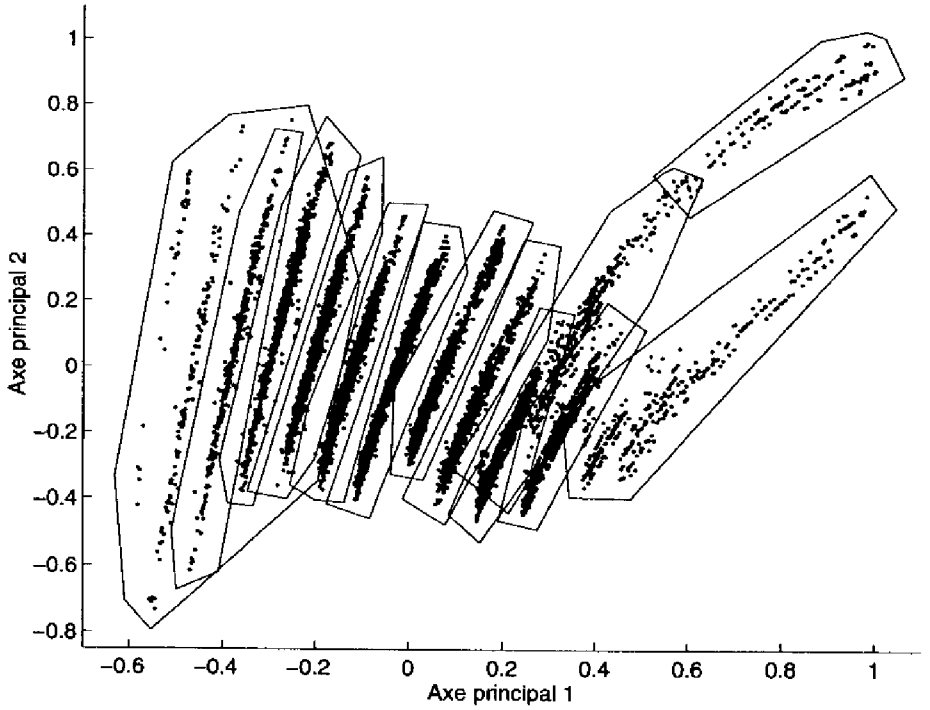

FIG. 6.24: Réponse globale du réseau. Il répond bon fonctionnement si le point est à l'intérieur d'un des polygônes et mauvais fonctionnement s'il se situe à l'extérieur de tous les polygônes.

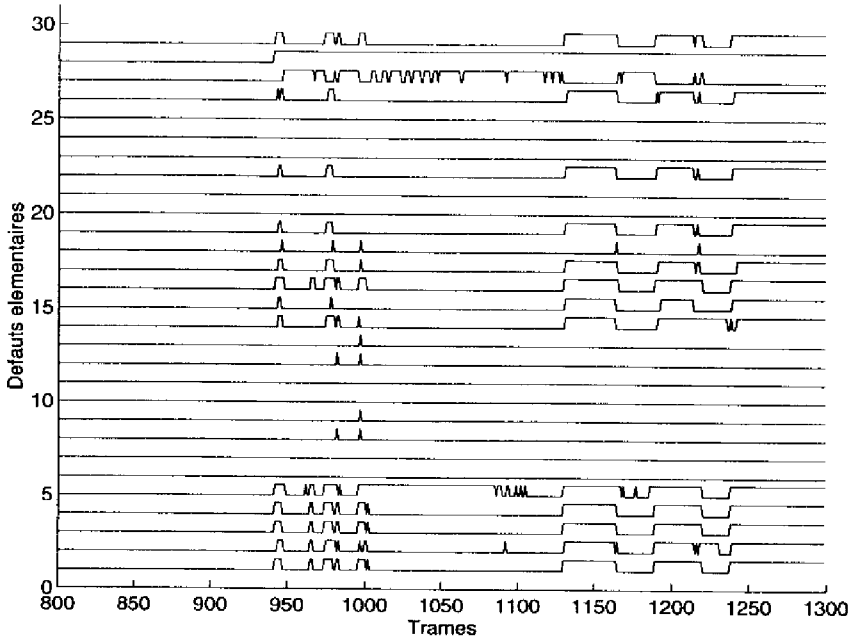

FIG. 6.25: Apparition des défauts élémentaires lors d'une panne d'essence, arrivée malencontreusement pendant les essais à cause d'une jauge défectueuse ! Les défauts élémentaires de 30 à 32 sont détectés par réseaux de neurones. Ils détectent des états de défauts non détectés par les autres diagnostics simples.

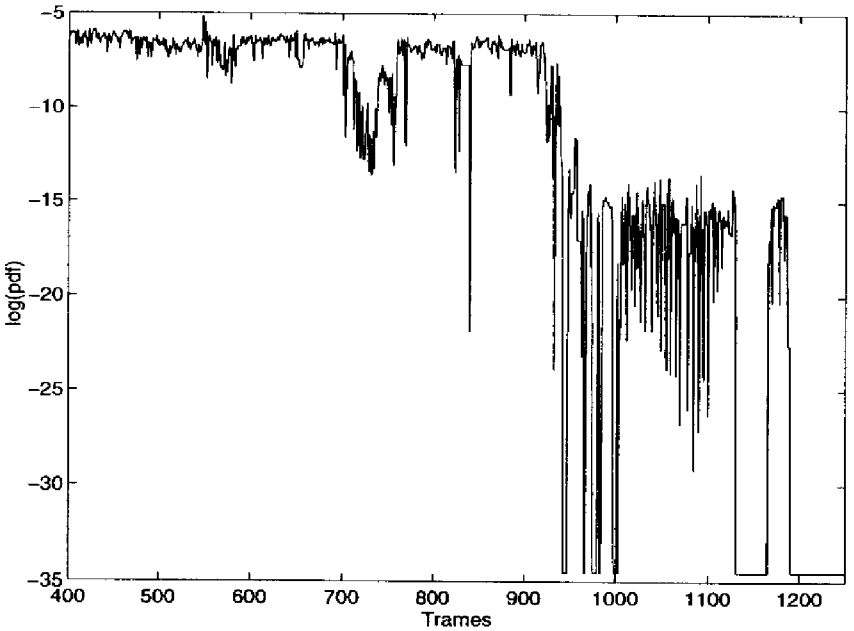

FIG. 6.26: Réponse d'un réseau gaussien durant la panne d'essence, appris avec les quatre premières composantes synthétiques représentant 95% de l'information. En comparant cette figure avec la figure précédente, on peut remarquer que le mélange de gaussiennes commence à détecter un problème dès $t = 700$, ce qui correspond à une trentaine de secondes avant les premiers effets visibles sur la voiture, environ à $t = 1150$.

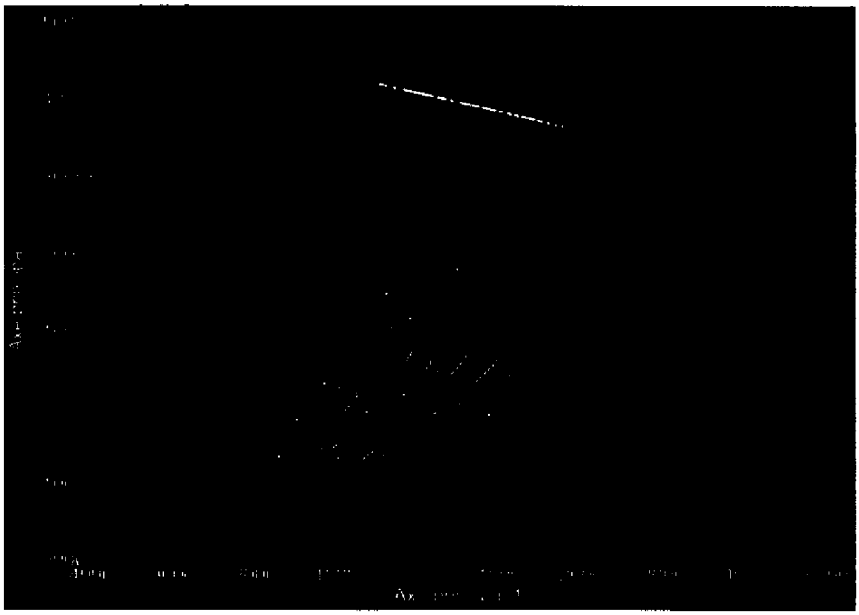

FIG. 6.27: Réseau gaussien appris sur les deux premières composantes synthétiques d'une ACP non normée. Ce réseau est sensible à un seul défaut père (points jaunes) relatif à l'allumage. Tous les autres défauts pères (points verts) ne sont pas détectés. Ce réseau est de type 1 : détection très ciblée par exclusion. Sur ce graphique, et ceux qui suivent, les points de bon fonctionnement ne sont pas répresentés, il y a seulement les projections des vecteurs d'états de 14 fichiers de pannes artificielles.

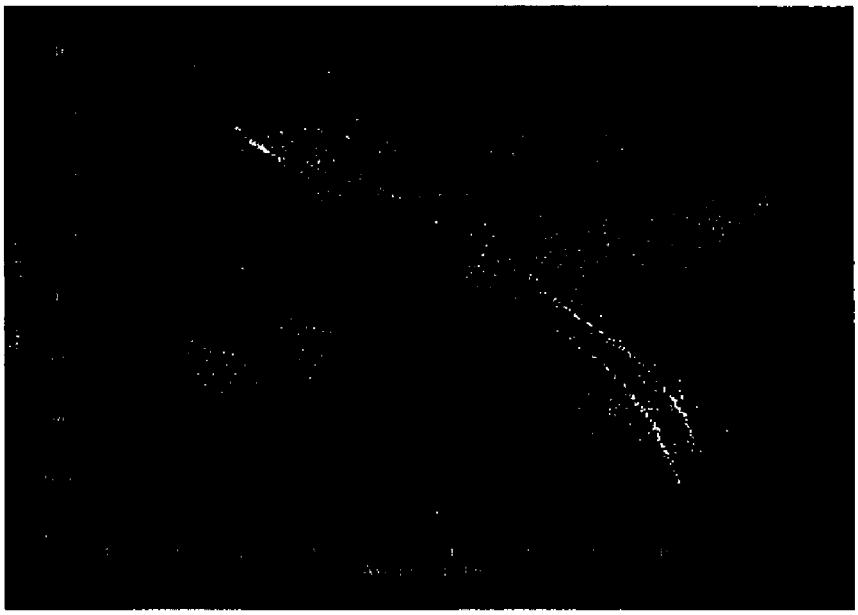

FIG. 6.28: Réseau appris sur les composantes 8 et 9 de l'ACP non normée. Ce réseau est sensible à 3 défauts pères relatifs à l'allumage (points cyans, magentas et blancs) plus un défaut relatif au relais de la pompe injection (points jaunes). Ce réseau est de type 3 sensible aux problèmes d'allumage.

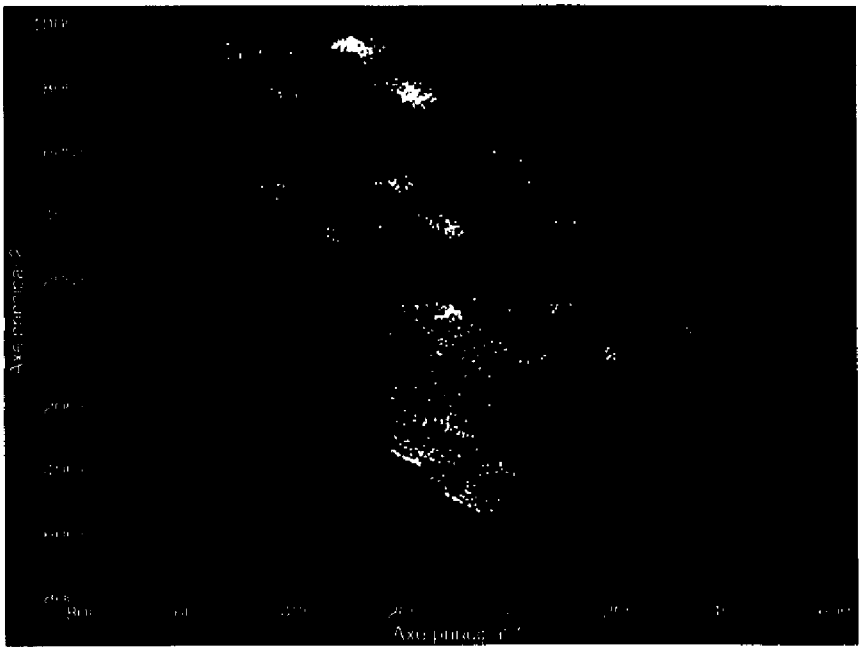

FIG. 6.29: Réseau appris sur les composantes 1 et 2 d'une ACP normée. Suivant le réglage de sensibilité, ce réseau peut être sensible à trois défauts pères ou à cinq. Ces défauts pères sont liés : 3 sont relatifs aux capteurs de température d'eau et d'air (problème de masse ou capteurs en circuit ouvert), et deux sont relatifs aux alimentations de capteurs (potentiomètre papillon et capteur pression).

Dans la figure 6.30 et les suivantes, on analyse la position des projections des défautgrammes des pannes dans les plans principaux d'une ACP simple de ces défautgrammes. Pour chaque défaut père, le codage des couleurs est le suivant :

p1 (* jaune) : *Référence masse capteurs température eau et air,*
p2 (* magenta) : *Référence masse capteur pression et potentiomètre papillon,*
p3 (* cyan) : *Alimentation allumage,*
p4 (* rouge) : *Commande allumage,*
p5 (* vert) : *Primaire bobine,*
p6 (* bleu) : *Vanne régulation ralenti 1,*
p7 (* blanc) : *Relais pompe injection,*
p8 (o jaune) : *Coupure des deux injecteurs,*
p9 (o magenta) : *Température eau,*
p10 (o cyan) : *Température air,*
p11 (o rouge) : *Curseur potentiomètre papillon,*
p12 (o vert) : *Alimentation capteur pression,*
p13 (o bleu) : *Alimentation potentiomètre papillon,*
p14 (o blanc) : *Electrovanne.*

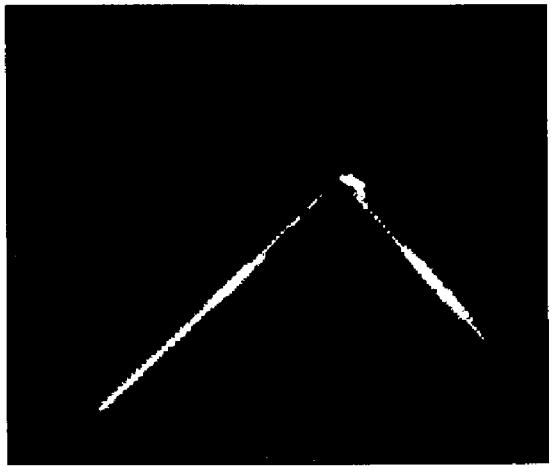

FIG. 6.30: Ce plan synthétique, constitué des deux premières composantes, est très intéressant, car les défauts proches fonctionnellement sont distribués dans des directions identiques.

FIG. 6.31: Ce plan synthétique est similaire au précédent : les défauts proches fonctionnellement se situent dans des directions identiques. Mais il est plus intéressant car la composante 3 *étale* les divers défauts pères, ce qui permet d'en identifier certains dans ce plan.

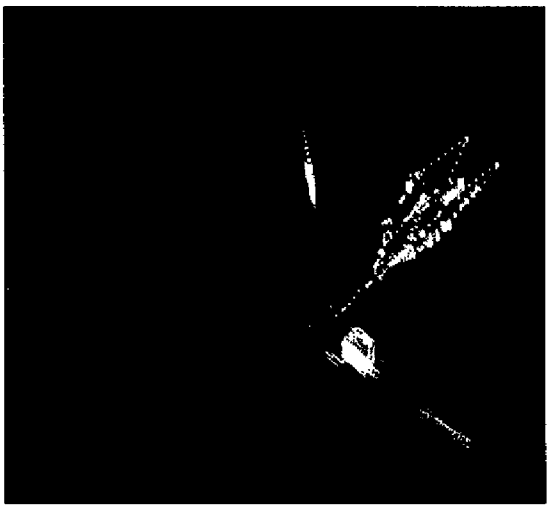

FIG. 6.32: Ce plan permet d'identifier très bien 2 défauts pères et moins bien deux autres. Dans ces quatre défauts, trois sont relatifs à l'allumage et l'autre au relais de la pompe injection. Ce sont les quatre défauts pères détectés par le réseau de la figure 6.28.

FIG. 6.33: Deux défauts sont parfaitement identifiés dans ce plan. Ce sont les défauts sur le capteur de température de l'air et le capteur de température de l'eau. C'est principalement la composante 7 qui fait s'opposer ces deux défauts.

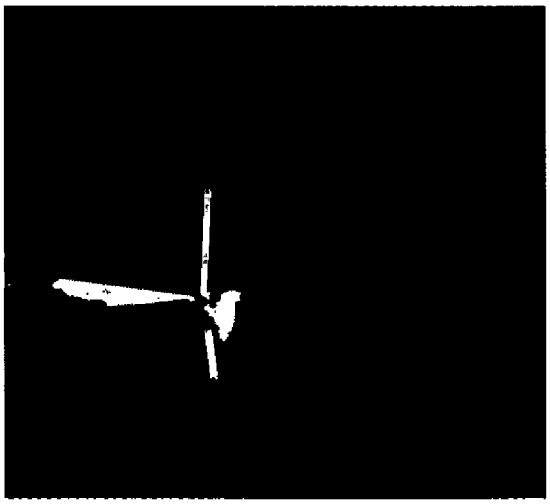

FIG. 6.34: Ce plan est la combinaison de deux composantes 3 et 9 qui étalent les divers défauts. En conséquence il est extrêmement discriminant. On peut pratiquement identifier 9 pères, dont 6 distinctement.

L'histoire du développement de la physique nous montre qu'une théorie serait bien présomptueuse en se flattant d'être définitive ; nous ne voyons guère les théories s'élever que pour crouler.

Mais, en s'écroulant, une théorie qui a été construite avec le désir sincère de parvenir au vrai ne disparaît jamais complètement ; parmi ses débris se trouvent toujours des matériaux propres à entrer dans la composition de quelque autre système plus parfait et plus durable...

PIERRE DUHEM

Conclusion générale

Dans cette thèse nous avons présenté deux axes de recherches qui ont été menés en parallèle : une approche théorique des réseaux de neurones binaires et une approche méthodologique du diagnostic par reconnaissance de formes. Par l'utilisation de méthodes statistiques, de procédures géométriques et de réseaux de neurones au sein d'une même méthodologie de diagnostic, l'objectif que nous nous étions fixé a pratiquement été atteint.

Notre première démarche pour la construction de réseaux de neurones binaires en se basant sur une étude de l'hypercube n'a pas complètement abouti. L'idée était de caractériser les parties linéairement séparables afin de bien définir les hyperplans pertinents dans le cas d'entrées booléennes et d'utiliser cette caractérisation pour concevoir un algorithme de construction par une recherche dans des espaces discrets. Cette étude n'a pas abouti à une méthode de construction de réseaux mais a fourni plusieurs propriétés nouvelles des parties linéairement séparables. Une propriété que nous avons longtemps prise pour une caractérisation s'est avérée être équivalente à d'autres propriétés déjà étudiées, qui ne sont que des conditions nécessaires. Mais la nouveauté de ce travail réside dans le type d'approche, en étudiant principalement les propriétés de graphes, et nous pensons que ce nouveau cadre peut certainement apporter d'autres résultats.
Pour la construction des réseaux de neurones, nous n'avons pas poursuivi cette voie car l'algorithme développé pour vérifier la conjecture s'est montré extrêmment puissant pour vérifier la séparabilité linéaire d'une base d'apprentissage. Nous avons donc changé d'orientation et exploité au maximum ce nouveau principe d'apprentissage d'une unité binaire. Ainsi le BCP et tous ses algorithmes dérivés peuvent être utilisés dans une large gamme d'algorithmes. Nous avons particulièrement développé l'association du BCP avec la stratégie Sequential Learning, ce qui nous a permis de développer un algorithme de construction multi-sorties très général, rapide et ayant apparemment de bons pouvoirs de généralisation. En utilisant une propriété de l'hypercube, nous avons développé une version encore plus rapide car le calcul des représentations internes n'est plus nécessaire. Malgré la construction d'une seconde couche cachée, cet algorithme permet l'apprentissage de plus grosses bases d'apprentissage sans diminution du pouvoir de généralisation. De plus l'implémentation VLSI de la seconde couche cachée et des neurones de sortie est très simple et ne nécessite que très peu de composants.
Le bon pouvoir de généralisation obtenu par les algorithmes proposés est certainement dû à un bon

placement des hyperplans : position intermédiaire entre les deux projections des exemples les plus proches et optimisation de la marge à tous les niveaux des algorithmes. Mais dans les algorithmes de construction que nous avons développé à partir du Sequential Learning, les potentialités en pouvoir de généralisation sont certainement limitées. La raison de cette limitation est valable pour tous les algorithmes de construction de réseaux de neurones binaires : l'apprentissage complet de la base d'apprentissage implique forcément un surapprentissage. Les algorithmes de construction de réseaux de neurones sont très intéressants par rapport aux modèles classiques car avec ces derniers il faut déterminer la structure optimale de manière empirique en faisant une multitude d'essais. De plus, l'usage des neurones binaires est plus intéressant pour le développement de circuits électroniques qui permettent d'exploiter le parallélisme intrinsèque des réseaux de neurones. Il apparaît donc fondamental de s'intéresser à ce problème de surapprentissage dans les algorithmes de construction de réseaux de neurones binaires. La littérature est quasi inexistante dans ce domaine. Dans le cas des algorithmes que nous avons développés, basés sur la stratégie Sequential Learning, une solution est peut-être d'utiliser une technique de *tolérance aux fautes*, c'est-à-dire de garder une position qui exclut beaucoup de d'exemples même si elle commet quelques erreurs. Une autre possibilité est peut-être d'utiliser la stratégie d'exclusion d'une manière générale, mais aussi la stratégie de minimisation des erreurs à certains moments et notamment vers la fin de la procédure. Il conviendrait de poursuivre les recherches dans cette direction, car le surapprentissage est *a priori* le seul problème des algorithmes de construction comme le MOBCPSL.

Trente quatre ans après le Perceptron, l'apprentissage d'une unité à seuil est à nouveau d'actualité. Beaucoup d'algorithmes nouveaux ont été proposés ces dernières années. De plus en plus, les auteurs essayent de relier ces nouveaux algorithmes à la théorie bayésienne : ils étudient le comportement de ces algorithmes par rapport à l'optimal bayésien dans les cas linéairement séparables ou non. Le BCP Prouvé converge vers l'optimal bayésien dans le cas linéairement séparable, car il est prouvé que cet optimal est le connecteur minimal. Mais dans le cas non linéairement séparable, il serait intéressant d'étudier la qualité de la solution trouvée par le BCP Min par rapport à l'optimal bayésien. Ce problème semble assez difficile analytiquement. Une solution est peut-être d'interpréter les coefficients de pondération comme des probabilités.

Ce retour en force de la théorie bayésienne dans le domaine des réseaux de neurones est caractéristique actuellement. La statistique intervient de plus en plus dans les réseaux de neurones et à tous les niveaux. Si des fondements théoriques solides unificateurs apparaissent dans le domaine des réseaux de neurones, la statistique en sera très certainement le coeur. Etant donné le nombre très important de publications théoriques dans les plus grandes revues internationales abordant des domaines comme la théorie bayésienne, la théorie de la VC-dimension (Vapnik-Chervonenkis), la théorie du PAC Learning (Probably Approximately Correct), la physique statistique et la théorie de l'information, ces notions feront certainement partie du cadre probabiliste de cette future théorie. On peut noter au passage que la physique statistique est depuis longtemps très présente dans le domaine des réseaux récurrents. *Statisticiens et connexionnistes, amis ou ennemis ?*, tel est le titre (assez significatif de la situation actuelle) d'un rapport de recherche [68] dans lequel l'auteur étudie les relations de plus en plus intimes entre ces deux domaines. Mais il distingue tout de même deux types d'études dans les réseaux de neurones : le *connexionisme d'ingénieur* et le *connexionisme biologique*. D'après lui, le second domaine qui est d'ailleurs l'origine du premier comme nous l'avons vu dans le chapitre 3, dont le but est d'essayer de modéliser les mécanismes du cerveau, ne sera pas trop touché par la déferlante probabiliste. Dans un autre article [177], l'auteur est plus catégorique et présente la majorité des modèles de réseaux de neurones comme n'étant rien d'autre que des méthodes statistiques : régression linéaire et non linéaire, analyse discriminante, analyse en composantes principales, projection poursuit... Il est vrai que l'auteur semble a priori être statisticien ! Il conclut quand même son article en disant que les deux domaines ont beaucoup de techniques communes, et qu'ils peuvent beaucoup s'apporter l'un à l'autre.

L'aboutissement du travail présenté dans ce mémoire est la méthodologie STANDING. Cette méthodologie atteint pratiquement les objectifs que nous nous étions fixés : *conception d'une méthode de diagnostic de pannes par reconnaissance de formes sans aucune information symbolique mais seulement avec des observations en bon et mauvais fonctionnement.* Mais alors qu'au début nous pensions utiliser principalement les réseaux de neurones comme outil de détection et d'identification, l'usage couplé de plusieurs techniques statistiques parut nécessaire, et notamment la recherche de corrélations et de caractéristiques par ACP, l'estimation de densité de probabilité et le filtrage par les algorithmes EM et SEM. Bien que des modèles de réseaux de neurones dédiés au calcul d'une ACP existent, les résultats sont en général approximatifs sur des bases limitées (il y a convergence asymptotique). Dans [177], l'auteur parle de ces méthodes comme étant : "*just inefficient algorithms for approximating principal components*". Il est montré que l'ACP classique est optimale, il nous a donc paru plus judicieux d'utiliser la formulation statistique. Le calcul de l'estimation de densité est lui purement du domaine de la statistique. La méthode EM que nous avons employée s'est révelée très puissante. De plus l'utilisation du mélange de gaussiennes nous a permis d'utiliser une méthode de classification automatique très pertinente et qui a donné de bien meilleurs résultats que des méthodes classiques comme le *k-mean*. D'autre part, dans [37], les auteurs comparent différentes versions de l'EM (EM, SEM, SAEM, MCEM). En moyenne, les meilleurs résultats en classification sont obtenus avec une association EM-SEM. Ce n'est pas exactement la même stratégie d'alternance empirique que nous avons développée, mais cela montre qu'une association EM-SEM est un bon choix parmi les méthodes de cette famille.

La méthodologie STANDING intègre donc au sein d'un même outil des techniques issues de divers domaines : statistiques (ACP et l'EM-SEM), méthodes de diagnostic simple (avec des prétraitements divers), méthodes géométriques (la génération automatique de bases d'apprentissage et le calcul d'enveloppes convexes), méthodes de classification automatique (*k-mean* pour initialiser l'EM et la classification à partir du mélange de gaussiennes) et une méthode de filtrage des *points extrêmes* (à partir de la densité de probabilité). Cette méthodologie repose sur deux principes : la modélisation des zones de bon fonctionnement pour la détection des défauts et la discrimination des diverses pannes dans un espace de probabilité d'apparition des défauts élémentaires. La validation de la méthodologie sur des données réelles a été satisfaisante, les 14 pannes artificielles ont pu être discriminées, et a montré les avantages de cette approche : grande sensibilité (réglable), abstraction des systèmes, rapidité du système implanté, généricité... De plus une grande partie du développement des réseaux (gaussiens ou binaires) est automatique. Enfin, l'utilisation des réseaux gaussiens a montré des potentialités intéressantes pour réaliser du diagnostic préventif. Ceci a été possible grâce à la malencontreuse (mais fort utile) panne d'essence qui est survenue lors d'essais.

Mais certaines études doivent être approfondies pour finir d'automatiser le processus, notamment au niveau de l'alternance des algorithmes EM-SEM, de l'initialisation du nombre de composantes, du choix du seuil d'élimination des composantes par le SEM et enfin de l'heuristique de détermination du seuil de filtrage. Ces problèmes ne nous semblent pas trop difficiles. En revanche, l'automatisation du choix des composantes principales est un problème qui nous semble assez délicat à résoudre. Dans STANDING, ce choix est fait visuellement par l'utilisateur en s'appuyant sur des données numériques telles que les corrélations et l'inertie des nuages de mauvais fonctionnement dans des plans principaux d'une ACP du bon fonctionnement.

Nous avons réalisé dernièrement certaines expériences dans lesquelles la détection de certaines pannes posait des problèmes, notamment les problèmes de masse du calculateur. Cette panne avait déjà posé des problèmes de détection avec la première maquette. Ce genre de pannes met en défaillance une grande partie des signaux, et il est assez difficile de les discriminer dans les plans issus de l'ACP des défautgrammes. De plus on peut se demander si la méthode pourra être appliquée de la même manière

avec plus de pannes prises en considération. Plus le nombre de pannes considéré est grand plus il peut être difficile de les discriminer, même en ajoutant des réseaux de neurones de détection dédiés. Pour cela, il faudra faire d'autres expérimentations. Une solution est d'adopter une approche modulaire, dans laquelle chaque module d'identification ne prendra en compte qu'un sous-ensemble de défauts élémentaires. On peut aussi faire des ACP différentes sur des sous-ensembles de données quantitatives.

Certains de ces problèmes pourraient être résolus par une décision finale prise par un petit système à règle et par l'utilisation ponctuelle d'informations symboliques. Ainsi nous pensons que le développement d'un système générique de diagnostic de systèmes complexes ne peut pas être complètement basé sur des données symboliques ou à l'inverse complètement basé sur des observations. L'avenir de ces systèmes de diagnostic semble être une association de plusieurs types de méthodes qui collaborent. Un concept qui peut être fondamental dans le développement de tels systèmes est la notion suivante : *utilisation des informations de tous types obtenues à moindre coût avec des méthodes appropriées qui collaborent.* Dans [42] une méthode de diagnostic à deux niveaux est proposée. Un système expert fait un premier diagnostic qui est infirmé ou confirmé par une approche de type reconnaissance de formes. Comme les approches par reconnaisance de formes permettent de réaliser une grande partie du travail simplement avec des observations, dont l'obtention est possible et ne coûte pas cher, il nous semble plus approprié d'en faire le coeur d'un système de diagnostic générique des systèmes complexes. De plus ce type d'approche paraît bien adapté à l'aspect préventif du diagnostic, et pourrait aussi être utile pour suivre l'évolution des caractéristiques d'un système, son vieillissement. L'adaptation du système de détection est un problème important. Elle suppose un système de conception incrémentale supervisée, car l'apparition ou l'évolution d'un nouveau mode de fonctionnement peut venir d'une panne proche ou d'un vieillissement normal du système. Pour mettre à jour les procédures de détection, il faut donc avoir ces informations. Ensuite des informations symboliques peuvent intervenir à plusieurs niveaux. Des connaissances de surfaces, simples et peu coûteuses, peuvent être très utiles dans une décomposition modulaire, dans le choix des sous-ensembles de variables traitées par une ACP ou dans une logique de décision qui prend la décision finale, sans pour autant développer un véritable système expert, ce qui est très difficile. On peut aussi très bien imaginer un système complexe dans lequel un sous-système peut facilement modéliser analytiquement. Dans ces conditions, il est important de prendre ceci en compte en réalisant un diagnostic interne local à ce sous-système. Ce module pourra alors venir se greffer en parallèle aux méthodes de diagnostic simple et aux méthodes multidimensionnelles avec réseaux de neurones, pour détecter de nouveaux défauts élémentaires.

Pour conclure ce manuscrit nous pouvons dire que l'approche frontale de la complexité avec une seule méthode nous semble insuffisante. Il est fondamental d'utiliser toutes les informations disponibles au moindre coût. Le diagnostic des systèmes complexes nous semble donc abordable par la synergie de plusieurs types de méthodes au sein d'un même outil, dont le coeur du système de détection serait du type reconnaissance de formes. Les réseaux de neurones et les statistiques auront vraisemblablement un rôle très important dans de tels systèmes. Avant d'aboutir à un tel système générique et opérationnel, le chemin à parcourir reste important. L'approche que nous avons développée, même si elle est uniquement du type reconnaissance de formes, est un premier pas dans cette direction et peut constituer l'embryon de futurs systèmes auxquels viendront s'ajouter d'autres modules de détection et d'identification, et un organe de décision finale.

Bibliographie

[1] I. ALEKSANDER et J. TAYLOR, éd. *Artificial Neural Network II.* Elvesier, Amsterdam, North-Holland, 1992.

[2] E. ALPAYDIN. Grow-and-learn : An incremental method for category learning. In *Proceeding International Neural Network Conference*, volume 2, pp. 761–764, Paris, 1990.

[3] D. ANDES, B. WIDROW, M. LEHR, et E. WAN. MRIII : A robust algorithm for training analog neural networks. In *International Joint Conference on Neural Networks*, pp. 533–536, Washington DC, Janvier 1990.

[4] P. ANDREAE, B. DAWKINS, et P. O'CONNOR. DYSECT : An incremental clustering. Rapport de recherche, 1993. URL (Statlib) : `http ://lib.stat.cmu.edu/general/dysect`.

[5] J.K. ANLAUF et M. BIEHL. The AdaTron : an adaptative perceptron algorithm. *Europhysics Letters*, 10, pp. 687–692, 1989.

[6] J.-M. ARNAUDIÈS et J. BERTIN. *Groupes, algèbre et géometrie.* Ellipse, Paris, 1993.

[7] T. ASH. Dynamic Node Creation in Backpropagation Networks. *Connection Science*, 1(4), pp. 365–375, 1989.

[8] K. BALAKRISHNAN et V. HONAVAR. Intelligent diagnosis systems. Rapport de recherche ISU-CS-TR 95-22, Department of Computer Science, Iowa State University, 1995.

[9] P. BALDI. Neural networks, acyclic orientations of the hypercube, and sets of orthogonal vectors. *SIAM Journal of Discrete Mathematics*, 1, pp. 1–13, 1988.

[10] P. BALDI. Neural Networks, Orientations of the hypercube, and Algebraic Threshold Functions. *IEEE Transactions on Information Theory*, 34, pp. 523–530, 1988.

[11] C.B. BARDER, D.P. DOBKIN, et H. HUHDANPAA. The Quickhull Algorithm for Convex Hull. Rapport de recherche, Geometry center Technical Report GCG53, Minneapolis, 1993. URL : `http ://www.geom.umn.edu/`.

[12] M. BASSEVILLE. Detecting changes in signals and systems – A survey. *Automatica*, 24(3), pp. 309–326, 1988.

[13] R. BATTITI. Accelerated Backpropagation Learning : Two Optimization Methods. *Complex Systems*, 3, pp. 331–342, 1989.

[14] R. BATTITI. First and second order methods for learning : between steepest descent and Newton's method. *Neural Computation*, 4, pp. 141–166, 1992.

[15] R. BATTITI et F. MASULLI. BFGS optimization for faster and automated supervised learning. In *Proceeding of International Neural Network Conference*, pp. 757–760, Paris, 1990.

[16] E.B. BAUM. When are k-nearest neighbor and back propagation accurate for feasible sized sets of examples. In *Neural Networks, Proceeding EURASIP Workshop*, pp. 2–25, Février 1990.

[17] E.B. BAUM et D. HAUSSLER. What size net gives valid generalization ? *Neural Computation*, 1, pp. 151–160, 1989.

[18] M.D. BEDWORTH, L. BOTTOU, J.S. BRIDLE, F. FALLSIDE, L. FLYNN, F. FOGELMAN-SOULIÉS, K.M. PONTING, et R.W. PRAGER. Comparison of neural and conventional classifiers on a speech recognition problem. In *IEEE First Conference in Artificial Neural Networks*, pp. 86–89, Londre, 1989.

[19] M. BERGOT et L. GRUDZIEN. Sûreté et diagnostic des systèmes industriels – Principaux concepts, méthodes, techniques et outils. *Diagnostic et sûreté de fonctionnement*, 5(3), pp. 317–344, 1995.

[20] A. BERMAK et D. MARTINEZ. A configurable serial-parallel multiplier building block for VLSI digital neural networks. Rapport de recherche 95263, LAAS–CNRS, Toulouse, Juin 1995.

[21] A. BERMAK et D. MARTINEZ. A variable-precision systolic architecture for ANN computation. In *Fifth International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, Lausanne, Février 1996.

[22] E. BERNAUER et H. DEMMOU. Les méthodes de détection, localisation et diagnostic. Rapport de recherche 95473, LAAS–CNRS, Toulouse, Novembre 1995.

[23] S.N. BHATT, F.R.K. CHUNG, F.T. LEIGHTON, et A.L. ROSENBERG. Efficient embeddings of trees in hypercubes. *SIAM Journal of Computing*, 21, pp. 151–162, 1992.

[24] F. BIEGLER-KÖNIG et F. BÄRMANN. A learning algorithm for multilayer neural networks based on linear least squares problems. *Neural Networks*, 6(1), pp. 127–131, 1993.

[25] C. BISHOP. Exact calculation of the Hessian matrix for multilayer perceptron. *Neural Computation*, 4, pp. 494–501, 1992.

[26] L. BOBROWSKI et J. SKLANSKY. Linear classifers by window training. *IEEE Transactions on Neural Networks*, 25(1), pp. 1–9, Janvier 1995.

[27] N.K. BOSE et A.K. GARGA. Neural network design using Voronoi diagrams. *IEEE Transactions on Neural Networks*, 4(5), pp. 778–787, 1993.

[28] J.D. BOSKOVIĆ et K.S. NARENDRA. Comparison of linear, nonlinear and neural-network-based adaptative controllers for a class of fed-batch fermentation processes. *Automatica*, 31(6), pp. 817–840, 1995.

[29] P. BOURRET, J. REGGIA, et M. SAMUELIDES. *Réseaux Neuronaux : une approche connexionniste de l'intelligence artificielle*. Teknea, Toulouse, 1991.

[30] K. BOUYOUCEFF. *Sur des aspects multirésolution en reconstruction d'images – Application au télescope HUBBLE*. Thèse, Université Paul Sabatier, Toulouse, 1993.

[31] M.L. BRADY et R. RAGHAVAN. Back Propagation fails to separate where perceptron succeed. *IEEE Transactions on Circuits and Systems*, 36(5), pp. 665–674, Mai 1989.

[32] W.L. BUNTINE et A.S. WEIGEND. Computing second derivatives in feed-forward networks : a review. *IEEE Transactions on Neural Networks*, 5(3), pp. 480–488, 1994.

[33] N. BURGES. A constructive algorithm that converges for real-valued input patterns. *International Journal of Neural Systems*, 5(1), pp. 59–66, 1994.

[34] C. CAMPBELL et C. Perez VICENTE. The Target Switch algorithm : A constructive learning procedure for feedforward neural networks. *Neural Computation*, 7, pp. 1245–1264, 1995.

[35] H. CAUSSINUS et A. RUIZ-GAZEN. Metrics for finding typical structures by means of principal component analysis. *Data science and its application*, pp. 177–192, 1995.

[36] H. CAUSSINUS et A. RUIZ-GAZEN. Projection poursuit and generalized principal component analysis, 1995. Article à paraître.

[37] G. CELEUX, D. CHAUVEAU, et J. DIEBOLT. On stochastic versions of the EM algorithm. Rapport de recherche 2514, INRIA, Mars 1995.

[38] G. CELEUX et J. DIEBOLT. Reconnaissance de mélange de densité et classification, un apprentissage probabiliste : l'algorithme SEM. Rapport de recherche 349, INRIA, 1984.

[39] J.P. CHANGEUX. *L'homme neuronal.* Collection Pluriel, Fayard, Paris, 1983.

[40] Y. CHAUVIN. A back-propagation algorithm with optimal use of hidden units. In *[188]*, pp. 519–526, 1989.

[41] B. CHAZELLE. An optimal convex hull algorithm in any fixed dimension. *Discrete Computational Geometry*, 10, pp. 377–409, 1993.

[42] C. COCCA. *Contribution à l'élaboration d'un système de diagnostic à deux niveaux application à la surveillance d'une pompe à chaleur.* Thèse, Université de Technologie de Compiègne, 1988.

[43] A. COHEN. *Ondelettes et traitement numérique du signal.* Masson, Paris, 1992.

[44] J.M. COMBES, A. GROSSMANN, et P. TCHAMITCHIAN. *Wavelets : time frequency methods and phase analysis.* Springer-Verlag, Berlin, 1989.

[45] P. COMON. Independant component analysis – A new concept ? *Signal Processing*, 36, pp. 287–314, 1994.

[46] C. CORTES. *Prediction of generalization ability in learning machines.* Thèse, University of Rochester, Rochester, New York, 1995.

[47] Y. Le CUN. Une procédure d'apprentissage pour réseau à seuil asymétrique. In *Cognitiva : A la Frontière de l'Intelligence Artificielle des Sciences de la Connaissance des Neurosciences*, pp. 599–604, Paris, 1985. CESTA.

[48] Y. Le CUN, J.S. DENKER, et S.A. SOLLA. Optimal Brain Damage. In *[189]*, pp. 598–605, 1990.

[49] G. CYBENKO, D.W. KRUMME, et K.N. VENKATARAMAN. Fixed hypercube embedding. *Information Processing Letters*, 25, pp. 35–39, 1987.

[50] R. DAVID et H. ALLA. Petri nets for modeling of dynamic systems – A survey. *Automatica*, 31(2), pp. 175–202, 1994.

[51] R. DAVID, J.-M. DION, et L.D. LANDAU. *Résultats et perspectives en automatique.* Traité des Nouvelles Technologies, série Automatique, Hermès, Paris, 1988.

[52] P. DEMARTINES et J. HÉRAULT. CCA : Curvilinear Component Analysis. In *Proceedings of the* 15^{th} *workshop GRETSI*, Juan-Les-Pins, France, Septembre 1995.

[53] P. DEMARTINES et J. HÉRAULT. Curvilinear Component Analysis : a self-organizing neural network for non linear mapping of data sets. Rapport de recherche, Berkeley, Décembre 1995. Soumis à IEEE Transactions on Neural Networks.

[54] A.P. DEMPSTER, N.M. LAIRD, et D.B. RUBIN. Maximum-Likelihood from incomplete data via the EM algorithm. *Journal of Royal Statistics Society*, 39, pp. 1–38, 1977.

[55] T. DENOEUX et R. LENGELLÉ. Initializing Back Propagation Networks With Prototypes. *Neural Networks*, 6, pp. 351–363, 1993.

[56] M.L. DERTOUZOS. *Threshold logic : A synthetic approach.* MIT Press, Cambridge, MA, 1966.

[57] E. DIDAY et AL.. *Optimisation en classification automatique.* INRIA, Rocquencourt, 1980.

[58] B. DUBUISSON. *Diagnostic et reconnaissance des formes.* Traité des Nouvelles Technologies, série Diagnostic et Maintenance, Hermès, Paris, 1990.

[59] B. DUBUISSON, C. CHABANON, et I. CRIGNON. Reconnaître les formes pour émettre un diagnostic. *Le nouvel automatisme*, pp. 49–55, Juin 1983.

[60] B. DUBUISSON, M.H. MASSON, et C. FRÉLICOT. Diagnostic par une approche reconnaissance de forme. In *Proceedings des Journées d'études S3, Sureté - Surveillance - Supervision, Détection et localisation de défaillances*, Paris, 1994.

[61] R.O. DUDA et P.E. HART. *Pattern classification and scene analysis.* John Wiley & Son, New York, 1973.

[62] O. DUFFAUT. *Problématique multi-modèle pour la génération d'arbres de test : application au domaine automobile.* Thèse, Ecole Nationale Supérieure de l'Aéronautique et de l'Espace, 1994.

[63] H. EDELSBRUNNER et W. SHI. An $O(nlog^2h)$ time algorithm for the three-dimensional convex hull problem. *SIAM Journal of Computing*, 20(2), pp. 259–269, 1991.

[64] I.Z. EMIRIS. A complete implementation for computing general dimensional convex hulls. Rapport de recherche 2581, INRIA, 1995.

[65] B.S. EVERITT. *Clusters analysis.* John Wiley & Son, New York, 1986.

[66] S.E. FAHLMAN. An Empirical Study of Learning Speed in Back-Propagation Networks. Rapport de recherche CMU-CS-88-162, Carnegie Mellon University, Pittsburgh,PA, 1988.

[67] S.E. FAHLMAN et C. LEBIERE. The Cascade-Correlation Learning Architecture. In *[189]*, pp. 524–532, 1990.

[68] A. FLEXER. Connectionists and statisticians, friends or foes? Rapport de recherche TR–95–06, The austrian research institute for artificial intelligence, 1995.
URL : `http ://www.ai.univie.ac.at/oefai/nn/arthur.html`.

[69] F. FOGELMAN-SOULIÉS et J. HÉRAULT, éd. *Neurocomputing : Algorithms, Architecture and Applications.* NATO ASI séries, Springer-Verlag, Berlin, 1989.

[70] P.M. FRANCK. Fault diagnosis in dynamic systems using analytical and knowledge redundancy – A survey and some new results. *Automatica*, 26(3), pp. 459–474, 1990.

[71] M. FREAN. The Upstart Algorithm : A method for constructing and training feedforward neural networks. *Neural Computation*, 2(2), pp. 198–209, 1990.

[72] M. FREAN. A Thermal perceptron learning rule. *Neural Computation*, 4(6), pp. 946–957, 1992.

[73] J.H. FRIEDMAN et W. STUETZLE. Projection pursuit regression. *Journal of the American Statistical Association*, 76, pp. 817-823, 1981.

[74] J.H. FRIEDMAN et J.W. TUKEY. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, 23, pp. 881–889, 1974.

[75] I.J. GABELMAN. *The functional behavior of majority (threshold) elements.* Thèse, Department of electrical engineering, University of Syracuse, 1961.

[76] S.I. GALLANT. Optimal linear discriminants. In *IEEE Proceeding 8th Conf. Pattern Recognition*, pp. 849–852, Paris, 1986.

[77] S.I. GALLANT. Three Constructive Algorithms For Network Learning. In *Proceeding of the 8th Annual Conference of the Cognitive Science Society*, pp. 652–660, 1986.

[78] S.I GALLANT. Perceptron-Based learning algorithms. *IEEE Transactions on Neural Networks*, 1, pp. 179–191, 1990.

[79] J.-D. GASCUEL et M. WEINFELD. ARIANE : une Architecture de Réseaux Intégrés Associatifs de NEurones. In *Proceeding* 3^{eme} *Symposium sur les architectiures nouvelles de machines*, Palaiseau, 1991.

[80] D.L. GRAY et A.N. MICHEL. A training algorithm for binary feedforward neural networks. *IEEE Transactions on Neural Networks*, 3(2), pp. 176–194, 1992.

[81] D. GRIES et I. STOJMENOVIC. A note on Graham's convex hull algorithm. *Information Processing Letters*, 25, pp. 323–327, 1987.

[82] G. GUGLIELMI, T. PARISINI, et G. ROSSI. Keynote paper – Fault diagnosis and neural networks : A power plant application. *Control in Engineering Practice*, 3(5), pp. 601–620, 1995.

[83] R. HABER et H. UNBEHAUEN. Structure identification of nonlinear dynamic systems – A survey on Input-Output approaches. *Automatica*, 26(4), pp. 651–677, 1990.

[84] S.J. HANSON, J.D. COWAN, et C. Lee GILES, éd. *Advances in Neural Information Processing Systems 5*, San Mateo, CA, 1993. Morgan Kaufman Publishers Inc.

[85] F. HARARY. *Graph theory.* Addison–Wesley, Reading, MA, 1969.

[86] F. HARARY. Cubical graphs and cubical dimensions. *Computational Mathematics and Applications*, 15(4), pp. 271–275, 1988.

[87] F. HARARY. The graph of a boolean function. *Journal of Theoretical Artificial Intelligence*, 2, pp. 163–169, 1989.

[88] F. HARARY, J.P. HAYES, et H.-J. WU. A survey of the theory of hypercube graphs. *Computational Mathematics and Applications*, 15(4), pp. 277–289, 1988.

[89] B. HASSIBI et D.G. STORK. Second Order Derivatives for Network Pruning : Optimal Brain Surgeon. In *[84]*, pp. 164–171, 1993.

[90] D.O. HEBB. *The organization of behavior.* John Wiley & Son, New York, 1949.

[91] G.E. HINTON. Adaptive learning procedures. *Artificial Intelligence*, 40, pp. 185–234, 1989.

[92] A. HOLLEY. Editorial. *Le courrier du CNRS*, 79, pp. 1, Octobre 1992.

[93] K. HORNIK. Approximation Capabilities of Feedforward Neural Networks. *Neural Networks*, 4, pp. 251–257, 1991.

[94] K. HORNIK. Some New Results on Neural Network Approximation. *Neural Networks*, 6, pp. 1069–1072, 1993.

[95] K. HORNIK, M. STINCHCOMBE, et H. WHITE. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, 2(5), pp. 359–366, 1989.

[96] J. HOWELL. Model-based fault detection in information poor plants. *Automatica*, 30(6), pp. 929–943, 1994.

[97] K.J. HUNT, D. SBARBARO, R. ZBIKOWSKI, et P.J. GAWTHROP. Neural networks for control systems – A survey. *Automatica*, 28(6), pp. 1083–1112, 1992.

[98] IGL. Présentation de SADT. Rapport de recherche, Société IGL, 1988.

[99] R. ISERMANN. Process fault detection based on modeling and estimation methods – A survey. *Automatica*, 20(4), pp. 387–404, 1984.

[100] R. ISERMANN. Fault diagnosis of machines via parameter estimation and knowledge processing – Tutorial paper. *Automatica*, 29(4), pp. 815–835, 1993.

[101] M. ISHIKAWA. A structural learning algorithm with forecasting of link weights. Rapport de recherche TR-90-7, Electrotechnical Lab., Tsukuba-City, 1990.

[102] C. JI, R.R. SNAPP, et D. PSALTIS. Generalizing Smoothness Constraints from Discrete Samples. *Neural Computation*, 2(2), pp. 188–197, 1990.

[103] A. JUDITSKY, H. HJALMARSSON, A. BENVENISTE, B. DELYON, L. LJUNG, J. SJÖBERG, et Q. ZHANG. Nonlinear black-box models in systems identification : Mathematical foundations. *Automatica*, 31(12), pp. 1725–1750, 1995.

[104] E.D. KARNIN. A Simple Procedure for Pruning Back-Propagation Trained Neural Networks. *IEEE Transactions on Neural Networks*, 1(2), pp. 239–242, 1990.

[105] M. KAROUIA, R. LENGELLÉ, et T. DENOEUX. Weight initialization in BP networks using discriminant analysis techniques. In *Proceeding Neuro-Nîmes 94*, Nîmes, 1994. (preprint version).

[106] H.J. KIM et J.G. LEE. Partial sum problem mapping into a hypercube. *Information Processing Letters*, 36, pp. 221–224, 1990.

[107] H.J. KIM et S.K. PARK. The geometrical learning of binary neural networks. *IEEE Transactions on Neural Networks*, 6, pp. 237–248, 1995.

[108] D.G. KIRKPATRICK et R. SEIDEL. The ultimate planar convex hull algorithm ? *SIAM Journal of Computing*, 15(1), pp. 287–299, 1986.

[109] S. KNERR, L. PERSONNAZ, et G. DREYFUS. Single layer learning revisited : a stepwise procedure for building and training a neural network. In *[69]*, 1989.

[110] T. KOHONEN. *Self-Organization and Associative Memory*. Springer-Verlag, Berlin, 1984.

[111] H.N KOIVO. Artificial neural networks in fault diagnosis and control. *Control in Engineering Practice*, 2(1), pp. 89–101, 1994.

[112] W. KRAUTH et M. MEZARD. Learning algorithms with optimal stability in neural networks. *Journal of Physic, série A*, 20, pp. 745–752, 1987.

[113] B.J.A. KRÖSE et P.P. Van Der SMAGT. *An introduction to neural networks.* Non publié, Amsterdam, 1991.

[114] J.K. KRUSCHKE. Creating local and distributed bottlenecks in hidden layers of back-propagation networks. In *Connectionist Models Summer School*, pp. 120–126, 1988.

[115] S. LABRÈCHE et H. POULARD. Caractéristiques des partitions optimales. *RAIRO - Revue de Recherche Opérationnelle*, 1996. En soumission finale.

[116] S. LABRÈCHE et H. POULARD. Parametrization of the principal cluster method. *Soumis à Journal of Classification*, 1996.

[117] S. LABRÈCHE, H. POULARD, et D. ESTÈVE. Classification between class criteria - Factorial presentation and optimal partition properties. In *Proceeding IFCS 96*, Kobe, Japon, Mars 1996.

[118] S. LABRÈCHE, H. POULARD, A. HERRERA, et D. ESTÈVE. Application à la veille technologique du choix de métriques dans les méthodes statistiques. In *Proceeding VST Sitef 95*, Toulouse, Octobre 1995.

[119] S. LABRÈCHE, H. POULARD, A. HERRERA, et D. ESTÈVE. Choix de métriques en ACP et en classification. In *Proceeding ASU XXVIII Journées de statistique*, Jouy en Josas, Mai 1995.

[120] P.-A. LAMONGIE. Apprentissage non supervisé des réseaux de neurones et estimation de densité de probabilité. Rapport de recherche, ENSEEIHT, 1994. Mémoire de DEA.

[121] R. LENGELLÉ et T. DENOEUX. Optimizing multilayer neural networks layer by layer without backpropagation. In *[1]*, pp. 995–998, 1992.

[122] I.C. LERMAN. Les représentation factorielles de la classification. *RAIRO - Revue de Recherche Opérationnelle*, 13(2), pp. 107–128, 1979.

[123] M. LESHNO, V.Y. LIN, A. PINKUS, et S. SCHOCKEN. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6, pp. 861–867, 1993.

[124] P.M. LEWIS et C.L. COATES. *Threshold logic.* John Wiley & Son, New York, 1968.

[125] M. MARCHAND, M. GOLEA, et P. RÚJAN. A convergence theorem for Sequential Learning in two-Layer perceptron. *Europhysics Letters*, 11, pp. 487–492, 1990.

[126] D. MARTINEZ. *Offset, une méthode de construction incrémentale de réseaux de neurones multicouches et son application à la conception d'un copilote automobile.* Thèse, Université Paul Sabatier, Toulouse, 1992.

[127] D. MARTINEZ, M. CHAN, et D. ESTÈVE. Construction of layered quadratic perceptrons. In *Proceeding Neuro-Nîmes 92*, pp. 655–665, Nîmes, 1992.

[128] D. MARTINEZ et D. ESTÈVE. The offset algorithm : Building and learning method for multilayer neural networks. *Europhysics Letters*, 18(2), pp. 95–100, 1992.

[129] D. MARTINEZ, H. POULARD, M.CHAN, A. HERRERA, L. ABADIE, et D. ESTÈVE. Incremental learning in a multilayer neural network for process-fault detection. In *Proceeding TOOLDIAG 93*, pp. 160–170, Toulouse, 1993.

[130] J. MARTINEZ. *Méthodes temps-fréquence appliquées à l'analyse de signaux issus d'écoulements turbulents.* Thèse, Ecole Nationale Supérieure de l'Aéronautique et de l'Espace, Toulouse, 1994.

[131] W.S. McCulloch et W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 9, pp. 127–147, 1943.

[132] Y. Meyer. *Wavelets and applications.* Masson, Paris, 1991.

[133] M. Mezard et J.-P. Nadal. Learning in feedforward layered networks : The Tiling Algorithm. *J. Phys. A*, 22(12), pp. 2191–2203, 1989.

[134] R. Milne. Strategies for diagnosis. *IEEE Transactions on Systems, Man & Cybernetics*, 17(3), pp. 333–339, Mai 1987.

[135] M.L. Minsky et S.A. Papert. *Perceptrons.* MIT Press, Cambridge, MA, 1969.

[136] B.G. Mirkin. The method of principal clusters. *Automation and remote control*, 10, 1987.

[137] M. Mollard. *Les invariants du n-cubes.* Thèse, Université Grenoble, 1981.

[138] M. Mollard. *Quelques problèmes combinatoires sur l'hypercube.* Thèse, Université Grenoble I, 1989.

[139] M. Møller. A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning. *Neural Networks*, 6(4), pp. 525–533, Juin 1993.

[140] M.C. Mozer et P. Smolensky. Skeletonization : A Technique for Trimming the Fat from a Network via Relevance Assessment. In *[188]*, pp. 107–115, 1989.

[141] I. Mozetic. Model-Based diagnosis : An overview. In *Advanced topics in Artificial Intelligence*, pp. 419–430. Springer-Verlag, Berlin, 1992.

[142] S. Muroga. *Threshold logic and its applications.* John Wiley & Son, New York, 1971.

[143] S. Muroga, T. Tsuboi, et C.R. Baugh. Enumeration of threshold functions of eight variables. *IEEE Transactions on Computers*, C-19(9), pp. 818–825, Septembre 1970.

[144] P.M. Murphy et D.W. Aha. *UCI Repository of machine learning databases.* Department of information and computer science, University of California, Irvine, CA, 1991.
URL : `ftp ://ftp.cs.colorado.edu :/pub/machine-learning-databases.`

[145] M. Muselli. On sequential construction of binary neural networks. *IEEE Transactions on Neural Networks*, 6(3), pp. 678–690, 1995.

[146] N.J. Nilsson. *Learning Machines : Foundations of trainable pattern-classifying systems.* McGraw-Hill, New York, 1965.

[147] S.J. Nowlan et G.E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4), pp. 473 493, 1992.

[148] C. Olivier, P. Courtellemont, O. Colot, D. de Brucq, et A. El Matouat. Comparison of histograms : a tool for detection. *Diagnostic et sûreté de fonctionnement*, 4(3), pp. 335–355, 1994.

[149] R. Parekh, J. Yang, et V. Honavar. Constructive neural network learning algorithms for multi-category pattern classification. Rapport de recherche ISU-CS-TR 95-15a, Department of Computer Science, Iowa State University, 1995.

[150] D.B. Parker. Learning-Logic : Casting the Cortex of the Human Brain in Silicon. Rapport de recherche TR-47, Center for Computational Research on Economics and Management Science, MIT, Cambridge, MA, 1985.

[151] H. PAUGAM-MOISY. *Optimisation des réseaux de neurones artificiels : analyse et mise en oeuvre sur architectures massivement parallèles.* Thèse, Ecole Normale Supérieure de Lyon, LIP-IMAG, 1992.

[152] B.A. PEARLMUTTER. Fast exact multiplication by the Hessian. *Neural Computation*, 6, pp. 147–160, 1994.

[153] Y. PENG et J.A. REGGIA. A connectionist model for diagnostic problem solving. *IEEE Transactions on Systems, Man & Cybernetics*, 19(2), pp. 285–298, Mars 1989.

[154] H. POULARD. Minimisation du nombre de connexions lors de l'apprentissage d'un réseau de neurones multicouches. Rapport de recherche 91451, LAAS-CNRS, Toulouse, Décembre 1991.

[155] H. POULARD. Barycentric Correction Procedure – A fast method of learning threshold unit. In *Proceeding WCNN 95*, volume I, pp. 710–713, Washington DC, Juillet 1995.

[156] H. POULARD. Improvement to the Barycentric Correction Procedure Sequential Learning. In *Proceeding ISANN 95*, pp. A3.23–A3.28, Hsinchu, Taiwan, Décembre 1995.

[157] H. POULARD et D. ESTÈVE. A convergence theorem for Barycentric Correction Procedure. *Soumis à Neural Computation*, 1995. En révision.

[158] H. POULARD, N. HERNANDEZ, et D. MARTINEZ. Two efficient multiple outputs constructive algorithms. *A soumettre à Connection Science*, 1996.

[159] H. POULARD et S. LABRÈCHE. Barycentric Correction Procedure – Efficient threshold unit training method for constructive algorithms. *Soumis à IEEE Transactions on Neural Networks*, 1995.

[160] H. POULARD et S. LABRÈCHE. STANDING – STAtistiques et Neurones artificiels pour un système de DIagNostic Générique. Rapport de recherche 95511, LAAS-CNRS, Toulouse, 1995. Confidentiel.

[161] H. POULARD et S. LABRÈCHE. A study of hypercube structures : Toward a characterization of linearly separable functions. Rapport de recherche 95494, LAAS-CNRS, Toulouse, Août 1995.

[162] H. POULARD et S. LABRÈCHE. Neural networks and real time diagnosis – A new methodology applied to automotive failure detection. In *Proceeding NEURAP 95*, pp. 322–329, Marseille, Mars 1996.

[163] B. RAFFIN et M.B. GORDON. Learning and generalization with minimerror - A temperature dependent learning algorithm. *Neural Computation*, 7, pp. 1206–1224, 1995.

[164] R.A. REDNER et H.F. WALKER. Mixture densities, maximum likelihood and the EM algorithm. *SIAM Review*, 26, pp. 195–239, 1984.

[165] R. REED. Pruning Algorithms — A Survey. *IEEE Transactions on Neural Networks*, 4(5), pp. 740–746, 1993.

[166] D.L. REILLY, L.N. COOPER, et C. ELBAUM. A Neural Model for Category Learning. *Biological Cybernetics*, 45, pp. 35–41, 1982.

[167] R. REITER. A theory of diagnosis from first principle. *Artificial Intelligence*, 32, pp. 57–95, 1984.

[168] B.D. RIPLEY. Statistical aspects of neural networks. In *Proceedings of SemStat*, Sandbjerg, Denmark, 1993.

[169] F. ROSENBLATT. *Principles of neurodynamics.* Spartan Books, Washington DC, 1962.

[170] A. RUIZ-GAZEN. *Estimation robuste d'une matrice de dispersion et projections révélatrices.* Thèse, Universite Paul Sabatier, Toulouse, 1993.

[171] P. RÚJAN. A fast method for calculating the perceptron with maximal stability. *J. Phys. I France*, 3(2), pp. 277–290, 1993.

[172] P. RÚJAN et M. MARCHAND. A geometrical approach to learning in neural networks. In *International Joint Conference on Neural Networks*, pp. 105–109, 1989.

[173] D. RUMELHART et J. MCCLELLAND, éd. *Parallel Distributed Processing : Explorations in the Microstructure of Cognition*, volume 1. MIT Press, Cambridge, MA, 1986.

[174] D.E. RUMELHART, G.E. HINTON, et R.J. WILLIAMS. Learning Internal Representations by Error Propagation. In *[173]*, chapitre 8. MIT Press, Cambridge, MA, 1986.

[175] Y. SAAD et M.H. SCHULTZ. Topological properties of hypercubes. *IEEE Transactions on Computers*, 37(7), pp. 867–872, 1988.

[176] G. SAPORTA. *Probabilités, analyse des données et statistique.* TECHNIP, Paris, 1990.

[177] W.S. SARLE. Neural networks and statistical models. In *In Proceeding of the nineteenth annual SAS users group international conference*, Avril 1994.

[178] T.J. SEJNOWSKI et C.R. ROSENBERG. Parallel networks that learn to pronounce English text. *Complex Systems*, 1, pp. 145–168, 1987.

[179] R. SETIONO et L. KWONG-HUI. Use of a quasi-newton method in a Feedforward Neural Network Constructive Algorithm. *IEEE Transactions on Neural Networks*, 6(1), pp. 273–277, 1995.

[180] R. SHONWILER. Separating the vertices of N-cubes by hyperplanes and its applications to artificial neural networks. *IEEE Transactions on Neural Networks*, 4, pp. 343–347, 1993.

[181] J. SIETSMA et R.J.F. DOW. Creating Artificial Neural Networks That Generalize. *Neural Networks*, 4(1), pp. 67–79, 1991.

[182] N. SIMON. Constructive Supervised Learning Algorithms for Artificial Neural Networks. Rapport de maîtrise, Delft University of Technology, Department of Electrical Engineering, Delft, Netherlands, Juin 1993.
URL : `ftp ://archive.cis.ohio-state.edu/pub/neuroprose/Thesis`.

[183] J. SJÖBERG, Q. ZHANG, L. LJUNG, A. BENVENISTE, B. DELYON, P.-Y. GLORENNEC, H. HJALMARSSON, et A. JUDITSKY. Nonlinear black-box modeling in system identification : A unified overview. *Automatica*, 31(12), pp. 1691–1724, 1995.

[184] D. SLEPIAN. On the number of symetry types of boolean function of n variables. *Canad. Journal of Mathematics*, 5, pp. 185–193, 1953.

[185] V.S. SRINIVASAN. Fault detection - Monitoring using petri nets. *IEEE Transactions on Systems, Man & Cybernetics*, 23(4), pp. 1155–1161, Juillet 1993.

[186] M.-C. SUBNER, M. GABRIEL, et L. CLAUDE. Utilisation de l'AMDEC pour générer la base de connaissances d'un système expert d'aide au diagnostic. *Diagnostic et sûreté de fonctionnement*, 2(2), pp. 133–153, 1992.

[187] G. THIMM et E. FIESLER. High order and multilayer perceptron initialization. Rapport de recherche 94–07, Institut Dalle Molle d'Intelligence Artificielle Perceptive (IDIAP), Martiny, Valay, Suisse, 1994. Submitted to IEEE Transactions on Neural Networks.

[188] D.S. TOURETZKY, éd. *Advances in Neural Information Processing Systems 1*, San Mateo, CA, 1989. Morgan Kaufman Publishers Inc.

[189] D.S. TOURETZKY, éd. *Advances in Neural Information Processing Systems 2*, San Mateo, CA, 1990. Morgan Kaufman Publishers Inc.

[190] V.N. VAPNIK. *Estimation of Dependencies Based on Empirical Data.* Springer-Verlag, New York, 1982.

[191] S.J. WAN. Cone algorithm : an extension of the perceptron algorithm. *IEEE Transactions on Systems, Man & Cybernetics*, 24(10), pp. 1571–1576, Octobre 1994.

[192] C.S. WEAVER. Some properties of threshold logic unit pattern recognition networks. *IEEE Transactions on Computers*, C-24(3), pp. 290–298, Mars 1975.

[193] A.S. WEIGEND, D.E. RUMELHART, et B.A. HUBERMAN. Back-Propagation, Weight-Elimination and Time Series Prediction. In D. TOURETZKY, éd., *Connectionist Models Summer School*, pp. 105–116, 1990.

[194] P.J. WERBOS. *Beyond Regression : New Tools for Prediction and Analysis in the Behavioral Sciences.* Thèse, Harvard University, 1974.

[195] N. WEYMAERE et J.-P. MARTENS. On the initialization and optimization of multilayer perceptrons. *IEEE Transactions on Neural Networks*, 5(5), pp. 738–751, 1994.

[196] D. WHITLEY et C. BOGART. The evolution of connectivity : prunning neural network using genetic algorithms. In *International Joint Conference on Neural Networks*, volume 1, page 134, Washington, DC, 1988.

[197] B. WIDROW. Generalization and information storage in networks of adaline neurons. In *Self-Organizing Systems*, pp. 435–461. Spartan Books, Washington DC, 1962.

[198] B. WIDROW et M.E. HOFF. Adaptive switching circuits. *IRE Western Electric Show and Covention Record*, 4, pp. 96–104, Août 1960.

[199] B. WIDROW et M. LEHR. 30 years of adaptive neural networks : Perceptron, Madaline and Backpropagation. *Proceedings of IEEE, Special Issue on Neural Network*, pp. 1415–1442, Septembre 1990.

[200] R. WINTER et B. WIDROW. MADALINE RULE II : A Training Algorithm for Neural Networks. In *Proceeding ICNN-88, San Diego*, pp. 401–408, 1988.

[201] G. ZWINGELSTEIN. *Diagnostic des défaillances.* Traité des Nouvelles Technologies, série Diagnostic et Maintenance, Hermès, Paris, 1995.

Annexe A : Théorème de convergence du BCP

A.1 Variations des barycentres et du vecteur w

C'est la démonstration des relations 5.11 5.12, 5.13 5.14.

$$
\begin{aligned}
b_1^{t+1} &= \frac{\sum_{i\in I_1} \alpha_i^{t+1} p_i}{\sum_{i\in I_1} \alpha_i^{t+1}} \\
&= \frac{\sum_{i\in I_1} \alpha_i^t p_i + \sum_{i\in J_1^t} \beta_i^t p_i}{\sum_{i\in I_1} \alpha_i^t + \sum_{i\in J_1^t} \beta_i^t} \\
&= \frac{\sum_{i\in I_1} \alpha_i^t p_i}{\sum_{i\in I_1} \alpha_i^t} \frac{\sum_{i\in I_1} \alpha_i^t}{\sum_{i\in I_1} \alpha_i^t + \sum_{i\in J_1^t} \beta_i^t} + \frac{\sum_{i\in J_1^t} \beta_i^t p_i}{\sum_{i\in I_1} \alpha_i^t + \sum_{i\in J_1^t} \beta_i^t} \\
&= b_1^t (1 - \sum_{i\in J_1^t} \eta_i^t) + \sum_{i\in J_1^t} \eta_i^t p_i \\
&= b_1^t + \sum_{i\in J_1^t} \eta_i^t (p_i - b_1^t) \\
&= b_1^t + \frac{\sum_{i\in J_1^t} \beta_i^t (p_i - b_1^t)}{\sum_{i\in I_1} \alpha_i^t + \sum_{i\in J_1^t} \beta_i^t} \\
&= b_1^t + \frac{(\sum_{i\in J_1^t} \beta_i^t)(b_{e_1}^t - b_1^t)}{\sum_{i\in I_1} \alpha_i^t + \sum_{i\in J_1^t} \beta_i^t} \\
&= b_1^t + \kappa^t e_1^t
\end{aligned}
$$

Donc

$$b_1^{t+1} - b_1^t = \sum_{i\in J_1^t} \eta_i^t (p_i - b_1^t) = \kappa^t e_1^t$$

Une démonstration similaire peut être faite et prouver que

$$b_0^{t+1} - b_0^t = \sum_{i \in J_0^t} \lambda_i^t (m_i - b_0^t) = \nu^t \, e_0^t$$

Ainsi

$$\begin{aligned} w^{t+1} - w^t &= (b_1^{t+1} - b_1^t) - (b_0^{t+1} - b_0^t) \\ &= \sum_{i \in J_1^t} \eta_i^t (p_i - b_1^t) - \sum_{i \in J_0^t} \lambda_i^t (m_i - b_0^t) \\ &= \kappa^t \, e_1^t - \nu^t \, e_0^t \end{aligned}$$

□

A.2 $\mathcal{F}_t$ est une bijection, calcul de $\mathcal{F}_t^{-1}$

Les relations entre les composantes des vecteurs y_t et x_t s'expriment par

$$\forall k \in J_1^t \quad (\sum_{i \in J_1^t} \beta_i^t + \sum_{i \in I_1} \alpha_i^t) \; \eta_k^t = \beta_k^t \tag{A.6}$$

$$\forall k \in J_1^t \quad (\sum_{i \in J_0^t} \delta_i^t + \sum_{i \in I_0} \mu_i^t) \; \lambda_k^t = \delta_k^t \tag{A.7}$$

La somme des équations A.6 pour $k \in J_1^t$ nous donne

$$(\sum_{i \in J_1^t} \beta_i^t + \sum_{i \in I_1} \alpha_i^t) \; \kappa^t = \sum_{i \in J_1^t} \beta_i^t$$

Ainsi $\sum_{i \in J_1^t} \beta_i^t$ peut s'exprimer comme une fonction de κ^t

$$\sum_{i \in J_1^t} \beta_i^t = \frac{\kappa^t}{1 - \kappa^t} \sum_{i \in I_1} \alpha_i^t \tag{A.8}$$

A partir de A.6 et A.8 on peut en déduire

$$\forall k \in J_1^t \quad \beta_k^t = \frac{\eta_k^t}{1 - \kappa^t} \sum_{i \in I_1} \alpha_i^t \tag{A.9}$$

De la même manière, on peut montrer

$$\forall k \in J_0^t \quad \delta_k^t = \frac{\lambda_k^t}{1 - \nu^t} \sum_{i \in I_0} \mu_i^t \tag{A.10}$$

La fonction $\mathcal{F}_t^{-1}$ est donc parfaitement définie par les relations A.9 et A.10. Donc $\mathcal{F}_t$ est une bijection de $\mathcal{A}_t = \mathcal{F}_t^{-1}(U_t)$ dans U_t.
Dans le cas ou l'on prend un paramètre de relaxation non égale à 1, on doit alors calculer $\mathcal{F}_t^{-1}(r_t y^t)$. Le résultat est alors

$$\forall k \in J_1^t \quad \beta_k^t = \frac{r_t \, \kappa^t / q_1^t}{1 - r_t \, \kappa^t} \sum_{i \in I_1} \alpha_i^t \quad \text{et} \quad \forall k \in J_0^t \quad \delta_k^t = \frac{r_t \, \nu^t / q_0^t}{1 - r_t \, \nu^t} \sum_{i \in I_0} \mu_i^t$$

A.3 Majoration de $\Delta^t(x^t)$

Lemme A.1 $\forall t/\ q^t > 0$ et $\theta^t \in K^t \implies \forall x^t \in U_t\ \ \Delta^t(x^t) < \mathcal{M}^t(x^t)$

Preuve A.1 *On considère le vecteur $x^t \in U_t$ en supposant qu'à l'itération t, on a $q^t > 0$ et $\theta^t \in K^t$. On va aussi supposer que $q_1^t > 0$ et $q_0^t > 0$. Une démonstration similaire et plus simple pourrait être faite en supposant $q_1^t = 0$ ou $q_0^t = 0$. En élevant au carré la relation 5.14 on obtient alors*

$$\begin{aligned}\Delta^t(x^t) &= \|w^{t+1}\|^2 - \|w^t\|^2 \\ &= (w^t + \kappa^t e_1^t - \nu^t e_0^t)^2 - \|w^t\|^2 \\ &= \kappa^{t2}\|e_1^t\|^2 + \nu^{t2}\|e_0^t\|^2 + 2\kappa^t\ w^t.e_1^t - 2\nu^t\ w^t.e_0^t - 2\kappa^t\nu^t\ e_1^t.e_0^t \end{aligned} \tag{A.11}$$

Premièrement comme $(b_1^t, b_{e_1}^t, b_0^t, b_{e_0}^t) \in (\mathcal{E}_1 \cup \mathcal{E}_0)^4$, on a la majoration

$$\kappa^{t2}\|e_1^t\|^2 + \nu^{t2}\|e_0^t\|^2 \ < \ \kappa^{t2} d^2 + \nu^{t2} d^2 \tag{A.12}$$

Le dernier terme peut être majoré par

$$-2\kappa^t\nu^t\ e_1^t.e_0^t \ < \ 2\kappa^t\nu^t\ \|e_1^t\|\,\|e_0^t\| \ < \ 2\kappa^t\nu^t d^2 \tag{A.13}$$

Le point $b_{e_1}^t$ est dans le demi-espace positif défini par $\mathcal{H}^t$, et le point $b_{e_0}^t$ dans le demi-espace négatif. Dans ces conditions on obtient $-w^t.b_{e_1}^t > \theta^t$ et $-w^t.b_{e_0}^t < \theta^t$. En utilisant la definition de K^t en 5.17, on obtient alors

$$\begin{array}{rcccccl} w^t.e_1^t &=& w^t.b_{e_1}^t - w^t.b_1^t &<& w^t.b_1^t - \gamma\|w^t\|^2 - w^t.b_1^t &=& -\gamma\|w^t\|^2 \\ -w^t.e_0^t &=& -w^t.b_{e_0}^t + w^t.b_0^t &<& -w^t.b_0^t - \gamma\|w^t\|^2 + w^t.b_0^t &=& -\gamma\|w^t\|^2 \end{array}$$

ce qui conduit aux inégalités

$$2\kappa^t w^t.e_1^t < -2\kappa^t\gamma\|w^t\|^2 \tag{A.14}$$

$$-2\nu^t w^t.e_0^t < -2\nu^t\gamma\|w^t\|^2 \tag{A.15}$$

En utilisant les majorations A.12, A.13, A.14 et A.15, on arrive

$$\Delta^t(x^t) < (\kappa^{t2} + \nu^{t2} + 2\kappa^t\nu^t)\,d^2 - 2\,(\kappa^t + \nu^t)\,\gamma\,\|w^t\|^2$$

mais $\omega^t(x^t) = \kappa^t + \nu^t$ donc $\Delta^t(x^t) < (\omega^t(x^t))^2 d^2 - 2\,(\omega^t(x^t))\,\gamma\,\|w^t\|^2$ et finalement

$$\Delta^t(x^t) < \mathcal{M}^t(x^t)$$

□.

A.4 Minimisation du majorant de $\Delta^t(x^t)$ dans U_t', méthode AL_0

Lemme A.2 $\forall t/\ q^t > 0$ et $\theta^t \in K^t \implies \Delta^t(x_m^t) < -\varepsilon_t < 0$

$$\text{avec}\quad x_m^t \in \arg\min_{x^t \in U_t'} \mathcal{M}^t(x^t) \quad \text{et} \quad \varepsilon_t = -\mathcal{M}^t(x_m^t) = \frac{\gamma^2\|w^t\|^4}{d^2}$$

Preuve A.2 *Dans le domaine U_t' le minimum absolu de $\mathcal{M}^t$ vérifie*

$$\nabla \mathcal{M}^t(x^t) = 0 \iff \frac{\partial G^t}{\partial \psi}(\omega^t(x^t))\nabla\omega^t(x^t) = 0 \iff \frac{\partial G^t}{\partial \psi}(\omega^t(x^t)) = 0$$

car le vecteur $\nabla\omega^t(x^t)$ est un vecteur constant avec toutes ses composantes égales à 1. Le minimum absolu de la fonction G^t, qui est une parabole est donc

$$\psi_m^t = \frac{\gamma\|w^t\|^2}{d^2} \quad \text{et} \quad G_t(\psi_m^t) = -\frac{\gamma^2\|w^t\|^4}{d^2}$$

L'ensemble des vecteurs x^t tels que $\omega^t(x^t) = \psi_m^t$ sont donc des minimums absolus de la fonction $\mathcal{M}^t$. L'existence de ces minimums peut être montrée par la méthode constructive suivante, qui est en fait une implémentation de la méthode AL_0. Un vecteur x_m^t doit être défini dans le domaine U_t'. On a $\|w^t\| < d$ et $\gamma < 1/2$ donc $\psi_m^t < 1/2$. Trois cas sont possibles :

- *si $E_1^t \neq \emptyset$ et $E_0^t \neq \emptyset$, on peut prendre $\kappa^t = \nu^t = \psi_m^t/2 < 1/4 < \xi$, ainsi $\forall k \in J_1^t\ \eta_k^t = \kappa^t/q_1^t$ et $\forall k \in J_0^t\ \delta_k^t = \nu^t/q_0^t$. Alors $\omega^t(x_m^t) = \kappa^t + \nu^t = \psi_m^t$ et $x_m^t \in U_t'$.*
- *si $E_0^t = \emptyset$ on peut prendre $\kappa^t = \psi_m^t < 1/2 < \xi$, et donc $\forall k \in J_1^t\ \eta_k^t = \kappa^t/q_1^t$. Ainsi $\omega^t(x_m^t) = \kappa^t = \psi_m^t$ et $x_m^t \in U_t'$.*
- *si $E_1^t = \emptyset$ on peut prendre $\nu^t = \psi_m^t < 1/2 < \xi$, et donc $\forall k \in J_0^t\ \delta_k^t = \nu^t/q_0^t$. Ainsi $\omega^t(x_m^t) = \nu^t = \psi_m^t$ et $x_m^t \in U_t'$.*

Avec cette méthode de construction $x_m^t \in \arg\min_{x^t \in U_t'} \mathcal{M}^t(x^t)$, et si on définit ε_t par

$$\varepsilon_t = -G_t(\psi_m^t) = -\mathcal{M}^t(x_m^t) = \frac{\gamma^2\|w^t\|^4}{d^2}$$

en appliquant le lemme A.1 on peut conclure

$$\Delta^t(x_m^t) < -\varepsilon_t < 0$$

□.

A.5 Prise en compte du paramètre de relaxation

Lemme A.3 $\forall x^t \in U_t \quad \forall r \in]0,1] \quad \Delta^t(rx^t) \leq r^2\,\Delta^t(x^t)$

Preuve A.3 *Pour alléger les écritures, on définira les vecteurs*

$$\forall i \in J_1 \quad s_i = p_i - b_1^t \quad \text{and} \quad \forall i \in J_0 \quad t_i = m_i - b_0^t$$

en élevant au carré la relation 5.13 pour tout x^t dans U_t et avec $r \in]0,1]$, on obtient

$$\begin{aligned}\Delta^t(rx^t) &= \|w^{t+1}\|^2 - \|w^t\|^2 \\ &= \sum_{i\in J_1^t} r^2\eta_i^{t2}\|s_i\|^2 + \sum_{i\in J_0^t} r^2\lambda_i^{t2}\|t_i\|^2 + \sum_{i\in J_1^t} 2\,r\,\eta_i^t\ w^t.s_i - \sum_{i\in J_0^t} 2\,r\,\lambda_i^t\ w^t.t_i - \sum_{(i,j)\in J_1^t\times J_0^t} 2\,r^2\eta_i^t\,\lambda_j^t\ s_i.t_i\end{aligned}$$

Mais $\forall i \in J_1^t\ w^t.s_i < 0$ et $\forall i \in J_0\ -w^t.t_i < 0$ et $0 < r^2 \leq r \leq 1$ alors

$$\forall i \in J_1 \quad 2\,r\,\eta_i^t\ w^t.s_i \leq 2\,r^2\,\eta_i^t\ w^t.s_i \quad \text{et} \quad \forall i \in J_0 \quad -2\,r\,\lambda_i^t\ w^t.t_i \leq -2\,r^2\,\lambda_i^t\ w^t.t_i$$

On peut donc facilement en déduire

$$\forall x^t \in U_t \quad \forall r \in]0,1] \quad \Delta^t(rx^t) \leq r^2\,\Delta^t(x^t)$$

□.

A.6 Démonstration du théorème de convergence avec paramètre de relaxation variable

Il faut d'abord montrer que

$$\min_{x^t \in U_t''} \Delta^t(x^t) \leq \min_{x^t \in U_t'} \Delta^t(x^t) < \min_{x^t \in U_t'} \mathcal{M}^t(x^t)$$

La première inégalité est évidente car on a $U_t' \subset U_t''$. La seconde peut être facilement déduite du lemme A.1. Grâce aux lemmes A.1,A.2 et A.3, on peut en déduire que pour les trois différents méthodes de calcul de x_m^t on a

$$\Delta^t(x_m^t) < -\varepsilon_t < 0 \quad \text{et} \quad \Delta^t(r_t x_m^t) \leq {r_t}^2 \Delta^t(x_m^t) \quad \text{alors} \quad \Delta^t(r_t x_m^t) < -{r_t}^2 \varepsilon_t < 0$$

- *supposons que C_1 et C_0 sont LS, alors $\|w_m\| > 0$ donc*

$$\varepsilon_t = \frac{\gamma^2 \|w^t\|^4}{d^2} > \varepsilon > 0 \quad \text{avec} \quad \varepsilon = \frac{\gamma^2 \|w_m\|^4}{d^2}$$

$$\text{ainsi} \quad \forall t \,/\, q^t > 0 \text{ et } \theta^t \in K^t \quad \Longrightarrow \quad \Delta^t(r_t x_m^t) < -{r_t}^2 \varepsilon < 0$$

Si on somme ces inégalités pour $t \in [0, p-1]$ on obtient $\|w^p\|^2 < \|w^0\|^2 - \varepsilon \sum_{t=0}^{p-1} {r_t}^2 < \|w^0\|^2 - p r^2 \varepsilon$. Donc $\exists t_0 \,/\, \|w^{t_0}\| < \|w_m\|$. Ce qui est absurde. Donc le terme de gauche de la dernière implication ne peut être vrai. Mais nous avons toujours $\theta^t \in K^t$ donc on déduit que $q^{t_0} = 0$. La propriété $(P1)$ est donc démontrée.

- *supposons maintenant que C_1 et C_0 sont NLS, donc $\|w_m\| = 0$. Aucun hyperplan ne peut être solution, et on a donc toujours $q^t > 0$. On definit la limite $N_{lim} = \sqrt{\frac{d^2}{\gamma}}$. Si on suppose que pour $t = 0$, $\|w^t\| \geq N_{lim}$, alors*

$$\varepsilon_t = \frac{\gamma^2 \|w^t\|^4}{d^2} > \varepsilon' > 0 \quad \text{avec} \quad \varepsilon' = \frac{\gamma^2 {N_{lim}}^4}{d^2}$$

$$\text{donc} \quad \forall t \,/\, \|w^t\| \geq N_{lim} \text{ et } \theta^t \in K^t \quad \Longrightarrow \quad \Delta^t(r x_m^t) < -{r_t}^2 \varepsilon' < 0$$

On peut en déduire $\exists t_0 \,/\, \|w^{t_0}\| < N_{lim}$. La suite $(u_t)_{t \geq t_0}$ est définit par $u_t = \|w^t\|^2$, donc $\varepsilon_t = Cte\, {u_t}^2$. Cette suite est strictement décroissante et vérifie la relation récursive

$$u_{t+1} < u_t - Cte\, u_t^2 \quad \text{avec} \quad Cte = \frac{\gamma^2 {r_t}^2}{d^2}$$

on a

$$ {N_{lim}}^2 = \frac{d^2}{\gamma} = \frac{\gamma {r_t}^2}{Cte} < \frac{{r_t}^2}{2\,Cte} \leq \frac{1}{2\,Cte} \quad \text{donc} \quad 0 < u_{t_0} < {N_{lim}}^2 < \frac{1}{2\,Cte}$$

On peut montrer facilement montrer qu'une telle suite converge vers 0. Donc $\|w^t\|$ converge vers 0. La propriété $(P2)$ est donc démontrée.

□

Thèse de Monsieur Hervé POULARD

Statistiques et réseaux de neurones pour un système de diagnostic Application au diagnostic de pannes automobiles

Résumé : Ce travail a été réalisé dans le cadre d'une convention CIFRE entre le LAAS-CNRS et la société ACTIA qui développe des outils d'aide au diagnostic de pannes automobiles. Le but était l'utilisation des réseaux de neurones artificiels pour la conception d'une nouvelle méthode de diagnostic de pannes automobiles sans modèle, ni information symbolique mais uniquement des observations du système en bon et en mauvais fonctionnement. C'est donc une approche du diagnostic de systèmes complexes par reconnaissance de formes. Après avoir mis au point le système d'acquisition, nous avons conçu une première maquette qui a démontré la faisabilité d'un tel système et l'intérêt des réseaux de neurones, mais qui a soulevé de nombreux problèmes. L'utilisation particulière des réseaux de neurones dans cette application a nécessité l'usage d'algorithmes de construction. Après une étude théorique des structures de l'hypercube qui n'a pas abouti à un algorithme de construction mais qui a fourni plusieurs résultats théoriques, nous avons développé une famille d'algorithmes pour la construction des réseaux de neurones binaires. Le coeur de ces outils est une nouvelle méthode d'apprentissage d'unités à seuil très performante dénommée Barycentric Correction Procedure (BCP). L'aboutissement est un algorithme novateur car très général (entrées quelconques et sorties multiples), rapide et avec un bon pouvoir de généralisation. Nous avons finalement mis au point une nouvelle méthodologie de diagnostic, dans laquelle l'utilisation de méthodes statistiques et d'analyse de données en collaboration avec les réseaux neuronaux parut nécessaire. Cette méthodologie utilise donc des techniques très diverses : analyse en composantes principales, estimation de densité de probabilité, classification automatique, calcul d'enveloppes convexes, génération géométrique de bases d'apprentissage, construction de réseaux de neurones binaires, réseaux de neurones gaussiens et méthodes de diagnostic simples. Cette méthodologie a été appliquée avec succès au problème de la détection de pannes automobiles et a aussi montré des potentialités pour le diagnostic préventif. Elle est de plus assez générique pour avoir de nombreuses applications potentielles.

Mots clés : Diagnostic par reconnaissance de formes, Hypercube, Réseaux de neurones, Algorithmes de construction, BCP, Analyse en composantes principales, Estimation de densité, Mélange de gaussiennes.

Statistic and neural networks for a diagnosis system Application to automotive failure detection

Abstract : This work supported by a CIFRE grant has been done at LAAS-CNRS and ACTIA which designs and produces electronic aiding tools for automotive failure detection. A new diagnosis method based on pattern recognition techniques including artificial neural networks is proposed. Thus only the system's faulty and nonfaulty behaviors are taken into account, to the exclusion of symbolic information or model of the systems. After designing the acquisition system, a first prototype was devised and showed the feasability of such a system as well as the advantages of neural networks. However a number of problems have been raised as a result. The use of feedforward neural networks required constructive strategies. After investigating the hypercube structures which furnished several theoretical results but no algorithm, a set of algorithms for binary neural networks construction was developed, based on a new efficient threshold unit training method referred to as Barycentric Correction Procedure (BCP). The final result is the first general binary unit-based constructive algorithm (real valued inputs and multiple outputs), to be fast while offering good generalization capabilities. Finally, a new diagnosis methodology has been developed in which statistical and data analysis methods combined with neural networks were required. This methodology is also based on methods derived from various frameworks : principal component analysis, density estimation, automatic clustering, convex hull computation, geometrical training set generation, binary neural network construction, gaussian neural networks and simple diagnosis methods. This methodology has been successfully applied to automotive failure detection and seems to have good potentialities for preventive diagnosis. Moreover its genericity allows it to be applied in various areas.

Keywords : Pattern recognition diagnosis system, Hypercube, Neural networks, Constructive algorithms, BCP, Principal component analysis, Density estimation, Gaussian mixture.