THÈSE de DOCTORAT de l'UNIVERSITÉ PARIS 6

Spécialité :

INFORMATIQUE

Présentée par

YANN LE CUN

pour obtenir le titre de DOCTEUR DE L'UNIVERSITÉ PARIS 6

Sujet de la thèse:

Modèles Connexionnistes de l'Apprentissage

soutenue le **15 Juin 1987**

devant le jury composé de:

M Jacques **PITRAT**

M Geoffrey **HINTON**

Mme Françoise **FOGELMAN-SOULIÉ**

M Bernard **ANGÉNIOL**

M Maurice **MILGRAM**

à mes parents

et à Isabelle et Kévin, ce travail leur a demandé autant d'efforts qu'à moi

J'adresse mes plus vifs remerciements à Monsieur le Professeur J. PITRAT, qui m'a fait l'honneur de présider le jury de cette thèse .

Mr B. ANGÉNIOL et Mme F. FOGELMAN-SOULIÉ ont accepté de juger cette thèse, qu'ils trouvent ici l'expression de mes remerciements .

Monsieur le Professeur G. HINTON m'a fait l'honneur de participer au jury. Je le remercie vivement d'avoir manifesté de l'intérêt pour ce travail à son début, et de l'avoir fait connaitre outre-Atlantique .

Je remercie Monsieur le Professeur M. MILGRAM pour ses conseils et ses encouragements qui ne furent pas limités à sa fonction de directeur de recherche .

Je remercie Mme F. FOGELMAN-SOULIÉ qui, par son énergie et sa gentillesse, a su faire du L.D.R. un lieu d'échange et de travail. Je la remercie également pour le plaisir que j'ai eu à travailler avec elle, ainsi qu'avec P. GALLINARI et les autres membres du LDR .

Ce travail n'aurait pas été possible sans l'appui dont j'ai bénéficié à l'E.S.I.E.E., de la part de E. VIVIAND, P. BILDSTEIN, et des membres du laboratoire d'Intelligence Artificielle et d'Analyse d'Image, particulièrement M. COUPRIE, D. BUREAU et G. BERTRAND. J'ai beaucoup apprécié l'environnement intellectuel et matériel de l'ESIEE .

Je remercie P. METSU, c'est ensemble que nous avons commencé à nous intéresser aux réseaux adaptatifs, beaucoup d'idées sont nées de nos nombreuses discussions .

Ces recherches ont été en partie financées par l'Ecole Nationale Supérieure des Mines de Paris .

Cette thèse a été écrite sur un AMIGA et tirée sur imprimante laser Agfa-P400, avec le logiciel de traitement de texte TEXimplanté sur le système PRIME de l'ESIEE. Merci aux TEX-wizards L. Chopinet et L. Breyton et au prime-guru T. Gautherot.

Table des matières

Liste des Figures

Chapitre

introduction

.1 L'apprentissage

L'apprentisage, c'est à dire la capacité d'acquérir de nouveaux comportements ou d'en modifier d'anciens par expérience, est une propriété de presque tous les animaux, y compris ceux que l'on peut difficilement qualifier d'intelligent [Kandel]. L'enjeu dont est actuellement l'objet l'intelligence artificielle a renouvelé l'intéret des chercheurs pour l'apprentissage. En effet à mesure que les techniques de représentation des connaissances se font plus nombreuses et efficaces, le principal goulot d'étranglement dans la réalisation d'un système basé sur la connaissance est précisément L'ACQUISITION de cette connaissance. La construction d'une base de connaissances nécessite la mise en oeuvre d'importants moyens humains et constitue la part la plus importante des efforts de conception d'un système expert. On comprend donc l'intérêt d'un système capable de construire sa base de connaissance à partir de données brutes. On peut y voir une des raisons de l'émergence récente de l'apprentissage en tant que sous-discipline de l'intelligence artificielle [Michalski & al. 83], [Michalski & al. 86]. Il est clair qu'envisager la réalisation d'un système général capable d'apprentissage est totalement irréaliste. C'est pourquoi on assiste àla floraison de nombreuses techniques, chacune étant adaptée à un problème particulier.

Les recherches sur "l'apprentissage artificiel" ont débuté très tôt dans l'histoire de l'IA [1]. Ces travaux et leurs successeurs ont suscité un excès d'enthousiasme et d'espoirs déçus dont les effets se font encore sentir aujourd'hui.

Ces dernières années ont vu l'apparition de nouvelles techniques regroupées sous le nom de CONNEXIONNISME dont le principe directeur est l'utilisation d'un réseau de

[1] voir les travaux de Samuel sur un jeu de dames en 54, et les travaux de Minsky sur un robot autonome la même annee

"processeurs" très simples interconnectés, et dont la connaissance réside précisément dans les connexions. Cette idée est en fait très ancienne, pour ne pas dire plus ancienne que l'informatique. L'idée sous-jacente, et assez présomptueuse, étant de copier l'architecture du cerveau. Ce dernier, qu'il soit humain ou animal est en effet le seul exemple de "machine" intelligente que nous connaissions. Il peut donc paraître naturel de s'en inspirer non seulement au niveau fonctionnel, mais aussi au niveau architectural.

Cette idée ancienne, et son application à l'IA et à la reconnaissance des formes, a déjà abouti à une impasse à la fin des années 60 avec l'arrêt des travaux sur le perceptron et les systèmes auto-organisateurs. Cela a contribué au déclin des méthodes de classification statistique, qui se sont au départ fort inspirées de ces modèles, au profit des méthodes structurelles et des systèmes à base de connaissance.

Le renouveau d'intérêt dans les systèmes connexionnistes est le fait de plusieurs raisons de natures très différentes. Tout d'abord, l'avènement de machines puissantes, et/ou à architecture parallèle bien adaptées à leur implantation. Egalement, la prise de conscience des limitations des méthodes "classiques" de l'IA pour ce qui concerne la perception (vision, audition) et l'apprentissage. Il est apparu nécessaire de trouver de nouveaux cadres théoriques et de nouvelles méthodes pour aborder ces problèmes. L'idée se répand qu'au delà de tout raffinement algorithmique, leur résolution nécessite une puissance de calcul énorme dont il est difficile de s'affranchir et qui va de pair avec un très haut degré de parallélisme. D'autre part on ne peut négliger le fait que, dans une certaine mesure, les travaux des précurseurs sont partiellement tombés dans l'oubli, ce qui a eu pour effet d'atténuer les conséquences du "syndrôme du perceptron" ayant conduit à le frapper d'interdit. Toutefois, la principale raison de ce renouveau est la découverte de nouvelles méthodes d'apprentissage levant les limitations des premiers modèles, en particulier les machines de Boltzmann [Hinton & Sejnowski 83], et l'algorithme de rétro-propagation [le Cun 85], [le Cun 86], [Parker 85], [Rumelhart & al. 86a] très largement décrit dans la suite.

Il est bon de mettre en garde le lecteur en insistant sur le fait que les systèmes connexionnistes ne sont QU'UN MOYEN DE PLUS de résoudre certains problèmes. En aucun cas il ne saurait être question de considérer ces méthodes comme une panacée universelle, ainsi que certains semblent le faire. La longue histoire des espoirs déçus de l'intelligence artificielle [2] conduit à une prudence extrême quant aux résultats possibles

[2] le perceptron, le General Problem Solver, les programmes d'échec toujours battus par les grand maitres, la traduction automatique, ... On consultera avec intérêt l'article de J. Pitrat dans le numéro spécial de La Recherche d'Octobre 1985

d'une nouvelle technique, quelle qu'elle soit.

.2 L'apprentissage dans l'Intelligence Artificielle

Les travaux sur l'apprentissage ont débuté très tôt dans l'histoire de l'intelligence artificielle, mais ses succès sont restés très marginaux, excepté dans le domaine de la reconnaissance des formes. Deux tendances se sont dessinées dès le départ: les méthodes plutôt heuristiques et/ou symboliques d'une part, et les méthodes plutôt numériques d'autre part. Selon Andler [in [Quinqueton % Sallantin 87]] et Newell c'est parce que l'apprentissage leur semblait un problème secondaire que les pionniers de l'intelligence artificielle, appartenant à la première tendance, se sont séparés de la cybernétique, appartenant à la seconde. [3]

La première tendance débute avec les travaux de Samuel sur le programme "Checkers" en 1954, puis avec le "Logic Theorist" et le GPS de Simon, Newell et Shaw, ces deux derniers systèmes sont répertoriés dans "l'apprentissage par découverte" selon la terminologie de [Michalski & al. 83] [Michalski & al. 86]. L'apparition de l'apprentissage en tant que sous-discipline de l'IA est assez récente. De nombreuses méthodes sont apparues avec les systèmes de représentation de connaissances utilisés dans les systèmes experts. Le système le plus connu est probablement Meta-Dendral de Buchanan, premier exemple d'apprentissage symbolique déductif. On peut citer les travaux très influents de Winston, basés sur la génération et l'appariement de descriptions structurelles, et ceux de Michalski fondés sur la transformation et la sélection de formules de la logique du premier ordre. D'autres travaux, plus spéculatifs, concernent la "découverte" de lois physiques ou de propriétés mathématiques (Lenat, Simon). Ces derniers travaux ont été très critiqués en raison du sens à donner au mot "découvrir", qui peut porter à confusion. On préfère souvent assimiler la notion d'apprentissage à celle "d'apprentissage à partir d'exemples" qui, quoique restrictive, contient une importante proportion des travaux, et est moins sujet à controverse. Ces méthodes sont exposées dans les ouvrages collectifs [Michalski & al. 83] et [Michalski & al. 86].

De nombreuses équipes françaises travaillent dans des domaines similaires. Un échantillon de ces travaux peut être consulté dans [Quinqueton % Sallantin 87]. Beaucoup d'entre eux concernent l'apprentissage à partir d'exemples, citons le système AGAPE de Ganascia et Kodratoff [in [Quinqueton % Sallantin 87]] basé sur l'appariement

[3]Cependant, certains chercheurs tels Minsky ou Nilsson ont tantôt appartenu à l'une ou à l'autre des tendances

structurel et la notion de "near miss", le système PLAGE de Gascuel [ibid.] mettant en oeuvre des méthodes "déductives" ou méthodes descendantes consistant en la sélection de concepts dans un ensemble prédéfini, ou encore le système CALM de Sallantin et Quinqueton [ibid.] basé sur la génération et la sélection de formules logiques et qu'il est possible de rapprocher des méthodes connexionnistes.

Des travaux ont également été menés à un niveau plus fondamental, tentant de formaliser L'INFÉRENCE INDUCTIVE, comme abstraction mathématique des processus d'apprentissage. Il s'agissait de formaliser le processus qui permet, à partir d'une collection de données expérimentales, de générer une représentation COMPACTE de ces données, sous forme de théorèmes, de prédicats, de programmes, ou de tout autre moyen de représenter des connaissances. L'idée étant qu'une représentation est d'autant meilleure qu'elle est plus compacte, plus simple. Ces notions sont très liées à la définition de complexité d'une séquence de symboles en tant que longueur du plus petit programme sur une machine de Turing universelle produisant cette séquence [Kolmogorov 68], [Solomonoff 64], [Chaitin 70], [Chaitin 74a]. Les travaux dans ce domaine sont nombreux, [Solomonoff 66], [Chaitin 74b], [Chaitin 75a]. On peut consulter [Chaitin 75b] pour une bonne introduction, et les travaux de Gold, Blum et Osherson pour les récents développements. Toutefois, leur application pratique à l'apprentissage ou à l'IA n'est que marginale, citons néanmoins les travaux de Cover sur les connexions entre ces notions et les méthodes statistiques classiques.

La seconde tendance regroupe tous les travaux dont les méthodes connexionnistes sont les héritières, nous y consacrons le paragraphe suivant.

.2.1 Un peu d'histoire

L'origine de la tendance "plutôt numérique" se trouve dans les travaux sur la cybernétique, et ceux de Warren McCulloch et Walter Pitts [McCulloch & Pitts 43], dont l'idée était d'expliquer comment un "neurone" très simplifié peut effectuer des inférences élémentaires, et comment un réseau constitué de ces neurones peut réaliser tout automate fini. Les premières simulations de systèmes neuronaux adaptatifs sur un ordinateur numérique sont dues à Farley et Clark [Farley & Clark 54], mais la première machine adaptative ayant remporté un certain succès est le perceptron [Rosenblatt 57], principalement en raison de l'existence d'un théorème de convergence de l'algorithme d'apprentissage [Rosenblatt 60]. Ce modèle a fait l'objet de nombreuses études au début des années 60 [Rosenblatt 61], [Block 62], [Block & al. 62]. Parallèlement, d'autres méthodes d'apprentissage étaient proposées, en particulier la célèbre procédure de Widrow et Hoff [Widrow & Hoff 60] ou "adaline", et basée sur

la minimisation itérative d'un critère quadratique. Beaucoup d'autres méthodes ont été proposées à la même époque. Les activités de cette communauté sont publiées dans de nombreux actes de conférences. Les articles de Minsky et Selfridge, et de Papert dans [Cherry 61] sont particulièrement intéressants. Les actes des conférences sur les "systèmes auto-organisateurs" regroupent une grande partie des travaux de l'époque [Yovits & Cameron 60], [Yovits & al. 62], [Von Foerster & Zopf 62].

L'élément de base de ces modèles était généralement le neurone formel de McCulloch et Pitts, modèle de neurone simplifié à l'extrême. Un neurone formel effectue une somme pondérée de ses entrées. Cette somme est comparée à un seuil, lorsqu'elle est supérieure au seuil, la sortie de l'élément est égale à +1, dans le cas contraire, elle est égale à -1 (ou 0). Un neurone formel peut donc effectuer une discrimination élémentaire définie par les pondérations. Le rôle de l'apprentissage est de modifier les pondérations. Selon le contexte, un tel élément est nommé neurone formel ou automate à seuil, ou encore élément linéaire à seuil (ELS). Les possibilités d'un ELS sont très limitées: il ne peut calculer que les fonctions dites LINÉAIREMENT SÉPARABLES, c'est pourquoi il est parfois appelé classifieur linéaire. Il est possible d'assembler de tels éléments en réseau, ce qui permet de calculer une très large classe de fonctions. Malheureusement, aucune des règles d'apprentissage proposées à l'époque ne pouvait fonctionner avec plus d'une couche d'éléments adaptatifs entre l'entrée et la sortie du système. Ils étaient donc limités aux fonctions linéairement séparables [4]. Ces limitations ont été très vite comprises, à la lumière des théorèmes de dénombrement des fonctions réalisables par un ELS [Winder 62], [Cover 65]. Quelques tentatives de généralisation des procédures d'apprentissage originales ont été proposées, basées sur des règles non supervisées [Block & al. 62], sur des éléments quadratiques et non plus linéaires, ou encore sur des éléments effectuant des votes majoritaires comme les "Multiple Adalines" ou "madalines" de Widrow [Von Foerster & Zopf 62] étudiées dans [Nilsson 65]. Aucun théorème de convergence n'existe pour ces procédures, et aucune ne s'est avérée totalement satisfaisante. Bien que certaines aient correctement fonctionné sur quelques problèmes, les recherches ont vite conduit à une impasse. Les chercheurs ont réorienté leur travaux sur des domaines connexes, abandonnant toute idée de fabriquer des "machines intelligentes", ce qui était leur but initial. B. Widrow a appliqué ses méthodes au traitement du signal adaptatif avec grand succès [5]. D'autres ont simplement changé leur vocabulaire, en ne parlant plus de neurones, mais de reconnaissance des formes

[4] le perceptron, bien que comportant deux couches d'opérateurs, est soumis à ces limitations car seule la dernière couche est modifiable

[5] il y a un "adaline" dans chaque modem

et de classification statistique. Le coup de grâce a été porté (involontairement selon les auteurs) par l'excellent livre de Minsky et Papert [Minsky & Papert 68] montrant les limitations théoriques du perceptron simple. L'ouvrage de Nilsson [Nilsson 65] fait le point sur une bonne partie des premiers travaux, mais la synthèse la plus complète est [Duda & Hart 73], qui de plus aborde de nombreuses autres méthodes statistiques. Dans un autre domaine, [Widrow & Stearn 85] présente les techniques de traitement du signal adaptatif, dont beaucoup d'idées peuvent être mises à profit pour l'analyse des systèmes connexionnistes.

.3 L'apprentissage et le problème de la mesure des performances

Le vrai problème de l'apprentissage est celui de la généralisation. La généralisation est la capacité à étendre une compétence à des exemples non appris. Par exemple, on espère qu'un système de reconnaissance des formes classera correctement des formes n'appartenant pas à l'ensemble d'apprentissage. La généralisation est conditionnée par la capacité du système à élaborer des représentations internes adéquates. Comment générer ces représentations internes? c'est tout le problème de l'apprentissage.

Il existe une méthode triviale d'apprentissage: la mémorisation in-extenso. Elle consiste à stocker TOUS les exemples de l'ensemble d'apprentissage avec leurs classes. On conçoit que cette méthode soit très dispendieuse en espace mémoire, ainsi qu'en temps puisqu'il faut comparer un nouvel exemple à tous les exemples déjà mémorisés. De plus cette représentation ne permet pas habituellement la généralisation [6].

Il est donc nécessaire de générer des représentations internes plus compactes, susceptibles de s'appliquer à une classe d'objets plus large que l'ensemble d'apprentissage. Cette notion de compacité (ou de simplicité) est fondamentale, elle est souvent utilisée comme critère de sélection des bonnes représentations dans les méthodes d'apprentissage basées sur la logique.

.3.1 Tester la généralisation

Une question essentielle se pose: comment tester la qualité d'une représentation interne générée par apprentissage? Deux méthodes principales sont possibles. La première, surtout utilisée en reconnaissance des formes, est de partager l'ensemble des exemples en un ensemble d'apprentissage et un ensemble de test, et de mesurer les performances

[6] sauf si l'on dispose d'une mesure de distance entre objets c'est la méthode des plus proches voisins

du système sur l'ensemble de test, après apprentissage. La seconde est d'interpréter la représentation extraite par le système (si tant est qu'elle soit interprétable) et de la comparer à un modèle connu par avance comme étant "le bon modèle". Cette méthode est souvent utilisée dans les domaines relevant classiquement de l'IA. Toutefois, elle ne peut être appliquée si l'on ne connait pas la solution par avance. En outre elle nécessite que les représentations générées soient interprétables ce qui n'est pas toujours le cas.

Il semble très difficile de formaliser la notion de généralisation. D'un point de vue objectif, aucune généralisation n'est meilleure qu'une autre puisque seules des données extérieures au problème (des exemples supplémentaires) permettront de mesurer sa qualité. En ce sens, l'apprentissage est un problème "mal posé", puisque la donnée des exemples ne suffit pas à le résoudre. On en est réduit à utiliser des critères subjectifs supposé pertinents, comme la compacité de la représentation générée, ou sa "simplicité" mesurée selon une méthode arbitraire.

.3.2 Recherche dans l'espace des représentations

Pour être en mesure de comparer des méthodes d'apprentissage, ou tout simplement d'évaluer leur performance, il faut pouvoir les tester sur plusieurs problèmes. En effet, d'une manière ou d'une autre, tout algorithme d'apprentissage se réduit à l'exploration (de préférence non exhaustive) d'un espace de représentations possibles. L'algorithme sélectionne la représentation en fonction des exemples et en se conformant à une stratégie de recherche. Mais la qualité réelle de l'algorithme dépend de la taille de cet espace. Prenons un exemple, imaginons qu'il s'agisse de "découvrir" la notion de nombre premier. Si l'espace à explorer ne contient que quelques dizaines de notions possibles (dont celle de nombre premier), il est clair que même une stratégie exhaustive (et exponentielle) finira par trouver la solution. Mais le système sera très limité, il ne pourra pas apprendre beaucoup de notions différentes. En revanche, si l'espace contient potentiellement toutes les propriétés des entiers naturels, alors la recherche risque d'être longue, quelque soit la stratégie de recherche employée. Il est important de remarquer que la complexité d'un concept appris n'influence pas du tout la difficulté de son apprentissage. C'est surtout le nombre des concepts "apprenables" qui la conditionne. Il semble que ce compromis entre les possibilités du système (sa généralité) et la complexité de l'apprentissage soit une nécessité. On ne peut pas espérer réaliser un système général d'apprentissage, et l'on doit se méfier des algorithmes exponentiels qui fonctionnent parfaitement sur de petits exemples particuliers, mais qui deviennent impraticables sur des problèmes de taille raisonnable en raison de la combinatoire.

En résumé, la mise en oeuvre d'un algorithme d'apprentissage se caractérise par un espace de représentations possibles, et par une stratégie de recherche dans cet espace. Les "bonnes" représentations sont sélectionnées à l'aide d'un critère subjectif supposé favoriser la généralisation.

.4 Les modèles connexionnistes de l'apprentissage

Après la parution du livre de Minsky et Papert, les travaux sur les réseaux neuronaux se sont raréfiés. Toutefois, quelques auteurs essentiellement japonais ont publié sur le sujet. Ces travaux ont principalement concerné les réseaux de neurones formels totalement connectés dans lesquels tout élément est connecté à tout autre à travers une matrice de pondérations. Amari [Amari 71], [Amari 72] étudie les propriétés de ces réseaux pour leur utilisation en tant que mémoires associatives, il propose l'idée de stocker des informations sous forme d'états stables du système, ou de séquences d'états. La règle de calcul des poids est la règle dite de Hebb [Hebb 49]: chaque poids est proportionnel à la covariance des activités des cellules qu'il relie. Simultanément Nakano [Nakano 71], [Nakano 72] propose un modèle identique appelé "Associatron" dans lequel les poids sont quantifiés sur 7 niveaux. Nakano a réalisé son associatron avec la technologie TTL standard de l'époque, selon une architecture très voisine des architectures systoliques modernes. En Europe, divers auteurs proposent des modèles de mémoires associatives linéaires basées sur la règle de Hebb [Kohonen 72], puis sur les pseudo-inverses [Kohonen & Ruohonen 73] [Kohonen 74], ou sur les théories "holographiques" de la mémoire de Longuet-Higgins, et les modèles binaires de Willshaw (voir dans [Willshaw 81]). Tous ces modèles sont également soumis à des limitations du même type que celles du perceptron. Fukushima propose le cognitron, un réseau de neurone multicouche pour la reconnaissance d'images [Fukushima 75] [Fukushima & Miyake 78], malheureusement, la règle d'apprentissage proposée est non supervisée, ce qui rend le système difficilement utilisable. En outre le modèle est extrêmement complexe et les simulations peu reproductibles. Little [Little 74] propose l'utilisation des méthodes de la physique statistique pour l'étude des réseaux de neurones.

De nombreux travaux de la fin des années 70 sont regroupés dans [Hinton & Anderson 81], en particulier le modèle "Brain-State-in-the-Box" de Anderson, les réseaux sémantiques distribués de Hinton, les cartes topologiques de Kohonen. Les travaux de Kohonen sont regroupés dans [Kohonen 84a].

En 1982 Hopfield propose un modèle identique au modèle d'Amari et à l'associatron,

dont il caractérise la dynamique à l'aide d'une fonction "d'énergie". Cet article très clair, va intéresser les physiciens en raison de l'analogie du modèle de Hopfield avec les verres de spins. De nombreuses publications vont alors apparaître sur la question dans les journaux de physique, certaines réinventant des modèles déjà existants. Il est amusant de constater que cet engouement a été suscité par un modèle qui NE LEVAIT PAS les limitations du perceptron et de ses variantes. Le perceptron et les raisons de son échec semblaient oubliés. Néanmoins le modèle de Hopfield et ses nombreuses variantes susciteront une importante quantité de publications, contenant de nombreux concepts nouveaux, et de nouvelles méthodes d'analyse issues de la physique statistique.

Puis Hinton, Sejnowski et Ackley proposent la "machine de Boltzmann", le premier modèle à lever les limitations du perceptron de façon satisfaisante [Hinton & Sejnowski 83], [Hinton & Al. 84c], [Ackley & al. 85], [Sejnowski & Hinton 85], [Sejnowski & al. 85]. Les machines de Boltzmann utilisent un ensemble de cellules dites "cachées" sans interaction directe avec l'extérieur, dont le rôle est de calculer les variables intermédiaires permettant de réaliser des fonctions non-linéairement séparables. Malheureusement la convergence de l'algorithme s'avère extrêmement longue, en raison du caractère probabiliste des éléments utilisés.

Ce défaut sera corrigé par l'algorithme de rétro-propagation proposé sous diverses formes dans [le Cun 85], [le Cun 86], [Parker 85] et [Rumelhart & al. 86a]. L'algorithme de rétro-propagation utilise des réseaux multi-couche de cellules "continues" et non plus binaires. La particularité de cet algorithme est qu'il modifie les poids des cellules (cachées) des couches intermédiaires. Ces cellules intermédaires élaborent les représentations internes que l'on espère adéquates et décomposent la relation entrée-sortie en suite d'étapes linéairement séparables [7]. Cet algorithme, bien que récent, a déjà fait l'objet de très nombreuses études et suscite un intérêt croissant [Hinton 86], [Hinton 86], [Sejnowski & Rosenberg 86], [le Cun & Fogelman 87a], [Fogelman & al. 87b], [Gallinari & al. 87].

.5 Plan de la thèse

En raison du caractère pluridisciplinaire du domaine et de l'éparpillement des publications relatives aux travaux des précurseurs, il a semblé utile de consacrer un chapitre à l'apprentissage des poids d'une cellule unique. Ce chapitre contient la description intuitive et formelle des algorithmes adaptatifs simples dit "à correction d'erreur". Principalement l'algorithme du perceptron et sa preuve de convergence, et les algorithme de moindres carrés dont l'algorithme de Widrow et Hoff. Les conditions de

[7] dans la mesure ou ce terme est applicable à des cellules non-binaires

convergence, ainsi que le temps de convergence des algorithmes de moindre carrés sont largement étudiés, afin d'introduire les méthodes d'accélération de la rétro-propagation décrites au chapitre 3. Auparavant sera présenté l'élément de base: le neurone formel, ainsi que les théorèmes de limitation le concernant. Le dernier sous-chapitre est consacré au perceptron et aux théorèmes de limitation de Minsky et Papert. Ce chapitre est essentiellement une synthèse bibliographique.

Le chapitre 2 présente les modèles "classiques" de mémoires associatives en particulier les travaux de Nakano, Amari, Kohonen, Hopfield et Anderson. On présentera quelques analyses originales des modèles d'apprentissage utilisant les matrices pseudo-inverses. Il sera mis l'accent sur l'identité entre les modèles basés sur les pseudo-inverses, et les modèles basés sur la minimisation d'un critère quadratique. La plupart de ces travaux est publiée dans [Fogelman & al. 87b]. Un nouveau modèle de réseau associatif est également présenté.

Le chapitre 3 est consacré aux algorithmes d'apprentissage pour réseaux à cellules cachées, et plus particulièrement à l'algorithme de rétro-propagation. Après la présentation des premières idées comme les Madalines de Widrow et les machines de Boltzmann, nous dérivons l'algorithme de rétro-propagation. Puis nous décrivons quelques simulations et propriétés de l'algorithme. Les variantes de l'algorithme de base sont présentées, en particulier l'algorithme HLM, proposé par l'auteur antérieurement à la rétro-propagation. Ensuite l'algorithme est dérivé à l'aide du formalisme Lagrangien, et quelques généralisations sont décrites. L'algorithme a été appliqué à plusieurs problèmes, dont la reconnaissance de chiffres manuscrits et le diagnostic des douleurs abdominales. Un sous chapitre est consacré à l'analyse de l'algorithme et aux techniques d'accélération de la convergence.

Le chapitre 4 contient le manuel du logiciel de simulation HLM. [8] HLM est un ensemble de logiciels de création, de manipulation et de simulation de réseaux utilisant l'algorithme de rétro-propagation et ses variantes.

[8] A ne pas confondre avec l'algorithme du même nom

Chapitre 1

Méthodes élémentaires d'apprentissage

L'objet de ce chapitre est de réunir de manière synthétique sous un même formalisme les différentes méthodes d'apprentissage qui peuvent se révéler utiles à l'analyse des modèles connexionnistes. Parmi celles-ci, nous traiterons essentiellement des méthodes concernant les classifieurs linéaires. En outre, il sera traité du comportement de convergence de divers algorithmes dans le cas linéaire, en s'appuyant pour ce faire sur la littérature concernant les filtres adaptatifs [Widrow & Stearn 85].

Nous nous référons principalement à deux ouvrages: [Duda & Hart 73] et [Nilsson 65].

Nous commencerons par analyser les capacités des classifieurs linéaires et de leurs dérivés, afin d'expliquer les raisons de leur relatif échec. La référence en ce domaine est [Minsky & Papert 68].

1.1 L'automate à seuil

L'automate à seuil, ou NEURONE FORMEL est apparu pour la première fois dans un article de MacCulloch et Pitts datant de 1943 [McCulloch & Pitts 43], dans lequel les auteurs le proposent comme modèle, très idéalisé, du neurone. Le neurone formel y est décrit comme un automate possédant plusieurs entrées booléennes $(e_1, e_2, \ldots, e_n)$ et une sortie également booléenne s. Sa fonction élémentaire est d'effectuer une somme pondérée de ses entrées. Si cette somme est supérieure à un seuil θ (fixé) l'état de l'automate (qui constitue également sa sortie) est +1, dans le cas contraire, il vaut 0 :

$$s = \begin{cases} 1 & \text{si } \sum_{i=1}^{n} w_i e_i > \theta \\ 0 & \text{sinon} \end{cases} \tag{1.1}$$

les w_i sont les pondérations qui définissent la fonction de transition de l'automate considéré (voir figure 1. 1).

En fait, il n'est pas utile de contraindre les entrées à être binaires. Cette restriction n'est intéressante que si l'on veut utiliser le formalisme de la logique ou de la théorie des automates. Dans tous les autres cas on considèrera des entrées à valeur réelle, Ce qui permettra de considérer le vecteur d'entrée comme un élément de R^n. Le nom attribué à l'opérateur défini par l'équation ci-dessus et ses variantes, dépend du contexte dans lequel on se place. En biologie théorique c'est le neurone formel, en reconnaissance des formes c'est le classifieur linéaire, en théorie des automates l'automate à seuil, et dans certains ouvrages l'élement linéaire à seuil ou ELS. Dans la suite nous nous référerons indifféremment (et parfois abusivement) à l'une ou l'autre des appellations.

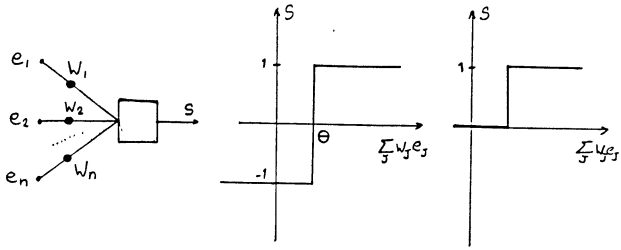


Figure 1. 1: Un automate à seuil, ou classifieur linéaire, reçoit des entrées e_i qui sont sommées à l'aide des pondérations w_i. Le résultat est transformé à l'aide d'une fonction à seuil de paramètre θ. La sortie est prise dans l'ensemble {0,1}, ou {-1,1}, selon le cas

L'équation 1.1 peut être réécrite en utilisant la fonction de Heaviside h:

$$s = h(a) \quad \text{avec la notation:} \quad a = \sum_{i=1}^{n} w_i e_i - \theta \tag{1.2}$$

Le terme a est la somme pondérée des entrées de l'ELS et est appelé ENTREE TOTALE. Cette quantité peut être considérée comme une fonction du vecteur d'entrée, elle est appelée dans ce cas FONCTION LINEAIRE DISCRIMINANTE.

Du point de vue de la théorie des automates, un neurone formel est un automate fini qui possède deux états, et dont la fonction de transition est définie par l'équation 1.1. Toutefois les termes "état" et "fonction de transition" sont ici utilisés dans un sens particulier. Dans la suite nous utiliserons ces termes même dans le cas ou le neurone formel est considéré comme une fonction combinatoire, c'est à dire, sans mémoire et par conséquent sans état ni fonction de transition. Cette distinction n'est utile que dans certains cas; typiquement lorsque les changements d'états sont SYNCHRONES, c'est à dire lorsqu'ils interviennent à des instants discrets bien définis. Dans ce cas l'équation 1.2 doit être légèrement modifiée pour tenir compte de la dimension temporelle. La fonction de transition devient

$$s(t) = h(a(t)) \quad \text{avec} \quad a(t) = \sum_{i=1}^{n} w_i e_i(t-1) - \theta \tag{1.3}$$

où t est un indice temporel discret.

1.1.1 généralisations et notations

Dans ce qui précède, les deux valeurs possibles de l'état sont notées 0 et 1 pour des raisons de cohérence avec la logique, néanmoins, il est plus commode dans certaines situations de noter les deux états -1 et +1. La fonction de Heaviside est remplacée par la fonction signe. Naturellement, d'un point de vue général, ces deux notations sont totalement équivalentes: tout ELS en 0,1 possède un équivalent en -1,+1. Si $(w_1,\ldots,w_n\theta)$ sont les paramètres d'un ELS en 0,1 les paramètres de l'ELS en -1,+1 équivalent sont

$$(w_1',\ldots,w_n',\theta')$$

où

$$w_i' = \frac{1}{2}w_i$$

et

$$\theta' = \theta - \frac{1}{2}\sum_{i=1}^{n} w_i$$

Nous pouvons simplifier les notations en considérant le seuil θ comme un poids particulier dont l'entrée correspondante vaut toujours -1. Le vecteur d'entrée ainsi modifié est appelé VECTEUR D'ENTRÉE AUGMENTÉ: $(-1,e_1,\ldots,e_n)$. Le seuil est alors tout simplement w_0 et l'équation 1.1 devient:

$$s = \begin{cases} +1 & \text{si } \sum_{i=0}^{n} w_i e_i > 0 \\ -1 & \text{sinon} \end{cases} \tag{1.4}$$

Cette notation est particulièrement utile lorsque l'on aborde le problème de l'apprentissage, le seuil est alors traité comme un poids ordinaire.

Deux généralisations des ELS viennent immédiatement à l'esprit. La première consiste à généraliser la forme de l'expression liant le vecteur d'entrée à l'entrée totale, par exemple en utilisant un polynôme de degré quelconque au lieu d'une simple combinaison linéaire. Ces éléments sont alors appelés CELLULES SIGMA-PI [Rumelhart & al. 86a] ou FONCTIONS POLYNOMIALES DISCRIMINANTES ou encore Φ-machines [Nilsson 65]. La deuxième généralisation concerne la forme donnée à la transformation non linéaire. La fonction à seuil est intéressante lorsqu'il s'agit de faire des discriminations, mais on peut vouloir utiliser une fonction plus complexe dans certaines applications. La sortie s'écrit sous la forme générale suivante:

$$s = f(a) \quad \text{avec} \quad a = \sum_{i=0}^{n} w_i e_i \tag{1.5}$$

où f est maintenant une fonction à valeur réelle, généralement choisie croissante et impaire (mais ceci n'est pas une restriction de principe). Dans ce contexte, il devient

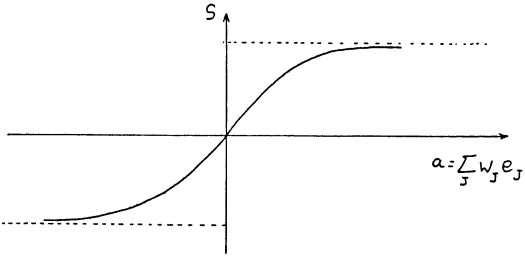


Figure 1. 2: fonction couramment utilisée pour la transformation non-linéaire

difficile de parler "d'automate", nous préfèrerons donc le mot CELLULE. La fonction f sera typiquement de la forme suivante:

$$f(x) = m\frac{\exp kx - 1}{\exp kx + 1} \tag{1.6}$$

cette fonction est impaire, strictement croissante, et possède deux asymptotes horizontales d'ordonnées $-m$ et $+m$, comme indiqué sur la figure 1. 2. En outre f tend vers la fonction signe lorsque k tend vers l'infini.

Dans la suite nous utiliserons les notations vectorielles ci-dessous:

$$E = \begin{pmatrix} e_0 \\ | \\ e_n \end{pmatrix} \qquad W = \begin{pmatrix} w_0 \\ | \\ w_n \end{pmatrix}$$

l'équation 1.5 s'écrit alors:

$$s = f(a) \quad \text{avec} \quad a = W^T E \tag{1.7}$$

où W^T est le vecteur W transposé.

Dans la suite les lettres minuscules serviront à désigner les quantités scalaires, les majuscules les vecteurs et les matrices.

Ce sous-chapitre a présenté les différents types de neurones formels, du plus spécifique au plus général. En résumé, ils se distinguent par les points suivants:

- La nature des entrées

- binaires: en {0,1}, ou {-1,+1}
- réelles

- La transformation non-linéaire de sortie
  - binaire à seuil: en {0,1}, ou {-1,+1}
  - à valeur réelle

- La fonction de combinaison des entrées
  - Linéaire avec un seuil (w_0) nul ou non-nul
  - Polynômiale de degré supérieur à 2

1.1.2 Séparabilité linéaire

Il est important d'évaluer la "puissance de calcul" des ELS, c'est à dire la complexité des fonctions qu'ils peuvent réaliser. Nous nous limiterons au cas d'un élément dont la transformation non-linéaire de sortie est binaire en -1,+1. Un ELS réalise une partition de son espace d'entrée en deux classes $\mathcal{E}^+$ et $\mathcal{E}^-$. Les vecteurs appartenant à la région $\mathcal{E}^+$ (resp. $\mathcal{E}^-$) produisent une sortie égale à +1 (resp. -1). La frontière entre ces deux zones est, dans le cas des ELS, un hyperplan. C'est la raison pour laquelle les fonctions calculables par un ELS sont dites LINEAIREMENT SEPARABLES . L'ensemble des bi-partitions linéairement séparables d'une collection de points est un sous ensemble strict de l'ensemble de toutes les bi-partitions possibles de cette collection. Il est intéressant et important d'estimer la proportion de fonctions linéairement séparables par rapport à toutes les dichotomies possibles. Mais ceci ne peut être fait dans le cas général, des hypothèses doivent être formulées concernant la nature de la collection de points à séparer.

Il existe une interprétation géométrique des fonctions linéaires à seuil, mais qui n'est, bien sûr, visualisable qu'en dimension 2 ou 3. Limitons nous dans un premier temps aux fonctions booléennes de 2 variables. Dans ce cas la situation est très simple: La sortie dépend du signe de $w_1e_1 + w_2e_2 - w_0$. L'équation

$$w_1e_1 + w_2e_2 - w_0 = 0$$

définit une surface séparatrice dans l'espace des entrées, en l'occurence une droite, d'un côté duquel la sortie sera +1 et de l'autre -1. L'espace des entrées est donc divisé en deux demi-espaces séparant les vecteurs d'entrée en deux classes. En dimension n

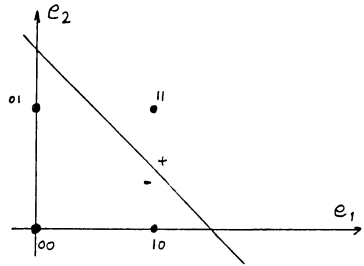


Figure 1. 3: exemple d'ELS réalisant une fonction booléenne ET à deux entrées. L'ensemble des quatre vecteurs d'entrée {(0,0),(0,1),(1,0),(1,1)}, est séparé par une droite orthogonale au vecteur de poids d'équation $W^T E = 0$. Ici, $W = (1.5,1,1)$, le point (1,1) est du côte positif, les 3 autres du côté négatif.

cette surface séparatrice est un HYPERPLAN de dimension $n-1$ orthogonal au vecteur de poids [1]. Lorsque $w_0 = 0$ cet hyperplan passe par l'origine.

La situation est claire sur la figure 1. 3 où l'on a représenté la droite séparatrice réalisant un ET à deux entrées. Le ET vaut 1 lorsque ses deux entrées valent 1, et 0 dans les trois autres cas. La droite sépare donc le point (1,1) des trois autres, en plaçant celui-ci du côté des $W^T E$ positifs, c'est à dire du même côté que l'extrémité du vecteur W, et les trois autres du côté négatif.

Il apparaît immédiatement que deux fonctions booléennes de deux entrées posent problème. La première est le OU EXCLUSIF (XOR). En effet, pour le XOR, les points (1,0) et (0,1), doivent être du côté positif, et les points (0,0) et (1,1) du côté négatif, ce qui, de manière évidente, est impossible à réaliser. La seconde fonction problématique est l'opposé du XOR, c'est à dire la fonction EQUIVALENCE. En dimension 2, le XOR et L'EQUIVALENCE sont les deux seules fonctions booléennes non-linéairement séparables sur les 16 possibles. Sur la figure 1. 4 sont représentées toutes les fonctions booléennes linéairement séparables de deux entrées.

[1]Selon que l'on considère l'entrée fictive e_0 comme une dimension à part entière de l'espace vectoriel de représentation, cet hyperplan est de dimension $n-1$ dans un espace de dimension n et ne passe pas (nécessairement) par l'origine, ou de dimension n dans un espace de dimension $n+1$ et passe par l'origine

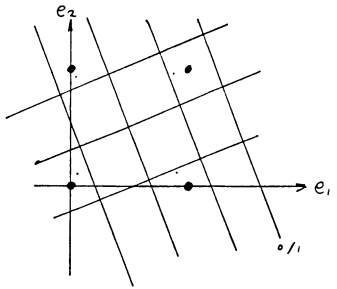


Figure 1. 4: 7 droites séparatrices correspondant aux 14 fonctions booléennes de 2 variables linéairement séparables.

La situation en dimension n est plus complexe, les 2^n vecteurs binaires possibles sont situés aux sommets de l'hypercube unité. Et il existe 2^{2^n} fonctions booléennes possibles de n entrées [2]. Malheureusement, la proportion de fonctions booléennes linéairement séparables de n entrées décroit EXPONENTIELLEMENT avec n [Nilsson 65]. Ce qui veut dire que, dès que n est grand, pratiquement AUCUNE fonction booléenne de n entrées n'est réalisable par un ELS simple.

1.1.3 Dénombrement des fonctions linéairement séparables

Le paragraphe suivant concerne l'estimation de la quantité $\mathcal{L}(n,p)$: le nombre de dichotomies linéaires de p vecteurs de dimension n (non nécessairement booléens). Le nombre de dichotomies possibles (quelconques) de p vecteurs est 2^p. Il sera fait l'hypothèse que les p vecteurs sont en POSITION GÉNÉRALE, c'est à dire qu'aucune famille de $n+1$ vecteurs choisis parmi les p n'est contenue dans un hyperplan de dimension $n-1$. Il est important de remarquer que cette condition n'est PAS VÉRIFIÉE si l'on s'intéresse aux fonctions booléennes [3] (c'est à dire aux vecteurs binaires). Néanmoins, ceci nous donne une borne supérieure au nombre de fonctions BOOLÉENNES linéairement séparables.

[2] c'est le nombre de séquences binaires de 2^n bits

[3] en dimension 3, les 8 vecteurs possibles forment les sommets d'un cube dont chaque face (contenue dans un plan) possède 4 sommets. La condition n'est, par conséquent, pas vérifiée dans ce cas.

estimation du nombre de partitions linéaires

De nombreux auteurs ont donné des dérivations de $\mathcal{L}(n,p)$ [Joseph 60], [Winder 62], [Cameron 60]. Le raisonnement suivant est dû à Cover [Cover 65] et repris par Nilsson [Nilsson 65]. L'idée est d'établir une relation de récurrence. Soit un ensemble $\mathcal{E}_{p-1}$ comportant $p-1$ points en position générale dans un espace de dimension n. Il y a $\mathcal{L}(n,p-1)$ dichotomies linéaires de $\mathcal{E}_{p-1}$. Construisons l'ensemble $\mathcal{E}_p$ en ajoutant un point E_p en position générale par rapport à ceux de $\mathcal{E}_{p-1}$. Nous allons regrouper les dichotomies de $\mathcal{E}_{p-1}$ en deux sortes: celles qui peuvent être définies par un hyperplan passant par E_p, et les autres. Notons $L_{E_p}(n,p-1)$ le nombre de dichotomies appartenant à la première catégorie. A toute dichotomie sur $\mathcal{E}_{p-1}$ de la deuxième catégorie correspond une et une seule dichotomie sur $\mathcal{E}_p$. Pour chaque dichotomie sur $\mathcal{E}_{p-1}$ de la première catégorie il en existe DEUX correspondantes sur $\mathcal{E}_p$; celles qui laissent E_p du côté positif et celles qui le laissent du côté négatif. Ces deux dichotomies sont construites en considérant un hyperplan passant par E_p que l'on déplace infiniment peu dans un sens ou dans l'autre. Ainsi, sans perturber la classification opérée sur $\mathcal{E}_{p-1}$ par cet hyperplan, nous pouvons placer E_p du côté positif ou du côté négatif construisant ainsi 2 dichotomies sur $\mathcal{E}_p$. Les dichotomies de la première catégorie (contraintes) étant au nombre de

$$\mathcal{L}(n,p-1) - L_{E_p}(n,p-1)$$

le nombre de dichotomies sur $\mathcal{E}_p$ est, de manière évidente, égal à

$$\mathcal{L}(n,p-1) - L_{E_p}(n,p-1) + 2L_{E_p}(n,p-1)$$

soit

$$\mathcal{L}(n,p-1) + L_{E_p}(n,p-1)$$

La situation en dimension 2 est clarifiée sur la figure 1. 5.

Il convient maintenant de calculer $L_{E_p}(n,p-1)$. Il va être montré que $L_{E_p}(n,p-1) = \mathcal{L}(n-1,p-1)$. Relions chacun des $p-1$ points de $\mathcal{E}_{p-1}$ à E_p par $p-1$ droites. Ces droites sont distinctes, compte tenu de l'hypothèse de position générale. Construisons un hyperplan H (de dimension $n-1$) qui coupe toutes ces droites, et considérons l'ensemble des points intersections de H et des droites. Ces points sont en position générale dans le sous-espace H de dimension $n-1$. De manière évidente, à toute dichotomie sur $\mathcal{E}_{p-1}$ définie par un hyperplan de dimension $n-1$ passant par E_p, correspond une dichotomie sur l'ensemble des points intersections. Cette dichotomie est réalisée par un hyperplan de dimension $n-2$ (cf figure 1. 6). Nous concluons donc que:

$$L_{E_p}(n,p-1) = \mathcal{L}(n-1,p-1)$$

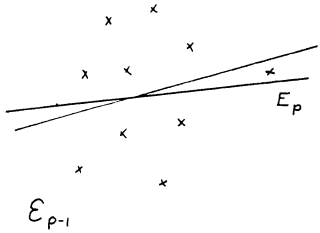


Figure 1. 5: Deux dichotomies construites par rotation d'une droite

Nous pouvons donc établir la relation de récurrence:

$$\mathcal{L}(n,p) = \mathcal{L}(n,p-1) + \mathcal{L}(n-1,p-1) \tag{1.8}$$

Nous disposons des conditions aux bornes:

$$\mathcal{L}(n,1) = 2 \qquad \mathcal{L}(1,p) = 2p$$

d'où nous déduisons:

$$\mathcal{L}(n,p) = 2\sum_{i=0}^{n} C_{p-1}^{i} \quad \text{pour } p > n \quad \text{et} \quad \mathcal{L}(n,p) = 2^p \quad \text{pour } p \leq n \tag{1.9}$$

où C_{p-1}^{i} est la notation usuelle du coefficient binômial.

Probabilité qu'une partition donnée soit linéairement séparable

En situation d'apprentissage supervisé, la partition est généralement définie par le superviseur qui fourni à l'opérateur de reconnaissance à la fois la collection des exemples et leur classe d'appartenance. Il est donc intéressant d'estimer la probabilité qu'une partition donnée soit linéairement séparable. Cover [Cover 65] donne une estimation de cette quantité non seulement pour les séparateurs linéaires, mais également pour les séparateurs d'ordre 2 (elliptiques, hyperboliques,...). Le résultat est le suivant:

Theoreme 1.1 Soit $\mathcal{E}$ un ensemble de p vecteurs de dimension n. Si aucun sous ensemble de $\mathcal{E}$ de cardinal $n+1$ n'est contenu dans un hyperplan de dimension $n-1$

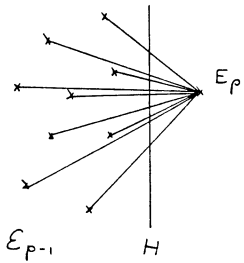


Figure 1. 6: réduction de dimension pour le calcul des partitions linéaires.

(hypothèse de la "position générale"), alors la probabilité $P(n,p)$ qu'une dichotomie sur $\mathcal{E}$ soit linéairement séparable est:

$$P(n,p) = 2^{1-p} \sum_{i=0}^{n} C_{p-1}^{i} \quad \text{pour} \quad p > n$$

et

$$P(n,p) = 1 \quad \text{pour} \quad p \leq n$$

Ce théorème est une conséquence directe du résultat précédent, si l'on considére que toutes les dichotomies sont équiprobables, et qu'elles sont au nombre de 2^p. Ce résultat concerne les ELS à $n+1$ entrées, c'est à dire avec un seuil w_0 éventuellement non nul. La fonction $P(n,p)$ est représentée sur la figure 1. 7. On constate que cette fonction décroît très vite dès que p dépasse $2(n+1)$, et que $P(n,2(n+1)) = \frac{1}{2}$. Cette décroissance autour de $p = 2(n+1)$ devient plus brutale à mesure que n augmente, pour devenir un seuil lorsque $n \to \infty$. Cover propose donc la définition suivante:

la CAPACITÉ d'un séparateur linéaire à n entrées ($n+1$ poids modifiables) C_n est donnée par:

$$C_n = 2(n+1)$$

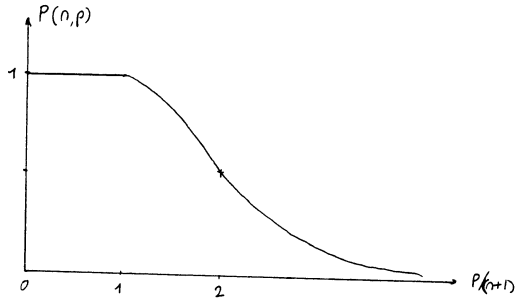


Figure 1. 7: Probabilité qu'une dichotomie choisie au hasard sur un ensemble de p vecteurs de dimension n en position générale soit linéairement séparable. Les abscisses sont exprimées en coordonnées réduites $p/(n+1)$

Ceci montre que la capacité d'un ELS est LINEAIRE en fonction de son nombre d'entrées. En situation réelle, ces résultats ne peuvent fournir qu'une indication approximative. En effet, il est rare que l'on puisse vérifier la validité de l'hypothèse de position générale, et lorsque c'est possible, c'est, la plupart du temps, pour montrer sa non-validité. Par ailleurs, il sera souvent fait usage d'éléments non binaires (utilisant la fonction sigmoïde de la figure 1.6 par exemple) pour lesquels, les résultats précédents n'ont qu'une valeur heuristique. Cela donne néanmoins une bonne estimation de ce que l'on est en droit d'espérer d'un classifieur linéaire.

1.1.4 Utilisation des classifieurs linéaires en Reconnaissance des Formes

Lorsqu'il y a plus de 2 classes

L'utilisation des ELS n'est évidemment pas restreinte aux problèmes à deux classes. Plusieurs généralisations directes à c classes sont accessibles. La plus simple consiste à disposer de c ELS possédant les mêmes entrées. La décision finale est définie comme l'indice de l'ELS dont l'entrée totale est maximale. Cette règle de décision définit une surface séparatrice linéaire par morceaux. Les zones de l'espace d'entrée correspondant à chacune des classes sont nécessairement convexes ce qui en limite les possibilités

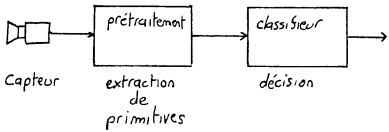


Figure 1. 8: Le schéma classique de la reconnaissance de formes.

[Duda & Hart 73]. Les opérateurs de ce type sont qualifiés dans la littérature de MACHINES LINÉAIRES.

Le schéma classique de la reconnaissance des formes

Malgré leurs limitations intrinsèques, les classifieurs linéaires sont extrêmement utilisés en reconnaissance des formes (RdF). Les principales raisons de ce succès sont leur simplicité de mise en oeuvre, et la somme des connaissances théoriques les concernant. Le schéma devenu classique de (presque) tout opérateur de RdF est représenté sur la figure 1. 8.

Il est composé d'un organe d'acquisition, d'un étage de prétraitement et d'extraction de primitives, suivi d'un opérateur de décision, ou de classification. Il est courant que les deux premières étapes soient dépendantes du problème à traiter alors que la dernière fasse appel à des techniques plus ou moins générales telles que les méthodes de classification statistique dont le classifieur linéaire est l'archétype, ou les méthodes structurelles [Miclet 84]. Dans la plupart des cas, seul l'opérateur de classification est modifiable par apprentissage, la partie prétraitement étant fixe. Pour un problème donné, l'importance du travail de conception est donc directement reliée aux tailles respectives de chacun des éléments: plus le classifieur est complexe (ou potentiellement complexe), moins l'étage de prétraitement le sera et, par conséquent, plus le travail de conception sera réduit.

L'utilisation d'un classifieur linéaire réduit l'élément de décision à sa plus simple

expression. L'étage de prétraitement doit alors être soigneusement conçu afin que les classes, après prétraitement soient linéairement séparables. Ce travail étant à refaire pour chaque nouveau problème, il serait souhaitable d'utiliser un classifieur aux possibilités moins limitées afin de réduire la complexité du prétraitement. L'idéal inaccessible étant de supprimer totalement le prétraitement et de l'intégrer dans le classifieur tout en conservant les capacités d'apprentissage.

1.1.5 les assemblages de neurones formels

Si les capacités d'un séparateur linéaire sont extrêmement limitées, il n'en va pas de même des ASSEMBLAGES de tels éléments. Les assemblage d'ELS, ou RÉSEAUX, sont de deux types principaux:

- Les réseaux combinatoires (sans boucle)
- Les réseaux itératifs (avec boucles)

ce paragraphe ne concerne que la première catégorie. Les réseaux combinatoires correspondent aux fonctions booléennes combinatoires, c'est à dire qui ne possédent pas d'état interne. La sortie à un instant t dépend uniquement de l'entrée à ce même instant (ou à un instant légèrement antérieur si l'on ne veut pas négliger le temps de transit des informations dans le réseau). Ils sont caractérisés par le fait que le graphe de connexion est sans circuit. En d'autres termes, il est possible d'étiqueter chaque élément à l'intérieur du réseau de manière à ce que la sortie d'un élément ne dépende que de la sortie d'éléments d'indices inférieurs au sien. On définit le NOMBRE DE COUCHES d'un réseau combinatoire comme étant la longueur du chemin le plus long reliant une entrée quelconque à une sortie quelconque. Nous pouvons énoncer le théorème suivant:

Theoreme 1.2 Toute fonction booléenne peut être réalisée par un réseau d'automates à seuil à deux couches.

Sans donner de démonstration rigoureuse de ce théorème, nous pouvons donner l'idée de sa preuve. une fonction logique peut être mise sous forme d'une disjonction de conjonctions (un OU de ET). Toute disjonction est réalisable par un automate à seuil, ainsi que toute conjonction [4]. Il s'ensuit que toute fonction booléenne peut être calculée par un réseau d'automates à seuil dont la première couche réalise une série de ET et la deuxième un OU des sorties de la première. Toutefois, il faut remarquer que

[4] les disjonctions et conjonctions peuvent comprendre des négations de variables d'entrée

pour réaliser certaines fonctions, le nombre de disjonctions nécessaires est énorme: éventuellement exponentiel en fonction du nombre d'entrées. Les conséquences de ceci seront explorées en détail dans le paragraphe 1.6 traitant du perceptron.

Un argument similaire sera développé ultérieurement pour montrer que toute fonction vectorielle réelle peut être approximée aussi près que l'on veut par un réseau à 3 couches constitué de fonctions croissantes linéaires par morceaux.

Il n'est pas inutile de remarquer que la décomposition d'une fonction booléenne en disjonction de conjonctions est à la base des PLA [5] souvent utilisés comme éléments de construction des circuits électroniques numériques. Cette remarque peut être étendue si l'on constate qu'avec les technologies actuelles, la réalisation d'un XOR nécessite au moins deux couches logiques, une couche logique correspondant à un transistor traversé par le signal. Dans les circuits logiques, un transistor est essentiellement utilisé en tant qu'élément à seuil. Certains auteurs proposent même l'utilisation d'éléments logiques à seuil pour la réalisation de circuits complexes plus compacts que les dispositifs classiques [Kobylarz 80].

Pour l'étude des propriétés logiques des réseaux de neurones formels on peut se reporter à [Minsky 67]. La décomposition de la plupart des circuits logiques de base en "portes logiques à seuil" y est étudiée (bascules, additionneurs, etc).

1.2 Apprentissage des poids d'un neurone formel

L'intérêt des modèles tels que les neurones formels réside dans le fait que la fonction qu'ils réalisent est paramétrée par un ensemble de variables CONTINUES: les pondérations. Il est donc aisé de définir une notion de distance entre ELS en utilisant la distance définie dans l'espace vectoriel des paramètres. Ceci sera largement mis à profit pour l'étude des algorithmes d'apprentissage. Ceux-ci reposent sur le calcul (généralement itératif) d'un vecteur de poids permettant de classer au mieux un ensemble d'exemples dont la classe est connue.

Le paragraphe suivant présente de manière très intuitive et informelle l'idée de base des algorithmes d'apprentissage itératifs pour ELS. Ils seront ensuite présentés de manière plus formelle en passant en revue une grande part des modèles existant dans la littérature.

[5] Programmable Logic Arrays

1.2.1 apprentissage par "essai et erreur"

L'apprentissage par correction d'erreur désigne une situation D'APPRENTISSAGE SUPERVISÉ dans lequel on dispose d'un ensemble d'exemples dont la classe d'appartenance est connue. On dispose donc d'un ensemble de formes (au sens large) $\{f_1, \ldots, f_p\}$ chacune associée à sa classe d'appartenance $\mathcal{E}^+$ ou $\mathcal{E}^-$. Il s'agit, dans le cas qui nous occupe, de trouver un vecteur de poids W qui permette à un ELS de répondre +1 lorsqu'une forme appartenant à $\mathcal{E}^+$ lui est présentée, et -1 dans le cas contraire. L'apprentissage itératif désigne le cas où, durant la session d'apprentissage, les formes sont présentées une à une et, si nécessaire, répétées jusqu'à l'obtention d'un résultat satisfaisant. Après chaque présentation (ou groupe de présentations), le vecteur de poids est éventuellement modifié, et c'est ce processus de modification des poids qui tiendra lieu d'apprentissage.

Chaque forme f_k est représentée par un vecteur de R^n, E_k. Les classes $\mathcal{E}^+$ et $\mathcal{E}^-$ sont associées respectivement aux valeurs de sortie +1 et -1. L'ENSEMBLE D'APPRENTISSAGE sera donc décrit par un ensemble de paires $\{(E_k, d_k)\}_{k=1..p}$ où d_k est la SORTIE DESIRÉE associée au vecteur E_k et est définie par:

$$d_k = +1 \leftrightarrow E_k \in \mathcal{E}^+ \quad d_k = -1 \leftrightarrow E_k \in \mathcal{E}^-$$

Plaçons-nous dans le cas où une forme E vient d'être présentée, en même temps que la sortie désirée associée d. L'ELS a calculé la somme pondérée de ses entrées a et a produit la sortie s égale au signe de a. Quatre cas peuvent se produire:

- $d = +1$ et $s = +1$
- $d = -1$ et $s = -1$
- $d = +1$ et $s = -1$
- $d = -1$ et $s = +1$

Nous pouvons considérer que dans les deux premiers cas, le vecteur de poids a fourni la bonne réponse, et qu'il n'est pas nécessaire de le modifier. Le troisième cas correspond de manière évidente à une situation où la somme pondérée des entrées est négative, alors qu'elle devrait être positive. Cet état de fait peut être corrigé si l'on augmente d'une petite quantité les poids dont l'entrée est positive, et si l'on diminue légèrement les poids dont l'entrée est négative (l'opération inverse peut être appliquée dans le quatrième cas). Par ce procédé, l'entrée totale a sera augmentée d'une quantité positive égale à la somme des modifications pondérées par leurs entrées respectives.

Ce processus aura donc tendance à rapprocher a de sa valeur idéale (ou de son ensemble de valeurs idéales) pour la forme considérée. L'application de cette règle de modification sera réitérée pour chaque forme de la base d'exemples, et l'on peut espérer que le processus convergera au bout d'un certain nombre de répétitions de la base, vers un vecteur de poids produisant la réponse correcte pour tous les exemples.

Prenons par exemple la règle de modification suivante dite RÈGLE DU PERCEPTRON [Rosenblatt 57]:

$$w_i \leftarrow w_i + \frac{1}{2}\lambda(d-s)e_i \quad \forall i \in [0,n] \tag{1.10}$$

où λ est une constante positive. Lorsque $d = s$ les poids ne sont pas modifiés. Lorsque $d = +1$ et $s = -1$ alors $d - s = 2$ et le poids d'indice i est modifié de la quantité λe_i. D'une manière générale, l'entrée totale a sera modifiée de la quantité:

$$\Delta a = \lambda(d-s)\sum_{i=0}^{n} e_i^2 = \lambda(d-s)||E||^2$$

si d est différent de s, cette quantité est du signe de d. L'application répétée de la règle d'apprentissage tendra donc à faire que a soit de même signe que d. La quantité $d - s$ (notée ϵ dans la suite) est un SIGNAL D'ERREUR qui guide l'apprentissage et qui intervient sous différentes formes dans tous les algorithmes d'apprentissage dits à CORRECTION D'ERREUR. Ce qui précède ne constitue pas une démonstration de convergence, mais juste une présentation intuitive du mécanisme de correction d'erreur. Il sera montré au paragraphe 1.3.1 que la règle du perceptron converge si une solution existe, c'est à dire si les vecteurs de la base d'exemples sont linéairement séparables.

Il est également possible d'interpréter géométriquement l'algorithme précédent. La situation en dimension 2 est représentée sur la figure 1. 9. Le vecteur de poids courant est orthogonal à la droite séparatrice. Un vecteur $\boldsymbol{E}$ de la classe $\mathcal{E}^+$ est présenté, et se trouve malencontreusement du côté négatif. L'application de la formule d'apprentissage ajoutera au vecteur de poids courant, un vecteur colinéaire à $\boldsymbol{E}$, et dans notre cas, de même direction, modifiant ainsi l'orientation de la droite. Celle-ci se rapprochera de l'extrémité du vecteur $\boldsymbol{E}$. Il est facile de voir que si cette opération est répétée, le vecteur $\boldsymbol{E}$ finira par se trouver du côté positif.

1.2.2 minimiser une fonction de coût

Avant d'aborder les différentes règles d'apprentissage de manière plus formelle, il convient d'exposer le cadre général dans lequel elles seront exprimées. Le problème du calcul des poids d'un ELS peut être très simplement formulé comme un problème

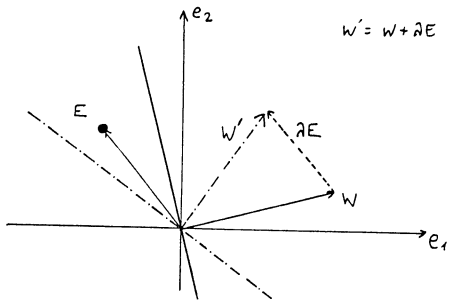


Figure 1. 9: rotation de la droite séparatrice avec la règle du perceptron. Un vecteur E malencontreusement classé dans $\mathcal{E}^-$. En traits pleins: l'ancien vecteur de poids et l'ancienne droite séparatrice correspondante. En pointillé: le nouveau vecteur de poids et la droite associée.

d'optimisation. Il s'agit de trouver un vecteur W qui minimise une certaine FONCTION DE COÛT. Une règle d'apprentissage sera définie, d'une part, par la forme de la fonction de coût et, d'autre part, par la technique de minimisation employée. Un exemple simple de fonction de coût serait le nombre d'exemples mal classés:

$$C(W) = \frac{1}{2} \sum_{k=1}^{p} | d_k - s(W, E_k) |$$

malheureusement, cette fonction ne possède pas les "bonnes" propriétés que l'on est en droit d'attendre d'une fonction de coût, en particulier, elle n'est pas continue, et à plus forte raison, pas dérivable: c'est une fonction constante par intervalles. Ceci interdit l'utilisation de méthodes classiques d'optimisation telles que l'algorithme du gradient. Nous lui préférerons donc d'autres fonctions moins immédiates, mais plus utilisables. Parmi celles-ci, le critère dit du perceptron.

$$C_p(W) = -\frac{1}{2} \sum_{E \in M(W)} d(E)\, W^T E$$

où $M(W)$ est l'ensemble des formes d'apprentissage mal classées par le vecteur de poids W et $d(E)$ est la sortie désirée associée au vecteur E. Ceci peut être exprimé

d'une autre manière

$$C_p(W) = \frac{1}{2}\sum_{k=1}^{p}(\, s(E_k) - d_k \,)\, W^T E_k$$

où s n'est plus considéré comme explicitement dépendant de W. Ce critère est nul lorsqu'aucune erreur de classification ne se produit sur l'ensemble d'apprentissage, il est strictement positif dans le cas contraire. Il est continu et dérivable presque partout. Un autre critère, largement utilisé est le critère des moindres carrés:

$$C_q(W) = \sum_{k=1}^{p}(d_k - W^T E_k)^2$$

ce critère peut sembler étrange dans la mesure où sa minimisation n'entraîne pas nécessairement la minimisation du nombre d'erreurs de classification. Néanmoins, ses qualités contrebalancent largement ce petit défaut, comme cela sera montré au paragraphe 1.4.

L'algorithme du gradient sera beaucoup utilisé dans la suite, quelques notations s'y rapportant vont être définies ici. Soit $g(W)$ une fonction différentiable, dont il s'agit de trouver un minimum. L'algorithme du gradient consiste, à partir d'un vecteur de paramètres initial W_0, à construire une séquence $W_0, W_1, W_2, \ldots$ au moyen de la formule de récurrence suivante:

$$W_{h+1} = W_h - \lambda_h \nabla g(W_h) \tag{1.11}$$

ici $\nabla g(W_h)$ représente le gradient de g au point W_h, et λ_h une "petite" constante strictement positive. Le vecteur W se modifie donc par petites étapes en suivant la direction opposée au gradient, c'est à dire la ligne de plus grande pente. Si les λ_h sont suffisamment petits, il est facile de montrer que $g(W_{h+1}) < g(W_h)$. En effet, l'équation 1.11 nous donne:

$$g(W_{h+1}) = g(W_h - \lambda_h \nabla g(W_h))$$

en développant g en série de Taylor au point W_h, nous avons:

$$g(W_{h+1}) = g(W_h) - \lambda_h (\nabla g(W_h))^T \nabla g(W_h) + O(\lambda_h^2)$$

où $O(\lambda^2)$ est une fonction qui tend vers 0 comme λ^2. On voit clairement qu'à la condition que $\nabla g(W_h) \neq 0$, il existe une valeur maximale de λ_h en-dessous de laquelle le terme $-\lambda_h (\nabla g(W_h))^T \nabla g(W_h) + O(\lambda_h^2)$ est négatif. Il s'en suit que $g(W_{h+1}) < g(W_h)$. La figure 1. 10 explique graphiquement le processus. Les conditions nécessaires à la convergence ont été longuement étudiées dans la littérature sur les problèmes d'optimisation. Précisons simplement qu'il est souvent nécessaire de faire l'hypothèse que la fonction g est convexe, et que par conséquent, elle possède un minimum unique.

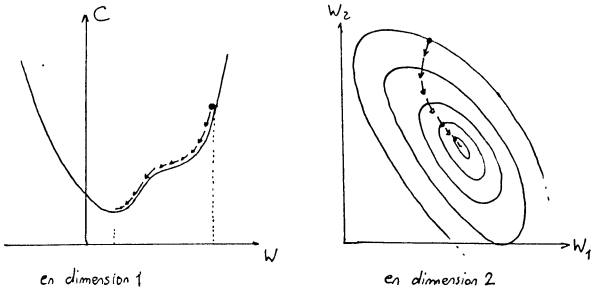


Figure 1. 10: l'algorithme du gradient. à chaque itération, le vecteur de paramètres W (ici à 1 et 2 dimensions) est modifié d'une quantité proportionnelle au gradient de la fonction de coût, et de direction opposée.

Il existe de nombreuses variantes à cet algorithme. Celle qui vient d'être décrite est connue sous le nom d'algorithme de la plus grande pente, sa principale qualité est sa simplicité. La vitesse et la précision de la convergence reposent entièrement sur le bon choix des λ_n. Si ceux-ci sont trop petits, le temps de convergence sera inutilement long, s'ils sont trop grands on s'expose au risque de la non-convergence. Les variantes reposent souvent sur le choix de λ, qui peut être remplacé par une matrice Λ. Toutefois, ceci nécessite souvent des calculs importants comme l'évaluation directe ou itérative de l'inverse du Hessien (méthodes du second ordre, moindres carrés récursifs). Les propriétés de convergence dans le cas d'un critère quadratique seront étudiées au paragraphe 1.4.1.

1.3 la règle du perceptron et ses variantes

La règle d'apprentissage du perceptron a été une des premières à apparaître [Rosenblatt 57]. Toutefois, l'idée des poids modifiables et son application à l'apprentissage est antérieure, et résulte certainement de la combinaison des travaux de McCulloch et Pitts, et de Hebb [Hebb 49]. Hebb propose la modification des connexions synaptiques comme support de l'apprentissage et de la mémorisation dans les systèmes nerveux animaux (voir aussi [Kandel]). Ces idées seront reprises par Clark et Farley

[Farley & Clark 54] qui effectueront les premières simulations de réseaux de neurones sur ordinateur. Les arguments heuristiques avancés par Rosenblatt pour présenter la procédure du perceptron sont proches de ceux présentés au paragraphe 1.2.1. Ces notions intuitives ont fait l'objet de nombreuses formalisations et généralisations, et l'on dispose de plusieurs démonstrations de convergence.

Le paragraphe précédent a introduit la fonction de coût $C_p(W)$ minimisée par la règle du perceptron:

$$C_p(W) = \frac{1}{2}\sum_{k=1}^{p}(s(E_k) - d_k)W^T E_k \tag{1.12}$$

le gradient de $C_p(W)$ est donné par:

$$\nabla C_p(W) = -\frac{1}{2}\sum_{k=1}^{p}(d_k - s(E_k))E_k \tag{1.13}$$

en considérant que s ne dépend pas explicitement de W. L'application de la procédure du gradient 1.11 donne immédiatement:

$$W_{h+1} = W_h + \lambda_h \frac{1}{2}\sum_{k=1}^{p}(d_k - s(E_k))E_k \tag{1.14}$$

cette procédure n'est pas à proprement parler la procédure du perceptron car elle fait intervenir TOUS les exemples mal classés pour UNE itération de la règle d'apprentissage.

Les propriété de convergence vont être étudiées sur une version différente dans laquelle une modification des poids est effectuée après CHAQUE présentation d'un vecteur d'entrée, comme cela est décrit par l'équation 1.10. Ceci peut être interprété comme une version stochastique de l'algorithme de minimisation 1.14 si l'on considère E comme un vecteur aléatoire.

1.3.1 Théorème de convergence de la règle du perceptron

De nombreuses démonstrations de convergence existent dont [Rosenblatt 60], [Joseph 60], [Block 62], et de nombreux autres (voir [Nilsson 65] pp 93 pour une bibliographie complète). Celle présentée ici est une variante de celle de Ridgway [Ridgway 62] reprise dans [Duda & Hart 73]. La règle du perceptron itérative est la suivante:

$$W_{h+1} = W_h + \frac{\lambda}{2}(d_h - s_h)E_h \tag{1.15}$$

ici λ est constant, c'est pourquoi cette règle est également appelée RÈGLE A PAS FIXE. De manière claire, l'hyperplan défini par W, ne dépend que de la direction de celui-ci et est indépendant de sa norme. Ceci amène deux remarques. La première

est que l'ensemble des solutions est un cône de section polygônale [6]. La deuxième est que λ n'intervient ici que comme un facteur d'échelle, et qu'il est donc possible, sans restreindre la généralité, et sans changer la nature de la solution, de choisir $\lambda = 1$. La règle obtenue est alors:

$$W_{h+1} = W_h + \frac{(d_h - s_h)}{2} E_h \tag{1.16}$$

en introduisant la notation $\epsilon_h = \frac{(d_h - s_h)}{2}$ pour désigner l'erreur, l'équation précédente se réécrit:

$$W_{h+1} = W_h + \epsilon_h E_h \tag{1.17}$$

Soit $\mathcal{E}^+$ et $\mathcal{E}^-$ deux ensembles de vecteurs linéairement séparables dont l'union $\mathcal{E} = \mathcal{E}^- \cup \mathcal{E}^+$ est l'ensemble d'apprentissage. On construit une séquence infinie d'éléments de $\mathcal{E}$ dans laquelle chaque élément de $\mathcal{E}$ apparaît une infinité de fois. Notons $S = E_0, E_1, \ldots, E_h, \ldots$ **cette séquence. Au temps** h**, le vecteur** E_h **est présenté, et le vecteur** W_{h+1} est calculé à l'aide de l'équation 1.17. Au cours de la session d'apprentissage, certains des vecteurs E_h vont donner lieu à une modification des poids et d'autres pas. Nous allons utiliser la sous-séquence $T = E^0, E^1, \ldots$ constituée uniquement des termes de la séquence S pour lesquels une correction a été apportée au vecteur de poids, c'est à dire E_h est présent dans T (sous forme d'un E^j) si $\epsilon_h \neq 0$.

Il s'agit de montrer l'existence d'une quantité positive, décroissante, et dont le pas de décroissance est borné inférieurement par une constante. Il en sera déduit que le **processus converge en un nombre fini d'étapes. Soit** $\tilde{W}$ **un vecteur solution, c'est à** dire pour lequel:

$$\forall E \in \mathcal{E}^+ \quad \tilde{W}^T E > 0$$
$$\forall E \in \mathcal{E}^- \quad \tilde{W}^T E < 0$$

ceci peut être exprimé autrement en utilisant $d(E)$, la sortie désirée associée à E:

$$\forall E \in \mathcal{E} \quad \tilde{W}^T E d(E) > 0 \tag{1.18}$$

avec:

$$d(E) = +1 \leftrightarrow E \in \mathcal{E}^+ \quad d(E) = -1 \leftrightarrow E \in \mathcal{E}^-$$

$\tilde{W}$ est solution du système d'inégalités 1.18. La quantité que nous voudrions décroissante serait la suivante

$$\| W - \tilde{W} \|$$

malheureusement, elle ne l'est pas en général, car, comme cela a été indiqué plus haut, si $\tilde{W}$ est solution du système 1.18 alors $\alpha\tilde{W}$ avec $\alpha > 0$ l'est aussi, et nous ne

[6] c'est un ensemble invariant par homothétie de coefficient positif, défini par l'intersection d'un ensemble fini de demi-espaces dont les frontières passent par l'origine

savons pas a-priori vers quelle solution particulière le processus converge. La quantité décroissante sera donc plus judicieusement:

$$|| W - \alpha\tilde{W} ||$$

L'équation 1.17 nous donne alors:

$$W_{h+1} - \alpha\tilde{W} = W_h - \alpha\tilde{W} + \epsilon^h E^h$$

E^h étant un membre de la sous-séquence T, et ϵ^h le terme d'erreur associé (non nul par construction) valant +1 ou -1. nous en tirons

$$||W_{h+1} - \alpha\tilde{W}||^2 = ||W_h - \alpha\tilde{W}||^2 + 2(W_h - \alpha\tilde{W})^T E^h\epsilon^h + ||E^h||^2(\epsilon^h)^2$$

le terme $W_h^T E^h\epsilon^h$ est nécessairement négatif puisque E^h appartient à la séquence T, et que, par conséquent, la réponse qui lui est associée est mauvaise. On déduit

$$||W_{h+1} - \alpha\tilde{W}||^2 \leq ||W_h - \alpha\tilde{W}||^2 - 2\alpha\tilde{W}^T E^h\epsilon^h + ||E^h||^2$$

le terme $2\alpha\tilde{W}^T E^h\epsilon^h$ est nécessairement positif, puisque $\tilde{W}$ est solution de 1.18. Il est clair que ce terme dominera le terme $||E^h||^2$ si α est choisi suffisamment grand. En particulier, si on pose:

$$e_{max} = \max_i ||E^i|| \qquad a_{min} = \min_i \tilde{W}^T E^i\epsilon^i > 0$$

l'équation précédente se réécrit

$$||W_{h+1} - \alpha\tilde{W}||^2 \leq ||W_h - \alpha\tilde{W}||^2 - 2\alpha a_{min} + e_{max}^2$$

en choisissant $\alpha = \frac{e_{max}^2}{a_{min}}$ nous obtenons

$$||W_{h+1} - \alpha\tilde{W}||^2 \leq ||W_h - \alpha\tilde{W}||^2 - e_{max}^2$$

donc la quantité $||W_{h+1} - \alpha\tilde{W}||^2$ décroit d'au moins e_{max}^2 à chaque modification. Au bout de q itérations:

$$||W_q - \alpha\tilde{W}||^2 \leq ||W_0 - \alpha\tilde{W}||^2 - qe_{max}^2$$

ce qui démontre le théorème de convergence, puisque la terme de gauche ne peut devenir négatif, il s'en suit que le nombre d'itérations q est nécessairement fini. Le nombre de modifications sera au plus égal à

$$q_{max} = \frac{||W_0 - \alpha\tilde{W}||^2}{e_{max}^2}$$

Etant donné qu'une correction n'intervient que si un vecteur est mal classé, et que d'autre part chacun des vecteurs apparaît un nombre infini de fois dans la séquence, nous concluons que lorsqu'il n'y a plus de correction tous les vecteurs de $\mathcal{E}^- \cup \mathcal{E}^+$ sont nécessairement bien classés. On doit cependant remarquer que l'expression du nombre maximum de corrections ne donne pas de résultat pratique sur la vitesse de convergence. La raison en est que α est inconnu ainsi que $\tilde{W}$. Par ailleurs, le nombre de corrections, n'est généralement pas égal au nombre de présentations (ou d'itérations).

Nous pouvons donc énoncer le théorème suivant:

Theoreme 1.3 de convergence du perceptron.

- Soient $\mathcal{E}^-$ et $\mathcal{E}^+$ deux ensembles de vecteurs linéairement séparables.
- Soit S une séquence de couples du type (E_h, d_h) où:
  - $E_h \in \mathcal{E}^- \cup \mathcal{E}^+$
  - chaque élément de $\mathcal{E}^- \cup \mathcal{E}^+$ apparait un nombre infini de fois
  - d_h est défini par: $d_h = +1 \leftrightarrow E_h \in \mathcal{E}^+$ et $d_h = -1 \leftrightarrow E_h \in \mathcal{E}^-$
- Soit la suite $W_0, W_1, \ldots$ construite à l'aide de la formule de récurrence suivante:

$$W_{h+1} = W_h + \frac{\lambda}{2}(d_h - s_h)E_h$$

où λ est une constante positive et s_h est donné par :

$$s_h = \begin{cases} +1 & \text{si} \quad W_h^T E_h > 0 \\ -1 & \text{sinon} \end{cases}$$

Alors il existe un entier q tel que pour tout $r > q$

- $W_r = W_q$
- $\forall E \in \mathcal{E}^- \cup \mathcal{E}^+$
  - $E \in \mathcal{E}^- \leftrightarrow W_r^T E < 0$
  - $E \in \mathcal{E}^+ \leftrightarrow W_r^T E > 0$

1.3.2 les variantes de la règle du perceptron

De nombreuses variantes de la règle du perceptron existent [Duda & Hart 73]. Un inconvénient de la règle du perceptron est qu'elle s'arrête dès qu'une solution est atteinte. Mais certaines solutions sont "meilleures" que d'autres vis-à-vis de la classification de formes supplémentaires. En particulier, il est ennuyeux d'aboutir à un hyperplan situé à une trop faible distance d'un des échantillons, on préfèrerait à cela un hyperplan en "position médiane" par rapport aux deux classes. Ce problème conduit à modifier légèrement le critère en introduisant une marge m en dessous de laquelle la valeur absolue de $W^T E$ ne doit pas descendre. Une modification des poids intervient alors à chaque fois que $d(E)W^T E < m$. Il se peut dans ce cas qu'une modification des poids intervienne même si la forme était correctement classée. [Duda & Hart 73] donne les conditions de convergence de cet algorithme. Certaines autres variantes sont des procédures itératives de résolution de systèmes d'inéquations linéaires transformées pour la circonstance. Hormis les méthodes de programmation dynamique modifiées on peut citer les algorithmes de relaxation, plus anciens que le perceptron [Agmon 54] qui peuvent s'exprimer comme une procédure de gradient appliquée à une fonction de la forme:

$$C_r(W) = \frac{1}{2} \sum_{E \in M(W)} \frac{(W^T E)^2}{E^T E}$$

où $M(W)$ est l'ensemble des formes E de $\mathcal{E}$ mal classées par le vecteur de poids W. Le gradient de ce critère présente l'inconvénient de tendre (trop) rapidement vers 0 à mesure que l'on s'approche de la zone des solutions, ralentissant de ce fait la convergence. Ce problème peut être résolu par l'introduction d'une marge m comme précédemment [Duda & Hart 73]. La procédure de minimisation associée est

$$W_{h+1} = W_h - \lambda_h \sum_{E \in M(W_h)} \frac{W_h^T E}{E^T E} E$$

comme la règle du perceptron, cette procédure peut être transformée pour effectuer une modification après chaque échantillon. Si au temps h l'échantillon E_h est mal classé par le vecteur de poids W_h on applique la formule suivante

$$W_{h+1} = W_h - \lambda_h \frac{W_h^T E_h}{(E_h)^T E_h} E_h$$

dans le cas contraire (si E_h est bien classé) on laisse $W_{h+1} = W_h$.

Les algorithmes présentés dans ce paragraphe convergent vers une solution A LA CONDITION QUE CELLE-CI EXISTE, mais ce n'est pas toujours le cas, et le comportement dans ces conditions n'est pas garanti. Différentes modifications de

l'algorithme de base ont été proposées pour remédier à ce problème. Citons le "pocket algorithm" [7] dont l'idée de base consiste à garder "dans sa poche" la meilleure solution trouvée, tout en continuant à effectuer des modifications. Citons également la méthode basée sur les ensembles flous [Keller & Hunt 85], dans laquelle l'incrément du vecteur de poids est multiplié par un coefficient d'appartenance à sa classe. Ce coefficient est fonction de la différence des distances entre la forme considérée et les barycentres des deux classes. Si une forme est située à mi-distance entre les deux barycentres, elle ne provoque pas de modification des poids.

1.4 les algorithmes de moindres carrés

A l'instar du perceptron, les algorithmes de moindres carrés reposent sur la minimisation d'un critère. Ce critère est quadratique en fonction de W et possède la forme suivante:

$$C_q(W) = \frac{1}{p}\sum_{k=1}^{p}(d_k - W^T E_k)^2 \tag{1.19}$$

ou encore

$$C_q(W) = \frac{1}{p}\sum_{k=1}^{p}\epsilon_k^2 \tag{1.20}$$

avec les notations habituelles. Lorsqu'il est utilisé pour calculer les poids d'un élément à seuil, ce critère est SOUS-OPTIMAL au sens où sa minimisation n'entraine pas nécessairement la minimisation du taux d'erreurs de classification. La contrepartie est que sa nature quadratique facilite son étude théorique et surtout que le comportement de l'algorithme associé ne dépend pas (trop) de la séparabilité ou de la non-séparabilité de l'ensemble d'apprentissage.

Minimiser le critère $C_q(W)$ avec un algorithme de gradient est extrêmement simple. Le gradient de $C_q(W)$ est donné par

$$\nabla C_q(W) = -\frac{2}{p}\sum_{k=1}^{p}\epsilon_k E_k$$

l'équation du gradient 1.11 donne l'algorithme dit de la PLUS GRANDE PENTE:

$$W_{h+1} = W_h + \lambda_h \frac{2}{p}\sum_{k=1}^{p}\epsilon_{kh} E_k \tag{1.21}$$

avec la notation

$$\epsilon_{kh} = d_k - W_h^T E_k$$

l'équation 1.21 a la même forme que celle de l'algorithme du perceptron, la différence réside dans le calcul de ϵ_{kh} qui dans un cas est binaire, et dans l'autre proportionnel à l'erreur. Comme toutes les procédures minimisant un critère global défini sur la totalité de l'ensemble d'apprentissage, celle-ci nécessite de passer en revue TOUS les exemples avant de pouvoir effectuer une modification de W. Elle peut être transformée, comme dans le cas du perceptron, afin de ne prendre en compte que le vecteur d'entrée courant et effectuer une itération d'apprentissage après chaque présentation. La procédure obtenue est la suivante:

$$W_{h+1} = W_h + 2\lambda_h \epsilon_h E_h \tag{1.22}$$

c'est la très célèbre règle de WIDROW ET HOFF [Widrow & Hoff 60]. Sous réserve de certaines hypothèses, on peut montrer que son comportement moyen approche celui de la procédure 1.21. On peut en effet remarquer que l'espérance mathématique du terme $-2\epsilon_h E_h$ est un estimateur sans biais du gradient du critère quadratique 1.19 à condition que les probabilités d'apparition des formes dans la séquence de présentation soient égales. Cette procédure est dite de "gradient stochastique" et s'avère être une forme de l'algorithme d'approximation stochastique de Robbins et Monro (1951) lorsque la suite λ_h est choisie décroissante, tendant vers 0. Néanmoins aucune démonstration de convergence n'existe dans le cas général.

On peut remarquer que, contrairement à l'algorithme du perceptron et ses dérivés, les méthodes basées sur les moindres carrés effectuent une modification des poids même si la réponse fournie EST BONNE. En effet le seul cas où aucune modification n'est effectué est lorsque

$$\epsilon = d - W^T E = 0$$

c'est à dire lorsque l'entrée totale est EXACTEMENT égale à la sortie désirée (-1 ou +1). En général, il n'existera pas de solution satisfaisant toutes ces contraintes, et après la période transitoire de convergence, les poids oscilleront autour de l'optimum.

Il est possible d'interpréter la minimisation du critère quadratique comme un moyen de trouver une solution approchée au système linéaire:

$$d_k = W^T E_k \qquad k = 1 \ldots p$$

il suffit que les formes E_k ne soient pas linéairement indépendantes pour que ce système n'ait pas de solution exacte. Cette situation se produit (entre autres) lorsque le nombre de formes p est supérieur au nombre d'entrées n, ou, plus généralement, lorsque les vecteurs d'entrée ne sont pas linéairement indépendants. La théorie relative à ce problème sera exposée au chapitre 2 dans le paragraphe sur l'apprentissage linéaire.

Lorsque l'on utilise des cellules non-binaires (avec la fonction sigmoïde 1.6 par exemple), il peut être intéressant de minimiser un critère "quadratique" de la forme

$$C_q(W) = \frac{1}{p}\sum_{k=1}^{p}(d_k - f(W^T E_k))^2 \tag{1.23}$$

la quantité minimisée est alors exactement l'erreur en sortie, contrairement au cas binaire, ou la non dérivabilité de la transformation non-linéaire réduit à utiliser des critères de substitution sous-optimaux. Il est important de remarquer que le critère 1.23 N'EST PAS QUADRATIQUE vis à vis de W, ce qui complique singulièrement l'analyse de sa convergence. L'algorithme de plus grande pente associé est

$$W_{h+1} = W_h + \lambda_h \frac{2f'(a_{kh})}{p}\sum_{k=1}^{p}\epsilon_{kh}E_k \tag{1.24}$$

f' est la dérivée de f, et on utilise les notations:

$$a_{kh} = W_h^T E_k$$

et

$$\epsilon_{kh} = d_k - f(a_{kh})$$

Les deux versions (déterministe et stochastique) de l'algorithme et leurs propriétés de convergence vont être passées en revue dans les paragraphes suivants. Nous nous intéresserons uniquement au cas linéaire dans un premier temps. De nombreux résultats en seront tirés pour l'étude des algorithmes de rétro-propagation décrits au chapitre 3.

1.4.1 Minimiser le critère des moindres carrés

Ce paragraphe traite des méthodes de minisation du critère des moindres carrés, et leur application aux systèmes adaptatifs. Les techniques décrites ici sont largement utilisées en télécommunication pour l'annulation d'écho ou le filtrage adaptatif, et en commande optimale pour l'identification de systèmes linéaires [Widrow & Stearn 85] [Taylor 83] [Macchi 84] [Honig & Messerschmitt 84]. Elles sont issues des travaux de la "cybernétique" et de la théorie des systèmes et plus particulièrement des travaux de Norbert Wiener. Les techniques relatives aux filtres adaptatifs seront particulièrement utiles. Nous nous réfèrons principalement à [Widrow & Stearn 85].

Plutôt que de considérer l'apprentissage statiquement en faisant intervenir directement les p exemples dans la fonction de coût, nous préférerons traiter le problème en considérant son caractère dynamique. Il sera donc fait l'hypothèse qu'à tout instant

h il est possible d'estimer la valeur de la fonction de coût sans nécessairement tenir compte de tous les échantillons. En utilisant la notation habituelle

$$\epsilon_h = d_h - W^T E_h$$

la fonction de coût utilisée sera la suivante

$$C(W) = \mathcal{M}[\epsilon_h^2] \tag{1.25}$$

où l'opérateur $\mathcal{M}[]$ est un opérateur linéaire de "moyennage". Dans le cas déterministe $\mathcal{M}[]$ correspondra simplement à une sommation normalisée sur tous les échantillons de la liste d'exemples. Ceci nous donne:

$$C(W) = \frac{1}{p}\sum_{k=1}^{p}(d_k - W^T E_k)^2 \tag{1.26}$$

Dans le cas stochastique, l'opérateur $\mathcal{M}[]$ sera l'espérance mathématique [7]

$$C(W) = \mathcal{E}[(d_h - W^T E_h)^2] \tag{1.27}$$

il s'agira dans ce dernier cas de trouver un bon estimateur du gradient. l'expression 1.25 peut être développée:

$$C(W) = \mathcal{M}[d_h^2 - 2W^T E_h d_h + W^T E_h E_h^T W] \tag{1.28}$$

soit en regroupant

$$C(W) = \mathcal{M}[d_h^2] - 2W^T \mathcal{M}[E_h d_h] + W^T \mathcal{M}[E_h E_h^T]W \tag{1.29}$$

nous utiliserons les notations suivantes:

$$P = \mathcal{M}[E_h d_h] \qquad R = \mathcal{M}[E_h E_h^T] \tag{1.30}$$

R est la matrice de covariance des vecteurs d'entrée, en supposant la stationnarité de E et d, R et P ne dépendent pas du temps. Ces notations nous permettent de réécrire 1.29 sous la forme:

$$C(W) = \mathcal{M}[d_h^2] - 2W^T P + W^T R W \tag{1.31}$$

R étant une matrice de covariance, elle est symétrique, et il sera fait l'hypothèse qu'elle est définie positive. On voit que $C(W)$ est une forme quadratique en W. L'équation 1.31 définit une surface qui, en l'occurence, est une PARABOLOÏDE. Les propriétés

[7] Il faudrait, pour être tout à fait correct, poser les hypotheses d'ergodicité adequates.

en seront explorées plus loin, une chose est néanmoins certaine: elle est convexe et par conséquent ne possède qu'un minimum. La valeur minimum de $C(W)$ est atteinte lorsque son gradient par rapport à W est nul, ce qui nous permet de donner une expression de la solution. Le gradient de $C(W)$ est

$$\nabla C(W) = 2RW - 2P \tag{1.32}$$

la solution minimisant $C(W)$ est notée $\tilde{W}$ et vérifie la condition $\nabla C(\tilde{W}) = 0$, soit

$$R\tilde{W} = P$$

si l'on suppose que R est non singulière on obtient:

$$\tilde{W} = R^{-1}P \tag{1.33}$$

cette équation est l'équation de WIENER ET HOPF. La généralisation de cette équation au cas multidimensionnel et au cas où R n'est pas inversible sera abordée dans le cas déterministe au chapitre 2 avec la théorie des pseudo-inverses et son application aux mémoires associatives linéaires. L'équation 1.32 dérivée une fois de plus par rapport à W donne le Hessien H de $C(W)$ c'est à dire la matrice des dérivées secondes

$$H = \nabla\nabla^T C(W) = 2R \tag{1.34}$$

le Hessien détermine l'allure de la paraboloïde de coût. Ses valeurs propres fixent les dimensions des axes principaux et conditionnent la vitesse de convergence comme cela sera mis en évidence dans les paragraphes suivants.

analyse de la fonction de coût

Il convient de transformer l'expression de la fonction de coût 1.31 pour la rendre plus facilement manipulable. Ceci sera effectué par un changement de variables exprimant les vecteurs dans un repère dont l'origine est le minimum de la fonction de coût, et dont les axes sont les directions principales de la paraboloïde. Commençons par la translation en effectuant le changement de variables

$$V = W - \tilde{W}$$

le critère 1.31 s'écrit:

$$C(V) = \mathcal{M}[d_h^2] - 2P^T(V + \tilde{W}) + (V + \tilde{W})^T R(V + \tilde{W})$$

en développant

$$C(V) = \mathcal{M}[d_h^2] - 2P^T V - 2P^T\tilde{W} + \tilde{W}^T R\tilde{W} + V^T RV + \tilde{W}^T RV + V^T R\tilde{W}$$

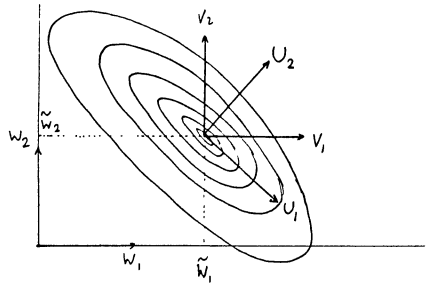


Figure 1. 11: changement de repère simplifiant l'expression du critère quadratique

en utilisant le fait que R est symétrique et en remplaçant $\tilde{W}$ par sa valeur donnée dans 1.33

$$C(V) = \mathcal{M}[d_h^2] - 2P^T V - 2P^T R^{-1} P + (R^{-1}P)^T R R^{-1} P + V^T R V + 2V^T R R^{-1} P$$

en regroupant et simplifiant

$$C(V) = \mathcal{M}[d_h^2] - P^T R^{-1} P + V^T R V$$

ou encore

$$C(V) = \mathcal{M}[d_h^2] - P^T \tilde{W} + V^T R V \tag{1.35}$$

puisque pour $V = 0$, $C(V)$ est minimum, nous noterons

$$C_{min} = \mathcal{M}[d_h^2] - P^T \tilde{W}$$

transformant ainsi 1.35

$$C(V) = C_{min} + V^T R V \tag{1.36}$$

on peut remarquer que le gradient de C par rapport à V est égal au gradient de C par rapport à W puisque ces deux vecteurs ne diffèrent que d'une constant $\tilde{W}$. On peut donc écrire

$$\nabla C(V) = \nabla C(W) = 2RV \tag{1.37}$$

Le deuxième changement de variables à opérer est une rotation, de matrice Q, alignant les axes de coordonnées sur les axes principaux de la paraboloïde (voir fig 1.

11)

$$U = QV \tag{1.38}$$

Il s'agit évidemment de choisir judicieusement Q. l'équation 1.36 se transforme en

$$C(U) = C_{min} + (Q^{-1}U)^T RQ^{-1}U$$

Q est une transformation unitaire, elle vérifie $Q^T = Q^{-1}$ ce qui permet d'écrire:

$$C(U) = C_{min} + U^T QRQ^{-1}U$$

nous choisirons Q de manière à ce que QRQ^{-1} soit diagonale. Ce choix est toujours possible compte tenu du fait que R est symétrique et par conséquent diagonalisable. L'équation précédente devient

$$C(U) = C_{min} + U^T LU \tag{1.39}$$

où L est une matrice diagonale composée des valeur propres de R qui, par construction sont toutes positives, et par hypothèse le sont strictement. Notons $l_0, \ldots, l_n$ les valeurs propres de R. Il est possible d'exprimer le gradient de C dans le nouveau système de coordonnées:

$$\nabla C(U) = 2LU$$

d'où l'on tire

$$\frac{\partial C}{\partial u_i} = 2l_i u_i \quad i = 0 \ldots n$$

et également

$$\frac{\partial^2 C}{\partial u_i \partial u_j} = \begin{cases} 2l_i & \text{si } i = j \\ 0 & \text{sinon} \end{cases}$$

les dérivées secondes ont une expression très simple dans le nouveau système de coordonnées, ceci sera très utile pour l'étude de la convergence des procédures d'apprentissage abordée dans le paragraphe suivant.

algorithme de la plus grande pente: conditions de convergence

L'utilisation d'une procédure de gradient pour minimiser $C(W)$ nécessite l'estimation du gradient à chaque itération.

$$W_{h+1} = W_h + \lambda_h \nabla C(W_h)$$

ce que l'on peut développer en

$$W_{h+1} = W_h + 2\lambda_h M[d_h E_h - E_h E_h^T W_h]$$

nous préfèrerons utiliser l'expression du gradient donnée par l'équation 1.37 soit

$$W_{h+1} = W_h - 2\lambda_h R V_h$$

en notant $V_h = W_h - \tilde{W}$. Si l'on retranche $\tilde{W}$ de chaque côté on obtient

$$W_{h+1} - \tilde{W} = W_h - \tilde{W} - 2\lambda_h R V_h$$

soit

$$V_{h+1} = V_h - 2\lambda_h R V_h$$

que l'on peut réécrire

$$V_{h+1} = (I - 2\lambda_h R) V_h$$

où I est la matrice identité. Nous allons supposer maintenant que λ est constant, ce qui nous permet de réécrire l'équation précédente sous la forme

$$V_{h+1} = T V_h \tag{1.40}$$

avec

$$T = I - 2\lambda R$$

1.40 est l'équation de récurrence d'une suite géométrique que nous voudrions convergente vers 0. Le terme général est

$$V_h = T^h V_0$$

on constate que cette suite converge vers 0 si T est un opérateur de contraction, c'est à dire s'il est tel que

$$||TV|| < ||V||$$

La situation devient plus claire lorsque l'équation 1.40 est exprimée dans le repère attaché aux axes principaux de la paraboloïde. L'équation 1.40 pré-multipliée par Q donne

$$U_{h+1} = Q T V_h \tag{1.41}$$

avec U défini par l'équation 1.38. Après substitution et manipulations algébriques

$$U_{h+1} = (QIQ^T - 2\lambda QRQ^T) U_h \tag{1.42}$$

soit

$$U_{h+1} = (I - 2\lambda L)U_h \tag{1.43}$$

la matrice L étant diagonale, ce système présente l'énorme avantage de ne pas être couplé ce qui rend son analyse très simple. L'équation de récurrence se réduit donc à $n+1$ récurrences indépendantes

$$u_{i(h+1)} = (1 - 2\lambda l_i)u_{ih} \quad i = 0 \ldots n \tag{1.44}$$

de manière évidente, tous les u_i convergeront vers 0 si

$$|1 - 2\lambda l_i| < 1 \quad i = 0 \ldots n \tag{1.45}$$

Ces conditions sont les conditions de convergence.

si l'on note l_{max} la plus grande valeur propre de R, la condition 1.45 se transforme en

$$0 < \lambda < \frac{1}{l_{max}} \tag{1.46}$$

cette condition assure la convergence de U vers 0 EN MOYENNE, et par conséquent celle de W vers $\tilde{W}$. Remarquons que si $0 < \lambda < \frac{1}{2l_i}$ la décroissance de u_i est exponentielle, alors que si $1/2l_{max} < \lambda < 1/l_i$ c'est une oscillation amortie. La valeur critique $\lambda = 1/2l_i$ assure une convergence en une seule itération (en supposant que l'estimation du gradient est exacte) et correspond à l'algorithme de Newton-Raphson qui sera étudié plus en détail dans la suite. Ceci nécessite toutefois de disposer d'un λ différent pour chaque coordonnée.

algorithme de la plus grande pente: vitesse de convergence

On peut facilement déduire de ce qui précède que le temps de convergence est directement lié au spectre de R. A chaque valeur propre de R est associée une constante de temps définie par

$$\tau_i = \frac{1}{1 - |1 - 2\lambda l_i|} \tag{1.47}$$

lorsque $\lambda < \frac{1}{2l_i}$,c'est à dire en dessous de la valeur critique, l'équation ci-dessus se réduit à

$$\tau_i = \frac{1}{2\lambda l_i} \tag{1.48}$$

Il est possible de calculer une valeur optimale de λ qui minimise le temps de convergence global τ_g [Honig & Messerschmitt 84]. Le temps de convergence global τ_g est donné par la plus grande constante de temps du système, c'est à dire

$$\tau_g = \max_i \tau_i = \max_i \frac{1}{1 - |1 - 2\lambda l_i|}$$

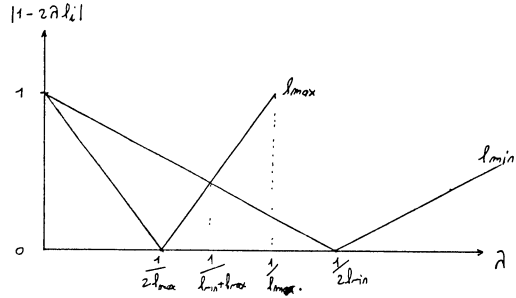


Figure 1. 12: paramètres de convergence associés à la plus grande et à la plus petite valeur propre de R

Il s'agira de trouver le λ qui minimise cette quantité. Le problème peut être simplifié en remarquant que la $i^{ème}$ constante de temps τ_i est une fonction croissante de la quantité

$$|1 - 2\lambda l_i|$$

qui est la raison de la suite géométrique des u_{ih}. Minimiser le temps de convergence est donc équivalent à rechercher la valeur de λ qui minimise

$$\max_i |1 - 2\lambda l_i|$$

La figure 1. 12 fait apparaitre de manière claire que la valeur optimale de λ (notée λ_{opt}) doit satisfaire la condition

$$|1 - 2\,\lambda_{opt}\, l_{max}| = |1 - 2\,\lambda_{opt}\, l_{min}|$$

nous savons que $\lambda_{opt} > 1/2l_{max}$ et que $\lambda_{opt} < 1/2l_{min}$ ce qui nous permet de transformer la condition précédente en

$$2\,\lambda_{opt}\, l_{max} - 1 = 1 - 2\,\lambda_{opt}\, l_{min}$$

d'où l'on tire

$$\lambda_{opt} = \frac{1}{l_{min} + l_{max}} \tag{1.49}$$

le choix de cette valeur a pour effet d'égaliser les constantes de temps associées à la plus grande et à la plus petite valeur propre de R. La première est soumise à un

régime de convergence oscillatoire, puisque λ est supérieur à la valeur critique associée, alors que la seconde subit une convergence exponentielle. En substituant λ_{opt} dans 1.48, on obtient la constante de temps optimale du processus de convergence τ_{opt}

$$\tau_{opt} = \frac{1}{2\frac{l_{min}}{l_{max}+l_{min}}}$$

nous pouvons exprimer cette quantité en fonction du rapport l_{max}/l_{min}

$$\tau_{opt} = \frac{\frac{l_{max}}{l_{min}} + 1}{2}$$

En pratique, cette équation donne une limite inférieure qu'il sera très difficile d'atteindre pour les raisons suivantes

- le gradient ne peut être qu'estimé et non calculé exactement, ce qui introduit un "bruit" susceptible de provoquer une divergence de l'algorithme.
- les valeurs propres sont rarement connues précisément.
- les valeurs réalistes de λ garantissant la convergence en présence de bruit sont souvent nettement inférieures à leur valeur optimale théorique

Ce résultat n'est donc qu'une indication d'un minimum inattaignable. Une hypothèse plus réaliste est de prendre

$$\lambda = \mu\frac{1}{2l_{max}}$$

avec $0 < \mu < 1$, qui par substitution dans 1.48 donne le temps de convergence global

$$\tau_g = \frac{l_{max}}{\mu l_{min}}$$

on voit clairement que le temps de convergence est d'autant plus long que les valeurs propres de R sont étalées. L'interprétation géométrique de ce résultat est très claire si l'on se souvient que le rapport entre les valeurs propres maximum et minimum de R détermine l'excentricité de la paraboloïde $C(W)$, plus exactement l'excentricité des ellipses de coût constant. La convergence sera plus rapide dans les directions des axes principaux les plus petits (correspondant aux valeurs propres les plus grandes), et ce sont eux qui fixeront la valeur de λ à ne pas dépasser. Selon les axes principaux les plus grands (les valeurs propres les plus petites), la convergence sera beaucoup plus lente (voir figure 1. 13).

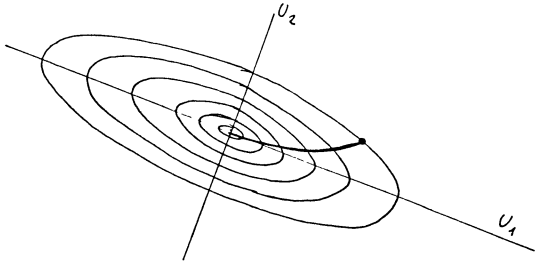


Figure 1. 13: convergence de l'algorithme de la plus grande pente

L'algorithme de Newton

L'algorithme de Newton est une extension au deuxième ordre de l'algorithme du gradient. Il consiste à approximer la fonction à minimiser par une surface d'ordre 2 (une forme quadratique), et à calculer directement la position du minimum. Le prix à payer est le calcul de l'inverse du Hessien de $C(W)$ à chaque itération. Il existe heureusement des méthodes itératives permettant d'éviter l'inversion du Hessien. Elles sont basées sur le théorème de Greville de décomposition des pseudo-inverses et sont utilisées dans les méthodes dites des moindres carrés récursifs. La méthode de Newton n'est utile que si l'une des deux conditions suivantes est remplie:

- le gradient ne peut être qu'estimé et pas calculé exactement
- le critère n'est pas quadratique

Dans le cas contraire (si les deux conditions sont simultanément fausses) la procédure de Newton n'est plus itérative (puisqu'elle converge en un coup) et se réduit au calcul de l'inverse du Hessien (de la matrice de covariance) et à l'application de l'équation de Wiener-Hopf 1.33.

L'algorithme a la forme suivante:

$$W_{h+1} = W_h - Z(W_h)$$

où $Z(W_h)$ est une fonction dont la nature est précisée ci-après. L'idée est d'approximer la fonction de coût autour de W_h par une forme quadratrique faisant intervenir le

gradient estimé et non le gradient réel, et de choisir le W_{h+1} au minimum de cette forme quadratique. L'approximation de la fonction de coût est de la forme

$$C(W) = C(W_h) + \tilde{\nabla}C(W_h)^T(W - W_h) + \frac{1}{2}(W - W_h)^T \tilde{H}(W - W_h) \qquad (1.50)$$

où $\tilde{\nabla}C(W_h)$ et $\tilde{H}$ sont respectivement les valeurs estimées du gradient et du Hessien au point W_h. La valeur de W qui minimise cette quantité est donné par

$$W - W_h = -\tilde{H}^{-1}\tilde{\nabla}C(W_h) \qquad (1.51)$$

d'où l'algorithme

$$W_{h+1} = W_h - \tilde{H}^{-1}\tilde{\nabla}C(W_h) \qquad (1.52)$$

Dans la pratique, on introduit un paramètre de convergence, généralement choisi entre 0 et 1:

$$W_{h+1} = W_h - \lambda_h\tilde{H}^{-1}\tilde{\nabla}C(W_h) \qquad (1.53)$$

avec les notations habituelles cette équation devient

$$W_{h+1} = W_h + 2\lambda_h\tilde{R}^{-1}\mathcal{M}[\epsilon_h E_h] \qquad (1.54)$$

l'algorithme de Newton réduit le nombre d'itérations en général, mais on se trouve confronté à la difficulté de calculer l'inverse de la matrice de covariance R^{-1}. Il faut de plus être dans la capacité d'estimer R. Si le critère est quadratique la trajectoire des poids se rapproche de la ligne droite au lieu de suivre les directions propres de la paraboloïde. L'algorithme de Newton pose de sérieux problèmes de convergence lorsque le critère n'est pas quadratique et/ou non convexe. La matrice Hessienne peut dans ce cas ne pas être inversible et/ou conduire à une divergence. La procédure de Newton peut être avantageusement utilisée lorsque R est diagonale: son inversion est alors triviale. Cette condition est remplie lorque les coordonnées des formes sont statistiquement indépendantes. Dans ce cas, la méthode de Newton est équivalente à la méthode du gradient dans laquelle on aura choisi un λ optimal différent pour chaque coordonnée. On peut provoquer artificiellement ce cas idéal en choisissant un prétraitement ad-hoc [8].

1.4.2 application au cas déterministe

La nature du problème qui nous intéresse interdit d'utiliser tous les raffinements des techniques d'optimisation. En effet, celles-ci reposent souvent sur l'hypothèse que la

[8] cette technique du "décorrélateur" est employée dans le traitement adaptatif de signal radar ou sonar [Taylor 83] et est à la base de la technique des filtres en treillis

valeur de la fonction à minimiser est simple à calculer en tout point de l'espace des W. Ce n'est pas le cas lorsqu'on s'intéresse à l'apprentissage puisque la valeur du critère en un point de l'espace des W dépend de toutes les formes de l'ensemble d'apprentissage. Dans le meilleur des cas, elles sont en faible nombre et connues par avance, dans le pire, elle sont générées au fur et à mesure de la session d'apprentissage et/ou sont en trop grand nombre pour qu'on puisse estimer le critère trop souvent et trop précisément. Les seules techniques utilisables sont donc celles qui sont robustes à une estimation grossière ou peu fréquente de la fonction de coût et de son gradient.

Ce paragraphe concernera le cas idéal où tout l'ensemble d'apprentissage est connu par avance. On définit le critère comme dans l'équation 1.26

$$C(W) = \frac{1}{p}\sum_{k=1}^{p}(d_k - W^T E_k)^2 \tag{1.55}$$

on se place délibérément dans le cas déterministe. L'algorithme de la plus grande pente donne

$$W_{h+1} = W_h + \lambda_h \frac{2}{p}\sum_{k=1}^{p}(d_k - W_h^T E_k)E_k \tag{1.56}$$

qui converge si λ_h est inférieur à $\frac{1}{l_{max}}$ où l_{max} est la plus grande valeur propre de $\frac{1}{p}\sum_{k=1}^{p} E_k^T E_k$

Cette formulation nous permet de définir des notations qui seront utilisées au chapitre 2. L'équation 1.55 peut en effet se réécrire en utilisant des notations matricielles. On définit le vecteur ligne $D = (d_1, \ldots, d_p)$ et la matrice $E = (E_1, \ldots, E_p)$ comportant $n+1$ lignes et p colonnes. 1.55 s'écrit:

$$C(W) = \frac{1}{p}(D - W^T E)(D - W^T E)^T \tag{1.57}$$

on peut considérer qu'il s'agit de trouver une solution approchée au système

$$W^T E = D \tag{1.58}$$

l'équivalent de l'équation de Wiener-Hopf peut être obtenu en post-multipliant par E^T de chaque côté

$$W^T E E^T = D E^T$$

et en multipliant par $(EE^T)^{-1}$ (en supposant que celle-ci existe)

$$W^T = D E^T (E E^T)^{-1}$$

le terme $E^T(EE^T)^{-1}$ est appelé la PSEUDO-INVERSE de E et est notée E^+. On vérifie aisément que $EE^+ = I$. L'équation précédente se réduit à

$$W^T = D E^+$$

On voit que si E est inversible, l'égalité $E^{+} = E^{-1}$ est vérifée. La pseudo-inverse peut être définie de manière plus générale, même dans le cas où (EE^T) n'est pas inversible par

$$E^{+} = \lim_{r \to 0} (E^T E + r^2 I)^{-1} E^T$$

Lorsque le système 1.58 est sous-déterminé, c'est à dire, lorsque E possède moins de colonnes que de lignes et que les colonnes sont linéairement indépendantes, il existe une infinité de solutions à 1.58. La pseudo-inverse donne la solution de norme minimum: celle qui minimise $W^T W$. Si le système est sur-déterminé, elle donne la solution minimisant le critère quadratique 1.57. De nombreux ouvrages de référence existent sur le sujet [Campbell & Meyer 79] [Rao & Mitra 71]. Ces notions seront utilisées au chapitre 2, pour les mémoires associatives linéaires [Kohonen 84a] [Fogelman & al. 87b], ainsi que pour l'algorithme de Ho-Kashyap [Duda & Hart 73].

1.4.3 cas stochastique: les moindres carrés moyens

La procédure de Widrow-Hoff (ou Least Mean Square algorithm), comme cela a été dit plus haut, est une version stochastique de l'algorithme de la plus grande pente où l'on estime le gradient de la fonction de coût en ne prenant en compte QU'UN SEUL échantillon. Une modification des poids est effectuée après chaque présentation d'un échantillon, et l'on espère que les effets des échantillons successifs se moyenneront pour donner approximativement le même comportement que l'algorithme déterministe. La procédure (déjà donnée par l'équation 1.22) est la suivante

$$W_{k+1} = W_k + 2\lambda_k (d_k - W_k^T E_k) E_k \tag{1.59}$$

Il est surprenant de constater que la procédure de Widrow-Hoff conduit en général à de MEILLEURES PERFORMANCES.

Plusieurs raisons peuvent êtres avancées pour expliquer ce fait. Premièrement, il est intéressant de disposer d'une procédure qui modifie les poids au fur et à mesure que de nouvelles données arrivent. Elle permet de suivre les variations des données d'apprentissage lorsque celles-ci ne sont pas stationnaires. Deuxièmement, les erreurs de mesure du gradient sont la cause d'une trajectoire erratique dont l'avantage est qu'elle fait explorer une plus grande zone de l'espace des poids. La troisième raison est que les poids sont couplés et que ce couplage s'effectue indirectement par l'intermédiaire de l'erreur. Plus les modifications de poids sont fréquentes, plus ce couplage pourra être fort. Il semble donc profitable d'effectuer le maximum de modifications possible, c'est à dire après chaque échantillon.

A notre connaissance, il n'existe AUCUNE démonstration de convergence de la procédure de Widrow-Hoff dans le cas général qui soit formellement irréprochable. Il existe de nombreuses démonstrations dans des cas particuliers. Par exemple, lorsque la séquence des λ_h est soigneusement choisie décroissante avec de bonnes propriétés, on démontre la convergence en probabilité pour des signaux stationnaires, c'est le cas de l'approximation stochastique de Robbins-Monro. La condition portant sur la suite des λ_h est la suivante ([Duda & Hart 73])

$$\sum_{h=0}^{\infty} \lambda_h^2 < \infty$$

et

$$\sum_{h=0}^{\infty} \lambda_h = \infty$$

La première condition assure la convergence vers 0 des λ_h, la deuxième assure que cette convergence est suffisamment lente pour que le vecteur de poids ne soit pas "bloqué" loin de la solution. Ces conditions sont en particulier satisfaites par le choix

$$\lambda_h = \frac{a}{b+h}$$

où a et b sont deux constantes positives. Les démonstrations de ceci ne seront pas développées ici.

Un autre exemple où une démonstration de convergence existe est lorsque la matrice de covariance des entrées est diagonale, mais c'est une hypothèse beaucoup trop restrictive pour être réaliste. Dans les autres cas (λ constant, entrées non indépendantes,...) il n'existe que des "preuves heuristiques" du type de celle du paragraphe 1.4.1 [Widrow & Stearn 85]. La difficulté du problème est à l'origine d'une myriade de techniques dont l'ingéniosité n'a d'égal que le nombre. Elle font intervenir les martingales (article de Métivier dans [Macchi 84]), ou la méthode dite des "équations différentielles ordinaires" introduite par Ljung, qui traite du cas λ décroissant avec entrées non-indépendantes. On trouvera une bonne synthèse de ces travaux dans [Macchi 84], on peut également se reporter à [Landau 81].

Widrow [Widrow & Stearn 85] traite le problème de la convergence en modifiant l'expression de l'algorithme pour y introduire un "bruit d'estimation" du gradient B. Le gradient estimé est la somme du gradient vrai et du bruit d'estimation

$$\tilde{\nabla} C(W_h) = \nabla C(W_h) + B_h$$

Widrow dérive par cette méthode le coût additionnel dû aux oscillations autour de la solution lorsque C_{min} est différent de 0 L'espérance mathématique du coût après

convergence est donné par

$$\mathcal{E}[C] = C_{min} + \lambda C_{min}\mathrm{Tr}(R)$$

où $\mathrm{Tr}(R)$ est la trace de R, c'est à dire la somme des termes de la diagonale. On voit que les oscillations au fond de la paraboloïde seront proportionnelles à λ, et sensiblement proportionnelles au nombre d'entrées si l'on considère que $\mathrm{Tr}(R)$ l'est.

convergence en moyenne et convergence du coût

Au paragraphe 1.4.1 nous avons donné les conditions nécessaires à la convergence de la MOYENNE des poids, ainsi qu'une estimation de la vitesse de convergence. Malheureusement "convergence en moyenne des poids" ne veut pas dire convergence du coût. L'espérance mathématique du vecteur de poids (son moment d'ordre 1) peut tout à fait converger sans pour autant que le coût qui lui est associé converge, ce dernier dépend en effet du moment d'ordre 2 du vecteur de poids. Honig et Messerschmitt [Honig & Messerschmitt 84] donnent une condition sur λ assurant la convergence du coût. Nous présenterons simplement le résultat sans démonstration

$$0 < \lambda < \frac{1}{\mathrm{Tr}(R)}$$

Cette condition est beaucoup plus restrictive que la conditon 1.46, mais elle possède l'avantage d'être plus facile à respecter: il est plus simple d'estimer la trace de R que la plus grande de ses valeurs propres. Ceci signifie qu'il existe une zone de valeurs de λ comprise entre $1/\mathrm{Tr}(R)$ et $1/l_{max}$ dans laquelle la convergence en moyenne des poids est assurée, sans que la convergence du coût le soit. Le vecteur de poids peut alors être sujet à d'importantes oscillations (non bornées) autour de la solution. Si l'on considère que les entrées ont des covariances approximativement égales, la condition ci-dessus se transforme en

$$0 < \lambda < \frac{1}{(n+1)e^2} \tag{1.60}$$

avec la notation

$$e^2 = \mathcal{M}[e_i^2] \quad i = 0 \ldots n$$

Toutes choses égales par ailleurs, λ devra donc être inversement proportionnel au nombre d'entrées.

Le temps de convergence des poids obtenu par ce choix est donné par l'équation 1.48

$$\tau = \frac{(n+1)e^2}{2l_{min}}$$

pour des données décorrélées (R diagonale) nous avons $l_{max} = e^2$, ce qui transforme l'équation ci-dessus en

$$\tau = \frac{(n+1)}{2}$$

c'est malheureusement un cas tout-à-fait idéaliste, et dans la pratique le temps de convergence sera plus long. Ce résultat donne néanmoins une loi approximative de proportionnalité entre le temps de convergence et le nombre d'entrées.

s'affranchir des variations d'amplitude des entrées

Il apparaît clairement au vu des résultats des paragraphes 1.4.1 et 1.4.3 que le temps de convergence, ainsi que la valeur optimale de λ sont directement reliés à l'amplitude des vecteurs d'entrée. En effet, supposons que l'on multiplie toutes les composantes des vecteurs d'entrée par un facteur k. La matrice de covariance se trouvera multipliée par un facteur k^2, ainsi que ses valeur propres. Si $k > 1$ les conditions de convergence pourront êtres violées, rendant l'algorithme divergent. Si $k < 1$ le temps de convergence sera multiplié par $1/k^2$, ce qui pourra être compensé en multipliant λ par $1/k^2$. Il faut donc adopter une attitude méfiante lorsque les vecteurs d'entrée sont sujets à des variations d'amplitude, une augmentation de celle-ci pouvant entraîner la divergence.

Une technique simple pour résoudre ce problème consiste à NORMALISER λ en fonction de la covariance des entrées. Nous proposons ici une variante du procédé proposé dans [Honig & Messerschmitt 84]. La valeur de λ est calculée à chaque itération avec la formule

$$\lambda_h = \frac{\mu_1}{\sigma_h^2 + \mu_2}$$

μ_1 et μ_2 sont des paramètres constants judicieusement choisis, et σ_h^2 est une estimation de la trace de R au temps h. L'objet de μ_2 est d'éviter que λ devienne trop grand lorsque la valeur de σ^2 diminue. On pourra utiliser pour σ un estimateur du type suivant

$$\sigma_h^2 = (1-\alpha)\sigma_{h-1}^2 + \alpha E_h^T E_h$$

la paramètre α doit bien sûr être choisi petit devant 1.

Nous allons maintenant proposer une généralisation de cette idée. Le raisonnement précédent peut être poussé plus loin en considérant chaque composante du vecteur d'entrée indépendamment. Plaçons-nous dans la situation où seule une composante du vecteur d'entrée est multipliée par un facteur k. Il serait judicieux de compenser l'effet de ceci en ne modifiant QUE le comportement du poids correspondant et pas celui des autres poids. Ceci conduit à l'utilisation d'un λ différent pour chaque composante dont le rôle est d'égaliser les vitesses de convergence des poids. L'avantage

est évident lorsque les entrées sont décorrélées. Considérons un exemple où la matrice de covariance est diagonale, et où tous les termes de la diagonale sont égaux à 1 sauf un seul qui est égal à k^2. Il est clair que la convergence la plus rapide sera obtenue en associant à cette dernière composante un λ particulier qui devra être $1/k^2$ fois plus grand que les autres. Le système ainsi corrigé se comportera "comme si" la matrice de covariance avait été égale à l'identité. Nous pourrons utiliser une règle du même type que la précédente. Soit λ_{ih} le pas attaché au poids i au temps h

$$\lambda_{ih} = \frac{\mu_1}{(n+1)\sigma_{ih}^2 + \mu_2}$$

avec

$$\sigma_{ih}^2 = (1-\alpha)\sigma_{ih-1}^2 + \alpha e_{ih}^2$$

Cette technique d'égalisation du spectre peut être vue comme une approximation de la méthode de Newton. L'inverse de la matrice de covariance utilisée dans la méthode de Newton est ici approximée par l'inverse des termes de sa diagonale.

Ce type de technique est très utile dans les réseaux multi-couches où l'on ne maîtrise pas les niveaux d'activité des cellules internes, et où la nécessité de s'en affranchir est cruciale.

quelques simulations

Quelques manipulations et expérimentations avec ce modèle sont utiles à une bonne compréhension intuitive des phénomènes. Certaines sont présentées ici, plus dans un but d'illustration que dans un but de validation expérimentale des résultats. Nous avons utilisé un sommateur linéaire à deux entrées e_0 et e_1 (l'entrée 0 n'est pas constante ici) ce qui permet de visualiser facilement la trajectoire du vecteur de poids dans un plan. Le problème choisi pour l'expérimentation est très idéalisé, mais possède l'avantage que tous ses paramètres peuvent être facilement contrôlés. Les vecteurs d'entrée sont calculés à l'aide de la formule suivante

$$e_0 = \sqrt{2}\cos z$$

et

$$e_1 = \sqrt{2}\cos(z+\theta)$$

où z est une variable aléatoire uniformément distribuée entre 0 et 2π, et θ un paramètre permettant de régler l'intercorrélation de e_0 et e_1. La matrice de covariance possède la forme suivante

$$R = \begin{pmatrix} 1 & \cos\theta \\ \cos\theta & 1 \end{pmatrix} \qquad (1.61)$$

en effet

$$R = \mathcal{M}[EE^T]$$

$$r_{00} = r_{11} = \frac{1}{2\pi} \int_0^{2\pi} 2\cos^2 z \; dz = 1$$

$$r_{01} = r_{10} = \frac{1}{2\pi} \int_0^{2\pi} [2 \; \cos z \; \cos(z+\theta)] dz = \cos\theta$$

Les valeurs propres sont égales à $1+\cos\theta$ et $1-\cos\theta$. La plus grande valeur de λ garantissant la convergence est donc

$$\lambda_{max} = \frac{1}{1+|\cos\theta|}$$

si l'on néglige l'effet des erreurs d'estimation du gradient. En vertu de 1.49, la valeur optimale de λ est

$$\lambda_{opt} = \frac{1}{l_{min} + l_{max}} = \frac{1}{2}$$

En comparant les deux équations précédentes, on voit immédiatement que la valeur optimale de λ est très proche de sa valeur maximale lorsque $\cos\theta$ est proche de 1, c'est à dire lorsque les entrées sont très corrélées. On constate ici que le choix du nombre d'entrées égal à 2 nous conduit à une situation particulière où $\mathrm{Tr}(R) = l_{min} + l_{max}$, la condition de convergence du coût 1.60 est donc respectée par le choix optimal de λ.

Quant au temps de convergence, il est proportionnel à

$$\frac{1}{1-|\cos\theta|}$$

Les vecteurs propres de R sont $(1,1)$ et $(1,-1)$. La sortie désirée est choisie de la manière suivante

$$d = -0.2e_0 + 0.4e_1$$

la solution idéale (de coût nul) est donc

$$\tilde{W}^T = (-0.2, 0.4)$$

La valeur de la fonction de coût peut être calculée très facilement pour tout W puisque R et $\tilde{W}$ sont connus par avance

$$C(W) = (W - \tilde{W})^T R (W - \tilde{W})$$

La figure 1. 14 montre la trajectoire des poids lorsque $\theta = \pi/2$, la matrice de covariance R est alors égale à l'identité, et la paraboloïde de la fonction de coût

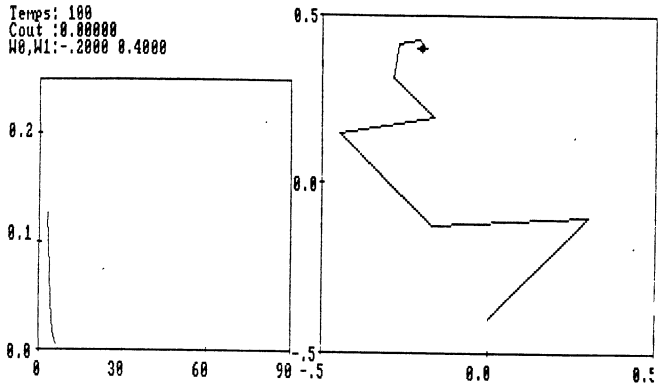


Figure 1. 14: Convergence de la procédure de Widrow-Hoff lorsque les entrées sont décorrélées. La courbe de droite est la trajectoire du vecteur de poids, la courbe de gauche représente la valeur de la fonction de coût en fonction du temps. Le vecteur de poids initial est (0,-0.4). $\theta = \pi/2, \lambda = 1/2$

est de révolution. On se trouve donc dans les conditions idéales. En choisissant $\lambda = \lambda_{opt} = 1/2$, le processus converge en une dizaine d'itérations. Ceci peut être interprété en disant qu'approximativement dix points sont nécessaires à l'algorithme pour estimer l'allure de la paraboloïde et la position de son minimum. La figure 1. 15 rend compte d'un cas plus réaliste où l'on a pris $\theta = 20°$, ce qui donne $\cos\theta = 0.94$, et $\lambda = 1/2$. On voit clairement apparaître deux directions de convergence associées aux deux valeurs propres de R. Selon la direction (1,1) associée à la valeur propre 1.94 la convergence est très chaotique, alors que selon l'autre direction (-1,1) associée à la valeur propre 0.06 la convergence est exponentielle et beaucoup plus lente. On doit remarquer que bien que λ soit très proche de sa valeur maximale théorique (0.5155), le processus reste convergent. Cette valeur théorique maximale de λ semble refléter assez fidèlement la réalité. L'expérience suivante représentée sur la figure 1.

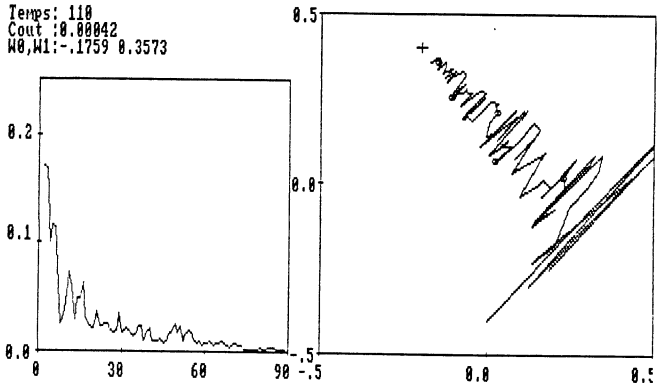


Figure 1. 15: Convergence de la procédure de Widrow-Hoff lorsque les entrées sont corrélées. La courbe de droite est la trajectoire du vecteur de poids, la courbe de gauche représente la valeur de la fonction de coût en fonction du temps. Le vecteur de poids initial est (0,-0.4). $\theta = 20°, \lambda = 1/2$

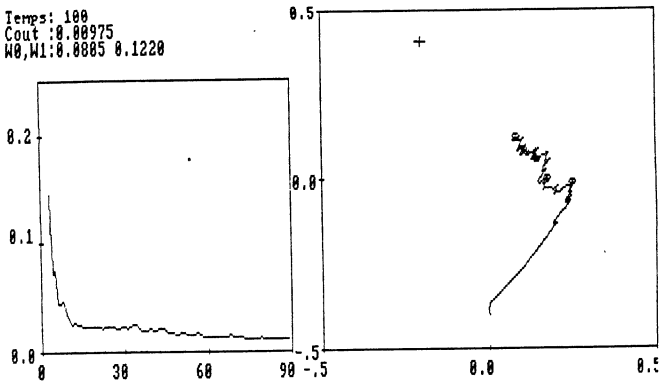


Figure 1. 16: Mêmes conditions que dans la figure 1. 15 mais les deux modes sont en régime exponentiel $\theta = 20°$,$\lambda = 0.1$

16 est effectuée dans les même conditions que la précédente, mais λ est maintenant choisi égal à 0.1 de manière à assurer une convergence exponentielle selon les deux directions. La courbe du coût en fonction du temps fait clairement apparaître qu'au début le mode associé à la direction (1,1) est dominant, et qu'à partir de l'itération 12, l'autre mode devient dominant. La trajectoire des poids est beaucoup moins chaotique que précédemment mais la convergence est plus lente. La situation s'aggrave très vite lorsque la corrélation augmente. La figure 1. 17 montre les trajectoires lorsque $\theta = 10°$ et $\lambda = 1/2$. La convergence devient extrêmement lente dans la direction (-1,1). On peut néanmoins objecter que selon cette direction le coût varie peu, et que la solution obtenue, bien qu'éloignée de la solution idéale, est acceptable. Les paragraphes suivants présenteront des variations de l'algorithme original qui tenteront de résoudre le problème.

D'autres simulations ont été effectuées avec des éléments à plus de deux entrées. Elles tendent à confirmer la validité de la condition 1.60.

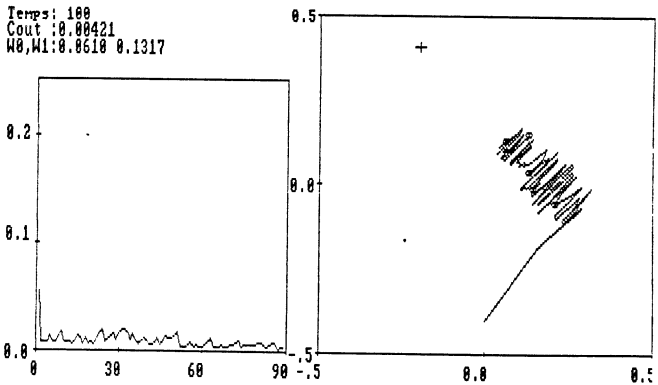


Figure 1. 17: Mêmes conditions que dans la figure 1. 15 mais avec $\theta = 10°$.

1.4.4 variantes

de nombreuses variantes de la procédure de base ont été proposées dans la littérature. Nous allons ici en présenter quelques exemples représentatifs.

moindre valeur absolue moyenne

Une simplification immédiate de la procédure de Widrow-Hoff consiste à quantifier le signal d'erreur utilisé pour l'apprentissage. Lorsque cette quantification est à deux niveaux, l'algorithme obtenu est dit "à renforcement binaire" et s'avère être très proche de la règle du perceptron

$$W_{h+1} = W_h + \lambda_h \, sgn(d_h - W_h^T E_h) \, E_h \tag{1.62}$$

où *sgn* est la fonction seuil qui vaut -1 lorsque son argument est négatif et +1 dans le cas contraire. L'équation 1.62 peut être réécrite en utilisant les notations habituelles

$$W_{h+1} = W_h + \lambda_h \, sgn(\epsilon_h) \, E_h \tag{1.63}$$

Gersho (dans [Macchi 84]) en étudie les propriétés de convergence. Il est facile de voir qu'elle est, en fait, une procédure de gradient stochastique appliquée à une fonction de coût de la forme suivante

$$C_a(W) = \mathcal{E}[\,|\,\epsilon\,|\,] = \mathcal{M}[\,|\,d - W^T E\,|\,]$$

Ce critère n'est pas partout dérivable ce qui interdit la convergence stricte du gradient stochastique. Cependant, il est surprenant de constater que cette procédure converge EN MOYENNE pour TOUTE valeur (positive) de λ (Gersho in [Macchi 84]). La solution optimale ne sera effectivement atteinte que si λ décroît avec le temps [9]. Dans le cas contraire, d'importantes fluctuations autour du minimum subsisteront. Gersho donne une borne à ces fluctuations

$$\mathcal{E}[C_a] = C_{a,min} + \frac{\lambda}{2} \mathcal{E}[E^T E]$$

elles sont donc directement proportionnelles à λ. L'analogie avec la règle du perceptron est évidente. Une différence essentielle les distingue néanmoins: le calcul de ϵ. Contrairement à la règle du perceptron, l'ensemble des solutions est ici réduit à un point (en général) et non à un hypercône. Il faut rappeler que le but de la procédure du perceptron est de trouver un hyperplan, c'est à dire une direction, alors qu'ici, le problème est de trouver un vecteur de poids. L'intérêt principal de cette procédure est la simplification des calculs nécessaires à l'apprentissage. Aucune multiplication n'est effectuée ce qui simplifie énormément les réalisations matérielles.

[9] et si la distribution des vecteurs d'entrée est stationnaire

moindres carrés récursifs

L'algorithme des moindres carrés récursifs est une version de l'algorithme de Newton dans laquelle l'inverse de la matrice de covariance est calculée itérativement à mesure de l'arrivée des données. Il est possible de dériver une version stochastique de cet algorithme en utilisant un estimateur de l'inverse de la matrice de covariance. L'algorithme est le suivant

$$W_{h+1} = W_h + \lambda_h R_h^{-1}(d_h - W_h^T E_h)E_h \tag{1.64}$$

ici R_h^{-1} est une estimation de R^{-1} au temps h. L'estimateur utilisé peut être le suivant

$$R_h = R_{h-1} + \lambda_h[E_h E_h^T - R_h]$$

l'utilisation de cet estimateur nécessite malheureusement l'inversion de R_h à chaque itération. C'est pourquoi on préfèrera plutôt utiliser directement une expression récursive de la quantité

$$K_h = \lambda_h R_h^{-1}$$

dont la mise à jour est effectué à l'aide de

$$K_h = \frac{1}{\gamma_h}\left[K_{h-1} - \frac{K_{h-1}E_h E_h^T K_{h-1}}{\gamma_h + E_h^T K_{h-1} E_h} \right]$$

avec la notation

$$\gamma_h = \frac{\lambda_{h-1}}{\lambda_h}[1 - \lambda_h]$$

on se reportera à [Widrow & Stearn 85] pour un traitement plus complet et à Ljung (in [Macchi 84]) pour une analyse de la convergence. Cette procédure permet la convergence en ligne (presque) droite dans l'espace des poids. Le prix à payer est la complexité des calculs à chaque itération.

1.5 comparaison

Parmi les algorithmes présentés tout au long du chapitre, certains sont spécifiquement adaptés aux automates à seuil. Leur objet est de trouver un hyperplan, c'est à dire une direction dans l'espace des poids. C'est le cas de la règle du perceptron et de ses variantes. D'autres, comme les procédures de moindre carrés ont pour but de trouver un vecteur de poids plutôt qu'une direction. La comparaison entre ces deux points de vue repose sur plusieurs critères. Selon que l'ensemble d'apprentissage est connu

par avance, ou qu'il s'agit d'apprendre une tâche susceptible d'évoluer au cours du temps et/ou comportant un certain degré d'aléatoire (distribution non stationnaire) les deux types d'algorithmes se comporteront différemment. Il est clair que la règle du perceptron est mieux adaptée à la première situation alors que la règle de Widrow-Hoff est mieux adaptée à la seconde. De même la qualité de la solution dépendra du caractère séparable des échantillons. On sait que la procédure du perceptron ne converge QUE SI l'ensemble d'apprentissage EST linéairement séparable, et peut ne donner aucune solution intéressante dans le cas contraire. Pour la règle de Widrow-Hoff nous savons qu'elle ne donne pas toujours la solution exacte dans le cas séparable, mais qu'elle fournit une solution approchée acceptable dans le cas contraire. La procédure de Ho et Kashyap, que nous ne détaillerons pas ici (voir [Duda & Hart 73]) est une tentative de solution de ce problème. Brièvement, l'idée est de minimiser un critère quadratique classique auquel on a rajouté un paramètre supplémentaire. Le critère (déterministe) utilisé est de la forme

$$C(W, Z) = \frac{1}{p} \sum_{k=1}^{p} (d_k z_k - W^T E_k)^2$$

où les z_k constituent un vecteur de paramètres qu'il s'agira d'optimiser en respectant la contrainte

$$z_k > 0 \quad k = 1 \ldots p$$

on voit que la minimisation de ce critère (en fonction de W et Z) ne contraint que le signe de $W^T E$.

Dans certains cas, comme celui des réseaux à couches, ou les applications de filtrage adaptatif, les cellules ne sont pas binaires. Il conviendra dans ce cas d'utiliser une procédure de minimisation appliquée à un critère bien choisi (par exemple: critère quadratique), les procédures de type perceptron étant ici inopérantes.

Lorsqu'on utilise une procédure de moindres carrés, le gradient stochastique (type Widrow-Hoff) semble plus efficace que le gradient vrai [10]. La raison en est sans-doute qu'avec le gradient stochastique une modification des poids est effectuée après chaque échantillon, alors qu'avec le gradient vrai la modification n'intervient qu'après accumulation des gradients de tous les échantillons de l'ensemble d'apprentissage. Comme cela a été indiqué plus haut, le couplage entre les dynamiques des poids s'effectue par l'intermédiaire de l'erreur et sera d'autant plus effective que les modifications de poids sont fréquentes.

[10] quand celui-ci est applicable

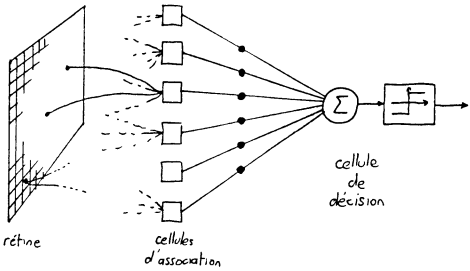


Figure 1. 18: un perceptron

1.6 le perceptron et les théorèmes de limitation

Ce sous-chapitre est consacré à l'étude du perceptron et de la critique qui en est faite dans le livre de Minsky et Papert [Minsky & Papert 68]. Les raisons de l'échec du perceptron sont souvent mal connues ou mal interprétées. Il semble utile de clarifier la situation sur quelques point précis afin d'élaborer des solutions permettant d'en lever les limitations. L'objet de prédilection sera le PERCEPTRON SIMPLE. Un perceptron simple est constitué d'une "rétine" ou tout autre senseur, sur laquelle on projette des images binaires. Chaque point de la rétine peut prendre deux états: noir ou blanc, que nous noterons par commodité 1 et 0. Vient ensuite une série de CELLULES D'ASSOCIATION (ou extracteurs de primitives). Chaque cellule d'association peut calculer une fonction booléenne quelconque, prédéterminée, dont les variables d'entrée sont constituées par un sous-ensemble des points de la rétine. Les sorties des cellules d'association sont combinées par une CELLULE DE DECISION qui est un simple ELS. Les poids de la cellule de décision sont éventuellement modifiables par apprentissage (voir fig 1. 18).

Il n'est pas inutile de préciser certains points:

- Selon cette définition, un perceptron PEUT CALCULER N'IMPORTE QUELLE FONCTION BOOLÉENNE.
- Plusieurs variations autour du perceptron simple existent. Dans tous les cas, il n'y a QU'UNE SEULE ET UNIQUE COUCHE DE POIDS MODIFIABLES.

Le premier point est facilement prouvable si l'on remarque qu'aucune hypothèse n'est faite sur la nature des cellules d'association. On peut donc imaginer un "perceptron simple" constitué d'une seule cellule d'association calculant la fonction désirée, et suivie d'une cellule de décision possédant une entrée unique dont la seule fonction est de recopier son entrée. Cet argument n'est pas constructif, et il existe une autre structure universelle de perceptron. Pour la construire, on pose l'hypothèse que la rétine est finie et comporte r points. Il "suffit" de disposer de $n = 2^r$ cellules d'association. Chacune d'elles produit un "1" pour une et une seule configuration de la rétine et produit "0" pour les $2^r - 1$ autres. Réciproquement, une configuration de la rétine "active" une cellule d'association et une seule. Les cellules d'association forment ce que l'on peut appeler un jeu de masques exhaustif. De manière évidente, toute fonction booléenne peut être calculée par un perceptron de ce type avec un vecteur de poids binaire [11]. Mais cette universalité coûte cher: le nombre de cellules d'association croît exponentiellement avec la taille de la rétine.

Ceci soulève une question intéressante à laquelle Minsky et Papert tentent de répondre: "quelle est la complexité d'un perceptron devant calculer une fonction booléenne donnée?". Naturellement, il n'y a pas de réponse générale à cette question, et l'on en est réduit à raisonner sur des cas particuliers susceptibles d'éclairer la situation. Un premier type de restriction concerne la structure du perceptron, et plus particulièrement, la complexité des cellules d'association. Il peut s'agir de limiter le nombre d'entrées de chaque cellule d'association, ou de les restreindre à prendre leurs entrées dans une zone particulière de la rétine, ou encore de limiter leur nombre. Un cas intéressant est celui où les cellules d'association sont elles-mêmes des ELS, c'est alors le GAMBA-PERCEPTRON, ou réseau d'ELS à deux couches, dont l'universalité a été démontrée au paragraphe 1.1.5. L'autre classe de restrictions concerne la nature des poids de la cellule de décision, et en particulier leur précision. A chacune de ces hypothèses correspond un théorème de limitation. Certains de ces théorèmes vont être succintement présentés.

Il est utile de définir plus précisément les différent éléments d'un perceptron. Notons $\phi_i(E)$ la sortie de la cellule d'association numéro i lorsque la forme E est présentée sur la rétine. La sortie du perceptron est donnée par:

$$s(E) = f(\sum_{i=0}^{q} w_i \phi_i(E))$$

où f est la fonction signe et W le vecteur de poids. Soit z_i le nombre de points de

[11] Dans un autre contexte, cette structure est appelée "mémoire RAM"

la rétine dont dépend ϕ_i. On définit L'ORDRE D'UN PERCEPTRON O comme étant

$$O = \max_i z_i$$

l'ordre d'un perceptron est donc le nombre maximum de connexions entre la rétine et une quelconque des cellules d'association. Un prédicat (c'est à dire une fonction booléenne) sera dit d'ordre O s'il existe un perceptron d'ordre O qui le réalise.

Minsky et Papert démontrent trois principaux théorèmes relatifs à l'ordre des prédicats. Le premier concerne le prédicat de convexité. Le perceptron doit répondre +1 lorsqu'une forme convexe lui est présentée et -1 sinon. Une forme est convexe si le fait suivant est vérifié pour tout couple de points appartenant à la forme: "soit deux points a et b appartenant à la forme, tout point c situé sur le segment $[a,b]$ appartient à la forme". Le prédicat de convexité est d'ordre 3. Il suffit en effet d'explorer tous les triplets de points alignés, et de vérifier que le fait ci-dessus est vérifié pour chacun d'eux. Les résultats sont ensuite combinés dans un ET, réalisé par la cellule de décision qui produit la réponse finale. Ceci nécessite quand même de disposer d'autant de ϕ_i qu'il y a de triplets de points alignés sur la rétine.

Un autre résultat, concerne le prédicat de parité. Le perceptron doit répondre +1 si le nombre de points de la forme est pair, et -1 s'il est impair. Le résultat est le suivant: avec une rétine de n points, il faut au moins un ϕ_i qui dépende de TOUS les points de la rétine. En d'autres termes, le prédicat de parité est d'ordre n.

Le troisième résultat concerne la connexité. Il est nécessaire auparavant de définir la notion de perceptron à diamètre limité: un perceptron est de diamètre d si la distance de deux point de la rétine dont dépend un même ϕ_i est au maximum égale à d. Chaque ϕ_i est contraint à prendre ses entrées à l'intérieur d'un "disque" de diamètre d. Pour le prédicat de connexité, un perceptron doit produire un 1 si la figure qui lui est présentée est en un seul morceau, et 0 s'il y a plusieurs composantes connexes. Le théorème est le suivant: "Le prédicat de connexité n'est pas calculable par un perceptron à diamètre limité". Les dimensions de la rétine sont, bien sûr, supposées plus grandes que le diamètre maximum du support des ϕ_i. Le principe de la démonstration est exposé sur la figure 1. 19. On dessine sur la rétine 4 figures constituées chacune de la réunion d'une partie gauche, supposée être vue par une partie des ϕ_i, et d'une partie droite, vue par un ensemble de ϕ_i disjoint du précédent Ce choix de deux ensembles de ϕ_i disjoints pour les parties gauche et droite est toujours possible compte tenu de l'hypothèse de diamètre limité: il suffit de dessiner des figures assez grandes. L'idée est de choisir les parties gauche et droite de manière à contraindre la cellule de décision à effectuer un XOR pour décider de la connexité. Un exemple de tels dessins est montré sur la figure 1. 19. Les dessins 0 et 3 ne sont pas connexes, et devront produire 0 en

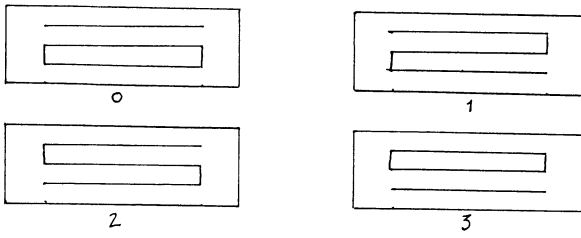


Figure 1. 19: Quatre figures utilisées pour le prédicat de connexité

sortie, quant aux dessins 1 et 2 il devront produire 1. Le groupe de ϕ, voyant la partie gauche ne peut, au mieux, que distinguer les figures 0 et 1 des figures 2 et 3. Quant aux ϕ, de la partie droite, ils ne peuvent que distinguer les figures 0 et 2 des figures 1 et 3. Notons a la réponse fournie par le groupe de gauche aux figures 0 et 1, $\bar{a}$ leur réponse aux figures 2 et 3. De même, b notera la réponse du groupe de droite aux figures 0 et 2 et $\bar{b}$ leur réponse pour les figures 1 et 3. La cellule de décision devra produire les résultats résumés dans la table ci-dessous

Figure	Gauche	Droite	Connexité
0	a	b	0
1	a	$\bar{b}$	1
2	$\bar{a}$	b	1
3	$\bar{a}$	$\bar{b}$	0

On peut assigner aux symboles a et b les valeurs 0 ou 1. Selon cette attribution, la table ci-dessus représentera la table de vérité du XOR ou de l'EQUIVALENCE. Nous savons que ces deux fonctions ne peuvent pas être calculées par la cellule de décision, puisque celle-ci est une fonction à seuil. Il en résulte que le prédicat de connexité n'est pas calculable par un perceptron simple à diamètre limité.

Ces théorèmes et les autres résultats du livre de Minsky et Papert sont intéressants à plus d'un titre. En particulier parce qu'ils donnent une relation entre le degré de parallélisme d'une machine et sa complexité structurelle, l'ordre d'un perceptron peut être considéré comme une mesure du degré de parallélisme: plus l'ordre est faible,

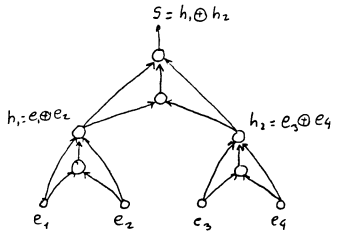


Figure 1. 20: Un réseau à seuil pour calculer le prédicat de parité

plus la machine est parallèle, et plus son applicabilité est restreinte. On pouvait se douter que le prédicat de parité était complexe à réaliser avec un perceptron: c'est un problème de nature essentiellement séquentielle. L'algorithme séquentiel de la parité est trivial, alors qu'un perceptron ne peut le résoudre sans comporter un élément global, dont la structure peut être quelconque. Le prédicat de parité possède une particularité très intéressante: la connaissance de $n-1$ bits d'entrée sur les n, ne donne AUCUNE information sur le résultat, tout dépendra du dernier bit. Cela ne signifie pas que le prédicat de parité ne soit pas parallélisable. On peut remarquer que le prédicat de parité sur n bits est réalisable avec un réseau d'ELS à $2\log_2 n$ couches comportant au total $2(n-1)$ cellules, ce qui représente un certain degré de parallélisme. Il suffit pour cela d'utiliser une série de XOR à 2 entrées (comportant chacun 2 ELS) architecturés en arbre binaire. Les feuilles de l'arbre sont les entrées et la racine est la sortie (voir 1. 20). Ce réseau, bien que parallèle, comporte $2\log_2 n$ couches, il est donc partiellement sériel. Ce schéma peut facilement être implémenté sur une machine parallèle à $\frac{n}{2}$ processeurs en un temps proportionnel à $\log n$, cette implémentation permet également la génération de la parité sur tous les segments initiaux de la suite de n bits dans le même temps.

1.7 conclusion du chapitre: limitations des modèles élémentaires

Les différentes méthodes d'apprentissage abordées dans ce chapitre possèdent en commun deux caractéristiques essentielles:

- elles sont limitées aux problèmes linéairement séparables.
- elles nécessitent la présence d'un signal d'erreur ou d'une sortie désirée pour chaque forme.

ces méthodes reposent essentiellement sur la nature LINÉAIRE, ou considérée comme telle, de la relation entrée-sortie. C'est à ce niveau que se situent leurs limitations. Il a été montré qu'un réseau d'ELS à deux couches pouvait calculer une fonction booléenne quelconque, il suffirait donc pour lever les limitations de modifier les algorithmes simples en tenant compte de cette structure en couches. Malheureusement, cette généralisation n'est pas immédiate, car pour calculer la sortie d'un réseau à deux couches il faut traverser deux transformations non-linéaires. Il est, par conséquent, très difficile d'évaluer l'effet de la modification d'un poids de la première couche sur la réponse finale. Ce problème peut être exposé de manière équivalente en disant qu'il est difficile de décider quelles sorties désirées doivent être assignées aux cellules internes (des premières couches). C'est le problème classique du "bouc émissaire": [12] lorsqu'une réponse résultant d'une cascade d'opérations non-linéaires est erronée, il est difficile de savoir quelle étape(s) est (sont) en cause, et, à plus forte raison, comment la (les) modifier. De nombreuses solutions partielles à ce problème ont été proposées dans le passé sans qu'aucune ne soit totalement satisfaisante. Les causes d'insatisfaction viennent soit du manque de généralité des structures proposées, soit de la difficulté à faire converger l'apprentissage. Certains auteurs [Minsky & Papert 68] ont même suggéré qu'il était vain de poursuivre les recherches dans cette direction. Ces dernières années ont vu apparaître des algorithmes d'apprentissage pour réseaux "à cellules cachées", dont les réseaux multi-couches sont un cas particulier et qui apportent une réponse à ce problème. Parmi ces procédures, la plus utilisée est l'algorithme de RÉTRO-PROPAGATION proposé par l'auteur et par deux autres groupes indépendamment [le Cun 85] [le Cun 86] [Rumelhart & al. 86a] [Parker 85] et qui fera l'objet du chapitre 3.

[12] le terme anglais est "credit assignment problem"

Chapitre 2

les réseaux de neurones

Le concept de ce qu'il est convenu d'appeler "réseaux neuronaux" [1] est appauru dès les travaux de Clark et Farley qui, les premiers, ont simulé sur un calculateur numérique un système de "neurones" rebouclé comportant des connexions variables dynamiquement [Farley & Clark 54]. La règle de modification des poids était la règle dite "de Hebb" qui confère à une pondération une valeur proportionnelle à une estimation de la covariance des cellules qu'elle relie. L'idée a été maintes fois reprise par la suite, en particulier dans [Block & al. 62] où le réseau Hebbien sert d'étage de prétraitement à un perceptron. à l'instar de Clarck et Farley, Block fait intervenir un DÉLAI de transmission des connexions. Les réseaux d'ELS totalement connectés et rebouclés utilisant la régle de Hebb ont été porposés comme mémoire associative dans [Nakano 71] et [Nakano 72], une réalisation matérielle y est même décrite. Ils ont été étudiés dans formellement dans [Amari 71] et [Amari 72]. Ces travaux se sont basés sur les idées de mémoires associatives linéaires de Willshaw et Longuet-Higgins proposées en 1969 (voir [Willshaw 81]), et sur leur étude par Kohonen [Kohonen 72]. Kohonen propose de nombreuses améliorations du modèle linéaire de base, en utilisant la théorie des matrices pseudo-inverses pour calculer les poids [Kohonen & Ruohonen 73] [Kohonen 74]. Ce modèle sera étudié dans [Reid & Sutherland Frame 75]. Les travaux de Kohonen sont regroupés dans les ouvrages [Kohonen 77] [Kohonen 81] et [Kohonen 84a].

Ces modèles ont fait l'objet de nombreuses publications récentes, abordant aussi bien les aspects théoriques que les réalisations matérielles avec des technologies électroniques ou optiques. Cet engouement a fait suite à la publication [Hopfield 82] qui a sucité l'intérêt de nombreux physiciens pour ces problèmes, et qui a, par contre-coup, réveillé l'intérêt des informaticiens. L'article de Hopfield montre certaines analogies entre un modèle de réseau d'ELS totalement connecté et certains cristaux aux propriétés étonnantes: les verres de spins. Le comportement de l'un et de l'autre sont décrits par une même FONCTION ÉNERGIE.

Beaucoup de travaux récents ont étudié les capacités des mémoires associatives à base de réseaux de neurones, et en ont proposé des variations.

[Murakami & Aibara 81] proposent une règle de calcul des poids basée sur la théorie Bayésienne de la décision, chaque neurone est un classifieur Bayésien, dont les entrées sont supposées indépendantes, ce modèle se compare très favorablement aux modèles Hebbiens de type "associatron".

Anderson (in [Hinton & Anderson 81]) propose le modèle "Brain-state-in-the-box" dans lequel les neurones ne sont pas binaires mais utilisent une transformation linéaire saturée. Anderson calcule les poids de sa mémoire associative à l'aide de la procédure

[1] traduction littérale de l'expression anglaise "neural networks"

de Widrow et Hoff décrite au chapitre précédent.

Hopfield [Hopfield 84] propose également un modèle utilisant des neurones non binaires. Il montre l'analogie entre ce système et un système à neurones binaires stochastiques.

Peretto [Peretto 84] utilise les méthodes de la physique statistique pour analyser le comportement du modèle de Hopfield. Il montre que le modèle binaire est une approximation raisonnable des systèmes de neurones réels, il étudie particulièrement le temps de relaxation et la capacité théorique de stockage en fonction de la taille du réseau et du bruit.

Stiles et Denq [Stiles & Denq 85] étudient les capacités de stockage des mémoires associatives linéaires basées sur les pseudo-inverses. Il étudient leur tolérance au bruit, et proposent un système de neurones à seuil totalement connectés dont les poids sont calculés à l'aide de la pseudo-inverse.

[Abu-Mostafa & Saint Jacques 85] étudient les capacités de stockage des réseaux à seuil indépendamment de la méthode de calcul des poids.

L'ouvrage [Bienenstock & al. 86] regroupe une bonne partie des travaux sur la question.

Une étude du comportement dynamique des réseaux d'automate est développée dans [Fogelman 85a] et [Goles 85]. L'influence des modes d'itération (parallèle, séquentiel, séquentiel par blocs, chaotique) est étudiée dans [Fogelman & Weisbuch 86b]. Une étude synthétique des modèles de mémoires associatives peut être consultée dans [Fogelman & al. 87b] [2]

2.1 Les réseaux d'automates à seuil

L'idée du réseau d'ELS totalement connecté a été très étudié par "l'école japonaise" au début des années 70. En particulier Nakano [Nakano 71] [Nakano 72] et Amari [Amari 71] [Amari 72]. Amari formalise les réseaux d'ELS totalement connectés: tout élément du réseau est connecté à tous les autres à travers une matrice de pondération W. Le terme w_{ij} est le poids de la connexion reliant l'élément j à l'élément i. L'état de l'élément i est noté x_i, l'état global des n éléments du réseau est noté par le vecteur X à n composantes (voir figure 2. 1). Amari étudie principalement une règle de mise à jour des états SYNCHRONE. A chaque instant t d'une échelle de temps discrète les états de tous les éléments sont mis à jour simultanément. L'évolution du réseau est

[2] Un extrait de cet article est fourni en Annexe

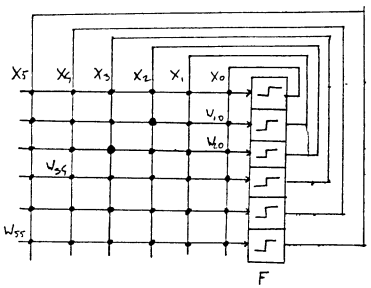


Figure 2. 1: Un réseau de neurones totalement connecté

donc gouvernée par l'équation:

$$X(t+1) = Sign[W\,X(t)] \tag{2.65}$$

Sign est une fonction vectorielle de $R^n \rightarrow R^n$ qui applique la fonction "signe" àchacune des composantes de son argument. En d'autres termes:

$$x_i(t+1) = sign(\sum_{j=1}^{n} w_{ij} x_j(t))$$

Un système de ce type est un automate fini comportant 2^n états. Sur ces états, certains sont STABLES, et certains appartiennent à un CYCLE LIMITE du **graphe d'itération**. Amari étudie le comportement du système en fonction de la matrice W.

Les travaux d'Amari visaient à formaliser "l'associatron" un modèle de réseau neuronal proposé par Nakano [Nakano 71] [Nakano 72]. L'associatron est un réseau d'ELS totalement connecté soumis à une règle de mise à jour synchrone. Nakano propose l'utilisation de son système en tant que MÉMOIRE ASSOCIATIVE. Les états stables du système représentent les objets ou les concepts mémorisés. Lorsqu'on place le réseau dans un état voisin d'un état stable (en forçant l'état d'un sous-ensemble d'éléments de l'extérieur), il évolue et se stabilise dans l'état stable (mémorisé) le plus proche. Il est donc possible de restaurer une information à partir d'une donnée incomplète ou bruitée. Le problème consiste à calculer la matrice W qui rend stables un ensemble de configurations choisies. Nakano utilise lui aussi la règle de Hebb. Si $X^1, X^2, \ldots, X^p$ sont les formes à mémoriser (les états à rendre stables), les poids sont

calculés à l'aide de:

$$w_{ij} = \sum_{k=1}^{p} x_i^k x_j^k$$

soit

$$W = \sum_{k=1}^{p} X^p (X^p)^T$$

Nakano a réalisé l'associatron avec la technologie standard de l'époque (à partir de circuits TTL standard). L'architecture qu'il a utilisé est très voisine des architectures systoliques actuelles. Dans son système, les pondérations sont quantifées sur 16 niveaux (4 bits) entre -8 et +7.

Amari étudie également la mémorisation de séquences, un état est associé à l'état suivant dans la séquence. Pour stocker la séquence d'états $X^1, X^2, \ldots, X^p, X^1, \ldots$ on utilise la règle

$$w_{ij} = \sum_{k=1}^{p} x_i^{k+1} x_j^k$$

Ces règles permettent de calculer à l'avance les pondérations, mais il est également possibles d'en imaginer une version adaptative, qui modifie les poids à mesure que les données arrivent. Par exemple:

$$w_{ij}(t+1) = (1-\alpha) w_{ij}(t) + \alpha x_i(t) x_j(t)$$

où α est une constante positive.

D'autres modes de mise à jour des états peuvent être imaginés. Au lieu de mettre à jour tous les éléments de manière synchrone, il est possible de découper l'ensemble des neurones en blocs, et de mettre à jour ces blocs dans un ordre prédéfini ou déterminé aléatoirement. C'est le mode d'itération séquentiel par blocs. C'est ce mode qui est utilisé dans les réseaux en couche. Un cas extrême est lorsque les blocs ne contiennent chacun qu'un seul neurone, c'est alors l'itération séquentielle, ou itération chaotique lorsque l'ordre de mise à jour est aléatoire.

2.2 Le modèle de Hopfield

Hopfield [Hopfield 82] étudie le comportement des réseaux d'ELS totalement connectés à l'aide d'une fonction énergie. Cette fonction est

$$H(X) = -\frac{1}{2} X^T W X = -\frac{1}{2} \sum_{ij} w_{ij} x_i x_j$$

Il montre que si la matrice W est symétrique cette fonction est décroissante sur la trajectoire, a condition d'utiliser un mode d'itération séquentiel ou chaotique. Il en

conclu que les cycles limites de la dynamique sont tous de longueur 1, en d'autres termes, l'évolution du réseau se termine toujours sur un état stable, minimum local de la fonction énergie $H(X)$. Cette propriété est mise à profit pour stocker des configurations stables. Hopfield utilise la classique règle de Hebb.

Nous pouvons interpréter cette règle comme un moyen de "creuser des trous" dans le paysage d'énergie de manière à rendre certains états stables en abaissant leur niveau d'énergie [le Cun 86].

Creuser un trou peut facilement s'effectuer en appliquant une procédure de gradient sur l'énergie, en faisant décroitre sa valeur pour la configuration à stocker. La procédure est la suivante (H est considérée comme une fonction de la matrice de poids W et de l'état à stocker X.

$$w_{ij} \leftarrow w_{ij} - \lambda \frac{\partial H(W,X)}{\partial w_{ij}}$$

soit

$$w_{ij} \leftarrow w_{ij} + \frac{\lambda}{2} x_i x_j$$

on retrouve la règle de Hebb. Malheureusement, ce procédé a le défaut que d'autres trous sont involontairement creusés à d'autres endroits, stabilisant des états non désirés. Ce problème est atténué avec les règles utilisant les pseudo-inverses.

2.3 L'apprentissage linéaire

Apprentissage linéaire désigne toutes les méthodes d'apprentissage basées sur l'algèbre linéaire. Ceci est indépendant des neurones utilisés qui, eux, peuvent être non-linéaires.

Nous renvoyons le lecteur à l'appendice A pour un exposé détaillé de ces méthodes.

Nous en donnons ici les idées essentielles. La condition de stabilité d'un réseau rebouclé est la suivante

$$X(t+1) = X(t)$$

soit

$$X = Sign[W X]$$

si nous voulons stocker dans le réseau les configurations $X^1, X^2, \ldots, X^p$ nous devons calculer la matrice W qui satisfait le système

$$X^k = Sign[W X^k] \qquad k = 1 \ldots p$$

Ce système est en général très complexe à résoudre. Nous préfèrerons en résoudre une version LINÉARISÉE (d'où le nom d'apprentissage linéaire)

$$X^k = W X^k \qquad k = 1 \ldots p$$

Cette condition est plus forte que la précédente, mais plus facilement exploitable. Si l'n note X la matrice dont les colonnes sont les X^k, le système précédent devient

$$X = WX$$

La solution générale de cette équation est

$$W = XX^+ + Z(I - XX^+)$$

où X^+ est la pseudo-inverse de X et Z est une matrice quelconque. en utilisant la notation $W_0 = XX^+$ l'équation précédente devient

$$W = W_0 + (I - W_0)$$

La matrice W_0 est la matrice de projection sur le sous espace engendré par les X^k. Lorsque les vecteurs X^k ne sont pas linéairement indépendants, il n'existe pas de solution exacte au système. La formule ci-dessus minimise alors le terme d'erreur

$$|| X - WX ||^2 = \mathrm{Tr}([(X - WX)^T(X - WX)])$$

L'équivalence entre la méthode des pseudo-inverses et les méthodes de minimisations de type Widrow-Hoff est claire.

Lorsque les X^k n'engendrent pas l'espace entier (lorsque le système est sous déterminé), W_0 est la solution de norme minimum. C'est cette solution qui est la meilleure lorsqu'il s'agit d'éliminer un bruit additif de moyenne nulle et décorrélé. Cependant, pour tout autre type de bruit, il convient de choisir une matrice Z adéquate donnant la solution particulière optimale pour le problème.

Chapitre 3

La rétro-propagation

3.1 Généraliser le perceptron

La plupart des modèles décrits au chapitre précédent (à vrai dire tous) sont sujets aux limitations intrinsèques des neurones formels: ils ne peuvent résoudre, en pratique, que des problèmes linéairement séparables. Ces limitations ont été clairement mises en évidence dès les premiers travaux sur le perceptron et sur les ADALINES de Widrow [Widrow & Hoff 60]. De nombreux algorithmes d'apprentissage ont été proposés au début des années 1960 pour tenter de les lever. Bien que certaines de ces tentatives se soient avérées fructueuses pour des applications particulières, aucune n'a atteint le degré de généralité suffisant. Les méthodes proposées à l'époque, ainsi que les récentes, sont fort variées, mais reposent toutes sur l'utilisation d'un réseau à plusieurs étages de traitement, plusieurs couches d'unités dont les poids sont modifiables, comportant éventuellement des boucles dans leur graphe de connexions. Il a été montré plus haut que ce type d'architecture pouvait calculer une large classe de fonctions, en fait, toutes les fonctions booléennes, et même les fonctions vectorielles réelles ainsi que le montrera la section 3.12. Il est clair que dans ce type d'architecture, les cellules des couches internes ne sont pas en relation directe avec l'extérieur, leur état n'est déterminé qu'indirectement par l'intermédiaire d'autres cellules, on les appelle pour cette raison les cellules cachées [Hinton & Sejnowski 83] [1]. Dans cette situation, l'apprentissage est beaucoup plus complexe, car lorsqu'une réponse erronée est produite par le réseau, il est difficile de savoir quelle(s) cellule(s) est (sont) en cause, et comment modifier ses (leurs) poids. En effet, seules les cellules de sortie disposent d'une information directement exploitable pour modifier leurs poids, aucun signal externe ne spécifie directement si les états des cellules cachées sont corrects ou non. Ce problème classique dans l'apprentissage est appelé "credit assignment problem" dans la littérature anglo-saxonne, ce que nous pouvons traduire par "problème du bouc-émissaire". Hinton qualifie les algorithmes avec et sans cellules cachées respectivement d'apprentissage difficile et facile. La distinction est justifiée par le fait que lorsqu'il y a des cellules cachées, la procédure d'apprentissage doit trouver des représentations internes adéquates en décidant des fonctions que doivent réaliser ces cellules cachées pour assurer un comportement correct en sortie. Ce problème de la génération de bonnes représentations internes, est un problème général pour lequel nous ne pensons pas qu'il existe de solution miracle. Cependant nous pouvons espérer simplifier la tâche de l'apprentissage en choisissant une architecture de réseau adaptée au problème. C'est une manière de mettre de la connaissance a priori pour orienter l'apprentissage et le

[1] Par analogie avec les "variables cachées" de la physique

contraindre à choisir un certain type de représentation.

Bien sûr, il est possible de généraliser le paradigme du classifieur linéaire différemment. Nous en avons déjà fait mention au chapitre 1 en citant les "Φ-machines", les classifieurs utilisant des surfaces séparatrices d'ordre supérieur à 1. Par exemple les cellules à base de fonction discriminante quadratique. La sortie d'une cellule quadratique (ou cellule sigma-pi) est donnée par l'équation

$$s = f(a) \quad \text{avec} \quad a = \sum_{i,j=0}^{n} w_{ij} e_i e_j$$

l'entrée e_0 est supposée être identiquement égale à -1. Si f est la fonction signe, la surface séparatrice associée à cette cellule est d'ordre 2 (hypersphère, paraboloïde, hyperboloïde, ellipsoide,...). Les possibilités sont évidemment supérieures à celles d'un classifieur linéaire. Le défaut de cette démarche est que le nombre de poids est directement dépendant de la dimension des entrées (il est égal au carré de celle-ci), nous ne pouvons donc pas facilement le contrôler et/ou le modifier. Nous n'avons pas le loisir de choisir une structure adaptée au problème à résoudre, contrairement aux réseaux multi-couches. Par contre, les algorithmes d'apprentissage sont extrêmement simples: ce sont les mêmes que pour une cellule linéaire.

Il est surprenant de constater que le renouveau d'intérêt pour les systèmes connexionnistes a été suscité par des travaux [Kohonen 84a] [Hopfield 82] qui ne levaient pas les limitations ayant conduit à leur abandon quinze ans auparavant. L'apport de concepts nouveaux, bien que limités, a suffi à créer un nouvel élan. Il semble toutefois que ce regain d'intérêt puisse être attribué à la conjugaison de quatre causes: premièrement, la possibilité donnée par les récents développements de la technologie de faire aboutir les idées (même simples) au niveau de l'application; deuxièmement, l'apport des méthodes issues de la physique statistique et l'analogie des réseaux de neurones avec les verres de spins; troisièmement, l'oubli dont ont été frappés les premiers travaux et leur échec; quatrièmement, la relative inefficacité des méthodes "classiques" de l'IA pour toute une classe de problèmes, comme la perception, le contrôle moteur, et d'une manière générale, les problèmes nécessitant une solution en temps réel. Ce n'est que récemment que des algorithmes d'apprentissage réellement nouveaux permettant l'utilisation de cellules cachées ont été proposés. L'objet de ce chapitre est de les présenter, en s'intéressant plus particulièrement à l'algorithme de rétro-propagation proposé par l'auteur et par deux autres équipes indépendamment.

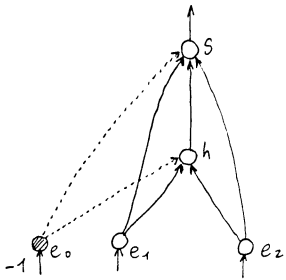


Figure 3. 1: Réseau minimal pour calculer le XOR

3.1.1 quelques exemples classiques

Comme nous l'avons signalé au chapitre 1, la plus simple des fonctions booléennes non-linéairement séparables est le OU EXCLUSIF (XOR) (ou l'EQUIVALENCE). Cette fonction peut être utilisée à titre illustratif. Le réseau minimal permettant de calculer le XOR avec des automates à seuil est représenté sur la figure 3. 1. Ce réseau contient deux cellules d'entrée e_1 et e_2, une cellule cachée h, et une cellule de sortie s. Les fonctions réalisées par s et h peuvent être choisies de plusieurs manières, par exemple h peut réaliser un ET-NON des deux entrées, et s réaliser la fonction (e_1 OU e_2) ET h. Ces deux fonctions sont évidemment linéairement séparables Ceci conduit à une analogie intéressante. Un algorithme d'apprentissage supposé calculer les poids de ce réseau pour réaliser le XOR se trouve, d'une certaine manière, dans la même situation qu'un concepteur de circuit logique devant décomposer en portes logiques élémentaires (e.g: en portes NAND) une fonction booléenne combinatoire définie par sa table de vérité. Pour une fonction donnée, il existe de nombreuses décompositions possibles. Le choix d'une décomposition s'effectue sur des critères de vitesse (minimisation du nombre de "couches logiques" traversées), ou d'encombrement (nombre d'éléments), les deux critères étant évidemment contradictoires. Il est intéressant de noter ici qu'il n'existe pas de méthode générale et automatique pour réaliser ce travail (autre que l'exploration exhaustive) [2]. Le choix de la structure optimale est même souvent rendu

[2]si ce n'était pas le cas, l'architecture des ordinateurs n'existerait pas en tant que problème technique, et cette thèse n'aurait pas d'objet

plus difficile par la présence de "trous" dans la table de vérité servant de définition, de combinaisons d'entrées pour lesquelles la valeur de sortie est indifférente. On en est donc réduit, même pour de "petites" fonctions, à employer des méthodes heuristiques, ou à se limiter à des structures particulières. C'est le cas des programmes de minimisation de PLA (Programmable Logic Array) qui tentent de réduire les fonctions booléennes multi-dimensionnelles sous forme normale conjonctive (OU de ET) en minimisant le nombre de monômes.

Revenons à l'exemple du XOR. Pendant l'apprentissage, le superviseur n'est pas censé connaître la structure interne du réseau, il est par conséquent hors de question qu'il puisse spécifier l'état désiré de la cellule cachée pour chaque configuration d'entrée. La seule information qu'il peut mettre à disposition est l'état idéal de la cellule de sortie s. La procédure d'apprentissage doit donc être en mesure de générer un "état désiré" pour la cellule cachée à partir de cette seule information.

Une idée communément admise est que les systèmes rebouclés du type de ceux du chapitre précédent ne sont pas concernés par les limitations de la séparabilité linéaire. C'est malheureusement faux dans la plupart des cas. Bien sûr, les possibilités de calcul théoriques d'un réseau rebouclé totalement connecté sont celles des automates finis, ce qui est plus que suffisant. Mais un réseau réalisant un automate quelconque à n entrées et m sorties doit en général comporter un nombre de cellules bien supérieur à $n+m$. Certaines cellules ne sont ni des cellules d'entrée, ni des cellules de sortie, à la manière de la cellule cachée de l'exemple précédent.

C'est pourquoi les règles de calcul des matrices de pondérations décrites au chapitre 2 sont loin de permettre d'atteindre cette limite. La raison essentielle est que lors de l'apprentissage, les états de TOUTES les cellules du réseau sont déterminés de l'extérieur. Il en résulte que certaines relations entre variables externes ne sont pas réalisables. En réalité, seules les relations linéairement séparables le sont. La situation est claire sur l'exemple suivant. Nous disposons d'un réseau totalement connecté comportant un nombre quelconque de cellules à seuil (disons 10000 cellules). Nous voulons stocker dans le réseau (i.e.: rendre des points stables de la dynamique) les quatre séquences binaires suivantes (on peut remplacer les 0 par des -1):

$$\begin{array}{cccccccc} 0 & 0 & 0 & x & x & \dots & x \\ 0 & 1 & 1 & x & x & \dots & x \\ 1 & 0 & 1 & x & x & \dots & x \\ 1 & 1 & 0 & x & x & \dots & x \end{array}$$

ici, $xx\dots x$ représente une suite quelconque de 9996 bits, la même suite pour les quatre séquences. Le résultat est le suivant: il n'est pas possible de stocker ces quatre

séquences et uniquement celles-ci. Le fait de rendre stable ces quatre séquences aura pour conséquence d'en rendre stable au moins quatre autres. En effet, les trois premiers bits des quatre séquences constituent la table de vérité du XOR. Dû à la réversibilité de cette fonction, chacun des trois premiers bits est un XOR des deux autres. Aucune information discriminante ne pouvant être tirée des 9996 bits restants, il est impossible de stocker ces 4 séquences sauf si l'on abandonne toute relation de dépendance entre les trois premiers bits. Dans ce cas les 8 configurations sur ces 3 bits seront stables, ce qui ajoute quatre état stables non désirés. Ceci revient à dire que la sous-matrice 3 par 3 des poids reliant les trois premières cellules est nécessairement positive diagonale [3]. Il est important de noter que ce résultat est indépendant de tout procédé de calcul de la matrice de poids. Les quatre autres configurations stables sont donc:

$$\begin{array}{cccccccc} 0 & 0 & 1 & x & x & \dots & x \\ 0 & 1 & 0 & x & x & \dots & x \\ 1 & 0 & 0 & x & x & \dots & x \\ 1 & 1 & 1 & x & x & \dots & x \end{array}$$

elles correspondent à l'opposé des séquences originales et représentent la table de vérité (elle aussi réversible) de l'EQUIVALENCE. En situation réelle, avec des séquences relativement indépendantes, ce genre de configuration a une probabilité faible de se produire, le cas qui vient d'être décrit est relativement pathologique. La question qu'il convient maintenant de se poser est: comment stocker les quatre configurations originales sans en stocker d'autres? Il suffit d'une seule cellule cachée pour résoudre le problème. Avec cette cellule supplémentaire, les quatre séquences deviennent "stockables":

$$\begin{array}{cccccccc} 1 & 0 & 0 & 0 & x & x & \dots & x \\ 0 & 0 & 1 & 1 & x & x & \dots & x \\ 0 & 1 & 0 & 1 & x & x & \dots & x \\ 0 & 1 & 1 & 0 & x & x & \dots & x \end{array}$$

la première cellule est la cellule supplémentaire (cachée). dans ces quatre séquences, chacun des quatre premier bits est une fonction linéairement séparables des trois autres. Il est ainsi possible de stocker ces séquences sans en rendre d'autres stables [4]. Malheureusement, l'état de la cellule supplémentaire ne fait pas réellement partie de la forme à stocker, il doit donc être généré par le réseau lui-même, et non pas spécifié

[3] éventuellement nulle

[4] à condition d'appliquer une itération séquentielle par blocs, le premier bloc contenant les cellules visibles, le second la cellule cachée

de l'extérieur. On voit que dans ce cas, l'état du réseau ne doit pas être totalement défini par le monde extérieur.

3.1.2 contenu du chapitre

Le chapitre se décomposera de la manière suivante. Nous commencerons par passer en revue les premières tentatives d'algorithmes d'apprentissage pour réseaux à cellules cachées. Puis nous décrirons rapidement l'algorithme de la machine de Boltzmann [Hinton & Sejnowski 83] [Hinton & Al. 84c] [Ackley & al. 85] [Sejnowski & Hinton 85] [Sejnowski & al. 85]: un des premiers algorithmes pour réseaux à cellules cachées ayant remporté un certain succès. Viendra ensuite la dérivation de l'algorithme de rétro-propagation proposé sous diverses formes dans [le Cun 85] [le Cun 86] [Parker 85] [Rumelhart & al. 86a], et étudié par de nombreux auteurs [le Cun & Fogelman 87a] [Fogelman & al. 87b] [Gallinari & al. 87] [Rumelhart & al. 86b] [Plaut & al. 86] [Sejnowski & Rosenberg 86]. Cette présentation sera illustrée de quelques simulations. Les principaux problèmes relatifs à l'algorithme de rétro-propagation seront abordés, en particulier ceux qui concernent la convergence. Nous aborderons également le problème de la généralisation qui, il ne faut pas le perdre de vue, est le problème principal de l'apprentissage. Plusieurs versions de l'algorithme de base existent. Nous présenterons ici l'algorithme HLM, très proche de la rétro-propagation "pure", et développé par l'auteur antérieurement à celle-ci [le Cun 84] [le Cun 85] [le Cun 86], cette procédure a été améliorée et modifiée par Plaut, Nowlan et Hinton [Plaut & al. 86]. Une dérivation originale de la rétro-propagation sera présentée qui utilise le formalisme Lagrangien. Il sera mis l'accent sur l'analogie entre cette formulation du problème et la théorie de la commande optimale. Diverses généralisations de la RP seront présentées, elles concerneront l'utilisation d'autres types de fonctions que les classiques "neurones" formels. Nous aborderons le problème des réseaux à boucles ou pseudo-boucles. Nous donnerons ensuite un théorème spécifiant que toute fonction vectorielle réelle définie sur un compact peut être approximée avec une précision arbitraire à l'aide d'un réseau à 3 couches. Puis nous décrirons des simulations effectuées sur des problèmes réels, en particulier un problème de diagnostic médical. Enfin nous passerons en revue les problèmes ouverts concernant les méthodes connexionnistes d'apprentissage.

3.2 Les premières idées

L'objet de cette section est de passer en revue les premières idées de procédures d'apprentissage pour réseaux en couches. Bien que ces tentatives n'aient pas toujours

été couronnées de succès, il semble utile de les présenter, ne serait-ce que pour profiter de leurs erreurs et pour mettre en évidence leur valeur heuristique.

Il existe une possibilité pour contourner le problème du bouc-émissaire: c'est d'abandonner le caractère supervisé de l'apprentissage. En effet lorsqu'il n'y a plus de sortie désirée, il n'est plus nécessaire de décider du bien-fondé de la réponse d'une cellule cachée. Un autre critère que l'erreur en sortie peut être alors utilisé. Il peut être de plusieurs natures. Ce peut être un critère supposé universel comme le flot d'information traversant le réseau, ou une mesure d'entropie [Pearlmutter & Hinton 86]. Ce peut être également un principe régulateur permettant de "répartir" le traitement parmi les cellules cachées. On qualifie généralement ce dernier type de modèle D'APPRENTISSAGE COMPÉTITIF. Il en existe de nombreuses variantes, mais l'idée de base qui leur est commune consiste à ne faire prendre part à l'apprentissage que les cellules dont le niveau d'activité est le plus élevé à un instant donné et dans un voisinage donné; il existe donc une sorte de "compétition" entre cellules pour avoir le droit de modifier leurs poids. De nombreux modèles d'apprentissage compétitif ont été proposés dans la littérature [Fukushima 75], [Fukushima & Miyake 78], [Miyake & Fukushima 84] [Rumelhart & Zipser 85], certains se sont même avérés extêmement efficaces pour des applications comme la segmentation des signaux de parole, et en sont au stade de la commercialisation [Kohonen 82], [Kohonen 84a], [Kohonen 84b].

D'autres règles non-supervisées utilisent simplement la règle de Hebb: le poids est proportionnel à la covariance des activités pré- et post-synaptiques. C'est une vieille idée [Block & al. 62] (article de Rosenblatt dans [Yovits & Cameron 60]) qui a été maintes fois reprise (réinventée)[Nakano 71][Kohonen 72][Amari 72][Hopfield 82], mais qui a le désavantage d'être totalement inefficace lorsqu'on l'applique aux cellules cachées: on aboutit en général à des situations de blocage où certaines cellules cachées se bloquent dans un état particulier.

Le problème de l'apprentissage non-supervisé est qu'il repose entièrement sur l'état initial des paramètres et, surtout, sur le critère choisi qui "supervise implicitement" le processus. Il est donc difficile de contrôler ce qui est appris, et par conséquent difficile de l'exploiter.

3.2.1 Multiple ADALINE et Comittee Machines

L'une des mieux étudiées des premières procédures d'apprentissage (supervisée) pour réseaux multi-couche est sans doute celle des MADALINES (pour Multiple ADAptive LINear Element) de Widrow, étudiée par [Nilsson 65] sous le nom de Comittee Machine. Un MADALINE (voir figure 3. 2) est constitué d'une couche d'éléments à seuil

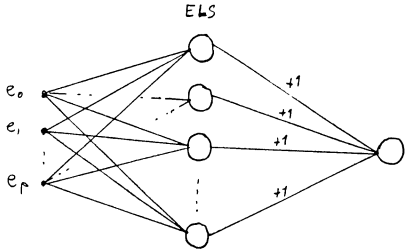


Figure 3. 2: Un "multiple adaline" ou "comittee machine"

classiques recevant tous le même vecteur d'entrée, et n'étant pas connectés entre eux. Tous les éléments de cette couche sont connectés à un élément de sortie à travers des poids fixes ayant tous la valeur 1. l'élément de sortie effectue simplement une décision de type vote majoritaire (d'où le nom comittee machine). Seuls les poids des cellules de la première couche sont modifiables par apprentissage, c'est un peu le contraire du perceptron. On utilise la règle suivante: lorsque la réponse produite par le réseau est correcte, aucune modification n'est effectuée; lorsque la réponse est erronée, on modifie les poids de certaines cellules choisies en nombre juste suffisant pour faire basculer le vote. Par exemple, si le nombre total de cellules est 9, et que 3 cellules ont donné la réponse correcte (et 6 la mauvaise réponse), il faudra modifier les poids de 2 des 6 "mauvaises" cellules. Le choix des 2 cellules à modifier (sur les 6 possibles) s'effectuera en cherchant à minimiser le changement total des poids conduisant à la réponse correcte. Il suffira donc de choisir les 2 cellules dont les sommes pondérées des entrées sont les plus proches de 0. La modification des poids des 2 cellules choisies est alors effectuée à l'aide de l'une des règles classiques (par exemple celle de Widrow et Hoff ou celle du perceptron) en utilisant la sortie désirée globale. Cette procédure se situe à mi-chemin entre les règles supervisées et l'apprentissage compétitif puisque seules certaines cellules participent à l'apprentissage à chaque instant, et qu'il est nécessaire de COMPARER les niveaux d'activités des cellules pour décider de la modification de leurs poids. La règle heuristique générale utilisée est ici, on le voit clairement, de toujours choisir la MODIFICATION MINIMUM des poids qui satisfait les contraintes du

problème. Autrement dit, entre deux possibilités de modification des poids conduisant à un effet équivalent sur la sortie, il faut choisir celle qui modifie le moins l'état actuel des poids. L'argument en faveur de cette règle DU CHANGEMENT MINIMUM pourrait être que l'arrivée d'un nouvel élément d'information doit perturber le moins possible le système de manière à ce qu'il oublie le moins possible ce qui a déjà été appris. Nous verrons dans la suite que cette règle heuristique peut être mise à profit dans d'autres situations.

Il n'existe pas de démonstration de convergence de la procédure qui vient d'être décrite, mais ses propriétés sont très bien détaillées dans [Nilsson 65]. On peut montrer, en particulier, que la norme du vecteur de poids est majorée. Widrow l'a utilisée au début des années 60 pour la reconnaissance de la parole (mots isolés) avec, semble-t-il, un certain succès. Une machine analogique a même été spécialement construite pour cette application [5]. Ces travaux ont été brusquement interrompus en 1963 lorsque B. Widrow a réorienté les objectifs de son laboratoire vers le traitement adaptatif du signal. Ce changement de cap a été commun à de nombreux laboratoires travaillant sur le sujet à l'époque.

3.2.2 Quelques autres idées

Diverses techniques ont été utilisées par la suite pour l'apprentissage dans les réseaux multicouches. La plus simple d'entre elles consiste à envoyer la même sortie désirée (ou le même signal d'erreur) à toutes les cellules du réseau. Ceci ne peut se concevoir que dans un problème à deux classes avec un réseau comportant une seule cellule de sortie. Dans la cas d'un problème multi-classe, il est possible de décider a-priori d'affecter chacune des sorties désirées à un groupe de cellules différent. Bobrowski étudie les propriétés de convergence de cette procédure [Bobrowski 78] [Bobrowski 81], il en donne une preuve de convergence: il prouve que le processus se stabilise, malheureusement, il n'est pas précisé si le point de convergence constitue une solution intéressante. Il semble que cela ne soit pas le cas.

Andrew [Andrew 65] propose l'idée du "significance feedback" dont on peut dire qu'elle est l'ancêtre de la rétro-propagation décrite plus loin. L'idée d'Andrew est la suivante: chaque connexion doit véhiculer deux signaux se propageant en sens inverses, l'un est le signal classique antérograde, et l'autre est un signal rétrograde représentatif de la sensibilité de la sortie globale par rapport à une variation du signal antérograde. Il espère ainsi pouvoir modifier les paramètres des cellules internes. Malheureusement

[5] discours de clôture de la conférence Neural Network for Computing, Snowbird, Utah, Avril 1987

aucun détail n'est donné sur l'algorithme tant du point de vue théorique que pratique. Andrew ne donne pas d'équation mais raisonne sur des cas particuliers. Il fait référence à des résultats préliminaires de simulation sans les détailler. Il est également partisan de la règle du changement minimum.

3.2.3 Apprentissage compétitif

Les modèles d'apprentissage compétitifs les plus connus sont sans doute ceux de Fukushima [Fukushima 75][Fukushima & Miyake 78]Miyake & Fukushima 84] et de Kohonen [Kohonen 82][Kohonen 84a][Kohonen 84b]. Nous décrivons rapidement le premier, et renvoyons à l'abondante littérature pour le second.

Les cognitrons et néo-cognitrons

Dans la plus pure tradition, les modèles proposés par Fukushima allongent la liste des modèles en "-tron". Les préoccupations des auteurs sont doubles, d'une part proposer un modèle fonctionnel des premières étapes des systèmes visuels animaux, et d'autre part exploiter ces idées pour réaliser une machine destinée à la reconnaissance des caractères. Le Cognitron utilise un modèle de neurone assez particulier et complexe que nous ne détaillerons pas. Ces cellules sont arrangées en une série de couches bidimensionnelles. Les connexions relient exclusivement les cellules d'une couche à celles de la couche suivante, c'est donc un réseau combinatoire [6]. Le schéma d'interconnexions est local en dimension deux, avec quelques connexions à longue distance réparties aléatoirement. La règle d'apprentissage est la suivante: lorsqu'une forme est présentée sur la couche d'entrée, les états de toutes les cellules sont calculés par propagation dans les couches, ensuite, le niveau d'activité de chaque cellule est comparé aux niveaux d'activité d'autres cellules contenues dans un voisinage à l'intérieur de la même couche. La cellule la plus active dans une région aura ses poids renforcés, les autres cellules de la région auront leurs poids diminués. Une connexion est renforcée ou diminuée proportionnellement au niveau d'activité de la cellule amont: c'est une version compétitive de la règle de Hebb. Selon les auteurs, ce processus développe la sélectivité du système et la spécificité de chaque cellule. Ainsi, des formes ressemblantes produiront des réponses approximativement identiques en sortie. La représentation générée par le système est de type local, peu de cellules d'une même couche sont actives en même temps, en général, une seule par région.

[6] d'autres publications de Fukushima et Miyake décrivent le "feed-back type cognitron" dans lequel la sortie (la dernière couche) est réinjectée sur l'entrée (la première couche)

Malheureusement, la notion de "ressemblance" entre images qui est extraite n'est pas facilement contrôlable. Elle dépend fortement de la configuration initiale des poids, des exemples présentés, et surtout, de l'architecture des connexions. Ainsi, un schéma d'interconnexions local tend à favoriser l'extraction de primitives locales, ce qui est avantageux pour des images, mais pas nécessairement intéressant pour d'autres types de données. Dans les derniers travaux, les auteurs se sont intéressés au problème de l'invariance par translation. Le modèle qu'ils proposent pour ce faire (le néo-cognitron) résoud le problème par "la force brute". L'idée est de dupliquer les premières couches du réseau de manière à ce qu'à chaque position possible d'une forme sur la rétine corresponde un morceau de réseau particulier. A ceci est ajouté un ensemble de cellules "pré-cablées" dont le rôle est d'extraire des primitives élémentaires. Ce cablage "à la main" de l'invariance par translation présente l'inconvénient de fournir un réseau de taille très importante, dans lequel le rôle de l'apprentissage est très réduit. La mise en oeuvre pratique de ce réseau semble de plus être totalement assujettie au bon choix d'un grand nombre de paramètres, qui, s'ils n'ont pas la bonne valeur, interdisent au système de fonctionner. Cette démarche semble très "ad-hoc".

3.3 La machine de Boltzmann

La machine de Boltzmann est un modèle très original présenté et décrit dans [Hinton & Sejnowski 83], [Hinton & Al. 84c], [Ackley & al. 85], [Sejnowski & Hinton 85], [Sejnowski & al. 85]. Nous le présentons brièvement, et proposons une nouvelle méthode de dérivation de la règle d'apprentissage.

3.3.1 Le modèle

Une machine de Boltzmann est un réseau d'éléments binaires non déterministes. Ces éléments peuvent prendre les états 0 et 1 avec la loi de probabilité suivante

$$\Pr(x_i = 1) = \frac{1}{1 + \exp -\beta a_i}$$

avec

$$a_i = \sum_j w_{ij} x_j$$

et β une constante assimilable à l'inverse d'une température. Lorsque β tend vers l'infini (température tendant vers 0) le système devient déterministe. Une cellule est dans l'état 1 avec une probabilité d'autant plus élevée que la somme pondérée de ses entrées a_i est importante. La fonction de probabilité est une fonction sigmoïde

croissante de 0 vers 1, c'est la distribution de Fermi-Dirac pour un système à deux états.

L'état de certains éléments peut être forcé de l'extérieur, ce sont les éléments d'entrée et de sortie. Les éléments de sortie sont en fait également des éléments d'entrée. Il peut ne pas y avoir d'éléments d'entrée purs.

Les éléments sont interconnectés à l'aide d'une matrice de pondérations SYMÉTRIQUE pour assurer la convergence du système vers des distributions d'états stationnaires.

Lors de l'apprentissage, le système fonctionne selon deux modes différents. Dans le premier mode, les états des cellules d'entrée sont fixés de l'extérieur, et l'on laisse le système "relaxer" vers une distribution d'équilibre, éventuellement accompagné d'un processus de "refroidissement", c'est à dire d'augmentation progressive de β. Ce procédé de recuit simulé permet de se rapprocher de minima absolus de la fonction énergie associée. Cette fonction énergie est la fonction de Hopfield [Hopfield 82]:

$$H(X,W) = -\frac{1}{2} X^T W X$$

L'utilisation d'une température non-nulle permet d'échapper aux minima locaux de l'énergie. La distribution d'équilibre obtenue ("l'équilibre thermique") est une distribution de Gibbs, le rapport des probabilités d'apparition de deux états est fonction de la différence d'énergie entre les deux états

$$\frac{\Pr(X^1)}{\Pr(X^2)} = \exp \beta(H(X^1) - H(X^2))$$

Dans le second mode, les états des cellules de sortie sont également forcés à leur valeur désirée, en plus des cellules d'entrée. Et l'on applique la procédure de recuit pour obtenir une distribution d'états stationnaire. Dans la suite, nous noterons Y l'état obtenu au cours de ce dernier mode.

La procédure d'apprentissage consiste à modifier les poids de manière à ce que la distribution obtenue avec le premier mode soit égale à celle obtenue avec le second. Les auteurs proposent une mesure de dissemblance entre ces deux distributions qu'ils minimisent à l'aide d'une procédure de gradient. La mesure utilisée est

$$G(W) = \sum_{\sigma} P(V_\sigma) \log \frac{P(V_\sigma)}{P'(V_\sigma)}$$

où $P(V_\sigma)$ est la probabilité que les cellules externes (d'entrée et de sortie) soient dans l'état σ lorsque cet état est fixé de l'extérieur (dans le deuxième mode), et $P'(V_\sigma)$ la probabilité que ces même cellules soient dans l'état σ lorsque seules les cellules d'entrée

sont fixées de l'extérieur. Un calcul assez complexe donne les dérivées partielles de G par rapport aux poids

$$\frac{\partial G(W)}{\partial w_{ij}} = -\beta(\mathcal{M}[y_i y_j] - \mathcal{M}[x_i x_j])$$

où $\mathcal{M}[.]$ est un opérateur de moyennage. l'expression $\mathcal{M}[y_i y_j]$ est la probabilité que les cellules i et j soient actives simultanément lorsque les sorties sont fixées de l'extérieur, et $\mathcal{M}[x_i x_j]$ cette même probabilité lorsque seules les cellules d'entrée le sont.

3.3.2 Une autre méthode

Au lieu de minimiser une mesure des probabilités des états, plutôt complexe à interpréter, nous allons utiliser une mesure d'énergie. La mesure que nous décidons de minimiser est la suivante

$$G'(W) = \frac{1}{2}(\mathcal{M}[X^T W X] - \mathcal{M}[Y^T W Y])$$

dont les dérivées partielles sont données par

$$\frac{\partial G'(W)}{\partial w_{ij}} = -(\mathcal{M}[y_i y_j] - \mathcal{M}[x_i x_j])$$

nous obtenons la même expression que précédemment au facteur β près [7]. L'interprétation de G' est simple: c'est la différence d'énergie moyenne entre le réseau libre et le réseau contraint par l'extérieur. L'énergie moyenne du second est, en général, supérieure à celle du premier, les contraintes externes ayant pour effet d'interdir l'accès aux niveaux d'énergie les plus faibles. Minimiser G' tend à rendre identiques les paysages d'énergie correspondant aux deux modes. Les paysages d'énergie étant égaux, les comportements le seront. Le réseau tendra en particulier à présenter les même configurations sur les cellules de sortie, que celles-ci soient déterminées de l'extérieur ou non. C'était le but recherché.

3.3.3 Conclusion

L'intérêt (éphémère) porté aux machines de Boltzmann a été suscité par les physiciens et à cause de l'analogie formelle avec les modèles des chaines de Markov cachées si largement étudiées pour la reconnaissance de la parole. Le défaut principal de la machine de Boltzmann est sa lenteur et la nécessité d'attendre "l'équilibre thermique"

[7] que l'on pourrait artificiellement introduire dans l'expression de G'

avant chaque modification des poids. Ceci a suscité des travaux visant à l'intégration de l'algorithme sur silicium [8]

3.4 La rétro-propagation: première dérivation

3.4.1 Introduction

La rétro-propagation est certainement l'un des plus simples et des plus efficaces algorithmes d'apprentissage pour réseau à cellules cachées. Bien que son apparition soit récente, il a fait l'objet de très nombreuses publications. L'idée de la rétro-propagation (RP) a été présentée dans [le Cun 85] et dans [Parker 85], mais l'article de référence le plus clair et le plus connu est [Rumelhart & al. 86a]. Il est important de noter que ces trois publications sont indépendantes, bien que les deux premières soient citées dans la troisième.

La RP est essentiellement une généralisation de la règle de Widrow et Hoff à une fonction de coût non-linéaire. On utilise des cellules dont la transformation non-linéaire est une fonction dérivable, ces cellules sont organisées en couches. L'algorithme permet de calculer rapidement les dérivées partielles de l'erreur en sortie par rapport à tous les poids du réseau, y compris ceux des cellules cachées. Il suffit ensuite d'appliquer une procédure de gradient pour minimiser, en fonction de tous les poids, un critère d'erreur sur les cellules de sortie. Du point de vue mathématique, cette méthode est une simple application de la règle de dérivation des fonctions composées. Il est étonnant qu'une idée aussi simple ne soit apparue que si récemment.

3.4.2 Les réseaux en couches

Considérons un réseau composé de trois types de cellules: les cellules d'entrée dont l'état est fixé de l'extérieur, les cellules de sortie, et les cellules cachées sans interaction directe avec le monde extérieur. Pour simplifier dans un premier temps, nous considérerons que ce réseau est combinatoire, c'est à dire sans boucle. Il est ainsi possible de définir un ordre de mise à jour des cellules (un mode d'itération séquentiel ou séquentiel par blocs) de telle manière qu'aucune cellule ne soit mise à jour sans que toutes les cellules lui envoyant leurs sorties ne l'aient été avant elle. Ainsi, l'état de toutes les cellules peut être calculé en une seule passe, par propagation. Les connexions d'un réseau combinatoire constituent un graphe sans circuit, la matrice des pondérations est donc isomorphe à une matrice triangulaire (à une permutation des lignes et des

[8] travaux de D Alspector, Communication à "Neural Network for Computing", Snowbird, Utah, 1987

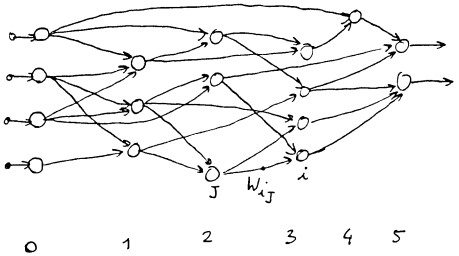


Figure 3. 3: Un réseau en couches, les couches sont numérotées par ordre croissant de l'entrée vers la sortie. le poids w_{ij} relie la cellule j à la cellule i.

colonnes près). Il est souvent commode de voir un réseau combinatoire comme une succession de couches de cellules. Les cellules à l'intérieur d'une même couche ne sont pas connectées entre elles, les seules connexions possibles vont des couches inférieures (les plus proches des entrées) vers les couches supérieures. Les connexions peuvent évidemment sauter une ou plusieurs couches (voir fig 3. 3). Après présentation d'une forme d'entrée par le forçage des états des cellules de la première couche, les couches sont mises à jour "dans l'ordre".

Dans le réseau, la cellule d'indice i produit une sortie notée x_i en effectuant une somme pondérée des sorties des cellules auxquelles elle est connectée, et en transformant cette somme, notée a_i, par une fonction non-linéaire dérivable f. La fonction f est typiquement une fonction sigmoïde (voir chapitre 1), mais tout autre choix est valable. Nous avons donc

$$x_i = f(a_i) \qquad a_i = \sum_{j=0}^{n} w_{ij} x_j \tag{3.66}$$

w_{ij} désigne évidemment le poids reliant la cellule j à la cellule i, w_{ij} est nul s'il n'existe pas de connexion entre les cellules j et i.

3.4.3 L'apprentissage

Une session d'apprentissage se déroule de la manière suivante. Les paires de vecteurs d'entrée et de sortie désirée sont présentées séquentiellement au réseau. La présentation

d'un vecteur d'entrée E_h s'effectue en forçant l'état des cellules d'entrée. Un vecteur de sortie désirée D_h est, quant à lui, présenté sur les cellules de sortie selon une méthode décrite plus loin. Naturellement, aucune information n'est donnée de l'extérieur concernant l'état des cellules appartenant aux couches intermédiaires.

Définition de la fonction de coût

A l'instar des règles d'apprentissage simples, il s'agira ici de trouver une configuration de poids qui minimise un certain critère. En l'occurence, ce critère ne doit directement dépendre QUE des états des cellules de sortie, et pas des cellules cachées. Pour un couple de formes entrée-sortie d'indice h: (E_h, D_h), nous supposons qu'à chaque cellule de sortie d'indice q est associée une sortie désirée d_{qh} ou, au pire, un signal d'erreur ϵ_{qh}. Nous pouvons choisir de minimiser le désormais classique critère quadratique (mais tout autre choix serait valable):

$$C(W) = \mathcal{M}[C_h(W)] = \mathcal{M}[\sum_{q \in U_s} (d_{qh} - x_{qh}(W))^2]$$

ici U_s est l'ensemble des cellules de sortie, x_{qh} est l'état de la cellule de sortie d'indice q, d_{qh} est l'état désiré de la cellule de sortie d'indice q. $\mathcal{M}[.]$ est l'opérateur habituel de moyennage qui peut être une somme normalisé sur les exemples dans le cas déterministe ou une estimation de la moyenne temporelle dans le cas stochastique. On utilise la notation C_h pour la valeur instantanée du coût, correspondant à un seul couple entrée-sortie d'indice h. Dans l'expression de C, les x_{qh} (les états des cellules de sortie) sont considérés comme des fonctions de la matrice de poids complète W, mais pour certains poids, cette dépendance est indirecte. Il est important d'insister sur le fait que la somme n'est effectuée QUE sur l'ensemble U_s des cellules de sortie puisqu'il n'y a pas de sortie désirée associée aux cellules cachées. La quantité élevée au carré dans la définition de la fonction de coût est un signal d'erreur associé à la cellule q que nous noterons de la manière habituelle

$$\epsilon_{qh}(W) = d_{qh} - x_{qh}(W) \tag{3.67}$$

l'expression du coût devient

$$C(W) = \mathcal{M}[C_h(W)] = \mathcal{M}[\sum_{q \in U_s} \epsilon_{qh}^2(W)]$$

Il s'agit maintenant de calculer les dérivées partielles de C par rapport à tous les w_{ij} de manière à être en mesure d'appliquer l'algorithme du gradient:

$$W_{t+1} = W_t - \lambda_t \nabla C(W_t)$$

Pour les poids directement attachés aux cellules de sortie, cette dérivée partielle est très simple à calculer. Pour les autres poids, attachés aux cellules cachées, la dépendance est indirecte. En effet, la relation de dépendance entre l'état d'une cellule de sortie et un poids interne est totalement non-linéaire, et dépend des poids et des états de presque toutes les autres cellules du réseau.

La linéarité de l'opérateur de moyennage nous permettra d'effectuer tous les calculs qui suivent sur les valeurs INSTANTANÉES de la fonction de coût et des autres variables (c'est à dire correspondant à un seul couple entrée-sortie), et de prendre la moyenne du résultat. Cependant, pour ne pas alourdir les notations nous n'écrirons pas l'indice h dans les équations.

Plutôt que de calculer directement les gradients par rapport aux poids, c'est à dire les quantités $\partial C/\partial w_{ij}$, nous préfèrerons calculer les gradients par rapport aux entrées totales a_i, c'est à dire les $\partial C/\partial a_i$. En effet, ceux-ci sont en nombre plus faible et permettent très simplement d'obtenir les gradients par rapport aux poids. En utilisant la définition de a_i:

$$a_i = \sum_{j=0}^{n} w_{ij} x_j$$

et la règle de dérivation des fonctions composées:

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial a_i} \frac{\partial a_i}{\partial w_{ij}}$$

c'est à dire

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial a_i} x_j$$

Calculons donc les $\partial C/\partial a_i$. Dans ce qui suit, nous utiliserons la notation suivante

$$y_i = -\frac{\partial C}{\partial a_i}$$

et nous considérons que toutes les variables utilisées (x, a, C) sont implicitement des fonctions de la matrice de poids W. Cette dépendance n'est pas écrite dans les équations pour ne pas alourdir les notations. Nous devons distinguer deux cas selon que la cellule d'indice i est une cellule de sortie ou non. Pour simplifier les calculs, nous ferons l'hypothèse que les cellules de sortie n'envoient pas leur état à d'autre cellules du réseau.

Les gradients attachés aux cellules de sortie

Si la cellule i est une cellule de sortie, le gradient est donné par

$$y_i = -\frac{\partial C}{\partial a_i} = -\frac{\partial \sum_{q \in U_s} \epsilon_q^2}{\partial a_i}$$

nous pouvons dériver pour obtenir

$$y_i = -\sum_{q \in U_i} \frac{\partial \epsilon_q^2}{\partial x_i} \frac{\partial x_i}{\partial a_i}$$

nous avons fait l'hypothèse que les cellules de sortie n'envoyaient pas leur état à d'autres cellules du réseau, en particulier, elles ne sont pas connectées entre elles, et par conséquent ne sont pas interdépendantes. Il en résulte que la quantité

$$\frac{\partial \epsilon_q^2}{\partial x_i}$$

est nulle si $q \neq i$. Nous en tirons l'identité suivante

$$y_i = -\frac{\partial \epsilon_i^2}{\partial x_i} \frac{\partial x_i}{\partial a_i}$$

soit

$$y_i = -\frac{\partial C}{\partial a_i} = -2\epsilon_i \frac{\partial \epsilon_i}{\partial x_i} \frac{\partial x_i}{\partial a_i} \tag{3.68}$$

d'après la définition 3.67 de ϵ en fonction des cellules de sortie on peut écrire

$$\frac{\partial \epsilon_i}{\partial x_i} = -1$$

ce qui transforme l'équation 3.68 en

$$y_i = -\frac{\partial C}{\partial a_i} = 2\epsilon_i f'(a_i) \tag{3.69}$$

Les gradients attachés aux cellules cachées

Nous allons maintenant dériver les gradients associés aux cellules cachées. Les étapes du calcul, ainsi que les relations de dépendance sont symbolisées sur la figure 3. 4. Nous allons montrer que le gradient attaché à une cellule peut être calculé à partir des gradients attachés à toutes les cellules auxquelles elle envoie sa sortie. Ceci permettra de calculer tous les gradients par propagation de l'erreur des sorties vers les entrées, d'où le nom de l'algorithme. Plaçons-nous dans le cas où il s'agit de calculer le gradient attaché à la cellule cachée d'indice i, c'est à dire la quantité

$$y_i = -\frac{\partial C}{\partial a_i}$$

Nous allons établir une relation de récurrence en supposant que tous les y_k des cellules auxquelles la cellule i envoie sa sortie sont connus au moment où nous désirons calculer

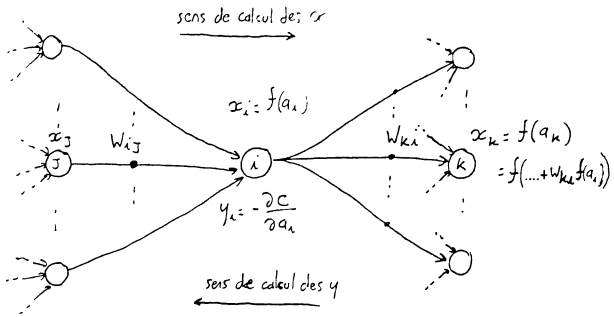


Figure 3. 4: Chaine des dépendances dans un réseau multicouche

y_i. Autrement dit, au moment de calculer y_i, nous faisons l'hypothèse que la proposition suivante est vraie:

$$w_{ki} \neq 0 \quad \Rightarrow \quad y_k \quad \text{connu}$$

en vertu de l'hypothèse de non circularité du graphe de connexion (réseau combinatoire), un ordre de calcul des y_i peut être défini de manière à ce que cette condition soit toujours respectée. On peut, par exemple, choisir l'ordre inverse de l'ordre naturel utilisé pour la mise à jour des états, c'est à dire parcourir les couches en sens inverse de la sortie vers l'entrée. L'expression des gradients est alors obtenue par une simple dérivation en chaîne:

$$y_i = -\frac{\partial C}{\partial a_i} = \sum_{k=0}^{n} -\frac{\partial C}{\partial a_k}\frac{\partial a_k}{\partial a_i} = \sum_{k=0}^{n} y_k \frac{\partial a_k}{\partial a_i}$$

en développant et en s'aidant de la figure 3. 4 pour exprimer $\partial a_k / \partial a_i$, nous obtenons

$$y_i = \sum_{k=0}^{n} y_k \frac{\partial a_k}{\partial x_i}\frac{\partial x_i}{\partial a_i}$$

soit

$$y_i = \sum_{k=0}^{n} y_k w_{ki} f'(a_i)$$

Nous obtenons donc une relation très simple

$$y_i = f'(a_i) \sum_{k=0}^{n} w_{ki} y_k \tag{3.70}$$

dont la ressemblance avec la formule de calcul des x est très claire. Le calcul du gradient attaché à une cellule utilise donc les poids qui la relient aux cellules qui sont

en aval. Ces poids sont utilisés "à l'envers" pour pondérer les gradients des cellules aval. Le calcul des y_i consiste donc en la présentation d'une sortie désirée sur les cellules de sortie, du calcul des gradients attachés à celles-ci à l'aide de la formule 3.69, puis du calcul des gradients des cellules internes en propageant à l'envers (de la sortie vers l'entrée) à l'aide de la formule 3.70. Chaque poids est donc utilisé de deux manières différentes pour les signaux antérogrades et rétrogrades.

Modifier les poids

Après les deux phases de propagation, antérograde pour les états, rétrograde pour les gradients, nous disposons pour chaque cellule de deux quantités: les x_i (ou a_i) et les y_i. Il s'agit maintenant de modifier les poids. L'algorithme du gradient donne la règle de modification classique

$$w_{ij} \leftarrow w_{ij} - \lambda \frac{\partial C}{\partial w_{ij}}$$

nous avons vu plus haut que

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial a_i} \frac{\partial a_i}{\partial w_{ij}}$$

ce que nous pouvons transformer en

$$\frac{\partial C}{\partial w_{ij}} = -y_i x_j$$

Tous les calculs ci-dessus ont été faits sur les valeurs instantanées, bien que l'indice h, n'ait pas été explicitement écrit. En accord avec les notations utilisées, cette équation concerne donc également la valeur instantanée des gradients du coût par rapport aux poids, il convient d'en utiliser la moyenne. La règle de mise à jour des poids est donc tout simplement

$$w_{ij} \leftarrow w_{ij} + \lambda \mathcal{M}[y_{ih} x_{jh}] \tag{3.71}$$

En résumé nous avons obtenu trois équations d'évolution et deux conditions aux bornes. La première équation (3.66) et la première condition aux bornes décrivent l'évolution des états, avec l'état de la couche d'entrée fixée par le vecteur d'entrée E_h

$$x_{ih} = f(a_{ih}) \qquad a_{ih} = \sum_{j=0}^{n} w_{ij} x_{jh}$$

La seconde équation (3.70) décrit l'évolution des gradients y_i

$$y_{ih} = f'(a_{ih}) \sum_{k=0}^{n} w_{ki} y_{kh}$$

avec la condition aux bornes 3.69 sur les cellules de sortie

$$y_{ih} = 2f'(a_i)(d_{ih} - x_{ih}) \qquad \forall i \in U_s$$

La troisième équation (3.71) décrit l'évolution des poids

$$w_{ij} \leftarrow w_{ij} + \lambda \mathcal{M}[y_{ih} x_{jh}]$$

Si l'on veut calculer les gradients exacts dans le cas déterministe, c'est à dire si les exemples sont connus à priori, il faut accumuler les termes $y_{ih}x_{jh}$ pour tous les vecteurs de l'ensemble d'apprentissage. Dans le cas contraire, on effectue une estimation de leur moyenne temporelle. En pratique, on utilisera souvent un gradient stochastique, en estimant le gradient avec sa valeur "instantanée" correspondant à une seule paire de vecteurs entrée-sortie désirée. Dans cette hypothèse, une modification des poids est effectuée après chaque présentation d'une forme, comme dans la procédure de Widrow-Hoff.

En résumé

En résumé, dans le cas stochastique, l'algorithme se décompose en trois phases:

- Présentation d'un vecteur d'entrée sur les cellules d'entrée et calcul de l'état du réseau (les x_i) par propagation, à l'aide de la formule

$$x_i = f(a_i) \quad a_i = \sum_{j=0}^{n} w_{ij} x_j$$

- Présentation d'un vecteur de sortie désirée sur les cellules de sortie et calcul des gradients relatifs aux a_i par rétro-propagation, à l'aide des formules

$$y_i = 2f'(a_i)(d_i - x_i) \quad \text{pour les cellules de sortie}$$

$$y_i = f'(a_i) \sum_{k=0}^{n} w_{ki} y_k \quad \text{pour les autres cellules}$$

- Application d'une itération de la procédure de gradient

$$w_{ij} \leftarrow w_{ij} + \lambda y_i x_j$$

L'algorithme obtenu est d'une surprenante simplicité, le nombre d'opérations à réaliser à chaque itération est proportionnel au nombre de poids. Lorsqu'il n'y a qu'une seule couche de poids (deux couches de cellules), cet algorithme se réduit à la procédure des moindres carrés décrite au chapitre 1. C'en est une généralisation directe.

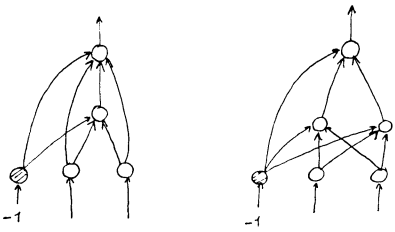


Figure 3. 5: Deux réseaux pour le XOR, l'état de la cellule hachurée est supposé constant égal à -1, les arcs qui en partent sont les seuils des cellules cibles

3.4.4 Propriétés élémentaires

Apprendre le XOR

Le premier problème sur lequel il convient de tester la RP est naturellement le XOR: le plus petit problème non-linéairement séparable. Ce problème est évidemment trop simple pour avoir un intérêt pratique quelconque, mais il illustre bien le fonctionnement de l'algorithme, et permet d'en dégager quelques propriétés élémentaires. On doit toutefois noter qu'il ne permet pas de tester les capacités de généralisation, puisqu'il s'agit d'apprendre "par coeur" une table de vérité. En outre, il ne faut pas négliger le fait que sur ce type de petit problème, l'algorithme trivial (et exponentiel) de stockage in-extenso [9] est de loin le plus performant.

Deux types de réseaux peuvent être utilisés pour le problème du XOR. Ils sont représentés sur la figure 3. 5.

Nous présentons ici quelques résultats de simulations réalisées avec la seconde structure (à deux cellules cachées). La fonction sigmoïde est

$$f(a) = m\frac{\exp ka - 1}{\exp ka + 1}$$

avec $m = 1.716$ et $k = 1.333$ (les raisons de ces valeurs sont données plus loin). Les valeurs binaires d'entrée et de sortie désirée sont -1 et +1. Les poids initiaux sont

[9] dans une mémoire dont l'adresse est constituée du vecteur d'entrée

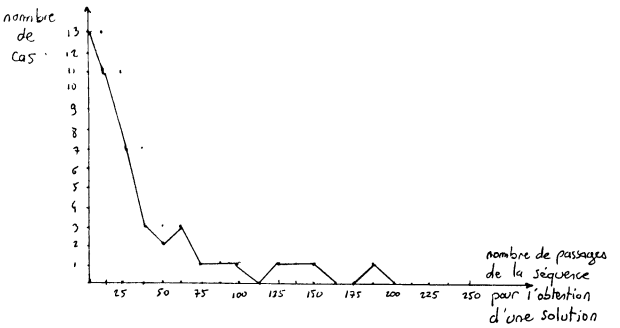


Figure 3. 6: Temps de convergence pour le XOR

choisis aléatoirement entre -2 et +2 avec une distribution uniforme. Le pas de gradient effectif λ est égal à 0.2. Une sortie est considérée comme correcte lorsque son signe est correct. Les 4 configurations sont présentées et répétées selon un ordre invariable: $(-1,-1)(-1,+1)(+1,+1)(+1,-1)$. Les poids sont modifiés après chaque présentation d'une configuration. Après chaque groupe de 4 passages de la séquence (toutes les 16 itérations d'apprentissage), on teste les performances, les 4 formes sont présentées et les réponses testées sans que les poids soient modifiés.

La figure 3. 6 montre le nombre de passages de la séquence nécessaires à l'obtention d'une solution correcte pour 50 configurations initiales des poids différentes. Dans 4 cas sur 50, la convergence n'a pas été obtenue après 400 présentations de la séquence. On voit que l'écart entre le temps le plus court et le temps le plus long est assez élevé. Le temps de convergence est très dépendant des conditions initiales, il semble qu'il existe une zone de départ, heureusement assez étendue, assurant une convergence rapide, et une autre zone plus petite, où la convergence est plus lente. Il est probable que cette dernière zone soit séparée de la zone des solutions par un plateau: une région de coût élevé où le gradient est presque nul.

Il semble que l'ordre de présentation des exemples influe sur les performances lorsqu'on utilise la procédure stochastique de mise à jour des poids [10]. En effet, il est préférable de choisir une séquence qui maximise le nombre de variations de la sortie

[10] Il est évident que si les gradients de tous les exemples sont accumulés avant les modifications des poids, leur ordre ne peut influer sur le résultat

désirée, l'apprentissage est plus efficace dans ce cas car la région de l'espace des poids explorée est plus grande. C'est pourquoi on a préféré la séquence

$$(-1,-1)(-1,+1)(+1,+1)(+1,-1)$$

à la séquence

$$(-1,-1)(-1,+1)(+1,-1)(+1,+1)$$

Les sorties désirées associées à la première séquence sont $-1,+1,-1,+1$, alors que celles de la seconde sont $-1,+1,+1,-1$.

La surface de coût

La fonction de coût associée à un réseau multicouche est beaucoup plus complexe que pour une cellule linéaire unique telle que celles du chapitre 1. Elle possède à peu près toutes les propriétés que l'on redoute:

- elle est non-convexe.
- elle possède des minima locaux
- le minimum global n'est pas unique
- elle est parsemée de "pièges" de deux natures: des plateaux et des ravins.

Les minima locaux ne semblent pas poser de sérieux problèmes pour deux raisons principales. La première raison est qu'ils sont rares. Les minima locaux n'existent généralement que lorsque le réseau a tout juste la taille nécessaire pour résoudre le problème. Ajouter des degrés de liberté (des poids supplémentaires) a pour effet de les faire disparaître. La seconde raison est que des techniques simples permettent d'échapper aux minima locaux. Une de ces techniques, que nous appellerons "pédagogie" par abus de langage, consiste à présenter plus souvent les formes incorrectement apprises. Ainsi, le poids statistique de ces formes dans le coût global augmente, et par contrecoup, le coût global lui-même augmente. Cette augmentation locale dans le temps et dans l'espace déforme la surface de coût, et permet éventuellement d'échapper au minimum local. Il faut toutefois se méfier des couplages et des oscillations que peut entraîner cette technique si elle est utilisée trop brutalement.

Les solutions, plutôt que d'être isolées, sont regroupées en zones. Il existe, en général, une infinité de solutions à un problème donné. Dans la plupart des cas, l'ensemble des solutions n'est pas connexe, ce qui facilite sa recherche.

Les deux principaux problèmes sont la présence de ravins et la présence de plateaux. Les ravins sont des régions où beaucoup de temps est perdu. Ce sont des zones de l'espace des poids dans lesquelles la dérivée seconde est importante (et positive) dans une direction, et les dérivées premières et secondes faibles dans une autre direction. La convergence dans ces zones est nécessairement lente, la situation est semblable à celle d'une cellule linéaire dont les entrées sont fortement corrélées. Dans la première direction, la convergence est rapide, mais la valeur du pas de gradient λ doit être suffisamment petite pour ne pas entraîner la divergence. Dans l'autre direction, en revanche, la convergence est très lente. C'est un phénomène similaire à celui décrit au chapitre 1 lorsque les entrées d'une cellule linéaire sont fortement corrélées. Toutefois la cause en est différente. Il est fréquent que les deux directions mentionnées ci-dessus correspondent aux poids de deux couches différentes, par exemple, les poids de la couche de sortie, et ceux d'une couche cachée. Souvent, les gradients relatifs aux poids de la couche de sortie sont importants, alors que ceux des couches cachées sont plus faibles. Hinton ([Rumelhart & al. 86a]) propose une méthode permettant d'accélérer la convergence dans les ravins. Elle consiste à affecter le vecteur de poids d'une inertie et d'un frottement visqueux. Ainsi les variations des poids sont plus rapides lorsqu'elles s'effectuent dans des directions constantes ou quasi-constantes, en revanche elles sont amorties lorsque la direction de la variation change fréquemment. La viscosité doit être élevée au début de l'apprentissage afin d'éviter les oscillations. En effet le gradient peut être très élevé au début de la phase d'apprentissage, mais il décroit à mesure que l'on s'approche de la solution (lorsque l'on est tombé dans un ravin). Le défaut de cette technique est qu'elle ne tient pas du tout compte de la CAUSE de l'existence des ravins. Sa mise en oeuvre est décrite plus précisément dans la partie "problèmes pratiques" ci-après.

Les plateaux de la surface de coût constituent certainement l'un des plus difficiles problèmes de la R.P. Un plateau est une zone où le coût est élevé, et où le gradient est quasi-nul sur une surface [11] importante. La présence de plateaux est directement liée à l'utilisation de cellules à fonction non-linéaire saturée. En effet, lorsque les poids d'une cellule sont trop importants, la somme pondérée de ses entrées est constamment dans la zone saturée de la fonction non-linéaire. Dans cette zone, la dérivée de la fonction est proche de zéro, cette cellule propagera donc un gradient très petit au cellules des couches inférieures. D'une manière générale, lorsque les poids sont trop grands, les gradients sont faibles, et l'apprentissage lent. Il est possible d'échapper à ce type de plateau en faisant systématiquement décroître les poids exponentiellement

[11] ou plutôt "hypersurface"

vers 0, c'est à dire en appliquant la formule de mise à jour

$$w_{ij} \leftarrow (1-\delta)w_{ij} + \lambda \ldots$$

où δ est une petite constante. Ceci est équivalent à ajouter dans la fonction de coût un terme proportionnel à la somme des carrés des poids. L'effet de ce terme sur le paysage du coût est d'en "soulever la périphérie", comme si l'on plaçait ce paysage dans un bol parabolique. Le danger de cette technique de décroissance des poids est qu'elle risque d'entraîner les poids dans un autre plateau situé autour de l'origine.

L'origine de l'espace des poids est entourée d'un plateau. Lorsque les poids sont nuls ou quasi-nuls, les gradients le sont également. Donc, lorsque les poids sont trop petits, l'apprentissage devient lent. Il peut même être rendu totalement inefficace, car l'origine de l'espace des poids est un point stable, un piège potentiel dont il est difficile de sortir.

Les plateaux et les ravins ne sont pas de natures fondamentalement différentes. Ils représentent deux aspects d'un même problème: l'importance du rapport entre le plus petit et le plus grand gradient possible. Comme dans le cas d'une cellule linéaire unique, c'est le plus grand gradient, qui fixera la valeur maximale admissible du pas de convergence. Et c'est le plus petit qui déterminera le temps de convergence.

3.5 Quelques exemples de simulations

3.5.1 Les problèmes pratiques

Comme pour la plupart des algorithmes, la théorie ne suffit pas à elle seule pour faire effectivement fonctionner le système. Une série de problèmes pratiques doivent être résolus au préalable. La plupart de ces problèmes concernent la valeur à donner aux paramètres. Nous avons fait des choix a priori, qui ont été conservés pour presque toutes les simulations.

Les cellules

Les cellules n'ont pas de seuil qui leur est spécialement affecté. Le seuil est un poids "normal" qui provient d'une cellule dont l'état est toujours -1. Il n'y donc pas de traitement particulier pour le seuil. La fonction des cellules est la fonction sigmoïde classique

$$f(a) = m\frac{\exp ka - 1}{\exp ka + 1}$$

Les valeur de m et k sont les suivantes

$$m = 1.7159 \qquad k = \frac{4}{3}$$

la fonction f est impaire et varie entre $-m$ et m. Ces valeurs, qui peuvent sembler étranges, assurent que $f(1) = 1$, et que la dérivée seconde de f est maximum en 1. Le respect de la première condition $f(1) = 1$ permet de plus facilement apprécier la taille des poids. Il est en outre possible d'imposer des sorties désirées égales à -1 et +1 lorsque les informations à produire en sortie sont binaires.

Les poids

Dans les simulations, nous utilisons un gradient stochastique: les poids sont modifiés après chaque présentation d'un couple entrée-sortie désirée. Les formules utilisées pour la modification des poids sont décrites ci-après. chaque poids w_{ij} est modifié d'une quantité Δw_{ij} mise à jour à l'aide de

$$\Delta w_{ij} \leftarrow \beta \Delta w_{ij} - \alpha \lambda_i y_i x_j$$

les poids sont modifiées à l'aide de

$$w_{ij} \leftarrow (1 - \delta) w_{ij} + \Delta w_{ij}$$

Voici la description des paramètres apparaissant dans ces expressions:

- λ_i est le pas de gradient attaché à la cellule i. En général nous avons choisi un λ différent pour chaque cellule, calculé de la manière suivante

  $$\lambda_i = \frac{\lambda}{F_i^{in}}$$

  où λ est une constante globale, et F_i^{in} le nombre d'entrées (Fan-in) de la cellule i. Ce mode de calcul permet d'égaliser les vitesses de convergence des diverses cellules: la trace de la matrice de covariance des entrées est proportionnelle au nombre d'entrées, ceci est donc une approximation du λ assurant la convergence du coût. Dans la suite nous n'indiquerons que la valeur du λ global, étant entendu que le pas de gradient associé à chaque cellule est calculé avec cette formule.

- α et β sont des paramètres permettant de "lisser" les variations trop rapides du gradient. Dans tous les cas, on a choisi $\alpha + \beta = 1$, cette hypothèse peut être faite sans perte de généralité. Prendre une valeur de β non-nulle ($\alpha < 1$) a pour effet d'intégrer les gradients (avec perte) et d'opérer à la manière d'un filtre passe-bas sur les variations de la fonction de coût.

- δ est un paramètre permettant de faire décroitre les poids exponentiellement vers 0 en l'absence de gradient. Lorsque δ est non-nul, la procédure minimise un compromis entre l'erreur en sortie et la norme de la matrice des poids (la somme des carrés de ses termes). En outre, après que l'équilibre ait été atteint, chaque poids a une valeur proportionnelle au gradient du critère relatif à ce poids. Ceci permet de mesurer l'importance d'un poids sur le résultat. Le maniement de ce paramètre est dangereux, car une valeur trop importante de δ fait immanquablement converger tous les poids vers 0.

Choisir les poids initiaux

Nous avons vu que deux zones de l'espace des poids étaient à éviter: trop loin de l'origine, ou trop près de l'origine. Lorsque certaines données sur le problème à résoudre sont disponibles, il est possible de choisir une configuration initiale qui en tienne compte. En l'absence de toute donnée sur le problème, on préfèrera choisir des poids assez grands, et utiliser une décroissance exponentielle, plutôt que de prendre des poids trop petits pouvant conduire à une catastrophe.

Ce problème des poids initiaux semble se poser totalement différemment lorsqu'on utilise une fonction non-linéaire variant entre 0 et 1 au lieu de $-m$ et $+m$. Dans cette situation, des poids nuls entraînent un niveau d'activité moyen des cellules de 0.5, et non de 0 comme lorsqu'on utilise une fonction impaire. Les variations de l'amplitude moyenne des cellules sont moins importantes. Ceci a peut-être pour effet de réguler l'amplitude des gradients et d'adoucir les ravins. En tout état de cause, le plateau de l'origine n'existe pas avec cette fonction. Il est néanmoins nécessaire de choisir des poids initiaux non nuls pour les cellules cachées (éventuellement aléatoires) de manière à briser la symétrie lorsque le schéma d'interconnexion ne le permet pas.

Malgré tout ce qui vient d'être dit, l'avantage des fonctions asymétriques n'est pas clair. Il semble que sur certains problèmes, les temps de convergence obtenus avec des fonctions symétriques soient très nettement plus courts qu'avec des non-symétriques.

Dans l'état actuel, nous manquons d'éléments pour choisir l'un ou l'autre en toute connaissance de cause. Pour les simulations, il est nécessaire de faire des choix a priori pour un certain nombre de paramètres. Nous avons choisi d'utiliser des cellules symétriques, bien que ce choix ne s'avère pas être le meilleur dans certains cas.

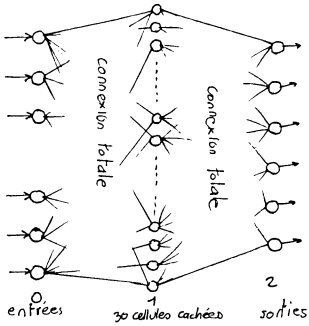


Figure 3. 7: Réseau utilisé pour la multiplication

3.5.2 Apprendre des fonctions booléennes

Il est possible d'utiliser la rétro-propagation pour synthétiser des réseaux à structure fixe calculant une fonction booléenne donnée. C'est une extension du problème du XOR à des fonctions plus complexes. Il n'est pas question ici d'apprentissage autre que l'apprentissage par coeur, puisqu'il n'y a pas (ou presque) de généralisation.

L'exemple choisi est la multiplication de deux nombres binaires à trois bits. On utilise un réseau comprenant 2 groupes de 3 cellules d'entrée pour les deux nombres à multiplier, et de 6 cellules de sortie pour le résultat. Le réseau comporte 30 cellules cachées (voir figure 3. 7).

Il y a 64 exemples possibles (de 0 fois 0 à 7 fois 7). Ce problème constitue un bon test, car la fonction devant être générée est assez complexe, et évidemment, non-linéairement séparable. La figure 3. 8 montre une courbe typique d'apprentissage pour $\lambda = 0.5$, $\alpha = 1$, $\delta = 0$. Une réponse est considérée comme correcte si les signes de toutes les sorties sont corrects.

Dans certains cas, on peut observer une généralisation élémentaire, trop élémentaire pour être intéressante. Elle intervient sur des exemples triviaux comme "0 fois x" ou "1 fois x", où la règle est facile à découvrir sur un petit nombre d'exemples.

Un réseau à 10 cellules cachées a également été essayé, mais n'a pas permis d'obtenir les 64 réponses correctes.

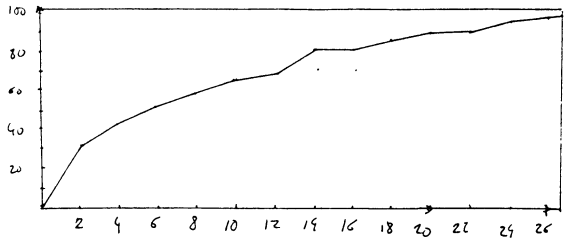


Figure 3. 8: Courbe d'apprentissage de la multiplication, en abscisse: le nombre de passage de l'ensemble d'apprentissage. En ordonnée: la proportion de réponses correctes

3.5.3 L'apprentissage non supervisé et l'extraction de codes

Un reproche est souvent formulé à l'égard de la RP: elle nécessite une sortie désirée pour chaque vecteur d'entrée, c'est une procédure d'apprentissage supervisée. Il existe de nombreux problèmes où les classes ne sont pas connues a priori, qui sont regroupées sous le vocable "classification automatique". Dans beaucoup de ces méthodes, il existe en fait un superviseur "implicite", qui est déterminé par l'espace de représentation des données. Elles sont en général basées sur une notion de distance dans cet espace qui est déterminée a priori, et qui conditionne la manière dont les formes vont être classées.

Notre sentiment est qu'il n'y a pas de différence fondamentale entre les procédures d'apprentissage supervisées et non-supervisées. Une méthode supervisée peut, dans la plupart des cas, être utilisée dans un mode non-supervisé. C'est le cas de la rétro-propagation qui peut fonctionner dans un mode "supervisé par les entrées".

L'utilisation la plus fréquente des méthodes de classification automatique est la compression des données. C'est le cas des méthodes de quantification vectorielle utilisées en communication et en reconnaisance de la parole. Dans d'autre cas, il s'agit de résumer une collection d'expériences ou d'échantillons statistiques. D'une manière générale, il s'agira de réduire la quantité d'information à manipuler.

Une architecture particulière de réseau peut être utilisée à cette fin. Ce type de

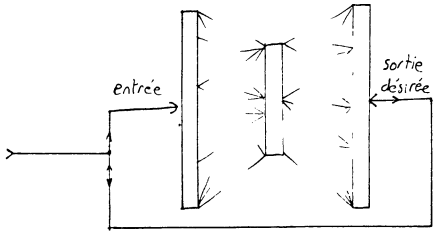


Figure 3. 9: Un réseau en "diabolo" pour la RP non supervisée

réseau est constitué d'une couche d'entrée, d'une ou plusieurs couches cachées, de taille inférieure à la couche d'entrée, et d'une couche de sortie identique à la couche d'entrée (voir figure 3. 9).

Au cours de l'apprentissage, le même vecteur est utilisé comme entrée et comme sortie désirée. Le rôle des cellules cachées est d'extraire un code compact des données d'entrée. La compacité et l'exactitude de ce code sont déterminées par la "largeur" de la plus petite couche cachée. Ce type de réseau fait actuellement l'objet de nombreuses recherches pour les problèmes de codage et de compression de données (image, parole).

Plusieurs méthodes peuvent être utilisées pour coder une image avec un réseau multi-couche. La plus simple est probablement d'utiliser un réseau en "diabolo" dont la couche d'entrée et la couche de sortie sont des fenêtres que l'on déplace sur l'image. Une ou plusieurs couches cachées sont intercalées entre la couche d'entrée et la couche de sortie, les états de la plus petite représenteront l'image codée (voir figure 3. 10). La génération du réseau peut s'effectuer sur une ou plusieurs images, selon l'utilisation, le code devra être d'autant moins compact (ou exact) qu'il représente plus d'images. La génération du réseau s'effectue en déplaçant la fenêtre sur l'image à coder, sans recouvrement entre les positions successives. Après apprentissage, le code de l'image est représenté par la donnée des poids des couches situées APRÈS la couche cachée la plus petite, et par la série des états de cette couche pour toutes les positions de la fenêtre.

Nous montrerons l'application de ces réseaux non supervisés aux mémoires asso-

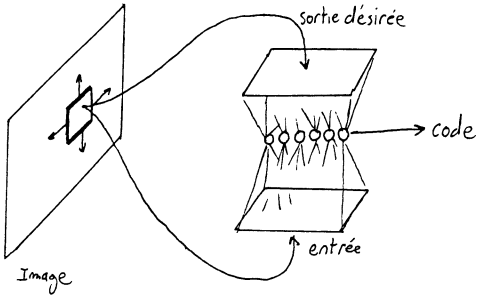


Figure 3. 10: un réseau auto-associatif pour le codage d'images

ciatives dans le sous-chapitre suivant. Auparavant nous décrivons un petit exemple intéressant.

L'encodeur 4-2-4

L'idée de "l'encodeur" est apparue dans [Hinton & Sejnowski 83]. Il s'agit d'un réseau comportant une couche d'entrée et une couche de sortie de même taille, et d'une couche cachée de taille inférieure. Le réseau doit reproduire le vecteur d'entrée sur la couche de sortie.

Nous présentons les résultats de simulation d'un encodeur 4-2-4, c'est à dire comportant 4 cellules d'entrée, 4 cellules de sortie, et 2 cellules cachées. La connexion est totale entre la couche d'entrée et la couche cachée, ainsi qu'entre la couche cachée et la couche de sortie. Quatre vecteurs différents peuvent être présentés à l'entrée, dans chacun d'eux, une seule cellule est à +1, les autres à -1. De manière évidente, les cellules de la couche cachée doivent générer une configuration différente pour ces quatre vecteurs. on pourrait penser que le seul code possible est le code binaire, mais ce serait oublier que les cellules peuvent prendre des états intermédiaires. Chacune des cellules de sortie effectue une discrimination linéaire. Il suffit donc que chacune des quatre configurations de la couche cachée soit séparable des trois autres par une droite pour que le code soit valide.

La figure 3. 11 représente les codes générés par les cellules cachées avec $\delta = 0$ et $\delta = 0.001$. Dans les deux cas on a pris $\lambda = 0.5$ et $\alpha = 1$, $\beta = 0$. On constate

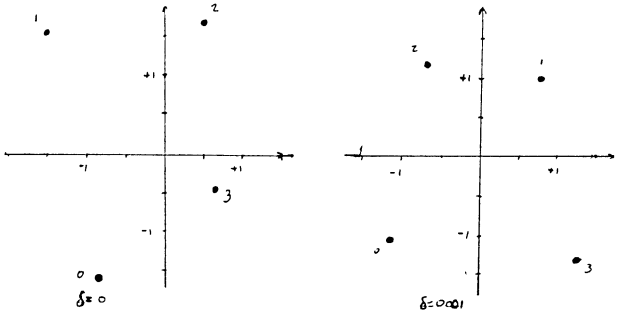


Figure 3. 11: Code généré par les cellules cachées de l'encodeur 4-2-4. En abscisse et en ordonnée sont les niveaux d'activité de la première et de la seconde cellule cachée. Chaque point correspond à un vecteur d'entrée. La figure de gauche résulte d'un apprantissage avec $\delta = 0$ et la figure de droite avec $\delta = 0.001$

que dans les deux cas, les quatre points correspondant aux quatre vecteurs d'entrée forment une figure convexe. Chaque point est séparable des autres par une droite. L'introduction d'une valeur non nulle pour δ rend la solution plus uniforme, les points tendant à former une figure symétrique autour de l'origine.

3.5.4 Un exemple: la reconnaissance de chiffres manuscrits

L'algorithme de rétro-propagation a été testé sur un petit problème de reconnaissance de chiffres manuscrits. Il s'agissait de tester l'algorithme sur un problème de taille raisonnable, et permettant d'examiner les capacités de généralisation.

La base d'exemples est constituée de 12 exemples de chacun des 10 chiffres digitalisés (avec une souris) dans une grille de 16 points par 13 points. Sur ces 120 caractères, 80 sont utilisés pour l'apprentissage et les 40 restants pour le test (voir figure 3. 12).

Les formes sont présentées sans prétraitement à l'entrée d'un réseau comportant 3 couches. La couche d'entrée contient évidemment 208 cellules, la couche cachée 30 cellules, la couche de sortie 10 cellules. La connexion est complète entre la couche d'entrée et la couche cachée, ainsi qu'entre la couche cachée et la couche de sortie. Les poids initiaux sont choisis aléatoirement entre -2 et 2 avec une distribution uniforme.

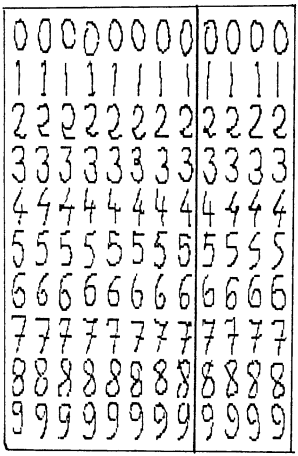


Figure 3. 12: Le Jeu de chiffres manuscrits. Les 80 caractères de gauche constituent l'ensemble d'apprentissage, les 40 autres l'ensemble de test

Figure 3. 13: Un échantillon de caractères bruités: chaque point est inversé avec une probabilité de 0.2

Ce réseau ne comporte aucune structure spécifique au problème. Il s'agit donc ici de tester les performances brutes de l'algorithme.

Il a paru nécessaire de bruiter les formes durant l'apprentissage de manière à contraindre la solution (voir figure 3. 13). Sans bruit sur les formes d'entrée, le problème est trop simple pour un réseau de cette taille, l'ajout de bruit permet d'éviter la découverte d'une solution particulière, trop dépendante de l'ensemble d'apprentissage. A chaque présentation, environ 20% des pixels sont inversés, cette proportion est diminuée progressivement jusqu'à 10% au cours de l'apprentissage. Les 100% de reconnaissance avec 10% de pixels inversés sont facilement atteints sur l'ensemble d'apprentissage. Une réponse est considérée comme bonne lorsque la cellule de sortie la plus active est celle correspondant à la classe correcte. En l'absence de bruit, les 100% sont atteints en 640 itérations, soit 8 passages de l'ensemble d'apprentissage.

Le taux de généralisation sur les 40 formes restantes est en revanche assez

médiocre: 87%5 à bruit nul (avec apprentissage en présence de bruit). Ce résultat n'est pas très étonnant: le réseau comporte 6540 poids, pour 120 fois 10 bits stockés. Il est clair que ce réseau est largement sous-contraint, on ne peut espérer un bon taux de généralisation. Sur le même problème et dans les mêmes conditions, un réseau à une seule couche (sans couche cachée) atteint 100% sur l'ensemble d'apprentissage en l'absence de bruit, mais seulement 65% sur l'ensemble de test. Ceci suggère que les primitives extraites par le réseau multi-couche ne sont pas si mauvaises.

Une possibilité de solution au problème de la généralisation est de réduire le nombre de poids, mais pour réduire ce nombre dans d'importantes proportions, il faut adopter un shéma d'interconnexion local entre l'entrée et la couche cachée, ce qui nous contraint à adapter le réseau au problème considéré. C'est un moyen de simplifier le problème de l'apprentissage en réduisant la taille de l'espace de recherche.

3.6 Les réseaux itératifs et pseudo-itératifs

Contrairement à ce que peut laisser croire ce qui précède, l'application de la RP n'est pas limitée aux réseaux sans boucle. L'hypothèse de non-circularité du graphe de connexion n'est qu'un moyen de simplifier la dérivation, mais elle n'est pas nécessaire.

3.6.1 Réseaux dynamiques

Permettre les boucles nécessite de prendre en considération la dynamique du système. Il n'est plus possible de raisonner uniquement sur les "états stables" obtenus par la propagation des signaux dans un réseau sans boucle. L'idée est de réinterpréter l'indice de couche d'un réseau multi-couche comme un indice temporel. On considère un réseau de structure quelconque, dont la matrice de connexion est W. On suppose que la mise à jour des cellules de ce réseau s'effectue selon un mode d'itération parallèle. L'évolution du réseau est donc décrite par l'équation

$$X(t+1) = F(W\,X(t)) \tag{3.72}$$

où t est un indice temporel, et F la fonction vectorielle appliquant f à chacune des composantes de son argument.

A l'instar d'un réseau multicouche sans boucle, les cellules sont divisées en 3 classes: les cellules d'entrée, de sortie, et les cellules cachées. Au temps t_0, l'état des cellules d'entrée est fixé de l'extérieur par un vecteur d'entrée $E(t_0)$. L'état des autres cellules du réseau est déterminé par leur état antérieur au temps $t_0 - 1$ à l'aide de l'équation 3.72. Au cours des instants suivants, le vecteur d'entrée est maintenu, ou

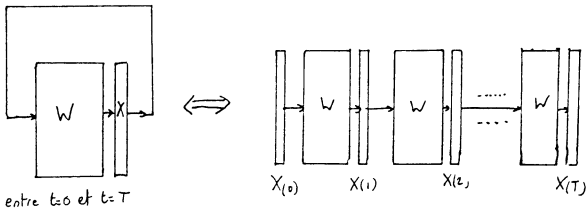


Figure 3. 14: Dépliement spatial des réseaux itératifs

éventuellement changé, l'état du réseau est calculé par l'application de l'équation 3.72. Au temps t_0+T, l'état des cellules de sortie est observé, et comparé à la sortie désirée $D(t_0+T)$ associée au vecteur d'entrée $E(t_0)$. Il s'agit maintenant de calculer les dérivées partielles de l'erreur finale par rapport aux poids. Ceci nécessite de rétro-propager les gradients dans le temps, et non dans l'espace. En effet on peut assimiler les états **successif** du **réseau** $X(t_0), X(t_0+1), X(t_0+2) \ldots X(t_0+T)$ **aux états des couches** d'un réseau sans boucle. On aboutit à un réseau dont les états successifs sont "dépliés" dans l'espace (voir figure 3. 14). Ce réseau présente quelques particularités: toutes les couches possèdent le même nombre de cellules, les matrices de pondération reliant une couche à la suivante sont toutes égales. Les gradients de la sortie finale par rapport aux $a_i(t)$ sont aisément calculables, avec la méthode classique:

$$y_i(t-1) = f'(a_i(t-1)) \sum w_{j,i} y_j(t) \tag{3.73}$$

Il est clair que l'influence d'un poids sur la sortie finale sera la somme de ses influences à chaque instant. En d'autre termes on peut écrire (on considère un seul couple entrée-sortie)

$$\frac{\partial C}{\partial w_{ij}} = \sum_{t=t_0+1}^{t_0+T} \frac{\partial C}{\partial a_i(t)} \frac{\partial a_i(t)}{\partial w_{ij}} \tag{3.74}$$

soit

$$\frac{\partial C}{\partial w_{ij}} = -\sum_{t=t_0+1}^{t_0+T} y_i(t) x_j(t-1) \tag{3.75}$$

Par conséquent, les poids sont modifiés à l'aide de

$$w_{ij} \leftarrow w_{ij} + \lambda \sum_{t=t_0+1}^{t_0+T} y_i(t) x_j(t-1) \tag{3.76}$$

On voit que la mise en oeuvre de cette procédure nécessite de stocker "l'histoire" du réseau entre les instants t_0 et $t_0 + T$ pour être en mesure d'effectuer une modification des poids.

Deux modes de mise à jour peuvent être effectués selon les échelles de temps choisies pour la mise à jour des états d'une part, et pour celle des poids d'autre part. Dans le premier mode, les poids sont modifiés à chaque instant. Dans le second mode, une modification des poids n'est effectuée que tous les T instants. Le vecteur d'entrée ne peut être changé lui aussi que tous les T instants.

La rétro-propagation classique est en fait un cas particulier de réseau itératif utilisé dans un mode spécial. Un réseau sans boucle est caractérisé par une matrice de connexion isomorphe à une matrice triangulaire (modulo une renumérotation des cellules). La constante T doit être égale au nombre de couches effectif du réseau. Il va de soi que le vecteur d'entrée doit être maintenu durant les T itérations entre le temps t_0 et le temps $t_0 + T$.

3.6.2 Réseaux mixtes ou pseudo-itératifs

Tous les intermédiaires entre les réseaux statiques et les réseaux dynamiques sont envisageables. On peut concevoir un réseau multi-couche classique comportant une couche cachée connectée à elle même. Il est possible de ne déplier spatialement QUE cette couche cachée en conservant un mode d'itération classique pour les autres couches. Un tel réseau est présenté sur la figure 3. 15. Seule la couche centrale est dépliée spatialement. Le résultat de ce dépliement peut être vu de deux manières différentes: soit l'on considère que c'est un réseau itératif utilisant un mode d'itération particulier, et utilisant la version itérative de la RP, soit l'on considère que c'est un réseau multi-couche sans boucle, classique, dont certains poids sont soumis à des contraintes d'égalité. Les deux points de vue sont équivalents.

3.7 Propager des états désirés: l'algorithme HLM

L'algorithme HLM [12] proposé dans [le Cun 84] [le Cun 85] et [le Cun 86] repose sur les mêmes idées de base que l'algorithme RP. Dans sa forme originale, il diffère de

[12] à ne pas confondre avec le logiciel de simulation du même nom

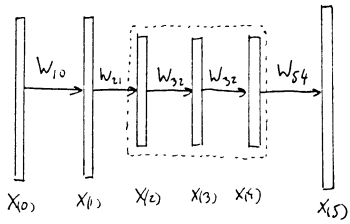


Figure 3. 15: Dépliement spatial d'une couche

l'algorithme RP essentiellement en ce que les variables rétro-propagées ne sont pas des gradients (ou des erreurs) mais des états désirés. L'algorithme HLM est plus le produit de l'intuition que du strict calcul formel, cependant, son analyse a posteriori a permis de dériver toute une classe d'algorithmes dans laquelle est incluse la rétropropagation.

3.7.1 Les états désirés

Ainsi que nous l'avons vu au chapitre 1, tous les algorithmes d'apprentissage pour une cellule unique nécessitent l'existence d'un état désiré, et dans les réseaux multicouche, seules les cellules de sortie en disposent d'un. Il est possible d'utiliser ces algorithmes classiques dans un réseau multicouche à condition d'affecter aux cellules cachées un pseudo-état désiré. L'algorithme HLM est un moyen de calcul des états désirés affectés aux cellules cachées en fonction de ceux des cellules de sortie.

Commençons par le décrire de manière informelle. Considérons dans un premier temps, un réseau composé de cellules à seuil, c'est à dire dont les cellules peuvent prendre uniquement les états -1 et 1 (nous supposons qu'il en est de même pour les états désirés). Nous allons poser la même hypothèse que lors de la dérivation de la RP: nous voulons calculer l'état désiré de la cellule d'indice i, et nous supposons que les états désirés de toutes les cellules auxquelles elle est connectée sont connus. Ces cellules, vers lesquelles la cellule i envoie sa sortie, seront nommées dans ce qui suit les "cellules aval". L'idée est de choisir l'état désiré de la cellule i de manière à "satisfaire au mieux" les autres cellules, de leur faciliter la tâche le plus possible.

Nous extrapolerons la règle après avoir examiné quelques cas particuliers. Par exemple, si la cellule i est reliée à une seule cellule (d'indice k) par l'intermédiaire d'un poids w_{ki} positif, il est clair que son état désiré devra être égal à celui de la cellule k pour satisfaire celle-ci au mieux et lui permettre de produire l'état désiré qui lui est attaché. Si w_{ki} est négatif, il devra être opposé. Si la cellule i est reliée à plusieurs autres cellules, ce qui est le cas le plus fréquent, il faudra choisir un compromis entre les états désirés de toutes ces cellules aval, toutes ne pourront pas être satisfaites en même temps. Une cellule aval d'indice k sera d'autant plus influencée par la cellule i que le poids w_{ki} est important en valeur absolue. Dans le compromis, il convient donc de donner un poids d'autant plus important à l'état désiré d'une cellule aval que le poids qui la relie à la cellule i est grand. En tenant compte de ces remarques, la règle de décision la plus simple pour choisir l'état désiré de la cellule i est la suivante

$$d_i = \begin{cases} 1 & \text{si } \sum_k w_{ki} d_k > 0 \\ -1 & \text{sinon} \end{cases}$$

ou encore

$$d_i = sgn(\sum_k w_{ki} d_k)$$

où *sgn* est la fonction à seuil qui vaut -1 lorsque son argument est négatif et +1 dans le cas contraire. Cette règle utilise les poids "à l'envers", les états désirés sont donc calculés par rétro-propagation à l'instar de la procédure "classique". Une fois cette rétro-propagation effectuée, nous disposons d'un état désiré pour chaque cellule, ce qui nous permet d'utiliser une quelconque des règles d'apprentissage décrites au chapitre 1, à condition qu'elle permette de traiter les signaux non-stationnaires.

Nous pouvons aboutir au même résultat que ci-dessus avec un argument légèrement plus formel. La règle heuristique qui vient d'être utilisée est qu'une cellule doit choisir son état (désiré) de manière à optimiser sa participation aux cellules aval. La participation de la cellule i à l'entrée de la cellule k est $w_{ki}x_i$. Nous pouvons choisir l'état désiré de la cellule i qui minimise le critère suivant

$$C_i = \sum_k (d_k - w_{ki} d_i)^2$$

où la somme est prise sur les cellules dont la cellule i est une entrée. La valeur de d_i qui optimise ce critère est celle qui optimise la participation de la cellule i aux cellules aval. La valeur optimale est caractérisée par

$$\frac{\partial C_i}{\partial d_i} = 0$$

soit

$$2\sum_k w_{kl}(d_k - w_{kl}d_l) = 0$$

d'où l'on tire

$$d_l = \frac{\sum_k w_{kl}d_k}{\sum_k w_{kl}^2}$$

Si l'on impose à d_l de prendre les valeurs -1 ou +1, la règle de choix optimale de d_l est la suivante

$$d_l = sgn(\sum_k w_{kl}d_k)$$

nous obtenons la même règle que précédemment.

Cette règle a l'avantage de la simplicité mais présente un important défaut: les états désirés des cellules cachées ne sont que très légèrement couplés.

3.7.2 Conditions sur les gradients, contraintes sur les états désirés

Toute une classe d'algorithmes peut être imaginée à partir d'une idée similaire, quoique plus formelle. Au lieu de raisonner sur le concept un peu trop spécifique d'état désiré, il est préférable d'utiliser celui plus général de critère local, déjà utilisé au paragraphe précédent. A chaque cellule du réseau, nous attachons un critère local C_l, qu'il s'agira d'une part de calculer, et d'autre part de minimiser. La définition, et le calcul, des critères locaux attachés aux cellules cachées sont des problèmes de même nature que la définition des états désirés de l'exemple précédent. On choisit d'affecter à chaque cellule cachée un critère local dont la minimisation entraîne la minimisation des critères attachés aux cellules de sortie. Les critères locaux des cellules cachées devront être calculés à partir des critères locaux des cellules de sortie, ou de ceux d'autres cellules cachées. Lorsque l'on utilise une procédure de gradient, un critère n'intervient sur les poids que par l'intermédiaire de son gradient, et non par sa valeur. C'est sur ces gradients qu'il convient d'établir une condition.

La condition la plus directe (qui mène à l'algorithme "classique" de rétro-propagation) est la suivante: minimiser le critère associé à une cellule doit entraîner la minimisation des critères des cellules aval. Plus exactement, le critère attaché à une cellule doit être choisi de manière à ce que sa minimisation (par modification des poids de la cellule considérée) entraîne celle des critères des cellules aval. Naturellement, tous les critères des cellules aval ne pourront pas être minimisés en même temps, mais nous pouvons choisir d'en minimiser la somme. Cette condition peut être exprimée par

l'équation suivante

$$\frac{\partial C_i}{\partial w_{ij}} = r \sum_k \frac{\partial C_k}{\partial w_{ij}} \tag{3.77}$$

où r est une constante positive arbitraire. En effet, les poids de la cellule i sont modifiés à l'aide d'une formule de gradient

$$w_{ij} \leftarrow w_{ij} - \lambda \frac{\partial C_i}{\partial w_{ij}}$$

la condition 3.77 exprime donc que la modification des poids de la cellule i doit faire décroître la somme des critères des cellules aval.

Nous allons développer la condition 3.77. Elle est équivalente à la condition suivante

$$\frac{\partial C_i}{\partial a_i} \frac{\partial a_i}{\partial w_{ij}} = r \sum_k \frac{\partial C_k}{\partial a_i} \frac{\partial a_i}{\partial w_{ij}}$$

soit

$$\frac{\partial C_i}{\partial a_i} x_j = r \sum_k \frac{\partial C_k}{\partial a_i} x_j$$

nous conserverons l'égalité équivalente

$$\frac{\partial C_i}{\partial a_i} = r \sum_k \frac{\partial C_k}{\partial a_i} \tag{3.78}$$

en développant

$$\frac{\partial C_i}{\partial a_i} = r \sum_k \frac{\partial C_k}{\partial a_k} \frac{\partial a_k}{\partial x_i} \frac{\partial x_i}{\partial a_i}$$

soit

$$\frac{\partial C_i}{\partial a_i} = r f'(a_i) \sum_k \frac{\partial C_k}{\partial a_k} w_{ki} \tag{3.79}$$

nous n'avons rien obtenu d'autre que la formule classique de rétro-propagation (à condition de choisir $r = 1$). Cependant, nous pouvons remplacer la condition 3.78 par une condition plus faible, ce qui nous permettra d'obtenir d'autres algorithmes. Une telle condition affaiblie peut porter sur les SIGNES des gradients, on peut également imaginer de contraindre la forme du critère en fonction de a_i.

Contrainte sur la forme du critère local

Nous pouvons imposer que les critères locaux soient d'une forme particulière. Un exemple intéressant est lorsque le critère est une fonction linéaire de a_i. La forme du critère local est alors très proche de la fonction d'énergie de Hopfield. Nous définissons les critères locaux par

$$C_i = -d_i a_i = -d_i \sum_j w_{ij} x_j$$

où d_i est contraint à prendre les valeurs -1 et +1. d_i joue le rôle d'une sortie désirée associée à la cellule i qu'il s'agit de calculer. La définition des critères locaux donne

$$\frac{\partial C_i}{\partial a_i} = -d_i$$

La condition 3.79 devient

$$d_i = rf'(a_i)\sum w_{ki}d_k$$

De manière évidente, cette condition ne pourra pas être satisfaite exactement, du fait da la contrainte sur les valeurs possibles de d_i. Il s'agira donc de choisir la valeur de d_i optimale, ce qui est effectué par la règle de décision

$$d_i = \begin{cases} +1 & \text{si} \quad rf'(a_i)\sum_k w_{ki}d_k > 0 \\ -1 & \text{sinon} \end{cases}$$

En faisant l'hypothèse que $f'(a_i)$ est toujours positif la règle ci-dessus se transforme en

$$d_i = sgn(\sum_k w_{ki}d_k)$$

c'est la règle utilisée dans l'algorithme HLM. Les variables rétro-propagées sont donc des états désirés. Seuls les états désirés des cellules de sortie sont déterminés de l'extérieur, et constituent une condition aux bornes. Ceux des cellules cachées sont calculés à l'aide de la formule ci-dessus. C'est la contrainte sur les valeurs possibles de d_i, et non la forme du critère qui a conduit à cette formule. L'hypothèse sur la forme du critère conduit à une règle d'apprentissage de type règle de Hebb, mais il n'est pas nécessaire de s'y limiter. Dans la pratique, on pourra utiliser une méthode plus efficace, comme par exemple la procédure de Widrow-Hoff.

Dans le cas stochastique, l'algorithme se compose des trois phases suivantes:

- Présentation d'un vecteur d'entrée sur les cellules d'entrée et calcul de l'état du réseau (les x_i) par propagation, à l'aide de la formule

$$x_i = f(a_i) \quad a_i = \sum_{j=0}^{n} w_{ij}x_j$$

- Présentation d'un vecteur de sortie désirée sur les cellules de sortie et calcul des gradients relatifs aux a_i par rétro-propagation, à l'aide des formules

$$d_i = sgn(\sum_{k=0}^{n} w_{ki}d_k) \quad \text{pour les cellules cachees}$$

- Application d'une itération d'une procédure d'apprentissage quelconque au niveau de chaque cellule

$$w_{ij} \leftarrow w_{ij} + \lambda G(x_i, d_i, x_j)$$

Un critère local quadratique

D'autres contraintes concernant la forme des critères locaux peuvent être utilisées. Une des plus directes est de choisir C_i de la forme

$$C_i = (d_i - x_i)^2$$

où d_i joue le rôle d'un état désiré. Par analogie avec la rétro-propagation "pure", nous utiliserons la notation suivante

$$y_k = -\frac{\partial C_k}{\partial a_k} = 2f'(a_k)(d_k - x_k)$$

la condition 3.79 devient

$$y_i = 2f'(a_i)(d_i - x_i) = rf'(a_i)\sum_k w_{ki}y_k \tag{3.80}$$

En choisissant $r = 1$ nous obtenons toujours la règle de rétro-propagation classique. En posant les hypothèses convenables sur $f'(a_i)$, cette condition se ramène à

$$d_i = x_i + \frac{r}{2}\sum_k w_{ki}y_k$$

Nous pouvons maintenant imposer des contraintes sur les valeurs possibles de d_i pour transformer la règle initiale et obtenir une variante de la rétro-propagation. La contrainte la plus simple est la suivante

$$d_i^2 = 1 \tag{3.81}$$

d_i est donc contraint à prendre les valeurs -1 et +1. Ce genre de contrainte permet d'éliminer certains défauts de la RP. En particulier, on est assuré que le niveau moyen d'activité d'une cellule sera de l'ordre de 1. Ceci permet d'éviter la convergence intempestive des poids et des niveaux d'activité vers 0 comme dans la RP classique, cela permet de "normaliser" les variables du réseau. De la même manière que dans l'exemple précédent, la condition 3.80 ne pourra pas être totalement satisfaite si l'on respecte la contrainte 3.81. Il conviendra appliquer une règle de décision optimale, c'est à dire choisir la valeur de d_i qui minimise la quantité suivante

$$(d_i - x_i - \frac{r}{2}\sum_k w_{ki}y_k)^2$$

avec la contrainte 3.81. De manière évidente, la règle de décision optimale est la suivante

$$d_i = \begin{cases} +1 & \text{si} \quad x_i + \frac{r}{2}\sum_k w_{ki}y_k > 0 \\ -1 & \text{sinon} \end{cases}$$

Soit

$$d_i = sgn(x_i + \frac{r}{2}\sum_k w_{ki}y_k)$$

avec toujours

$$y_k = 2f'(a_k)(d_k - x_k)$$

Le paramètre r permet de contrôler la probabilité que la sortie désirée soit de signe différent de la sortie effective x_i. Si $r = 0$ les cellules sont découplées et n'influencent pas mutuellement leurs états désirés. Plus r augmente, plus l'état désiré d'une cellule devient dépendant des états désirés des cellules aval, et plus il devient indépendant de l'activité propre de la cellule. Il est possible de choisir une valeur de r différente pour chaque cellule, de manière à contrôler leurs comportements indépendamment.

Dans le cas stochastique, les trois phases de l'algorithme sont les suivantes:

- Présentation d'un vecteur d'entrée sur les cellules d'entrée et calcul de l'état du réseau (les x_i) par propagation, à l'aide de la formule

$$x_i = f(a_i) \quad a_i = \sum_{j=0}^{n} w_{ij}x_j$$

- Présentation d'un vecteur de sortie désirée sur les cellules de sortie et calcul des gradients et des états désirés par rétro-propagation, à l'aide des formules

$$d_i = sgn(x_i + \frac{r}{2}\sum_{k=0}^{n} w_{ki}y_k)$$

$$y_i = 2f'(a_i)(d_i - x_i)$$

la première des formules ci-dessus ne s'applique qu'aux cellules cachées, la seconde s'applique à toutes les cellules.

- Application d'une itération de la procédure de gradient

$$w_{ij} \leftarrow w_{ij} + \lambda y_i x_j$$

Condition sur les signes des gradients

Au lieu d'imposer des contraintes sur la forme du critère et les valeurs possibles des états désirés, nous allons modifier (affaiblir) la condition 3.79. Au lieu d'imposer l'égalité des gradients (au facteur r près), nous pouvons imposer l'égalité des signes des gradients. De cette manière, nous sommes assurés que les critères locaux croissent et décroissent ensemble. La condition 3.79 affaiblie est

$$sgn\left(\frac{\partial C_i}{\partial a_i}\right) = sgn\left(f'(a_i)\sum_k \frac{\partial C_k}{\partial a_k}w_{ki}\right)$$

soit

$$sgn\left(\frac{\partial C_i}{\partial a_i}\right) = sgn\left(\sum_k \frac{\partial C_k}{\partial a_k} w_{ki}\right)$$

puisque f' est toujours positive. Avec les notations habituelles

$$sgn(y_i) = sgn\left(\sum_k y_k w_{ki}\right)$$

Cette condition ne donne qu'une information sur le signe de y_i, mais pas sur sa valeur absolue. Nous pouvons calculer celle-ci à l'aide de règles empiriques, en choisissant judicieusement la fonction g intervenant dans la formule suivante

$$y_i = g\left(\sum_k y_k w_{ki}\right)$$

la fonction g doit évidemment conserver le signe de son argument, en dehors de cette condition son choix s'effectue sur des critères expérimentaux.

3.7.3 Simulations

Nous allons décrire quelques simulations [13] utilisant une version encore simplifiée de l'algorithme HLM.

Modèle utilisé

Le modèle utilisé pour les simulations est une version simplifiée de l'algorithme HLM. Il utilise des cellules binaires (à seuil) dont les sorties peuvent prendre les valeurs -1 et 1. L'état d'une cellule est donc donné par

$$x_i = sgn\left(\sum_j w_{ij} x_j\right)$$

Ces cellules sont organisées en couches. Les états désirés sont présentés sur les cellules de sortie situées sur la dernière couche. Les états désirés des cellules appartenant aux couches intermédiaires sont calculés par rétro-propagation, avec la formule

$$d_i = sgn\left(\sum_k w_{ki} d_k\right)$$

Le schéma de calcul des états désirés est donc totalement identique à celui des états effectifs, excepté que c'est la TRANSPOSÉE de la matrice de poids qui est utilisée. Les pondérations sont modifiées à l'aide d'un algorithme de gradient stochastique de type Widrow-Hoff mis en jeu au niveau de chaque cellule:

$$w_{ij} \leftarrow w_{ij} + \lambda(d_i - a_i)x_j$$

[13] effectuées à l'automne 1984

avec la notation habituelle

$$a_i = \sum_j w_{ij} x_j$$

Les équations d'évolution des x et des y peuvent être écrites en utilisant les notations matricielles

$$X(t+1) = Sgn(W\,X(t))$$

$$Y(t+1) = Sgn(W^T Y(t))$$

où t est un indice temporel, et Sgn la fonction vectorielle appliquant la fonction sgn indépendamment à chacune des composantes de son argument. Un vecteur d'entrée est présenté en forçant les x des cellules d'entrée aux valeurs correspondantes. De même, un vecteur de sortie désirée est présenté en forçant les y des cellules de sortie. Après qu'une paire entrée-sortie ait été présentée au réseau, les deux équations ci-dessus sont appliquées jusqu'à stabilisation de X et de Y. Ceci correspond aux phases de propagation antérograde et rétrograde. Après obtention d'une configuration stable, ce qui arrive nécessairement si le réseau ne comporte pas de boucle, une itération de modification des poids est effectuée.

Architecture du réseau

Nous avons utilisé un réseau à structure fixe paramétrable, principalement destinée à la reconnaissance de (très) petites images. La taille des images est en effet de 64 pixels arrangés en un carré de 8 par 8. Il s'agissait plus de valider l'algorithme que de résoudre un problème réel de reconnaissance des formes.

Le réseau est constitué d'une série de couches de 64 cellules. Chaque couche est connectée à la précédente selon un schéma de connexion local en dimension deux. En d'autres termes, chaque cellules d'une couche prend ses entrées sur son homologue dans la couche immédiatement inférieure, ainsi que sur ses voisines (voir figure 3. 16). La taille du voisinage de connexion, et par conséquent le nombre de connexion par cellule, est de 9, 25 ou 49 cellules (carré 3 par 3, 5 par 5 ou 7 par 7). Dans ce schéma d'interconnexion, les plans successifs de cellules sont supposés posséder une topologie torique, pour éviter les effets de bords: les cellules des bords gauche et droit sont voisins, ainsi que les cellules des bords haut et bas. Ce schéma de connexion local semble bien adapté aux problèmes de reconnaissance d'image.

La dernière couche possède une structure différente des autres. Le nombre de ses cellules est variable, et celles-ci sont connectées à TOUTES les cellules de l'avant-dernière couche.

Bien entendu, il n'y a pas de connexion à l'intérieur d'une même couche.

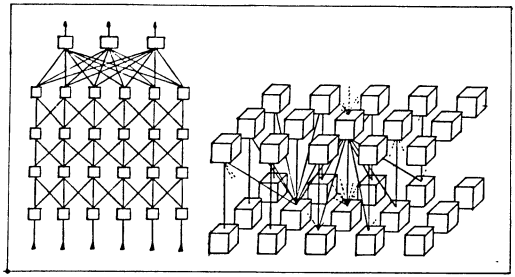


Figure 3. 16: Schéma d'interconnexion du réseau HLM

Un premier exemple

La première chose à tester est la capacité de généralisation. Comme nous l'avons déjà signalé, la généralisation est un problème mal posé. Dans l'absolu, aucun critère objectif ne peut décider si une généralisation est correcte ou non. Nous nous sommes donc fiés à la subjectivité de l'observateur, ce qui est la seule stratégie possible pour juger de la qualité de l'apprentissage.

Il s'agit de reconnaître les six premiers caractères de l'alphabet, grossièrement digitalisés "à la main" sur une grille de 8 points par 8 points. Ces caractères sont directement présentés sur la couche d'entrée du réseau, les points noirs correspondant à une cellule dont l'état est +1 et les points blancs à une cellule dont l'état est -1 [14]. Il y a cinq exemples de chacun des six premiers caractères de l'alphabet. Chacun de ces 30 échantillons peut être présenté dans quatre positions différentes sur la couche d'entrée. Il y a donc au total 120 formes différentes. Le réseau comporte 6 cellules de sortie, chacune d'elles représentant un caractère différent (voir **figure 3. 17**).

La session d'apprentissage peut se dérouler selon plusieurs modes selon que les stratégies suivantes sont actives ou non:

- les formes peuvent être choisies selon une séquence préétablie (mode "séquentiel") ou par un tirage aléatoire avec remise (mode "aléatoire").
- une forme peut être présentée telle quelle (mode "sans bruit") ou avec un certain

[14] Il va de soi que cette stratégie serait inopérante dans un problème réel de reconnaissance de caractères

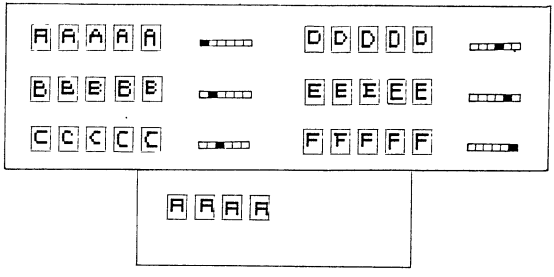


Figure 3. 17: Ensemble des caractères (non bruités) utilisés au cours de l'apprentissage avec les réponses associées. Chaque forme est présentée dans quatres positions différentes sur la couche d'entrée.

nombre de pixels choisis au hasard inversés (mode "avec bruit").

- il est possible de choisir une nouvelle forme après chaque itération de modification des poids (mode "normal") mais il est également possible de conserver la forme jusqu'à ce qu'elle produise une réponse correcte (mode "pédagogie"). Nous examinerons les avantages de ce dernier mode dans la suite.

Les 120 formes sans bruit sont apprises avec 100% de bonnes réponses par un réseau à 5 couches de poids (y compris la couche de sortie) et de connectivité 25. Ce problème n'est pas trivial, bien que simple. En terme de distance de Hamming, des formes très proches doivent être discriminées, et d'autres, très éloignées, doivent être regroupées dans la même classe. Par exemple, la distance entre le troisième B et le deuxième E est de deux pixels seulement (il en est de même de leurs 3 versions translatées), alors que la distance entre ce même B et sa version translatée vers le bas et la droite est de 30 pixels.

Les poids initiaux ont des valeurs aléatoires comprises entre -0.5 et 0.5 avec une distribution uniforme. L'algorithme ne peut pas "démarrer" avec des poids nuls. La stratégie d'apprentissage utilisée est d'activer le mode "aléatoire" au début, puis d'activer le mode "séquentiel" lorsque toutes les formes produisent une réponse correcte sauf 3 ou 4. Il est également préférable d'activer le mode "pédagogie" dans cette dernière phase. La valeur de λ doit être choisie intialement assez élevée, aux

alentours de 0.1, puis doit être progressivement diminuée et tendre vers 0, cette opération est effectuée manuellement. La vitesse de décroissance dépend de la vitesse de l'apprentissage. Cette diminution progressive de la valeur de λ est rendue nécessaire par le caractère instable de l'algorithme. En effet l'adéquation de la sortie à la sortie désirée ne modifie en aucun cas les variables rétro-propagées. Il n'y a pas de mécanisme auto-régulateur comme dans la RP classique où les gradient tendent vers 0 à mesure que la sortie se rapproche de la sortie désirée. L'algorithme de Plaut, Nowlan et Hinton décrit au paragraphe 3.7.4 tente de résoudre ce problème.

Comme dans le cas de la RP classique, le mode "pédagogie" peut être vu comme un moyen d'échapper à un minimum local de la fonction de coût. Ce mode tend à rendre plus fréquente la présentation des formes les moins bien reconnues. Ceci a pour conséquence d'augmenter localement dans le temps et dans l'espace (des poids) la valeur du coût, le résultat est une "déformation" de la surface de coût faisant disparaître le minimum local dans lequel le vecteur de poids est piégé. Cette technique peut s'avérer dangereuse et mener à des instabilitées ou des oscillations en raison du couplage entre ces déformations et la trajectoire du vecteur de poids.

Le nombre de cellules nécessaires pour résoudre ce problème est assez important, ce qui nous fait conclure que cet algorithme n'est pas très économe. Il semble que cela soit dû au faible couplage entre les cellules cachées. Contrairement à la RP, il n'existe ici aucune information qui permette d'éliminer les situations (de redondance) où deux cellules réalisent la même fonction. Ceci doit être compensé par une augmentation du nombre de cellules nécessaires à la résolution d'un problème.

Nous avons également testé la méthode sur le même problème en présence de bruit. Durant la session d'apprentissage, chaque exemple est présenté après inversion de 5 à 7% des ses pixels choisis au hasard.

Le réseau utilisé comporte 6 couches de poids avec une connectivité de 49 (7 couches de cellules en comptant la couche d'entrée).

La figure 3. 18 montre la classification opérée par le réseau sur de tels exemples. Les confusions les plus fréquentes se produisent entre E et B, le bruit pouvant facilement transformer l'un en l'autre.

La figure 3. 19 montre la classification opérée par le réseau sur des formes non apprises bruitées et non-bruitées. Ces exemples premettent de se faire une idée des primitives extraites par le réseau. Le sixième exemple de A montre que la reconnaissance de cette lettre est liée à la présence d'une barre verticale sur le côté droit. Le côté gauche est commun à presque tous les exemples (à une translation près), et n'est, par conséquent, pas discriminatif. Le premier exemple de B montre que celui-ci est

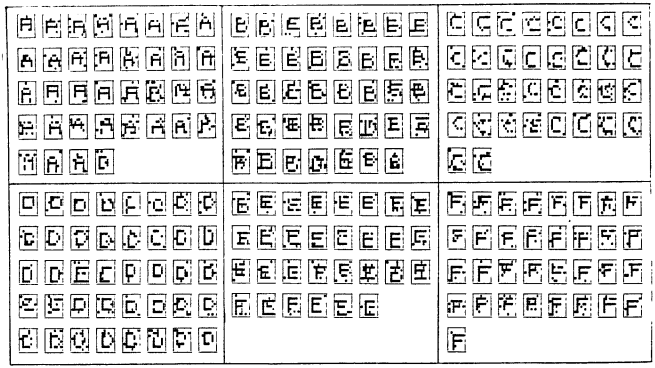


Figure 3. 18: Exemple de classement opéré par le réseau sur des images bruitées.

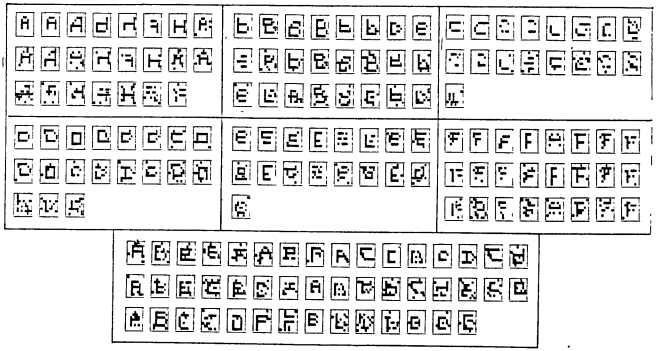


Figure 3. 19: Ensemble de test. Les formes regroupées en bas de la figure sont celles pour lesquelles aucune des 6 configurations désirées n'a été produite.

caractérisé par la présence d'une boucle dans la partie inférieure [15].

Le réseau a été construit de manière "incrémentale". On débute l'apprentissage avec un réseau comportant une seule couche cachée, puis lorsque les performances stagnent, on rajoute une couche cachée, que l'on intercale entre la première couche cachée et la couche de sortie. Les poids initiaux de cette nouvelle couche sont nuls, sauf les poids liant une cellule à son homologue qui, eux, sont égaux à 1. Cette configuration initiale des poids assure que l'ajout de la nouvelle couche ne perturbe pas (trop) le réseau, et que le changement est progressif.

Comme on a pu le constater, l'algorithme HLM présente l'énorme défaut que sa mise en oeuvre nécessite toute une panoplie de techniques, ou plutôt de "trucs", rendus nécessaires par son instabilité. Ceci, ajouté au fait qu'il est très peu économe en cellules et en poids, nous a conduits à ne pas poursuivre son étude. Cependant, certaines caractéristiques qui lui sont propres semblent intéressantes à étudier, l'une d'elles va être brièvement exposée au paragraphe suivant. En conclusion, il est certainement possible d'améliorer l'algorithme en en conservant le principal avantage: la simplicité. C'est le but que nous avons poursuivi en présentant les variantes de la procédure de base. Mais leurs performances doivent être testées par simulation.

3.7.4 Variantes

La procédure de Plaut-Nowlan-Hinton

Plaut, Nowlan et Hinton ([Plaut & al. 86]) proposent une variation de l'algorithme HLM qui utilise également la rétro-propagation d'états désirés. La différence réside dans la présence d'un FACTEUR DE CRITICALITÉ associé à chaque cellule. L'état désiré de la cellule i est calculé à l'aide de la formule suivante

$$d_i = sgn\left(\sum_k w_{ki} d_k p_k\right)$$

où p_k est le facteur de criticalité associé à la cellule k. On retrouve l'algorithme HLM en prenant $p_k = 1$ pour tout k. Le facteur de criticalité de la cellule i est mis à jour en même temps que l'état désiré à l'aide de la formule

$$p_i = \frac{|\sum_k w_{ki} d_k p_k|}{\sum_k |w_{ki} d_k p_k|}$$

ainsi, si toutes les cellules aval "proposent" le même état désiré, p_i sera égal à 1, sa

[15] Ce qui est une caractéristique commune au B majuscule et minuscule

valeur maximum. Au contraire, si les "propositions" des cellules aval sont exactement contradictoires, p_i sera égal à 0. Dans ce dernier cas, les poids de la cellule i ne seront pas modifiés, et son état désiré n'influera pas sur ceux des cellules amont. La règle d'apprentissage utilisée est une variante de la règle des moindres carrés qui tient compte du coefficient p_i

$$w_{ij} \leftarrow w_{ij} - \lambda p_i (d_i - x_i) x_j$$

L'intuition des auteurs est que les lois d'échelle de cet algorithme sont meilleures que celles de la RP. Les temps de convergence semblent croître moins vite avec la taille du réseau qu'avec la RP classique. Malheureusement, il semble également que pour un même problème, cette procédure nécessite un plus gros réseau. Les cellules cachées sont moins couplées qu'avec la RP, il en résulte une moins bonne performance sur les problèmes très contraints. Cette situation se produit avec des réseaux dont la taille est proche de la taille minimale pour le problème.

Conclusion

Les techniques décrites dans ce sous-chapitre tentent de modifier la rétro-propagation pour la rendre plus "régulière". Dans la procédure classique, les cellules de sortie jouent un rôle différent des autres, elles ont une référence extérieure qui fixe leur niveau d'activité. Les cellules cachées, quant à elles, n'ont pas ce type de contrainte, aucun processus de régulation n'assure que leur niveau d'activité ne s'écartera pas de la zone des valeurs "raisonnables". C'est sans doute une des raisons qui interdit à la RP de fonctionner avec plus de 5 ou 6 couches. Les contraintes sur les états désirés permettent de "normaliser" les niveaux d'activité et les poids.

Cette amélioration est au prix d'une perte de stabilité de l'algorithme. Le régime de convergence est beaucoup plus chaotique qu'avec la RP pure. Sur des problèmes nécessitant un ajustage fin des poids des cellules cachées, ces nouveaux algorithmes sont moins performants que la RP. En revanche, sur des problèmes moins contraints, il semble que les performances soient légèrement meilleures, particulièrement en ce qui concerne la vitesse d'apprentissage. En contrepartie, sur un même problème, il sera souvent nécessaire de disposer d'un plus grand nombre de cellules et de poids avec ces procédures qu'avec la RP. Ceci est probablement dû au fait que les redondances sont moins bien éliminées en raison du faible couplage entre cellules cachées.

3.8 La rétro-propagation: une minimisation sous contraintes

Nous présentons ici une dérivation originale de la RP qui la fait apparaître sous un jour nouveau. Nous utiliserons le formalisme Lagrangien qui permettra, à partir d'une unique fonction, de dériver la RP et d'en imaginer des variantes.

3.8.1 Un premier Lagrangien

Pour simplifier, dans un premier temps, nous considèrerons que le réseau étudié est à couches strictes, c'est à dire que les connexions relient uniquement une couche à la suivante immédiate, sans sauter de couche. Les notations utilisées sont les suivantes. Les couches du réseau sont numérotées de 0 à N, la couche 0 est la couche d'entrée, la couche N la couche de sortie. L'état d'une couche d'indice k est désigné par un vecteur $X(k)$, l'état global du réseau X est un vecteur constitué des $X(k)$ "mis bout à bout". La couche d'indice $k-1$ est reliée à la couche k par l'intermédiaire d'une matrice de connexions $W(k)$. Le vecteur des sommes pondérées des entrées de la couche k (son vecteur d'entrée totale) est noté $A(k)$ et est défini comme à l'habitude par

$$A(k) = W(k)X(k-1)$$

L'équation de propagation des états dans le réseau est alors simplement

$$X(k+1) = F(W(k)X(k-1)) = F(A_k) \quad \forall k \in [1, N]$$

$X(0)$ étant égal au vecteur d'entrée E donné de l'extérieur. La fonction F est une fonction vectorielle qui applique la fonction scalaire f à chaque coordonnée de son argument, en d'autres termes, si Z est un vecteur, $F(Z)$ est un vecteur de même dimension défini par

$$F(Z)_i = f(z_i)$$

Nous noterons cette fonction F indépendamment de la dimension de son argument bien que cette dernière soit variable. Le vecteur des sorties désirées est noté D, il est de même dimension que $X(N)$.

On définit la fonction de Lagrange suivante

$$L(W, X, B) = (D - X(N))^T(D - X(N)) + \sum_{k=1}^{N} B(k)^T(X(k) - F[W(k)X(k-1)]) \quad (3.82)$$

Cette fonction est la somme d'un terme de coût, et de k termes représentant les contraintes. Les termes de contraintes sont le produit d'un vecteur $B(k)$, le multiplieur

de Lagrange, et d'un terme qui est nul lorsque la contrainte est respectée. La contrainte est donc ici

$$X(k) = F[W(k)X(k-1)] \quad \forall k \in [1, N]$$

c'est simplement l'équation de propagation des états dans le réseau. Elle ne fait que traduire la relation de dépendance de $X(N)$ vis à vis des matrices $W(k)$. Pour être tout à fait correct, nous devrions définir la fonction L comme une somme portant sur les formes de l'ensemble d'apprentissage, nous avons omis ce détail pour ne pas alourdir les notations.

En vertu du principe de Pontryagin [Athans & Falb 66], le comportement du réseau est décrit par la condition d'optimalité

$$\nabla L(W, X, B) = 0$$

Ce qui se transforme en trois séries de conditions

$$\frac{\partial L(W, X, B)}{\partial B} = 0 \tag{3.83}$$

$$\frac{\partial L(W, X, B)}{\partial X} = 0 \tag{3.84}$$

$$\frac{\partial L(W, X, B)}{\partial W} = 0 \tag{3.85}$$

Chacune de ces conditions se décompose en N conditions correspondant aux N couches du réseau. Nous allons maintenant développer ces trois conditions qui nous donneront chacune une loi d'évolution.

La première condition

La condition 3.83 se décompose en N conditions

$$\frac{\partial L(W, X, B)}{\partial B(k)} = 0 \quad \forall k \in [1, N]$$

et nous donne tout simplement la contrainte

$$X(k) = F[W(k)X(k-1)] \quad \forall k \in [1, N]$$

qui n'est autre que la règle de mise à jour des états.

La seconde condition

La condition 3.84 se décompose également en N conditions qui sont de deux types différents selon que l'indice de couche considéré est égal à N ou non. En effet nous décomposons la condition en deux sous-parties. D'une part

$$\frac{\partial L(W,X,B)}{\partial X(N)} = 0$$

et d'autre part [16]

$$\frac{\partial L(W,X,B)}{\partial X(k)} = 0 \quad \forall k \in [1, N-1]$$

La première sous-condition donne

$$B(N) = 2(D - X(N)) \tag{3.86}$$

le développement de la seconde est plus complexe

$$B(k) = W^T(k+1)\nabla F(A(k+1))B(k+1) \tag{3.87}$$

Dans cette équation, le terme

$$\nabla F(A(k+1))$$

est la matrice Jacobienne de F au point $A(k+1)$, étant donné le F choisi, cette matrice est diagonale et son $i^{ème}$ terme est donné par

$$\nabla F(A(k+1))_{ii} = f'(a_i(k+1))$$

Nous pouvons simplifier les notations en transformant les deux équations 3.86 et 3.87 à l'aide du changement de variables suivant

$$Y(k) = \nabla F(A(k))B(k)$$

Les deux équations se transforment en

$$Y(N) = 2\nabla F(A(N))(D - X(N)) \tag{3.88}$$

$$Y(k) = \nabla F(A(k))W^T(k+1)Y(k+1) \quad \forall k \in [0, n-1] \tag{3.89}$$

Ces deux équations décrivent tout simplement la règle de mise à jour des gradients par propagation rétrograde. Les variables rétro-propagées apparaissent naturellement dans les calculs, ce sont les multiplieurs de Lagrange.

[16] il n'y a pas de condition pour $k = 0$ car $X(0)$ est fixé de l'extérieur et ne constitue pas une variable du problème

Troisième condition

La condition 3.85 ne donne pas directement de méthode de calcul de la matrice W, mais donne une condition qu'elle doit respecter. L'équation 3.85 se transforme en N conditions

$$\frac{\partial L(W,X,B)}{\partial W(k)} = 0 \quad \forall k \in [1,N]$$

ce qui donne

$$\nabla F(A(k))B(k)X^T(k-1) = 0 \quad \forall k \in [1,N]$$

ou encore, en utilisant le même changement de variable que précédemment

$$Y(k)X^T(k-1) = 0 \quad \forall k \in [1,N]$$

La matrice de poids qui rend cette identité vraie est facilement calculable à l'aide d'une procédure itérative de minimisation

$$W(k) \leftarrow W(k) - \lambda \frac{\partial L(W,X,B)}{\partial W(k)}$$

soit

$$W(k) \leftarrow W(k) + \lambda Y(k)X^T(k-1) \tag{3.90}$$

ce qui est identique à la règle de mise à jour des poids obtenue de manière plus classique.

Le résultat

Les résultats combinés fournis par les conditions d'optimalité sont donc les suivants

$$X(k) = F[W(k)X(k-1)] \qquad \forall k \in [1,N]$$

$$Y(k) = \nabla F(A(k))W^T(k+1)Y(k+1) \qquad \forall k \in [0,n-1]$$

$$W(k) \leftarrow W(k) + \lambda Y(k)X^T(k-1) \qquad \forall k \in [1,N]$$

avec les conditions aux bornes

$$X(0) = E$$

$$Y(N) = 2\,\nabla F(A(N))\,(D - X(N))$$

C'est donc exactement l'algorithme de rétro-propagation que nous venons d'obtenir à partir de la minimisation d'une seule fonction de Lagrange. Elle joue le rôle d'une fonction d'énergie, de manière analogue à la fonction énergie d'un réseau de Hopfield ou d'une machine de Boltzmann. Cette manière de voir le problème est susceptible

d'apporter des généralisations intéressantes de la procédure de base. Il est intéressant de remarquer que les multiplieurs de Lagrange apparaissent naturellement comme étant les variables propagées à l'envers. Ce résultat est connu en contrôle optimal. Lorsque le système est linéaire (ce qui n'est pas notre cas), le multiplieur de Lagrange est nommé l'état adjoint du système, et est calculé à rebours dans le temps [Athans & Falb 66] [17]. La RP n'est donc pas une idée si nouvelle, au moins dans le cas linéaire.

Le cas linéaire ne présente pour nous aucun intérêt, car toute composition de transformations linéaires reste linéaire Un réseau multi-couche composé de cellules linéaires est donc équivalent à un réseau mono-couche. Cette remarque n'est pas totalement gratuite: il peut être envisageable d'utiliser la rétro-propagation pour DÉCOMPOSER une transformation linéaire (une matrice) en produit de matrices plus "simples", ou de structures particulières. Ainsi, en imposant une structure de connexions, on peut décomposer une matrice en produit de matrices bandes. Néanmoins, il n'est pas du tout certain que cette méthode présente un avantage particulier sur les méthodes classiques de décomposition.

3.8.2 Quelques généralisations

La fonction de Lagrange introduite ci-dessus peut être modifiée de différentes façons, certains cas intéressants vont être explorés.

La décroissance des poids

Il est très simple de modifier le Lagrangien pour imposer une décroissance exponentielle des poids vers 0. Il suffit d'ajouter dans le Lagrangien le terme

$$u \sum_k \|W(k)\|^2 = u \sum_{i,j,k} w_{ij}^2(k) = u \sum_k \mathrm{Tr}(W(k)^T W(k))$$

Ainsi, le gradient de L par rapport aux $W(k)$ est transformé en

$$\frac{\partial L}{\partial W(k)} = -Y(k)X^T(k-1) + uW(k)$$

La formule d'apprentissage devient

$$W(k) \leftarrow (1 - \lambda u)W(k) + \lambda Y(k)X^T(k-1)$$

en l'absence de gradient, les poids subissent une décroissance géométrique de raison $1-\lambda u$. Le choix de u est évidemment critique, car une valeur trop importante entraine une convergence inexorable de tous les poids vers 0, indépendamment des valeurs des gradients.

[17] l'indice temporel discret est l'équivalent de notre indice de couche

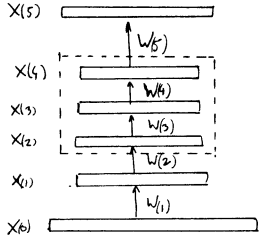


Figure 3. 20: Réseau pseudo-itératif. Les poid reliant la couche 2 à la couche 3 sont égaux à ceux qui relient la couche 3 à la 4. Les couche 2, 3 et 4 représentent l'état d'une couche unique à des instants successifs.

Réseaux pseudo-itératifs et contraintes sur les poids

Il est possible de modifier la fonction L de manière à introduire des contraintes sur les matrices de poids obtenues. Une contrainte intéressante est celle qui conduit aux réseaux pseudo-itératifs décrits plus haut. Comme nous l'avons vu, un réseau pseudo-itératif se construit en "dépliant" dans l'espace les états successifs d'un réseau rebouclé. Il est ainsi possible d'effectuer une rétro-propagation dans le temps, en considérant chaque itération temporelle comme une couche fictive. Cette situation est en fait un cas particulier d'une contrainte d'égalité que l'on peut imposer aux poids. Prenons l'exemple du réseau représenté sur la figure 3. 20. La particularité de ce réseau est que les matrices de connexions $W(3)$ et $W(4)$ sont égales. Cette contrainte peut facilement être incorporée dans le Lagrangien décrivant ce réseau en identifiant les deux matrices. Dans le cas général, le Lagrangien est transformé en

$$L(W,X,B) = (D - X(N))^T(D - X(N)) + \sum_{k=1}^{N} B(k)^T(X(k) - F[W(\phi(k))X(k-1)])$$

où ϕ est une application, généralement non-surjective, de l'ensemble d'indices $[1,N]$ sur lui-même. Si deux valeurs k_1 et k_2 ont la même image par ϕ, cela signifie que les matrices de poids des deux couches correspondantes sont égales. Les équations d'évolution obtenues à partir des conditions d'optimalité de ce Lagrangien sont identiques aux

précédentes, exceptée la règle de modification des poids qui devient

$$W(k) \leftarrow W(k) + \sum_{l \in \phi^{-1}(k)} \lambda Y(l) X^T(l-1) \quad \forall k \in \phi([1,N])$$

ici $\phi^{-1}(k)$ est l'ensemble des entiers de $[1,N]$ dont l'image par ϕ est k, et $\phi([1,N])$ est l'ensemble des images de l'intervalle $[1,N]$ par ϕ. Tout se passe comme si certaines variables de poids étaient "mises en commun" par plusieurs arcs. Cette propriété peut être utilisée avantageusement pour la réalisation informatique des réseaux pseudo-itératifs.

La contrainte peut être poussée plus loin si l'on remarque que cette mise en commun de poids peut ne pas concerner nécessairement les poids de couches successives, mais peut concerner un ensemble de poids quelconque. Par exemple, si la fonction que doit réaliser le réseau est invariante par une transformation géométrique T (translation, symétrie,...), il suffit d'imposer une contrainte d'égalité sur les poids d'une même couche qui prend en compte cette propriété d'invariance. Dans ce contexte, l'application ϕ transforme les indices des poids, et non plus simplement les indices des couches.

3.9 Un formalisme général

Nous proposons un formalisme général permettant de décrire la plupart des modèles présentés jusqu'ici. Il ne nous semblait pas souhaitable d'utiliser ce formalisme dès le départ pour ne pas distraire le lecteur des idées essentielles.

3.9.1 Notations

Nous considérons un réseau de structure quelconque, dont la matrice de connexion est notée W. L'état à un instant t est un vecteur de dimension n noté $X(t)$. L'entrée du réseau à l'instant t est notée $E(t)$, et la sortie désirée $D(t)$, E est un vecteur de dimension p et D un vecteur de dimension q. L'état du réseau est mis à jour à l'aide de l'équation

$$X(t+1) = F(\, W X(t) + P E(t)\,) \tag{3.91}$$

dans cette équation, F est la fonction vectorielle de $R^n \rightarrow R^n$ appliquant f à chacune des composantes de son argument, P est la MATRICE D'ÉCRITURE du réseau, c'est une matrice n par p. Cette matrice P sera souvent de la forme:

$$P = \begin{bmatrix} I \\ 0 \end{bmatrix}$$

où I est l'identité. Ainsi les p premières cellules du réseau sont des cellules d'entrée. Bien sûr, il n'est pas obligatoire d'utiliser un P de cette forme. L'entrée d'une donnée n'est plus effectuée par forçage des états, mais par l'ajout d'un terme dans l'entrée totale.

La sortie du réseau est notée $S(t)$ et est de dimension q. Elle est calculée à partir de l'état à l'aide de la matrice de lecture Q:

$$S(t) = QX(t)$$

les dimensions de Q sont q par n. Comme pour la matrice d'écriture, on utilisera fréquemment une matrice Q de la forme

$$Q = [\ 0\ \ I\ ]$$

pour que seules les q dernières cellules du réseau soient des cellules de sortie.

3.9.2 Le Lagrangien

A un instant τ, l'évolution du réseau est totalement décrite par la fonction scalaire suivante:

$$L(X,B,W) = C(D(\tau),S(\tau)) + \sum_{t=\tau-T}^{\tau} B^T(t)[X(t) - F(WX(t-1) + PE(t-1)] \qquad (3.92)$$

La fonction C est une fonction de coût ne dépendant que des sorties du réseau et des sorties désirées. Le paramètre T fixe l'éloignement dans le passé du dernier évènement dont il est tenu compte pour l'apprentissage. On néglige les dépendances entre vecteurs d'états séparés de plus de T unités de temps. La fonction de coût peut être le classique critère quadratique

$$C(D,S) = ||D-S||^2 = (D-QX)^T(D-QX)$$

La minimisation de L par rapport à X, B, et W est réalisable par rétro-propagation temporelle. La condition d'optimalité

$$\nabla L = 0$$

respectée à chaque instant t donne les trois équations classiques

$$X(t) = F[WX(t-1) + PE(t-1)]$$

$$Y(t) = \nabla F(A(t))[W^T Y(t+1) + 2(D(t) - QX(t))]$$

$$W \leftarrow W + \lambda Y(t) X^T(t-1)$$

avec les notations habituelles.

Ces équations s'appliquent à tous les types de réseaux, itératifs ou non.

3.10 Application de la RP aux mémoires associatives

Le chapitre précédent a présenté plusieurs exemples de stockage associatif. Ils étaient tous basés sur l'hypothèse implicite ou explicite que durant la phase de stockage, on pouvait faire abstraction du caractère itératif du processus de restauration. Cette simplification était destinée à justifier l'emploi d'algorithmes d'apprentissage classiques, ne concernant que les systèmes qui possèdent une seule couche de poids modifiables entre leur entrée et leur sortie. Nous nous proposons d'utiliser l'algorithme RP sur cette application de mémoire associative et d'en comparer les performances avec les méthodes "classiques" du chapitre 2. Nous montrons que les performances des réseaux multicouche sont nettement supérieures en ce qui concerne le nombre de formes stockables, leur attractivité et la tolérance au bruit.

Nous présentons les résultats de deux séries d'expériences. L'une concerne le stockage de caractères alphabétiques digitalisés manuellement sur une grille de 8 points par 8 points. L'autre présente un caractère plus systématique, et concerne le stockage de vecteurs binaires aléatoires de 32 bits en auto- ou hétéro-association. Dans les deux cas, nous avons comparé des réseaux avec une ou plusieurs couches. Dans le cas de l'auto-association, la sortie du réseau est binarisée (seuillée) et réinjectée sur l'entrée. Pour garder les choses comparables, le nombre de poids est conservé constant quelle que soit la structure employée.

En testant le stockage de vecteurs aléatoires, nous tentons de répondre à la question: les réseaux multicouches sont-il intrinsèquement meilleurs que les réseaux à une couche? C'est une question importante à laquelle il est préférable de répondre avant d'engager des efforts sur la question. Comme on peut s'en douter, nous y répondrons par l'affirmative. Pour ce faire nous comparerons des réseaux avec une ou plusieurs couches comportant dans tous les cas le même nombre de poids. Cette condition est nécessaire pour rendre les choses comparables.

3.10.1 Première expérience

Le problème

Il s'agit de stocker les 18 premiers caractères de l'alphabet dans une mémoire associative distribuée. Les caractères ont été dessinés manuellement en deux niveaux sur une grille de 8 points par 8 points. Ils sont représentés par des vecteurs binaires (en -1,1) à 64 composantes. C'est, bien sûr, un problème d'auto-association. Le critère de

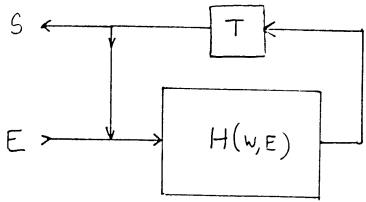


Figure 3. 21: Une mémoire associative itérative

qualité recherché est la taille des attracteurs au sens de la distance de Hamming. En d'autres termes, on mesurera la probabilité qu'une forme soit correctement restituée en fonction du nombre de bits faux de la forme présentée en entrée. Il est important de remarquer que ce critère de test est totalement arbitraire, nous aurions pu choisir un test différent, lié à une distance différente de la distance de Hamming.

Les Réseaux

Une mémoire associative itérative est représentée sur la figure 3. 21. Un vecteur E^0 est présenté à l'entrée, il est transformé par la fonction (éventuellement non-linéaire) H qui dépend d'un vecteur (d'une matrice) de paramètres W tenant lieu de mémoire, le résultat de cette transformation est à son tour transformé par une fonction de rebouclage T, qui dans notre cas, est un simple seuillage des coordonnées. Le vecteur obtenu après une passe dans le réseau est, espérons-le, débarrassé d'une partie du bruit qui était contenu dans le vecteur d'entrée initial. Il est donc intéressant de le réinjecter à l'entrée pour à nouveau éliminer une partie du bruit résiduel. On espère qu'en réitérant ce processus, le système se stabilisera sur le vecteur "pur" débarassé de tout bruit.

La condition de stabilité du système est évidemment

$$E = T(H(W,E))$$

On considérera qu'un vecteur E est stocké dans la mémoire si ce vecteur vérifie la condition ci-dessus. Il constitue un état stable du système. Le problème de la

mémorisation est de trouver un W tel que la condition de stabilité soit vérifiée pour les p vecteurs $E_h \quad h = 1 \ldots p$ que l'on veut stocker. Plutôt que d'utiliser la condition de stabilité telle quelle, on préfère choisir judicieusement la fonction T de manière à pouvoir utiliser la condition

$$E = H(W, E)$$

Si par exemple T est un seuillage coordonnée par coordonnée, la condition ci-dessus est plus forte que la condition initiale: si elle est vérifiée, alors la condition initiale l'est aussi. L'avantage de cette dernière condition est sa plus grande simplicité d'exploitation. Le problème global devient alors celui de satisfaire

$$E_h = H(W, E_h) \quad \forall h \in [1, p]$$

Nous pouvons le résoudre de manière itérative en le présentant comme un problème d'optimisation: trouver W qui minimise le critère

$$C(W) = \sum_{h=1}^{p} (E_h - H(W, E_h))^T (E_h - H(W, E_h)) \tag{3.93}$$

Cette formulation du problème permet en outre de trouver des solutions approchées (n'annulant pas le critère) lorsqu'aucune solution exacte n'existe. Lorsque H est la fonction réalisée par un réseau multicouche, ce problème est aisément résolu avec l'algorithme RP, il suffit de présenter le même vecteur comme entrée et comme sortie désirée. H représente alors la "fonction de transfert" du réseau, sa relation entrée-sortie.

Les réseaux utilisés dans les simulations sont de trois types:

Type 0 Un réseau dans lequel la fonction H est une transformation linéaire de matrice W, calculée à l'aide de la procédure d'orthogonalisation de Gram-Schmidt (voir [Fogelman & al. 87b]), et soumise à la contrainte de diagonale nulle [18]. La matrice obtenue est donc la solution de norme minimum avec diagonale nulle. Le réseau comporte $64^2 - 64 = 4032$ poids.

Type 1 Un réseau comportant une couche de cellules utilisant la fonction sigmoïde 1.6

$$f(a) = m \frac{\exp ka - 1}{\exp ka + 1}$$

avec $m = 1.716$ et $k = 1.333$, ce qui assure que $f(1) = 1$. Ce réseau est représenté sur la figure 3. 22, il ne possède qu'une seule couche de poids entre son entrée et sa sortie. Ses poids sont calculés à l'aide de la procédure de

[18] Cette simulation a été effectuée par Patrick Gallinari et Sylvie Thiria

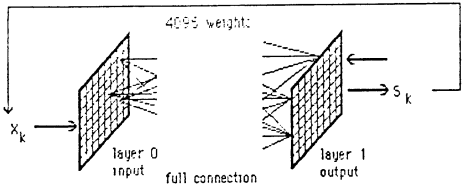


Figure 3. 22: Réseau auto-associatif de type 1

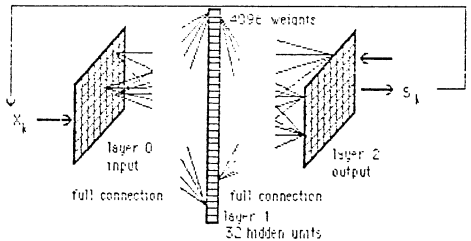


Figure 3. 23: Réseau auto-associatif de type 2

Widrow-Hoff, c'est à dire un gradient stochastique sur le critère 3.93. Il n'y a pas de connexion sur la diagonale, le nombre de poids est donc égal à 4032 comme dans l'exemple précédent

Type 2 Un réseau à deux couches de poids modifiables utilisant les mêmes cellules que le réseau de type 1, et dont les poids sont calculés par rétro-propagation sur le critère 3.93. Pour que les choses restent comparables, ce réseau est choisi de manière à comporter approximativement le même nombre de poids que les précédents. Il est représenté sur la figure 3. 23, il comporte une couche de cellules cachées contenant 32 cellules. Le nombre de poids est donc égal à 4096.

Pour les réseaux de types 1 et 2, deux techniques d'apprentissage ont été utilisées. La première, la plus simple, se déroule de la manière suivante: les 18 formes sont présentées séquentiellement au réseau jusqu'à l'obtention d'une solution "raisonnable". Chaque itération d'apprentissage consiste en la présentation d'une forme en tant qu'entrée et en tant que sortie désirée suivie d'une modification des poids. La deuxième technique d'apprentissage diffère de la première en ce que les formes présentées en entrée sont des versions bruitées des formes à stocker. On espère ainsi améliorer l'attractivité des formes stockées. Une itération d'apprentissage se déroule comme suit: choix d'une formes, inversion (multiplication par -1) d'un nombre fixé de composantes choisies au hasard, présentation de cette forme bruitée sur les entrées, présentation de la forme "pure" en sortie désirée, et modification des poids. Au cours d'une présentation des 18 formes, le nombre de composantes inversées (le bruit) est maintenu constant (par exemple égal à 5 bits inversés), au cours de la série suivante, on lui donne une valeur légèrement inférieure (ex: 3), puis on fait décroître cette valeur au cours des séries suivantes jusqu'à 0. Puis, à nouveau, on recommence toute la séquence en repartant d'un bruit important (ex: 5) etc... jusqu'à l'obtention d'une solution satisfaisante. Parallèlement la valeur de λ est diminuée (par exemple de 0.5 à 0.01) [19]. On voit qu'en présence de bruit, le problème considéré n'est pas à proprement parler un problème d'auto-association. Dans l'absolu, les formes d'entrée et de sortie désirée sont différentes. Néanmoins, on s'est efforcé à choisir un protocole qui permette d'obtenir 100% de réussite pour un bruit nul, c'est à dire de rendre stable (sinon attractif) chacun des 18 caractères.

En l'absence de bruit, le système est largement sous-déterminé, puisqu'il y a 18 formes de dimension 64, soit 18 fois 64 équations pour environ 4000 inconnues. En

[19]La valeur de λ effectivement utilisée pour une cellule est égale à cette valeur divisée par le nombre d'entrées de la cellule considérée

présence de bruit la situation est inversée, car il existe de très nombreuses manières différentes de bruiter les 18 formes. Le nombre de formes effectivement stockées est alors très important, on peut toutefois objecter que ces formes sont fortement interdépendantes.

Au cours de la phase de restauration, des vecteurs bruités sont placés à l'entrée du réseau, puis la sortie du réseau est calculée. Le vecteur obtenu est seuillé composante par composante. Le vecteur binaire résultant de ce seuillage est réintroduit à l'entrée du réseau. Ce processus est répété cinq fois au total.

Résultats et discussion

Les résultats des tests sont donnés sur la figure 3. 24. En abscisse: le nombre de bits inversés sur le vecteur d'entrée, en ordonnée: le pourcentage de vecteurs correctement restitués (exactement restitués). 18 vecteurs ont été stockés dans le réseau. Cinq courbes sont représentées sur la figure. Les deux courbes extrêmes correspondent à un réseau de type 2 (à deux couches de poids), la plus haute correspond à un apprentissage avec bruit, et la plus basse à un apprentissage sans bruit. La courbe centrale est produite par un réseau de type 0 (une couche, algorithme de Gram-Schmidt). Les deux courbes restantes sont celles d'un réseau de type 1, (une couche, procédure de Widrow-Hoff), la meilleure des deux correspond à un apprentissage avec bruit, et la plus mauvaise à un apprentissage sans bruit.

Les performances du réseau multicouche s'avèrent être nettement meilleures que les autres, seulement lorsque l'apprentissage s'est déroulé en présence de bruit. Au contraire, si l'apprentissage ne s'effectue que sur les formes pures, le réseau à une couche a de meilleures performances que le réseau multicouche.

En général, les performances sont meilleures avec bruit lors de l'apprentissage que sans bruit.

La raison de ceci est que, pour éliminer le bruit, le système doit se référer à un modèle de ce bruit. Deux méthodes sont possibles: se référer à un modèle a priori, prévu par le concepteur, et inclu dans le système dès le départ. La seconde méthode est de construire le modèle au cours de l'apprentissage. C'est la seconde méthode qui est employée ici lorsque l'apprentissage s'effectue en présence de bruit: le système dispose de l'information nécessaire à l'élimination du bruit, puisqu'il dispose à la fois d'un vecteur bruité et de ce même vecteur non bruité. En revanche, c'est la première méthode qui est implicitement utilisée lorsqu'aucun bruit n'est présent lors de l'apprentissage. Le système ne disposant d'aucune information sur le bruit à éliminer, le modèle de bruit auquel il est fait implicitement référence est déterminé par la nature

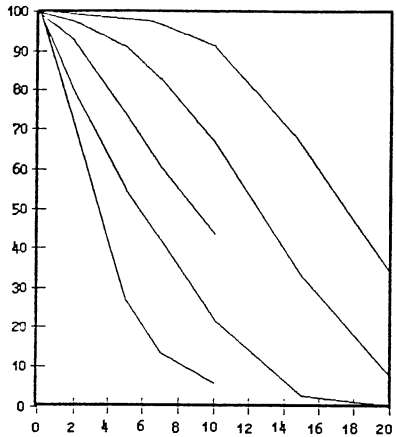


Figure 3. 24: Performances de plusieurs mémoires associatives. En abscisse: le nombre de bits inversés sur le vecteur d'entrée, en ordonnée: le pourcentage de vecteurs correctement restitués. Significations des courbes: de haut en bas, réseau de type 2 avec apprentissage bruité, type 1 avec apprentissage bruité, type 0, type 1 avec apprentissage sans bruit, type 2 apprentissage sans bruit. Chaque point est une moyenne sur 50 tirages de bruit pour chaque forme

du réseau. Ainsi, lorsqu'on utilise un réseau de type 0, et que l'on calcule la solution de norme minimum (avec la méthode de Gram-Schmidt), on fait l'hypothèse implicite que le bruit à éliminer est un bruit additif de moyenne nulle et décorrélé (matrice de covariance diagonale) [Fogelman & al. 87b]. Les performances du réseau en présence d'un bruit de nature différente seront fonction de la ressemblance entre ce bruit et un bruit additif décorrélé, de moyenne nulle. La méthode de l'inversion de bits choisis au hasard ne rentre pas dans ce modèle mais n'en est pas très éloigné, c'est pourquoi les performances du réseau de type 0 ne sont pas si mauvaises.

Il est intéressant de remarquer que l'ajout d'un bruit additif décorrélé de moyenne nulle pendant l'apprentissage d'un réseau de type 1 conduit à la solution de norme minimum (la même que celle du réseau de type 0), car c'est elle qui est optimale dans ce cas. L'utilisation d'un paramètre de décroissance exponentielle des poids (δ) aboutit au même résultat.

L'écart des performances entre les apprentissages avec et sans bruit sont moins importantes pour le réseau de type 1 que pour le réseau de type 2. L'explication est sans doute que les contraintes qu'imposent la structure mono-couche sont plus fortes que celles de la structure multicouche, bien qu'elles possèdent le même nombre de poids. Il n'y a pas de modèle de bruit a priori dans le réseau multi-couche comme il y a dans le réseau mono-couche. Le réseau mono-couche ne peut s'écarter que faiblement de ce modèle. En revanche, on dispose de plus de possibilités avec le réseau multi-couche. C'est ce qui explique les très mauvaises performances du réseau multi-couche en l'absence de bruit, aucune hypothèse implicite n'étant faite sur la nature du bruit, le système n'a aucune raison d'éliminer un type de bruit plutôt qu'un autre.

En réalité, il est surprenant que l'on puisse espérer du système qu'il élimine le bruit sans qu'on lui en donne un modèle. L'inversion de bits est sans doute l'un des plus simples à éliminer, mais on pourrait imaginer de vouloir s'affranchir des translations, ou d'autres types de transformations compliquées. Ainsi, si l'on veut stocker des mots dans le réseau et pouvoir corriger les fautes d'orthographe, il faut s'affranchir des omissions ou des insertions de symboles, et par conséquent, pouvoir effectuer une mémorisation invariante par translation. Il est évident que ce type de "bruit" ne peut pas être éliminé spontanément par le système. La confusion souvent rencontrée sur ce sujet est sans doute due à la mesure quasi-systématique du bruit à l'aide de la distance de Hamming. Il semblerait plus général d'assimiler la notion de bruit à celle de transformation, et la notion d'élimination du bruit à celle de réponse invariante par ces transformations. Ceci replonge le problème de la mémorisation associative dans celui plus général de la reconnaissance des formes.

Il est important de noter que les vecteurs utilisés, quoique linéairement indépendants, sont corrélés.

La conclusion de ces expériences est qu'à nombre de poids égal, un système à deux couches est plus performant qu'un système à une couche.

Toutefois, on doit signaler que l'implémentation de ces mémoires sur une machine massivement parallèle serait moins performante dans les cas du réseau multi-couche que dans dans le cas du réseau à simple couche, puisqu'il faut calculer l'état de la couche cachée AVANT de pouvoir calculer celui de la couche de sortie. Multiplier le nombre de couches, c'est franchir un pas en direction de la sérialisation.

3.10.2 Seconde expérience

voir appendice B. [20]

3.11 Application à un problème de diagnostic médical

Nous avons appliqué la RP à un problème de diagnostic des douleurs abdominales. Ce travail a été effectué en collaboration avec J. L. Golmard de l'INSERM au CHU de la Pitié-Salpétrière.

3.11.1 Le problème

On dispose d'une base de données réunissant les descriptions de 5765 malades arrivés aux urgences de l'hôpital avec des douleurs abdominales. Cette base de données a été collectée par l'Association de la Recherche Chirurgicale dans plusieurs hopitaux français. C'est l'une des plus importantes au monde sur le sujet.

Il s'agit évidemment d'utiliser cette base de données pour générer et tester un réseau destiné à faire un diagnostic. Le diagnostic est important, mais ce qui intéresse surtout le chirurgien, c'est de savoir s'il est nécessaire d'opérer.

La base de données

Chaque cas est décrit par 132 symptômes qualitatifs ou semi-quantitatifs, assortis de l'un des 22 diagnostics possibles. Le diagnostic fourni est supposé être le diagnostic vrai, mais on ne peut en être totalement assuré. Sur les 132 signes, environ 80% sont connus pour chaque patient en moyenne. Les diagnostics sont supposés être exclusifs,

[20] Travail effectué en collaboration avec Patrick Gallinari

mais ce n'est pas toujours le cas. Les diagnostics possibles sont les suivants (les nombres entre parenthèses sont les pourcentages de cas):

- Affection pulmonaire (0.31)
- Anévrysme de l'aorte (0.45)
- Appendicite (26.84)
- Cancer sans occlusion (0.92)
- Cholécistite (10.72)
- Colique néphrétique (3.40)
- Douleur gynéco-fonctionnelle (0.64)
- Douleur non-spécifique (24.19)
- Fausse couche (0.17)
- Foyer Suppuré (0.31)
- Grossesse extra-utérine (0.95)
- Hernie étranglée (3.01)
- Infarctus du mésentère (1.06)
- Infection urinaire (1.46)
- Kyste ovarien (1.51)
- Pancréatite (3.5)
- Perforation d'ulcère (3.87)
- Péritonite primitive ou intestinale (1.39)
- Poussée d'ulcère (2.34)
- Occlusion organique primitive (8.34)
- Salpingite (2.48)
- Sigmoïdite (2.13)

Afin d'être utilisable par le réseau, les symptômes sont recodées à l'aide de descripteurs binaires. Les signes qualitatifs sont codés avec 2 bits, et les signes quantitatifs avec autant de bits qu'il y a de valeurs possibles du descripteur. Pour ces derniers, on utilise un codage par place. Un descripteur qualitatif binaire est codé de la manière suivante: lorsque le signe est présent la configuration des deux entrées correspondantes sera $(+1,-1)$, lorsqu'il est absent $(-1,+1)$, et lorsqu'il est inconnu $(-1,-1)$.

Après recodage, les cas sont décrits par 316 bits.

Les cas

Il est important de mettre en avant les particularités de la base de données. Les descriptions ainsi que les diagnostics finaux proviennent d'origines très diverses (différents hôpitaux, différents praticiens), il en résulte des disparités dans les descriptions d'un cas à l'autre. En particulier en ce qui concerne certains signes, comme la localisation et l'importance de la douleur, dont l'évaluation repose sur la subjectivité du praticien et sur ses habitudes, ainsi que sur celles du patient.

Les deux maladies les plus fréquentes sont l'appendicite (près de 27% des cas), et les douleurs non-spécifiques (plus de 24%). Le diagnostic des appendicites est assez complexe car les symptômes peuvent revêtir des formes très atypiques. Quant aux douleurs non-spécifiques (DNS), elles regroupent les non-maladies (douleurs d'origine psychologique ou sans cause organique apparente), et les maladies bénignes ne nécessitant pas de traitement. Cette classe, contrairement aux autres, ne présente aucune unité ni cohérence, c'est une sorte de "classe poubelle". Parmi les maladies bénignes classées dans les DNS, certaines sont très difficilement distingables de l'appendicite, et conduisent de toute façon à une intervention chirurgicale en raison du risque que causerait la non-opération d'une vraie appendicite. Pour près de la moitié des opérations de l'appendicite, il s'avère a posteriori que le patient n'avait pas la maladie. Mais ce type de situation est infiniment préférable à une appendicite non-opérée.

Certaines maladies sont très peu fréquentes (10 cas de grossesses extra-utérines en tout, 18 cas de foyers suppurés), et sont difficile à prendre en compte par une méthode essentiellement statistique comme la RP.

3.11.2 Les réseaux

Pour le test du système, la base de données est divisée en deux catégories. La première contient 4027 cas et constitue l'ensemble d'apprentissage, soit environ 70% des cas. La seconde contient les 1740 cas restants et constitue l'ensemble de test.

De nombreux réseaux ont été essayés. Le plus simple est constitué d'une couche d'entrée de 316 cellules et d'une seule couche, la couche de sortie comprenant 22 cellules, soit au total 6974 poids en comptant les seuils. Les performances de ce réseau qui ne comporte pas de cellule cachée sont de 59.8% sur l'ensemble d'apprentissage, et 54.2% sur l'ensemble de test. Ceci nécessite 8 passages des 4027 exemples, avec $\lambda = 0.1$ et $\alpha = \beta = 0.5$. A titre de comparaison, nous avons utilisé la méthode Bayésienne sur le même problème, avec hypothèse d'indépendance des signes, et calcul incrémental des vecteurs de probabilité. Les performances sont de 53.5% sur l'ensemble d'apprentissage et 44.4% sur l'ensemble de test.

Nous avons utilisé une technique de construction incrémentale de réseau consistant à ajouter, après apprentissage, une couche supplémentaire au réseau ci dessus. Cette couche comporte également 22 cellules qui se substituent aux anciennes cellules de sortie. La matrice de poids entre la nouvelle couche de sortie et l'ancienne est initialement égale à la matrice identité. Tous les poids son nuls, sauf ceux qui relient une cellule de l'ancienne couche de sortie à son homologue dans la nouvelle. Ainsi, la fonction du réseau n'est pas perturbée par cette modification. Quelques passages de la base d'exemples permettent d'obtenir 61.5% sur l'ensemble d'apprentissage.

Nous avons testé de nombreuses structures à connexion complètes comportant jusqu'à 17000 poids et 2 couches cachées. Aucune n'a permi d'obtenir des résultats substantiellement meilleurs que 60% sur l'ensemble d'apprentissage et 55% sur le test. La multiplication du nombre de poids tend plutôt à faire décroître les performances sur le test. Il est difficile d'analyser les causes précises de ce résultat, on peut néanmoins avancer quelques hypothèses. La première est que notre connaissance du comportement de l'algorithme RP avec de grands réseaux est (était) encore trop faible. Il semble clair que l'application de l'algorithme RP tel-quel ne peut être efficace. Les statistiques de simulation montrent clairement de très fortes inégalités dans les vitesses de convergence des poids des différentes couches. Deux phénomènes essentiels peuvent se produire. En raison de la non-cohérence de la base de donnée, les poids ont tendance à tendre vers des valeurs assez faibles, ce qui rend également faibles les niveaux d'activité des cellules. Les gradients deviennent alors très petits, particulièrement ceux des cellules cachées. Les poids ne bougent plus. C'est le problème du plateau de l'origine, déjà mentionné. Il est possible que l'utilisation de cellules avec une fonction non-symétrique permette de résoudre ce problème. Le deuxième phénomène, très relié au précédent, est celui des ravins. Nous pensons pouvoir mettre en oeuvre des méthodes d'accélération de convergence pour résoudre ce problème, comme cela sera décrit dans la suite.

Un réseau adapté au problème

Le travail décrit ci-après est en cours de développement, nous ne présentons ici que des résultats préliminaires.

Afin d'améliorer la généralisation, nous avons travaillé sur un réseau à nombre de poids réduit, dont la structure d'interconnexion a été spécifiquement adaptée au problème. Un ensemble de diagnostics dits de haut niveau a été conçu, devant être identifiés par les cellules cachées. Les diagnostics de haut niveau (DHN) sont les suivant

- Inflammation
- Inflammation digestive
- Occlusion haute
- Occlusion basse
- Irritation péritonéale
- Péritonite
- Affection gynécologique
- Affection urinaire
- Atteinte de l'intestin
- Affection gastro-duodénale
- Occlusion
- Localisation
- Appendicite/DNS

Chaque DHN correspond à une classe d'affections particulière regroupant plusieurs maladies, avec des recouvrements possibles. Chacun de ces DHN est affecté à un groupe de 4 à 10 cellules cachées. La présence ou l'absence d'un DHN N'EST PAS CONNUE lors de l'apprentissage. Ils servent uniquement à nommer des groupes de cellules cachées, sans préjuger de leur fonction.

L'utilisation des cellules cachées, et leur relation avec les DHN, sont déterminés par la CONNECTIVITÉ du réseau. l'ensemble des interconnexions a été défini en accord avec les RELATIONS CAUSALES CONNUES entre symptômes, diagnostics de haut niveau et diagnostics finaux. Trois types de connexion sont permis:

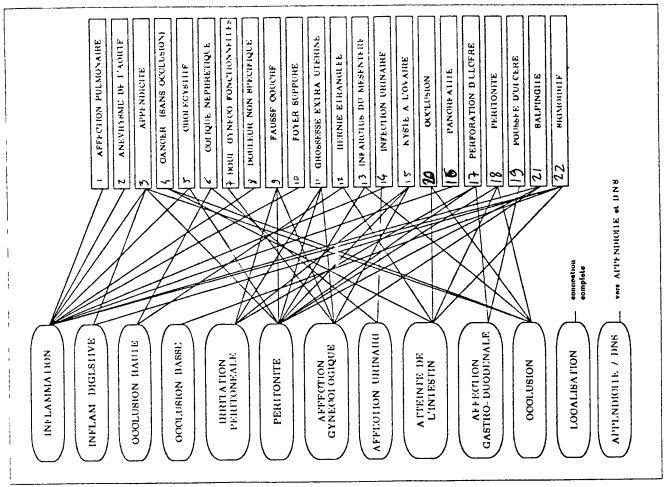


Figure 3. 25: liaisons entre d.h.n et diagnostics finaux

- des symptômes vers les diagnostics de haut niveau
- des diagnostics de haut niveau vers les diagnostics finaux
- des symptômes vers les diagnostics finaux

Dans ces trois classes, une connexion a été placée si une relation causale médicalement connue lie l'objet source à l'objet cible. Par exemple, le sexe conditionne la possibilité d'une affection gynécologique, il y a donc une connexion entre le signe sexe et le DHN "affection gynécologique". Autre exemple: les deux diagnostics "perforation d'ulcère" et "poussée d'ulcère" sont des affections gastro-duodénales, nous établissons donc une connexion entre le DHN "affection gastro-duodénale" et ces deux diagnostics. Les connexions entre les d.h.n et les diagnostics finaux sont représentées sur la figure 3. 25.

Les deux derniers DHN de la liste sont particuliers. Les cellules du DHN "Localisation" prennent leurs entrées sur les signes décrivant la localisation de la douleur. Elles sont destinées à extraire une représentation pertinente du siège de la douleur,

elles sont reliées à tous les diagnostics. Les cellules du DHN "DNS/Appendicite" sont au nombre de 10. Elles sont destinées à faciliter la discrimination entre la DNS et l'appendicite car c'est la confusion la plus fréquente. Ces cellules prennent leurs entrées sur la totalité des signes, mais ne sont connectées qu'aux deux diagnostics "DNS" et "Appendicite".

Le réseau obtenu comporte environ 3380 poids et 400 cellules dont 62 cellules cachées. Un nouveau réseau a été construit en utilisant la méthode incrémentale décrite plus haut, consistant à ajouter une couche non-perturbatrice après apprentissage. Nous avons testé ce dernier réseau dans des conditions différentes des précédents.

Après avoir calculé ses sorties, le système ordonne les cellules de sortie par niveaux d'activités décroissants. Les trois cellules les plus actives sont retenues comme diagnostics proposés.

Nous avons mesuré le pourcentage de fois que le diagnostic correct se trouve dans ces trois premières réponses. Ce pourcentage est de 75.3% sur l'ensemble d'apprentissage et 71.3% sur l'ensemble de test. La confusion de loin la plus fréquente se produit entre l'appendicite et la DNS. De nombreuses DNS sont classées en appendicite, mais comme nous l'avons signalé, ces erreurs sont inévitables. La confusion inverse ne se produit heureusement qu'exceptionnellement. Ces confusions DNS/Appendicite interviennent pour environ 40% des erreurs.

3.11.3 Conclusion partielle

Il est encore trop tôt pour tirer des conclusions définitives de ces résultats partiels. Il est difficile de dire si les possibilités d'amélioration sont à chercher dans l'algorithme lui-même ou dans la manière dont on l'utilise. Notre connaissance du comportement de l'algorithme pour de grand réseaux et des données bruitées est encore trop parcellaire. Il faudrait pouvoir effectuer des investigations systématiques. Mais celles-ci nécessiteraient l'emploi de machines vectorielles pour pouvoir être menées à bien, tant la puissance de calcul nécessaire est importante. Notre sentiment est que la limitation des performances est due à un mauvais choix des paramètres et/ou de structure de réseau, mais ce n'est qu'un avis spéculatif qu'il convient de confirmer.

Deux voies de recherches s'ouvrent à nous. D'une part l'étude du comportement de convergence de la RP sur de grands réseaux, et d'autre part, la mise au point d'outils et de techniques de conception de réseaux adaptés au problème. Il semble clair que l'apprentissage ne peut que difficilement aboutir sans une bonne dose de connaissances initiales, ces outils devraient nous permettre d'intégrer cette connaissance dans le réseau initial. On peut imaginer un compilateur de réseau pouvant générer un

graphe de connexion à partir d'une description de sa fonction en terme de règles de production. Le rôle de l'apprentissage est alors d'étendre l'applicabilité de la structure proposée.

En conclusion, le travail qui vient d'être décrit est loin d'être achevé. De nombreuses idées attendent d'être mises à l'épreuve des simulations.

Néanmoins, il est clair que c'est un problème difficile sur lequel beaucoup de techniques ont été testées, sans qu'aucune ne se soit avérée nettement supérieure aux autres. [21]. Il est possible que les performances décrites ici soient la limite de ce qu'il est possible de faire avec cette base de données, en raison de ses incohérences, indépendamment de la méthode utilisée. On peut signaler qu'en situation d'urgence, un praticien ne donne le bon diagnostic qu'environ 60% du temps, ce qui est comparable aux résultats obtenus ici. [22]

3.12 Approximation des fonctions vectorielles réelles

L'intérêt de ce sous-chapitre est purement théorique. Il vise à fournir un théorème d'existence d'un réseau universel pour les fonctions vectorielles réelles, du même type que le résultat d'universalité des réseaux binaires à deux couches. On montre que toute fonction vectorielle réelle bornée, définie sur un compact peut être approximée aussi près que l'on veut à l'aide d'un réseau à trois couches. Le réseau dont il s'agit est différent de ceux qui ont été présentés dans les sections précédentes. On a formulé quelques hypothèses permettant de simplifier le problème. Le réseau est constitué de cellules effectuant une somme pondérée de leurs entrées, mais dont la transformation de sortie est une fonction linéaire par morceaux. Cette fonction $f(a)$ est constante, égale à 0, pour $a < -1$, constante égale à $+1$ pour $a > 1$, linéaire croissante du point (-1,0) au point (1,1) (voir figure 3. 26). Nous considérerons pour simplifier, que les cellules de sortie sont linéaires.

3.12.1 en dimension 1

Il s'agit de construire un réseau pouvant calculer une fonction réelle. Le réseau est représenté sur la figure 3. 27. Il comporte deux couches de poids. La première couche sert à "paver" un intervalle de la droite réelle. Elle comporte une série de cellules allant par paires. La différence des sorties de deux cellules d'une paire est une

[21] L'équipe de l'INSERM a testé sur cette base de données un systeme expert, ainsi qu'un système d'apprentissage basé sur la logique (travaux de O Gascuel)

[22] statistiques officieuses

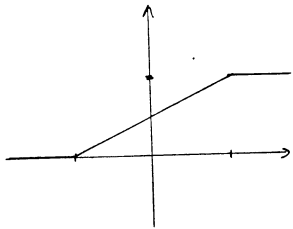


Figure 3. 26: Fonction linéaire par morceaux utilisée pour la transformation de sortie des cellules.

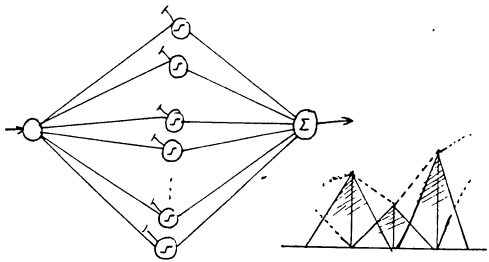


Figure 3. 27: Réseau universel pour l'approximation de fonctions à variable réelle

fonction triangulaire. Ce triangle peut être translaté sur la droite réelle en modifiant les seuils des cellules de la paire. Ainsi, nous pouvons répartir les fonctions triangulaires sur l'intervalle d'approximation. Le seuil de la première cellule d'une paire est égal à celui de la seconde cellule de la paire précédente. De cette manière, toute fonction peut être approximée linéairement en effectuant une somme pondérée des fonctions triangulaires. Les poids reliant les cellules "cachées" aux sommateurs de sortie sont égaux en valeur absolue aux valeurs de la fonction à approximer au maximum de la fonction triangulaire considérée. Les poids des deux cellules d'une même paire sont l'opposé l'un de l'autre.

Les fonctions vectorielles d'une variable réelle peuvent être approximées de la même façon en ajoutant le nombre de sommateurs désiré en sortie.

Ce résultat doit être aisément transposable aux réseaux utilisant des cellules sigmoïdes classique, nous ne l'avons cependant pas vérifé.

Un réseau possédant cette structure de départ est capable d'apprendre très vite une série d'associations: une seule présentation suffit puisque seuls deux poids (opposés l'un de l'autre) sont actifs à un instant donné. Il n'y a aucun couplage entre cellules cachées, leurs poids sont totalement indépendants. Cette structure est l'équivalent analogique de la mémoire RAM. Evidemment, aucune généralisation n'est possible, cela ressemble plus à de la mémorisation qu'à de l'apprentissage.

Le principal défaut de ce réseau est sa taille. Le nombre de cellules "cachées" croit très vite avec la précision de l'approximation.

Il est donc possible d'apprendre très rapidement, à condition d'abandonner la généralisation, et de disposer d'un important stock de cellules.

3.12.2 en dimension n

Ce résultat peut être généralisé aux fonctions vectorielles de dimension quelconque, mais il faut une couche supplémentaire. L'idée est de paver l'espace non pas avec des triangles, mais avec des pyramides, ou leur analogue multidimensionnel. Pour générer une fonction pyramidale multidimensionnelle, il suffit de disposer d'une fonction triangulaire dans chacune des dimensions, d'en faire la somme, et de transformer cette somme à l'aide de la fonction linéaire par morceaux. Les paramètres de cette transformation sont choisis de manière à ce que la somme pondérée des entrées ne soit supérieure à -1 que dans la zone où toutes les fonctions triangulaires sont non-nulles en même temps. Le réseau obtenu est représenté sur la figure 3. 28 Le nombre de cellules de la première couche cachée est égal à deux fois le nombre de fonctions triangulaires. Le nombre de cellules de la seconde couche cachée est égal au nombre

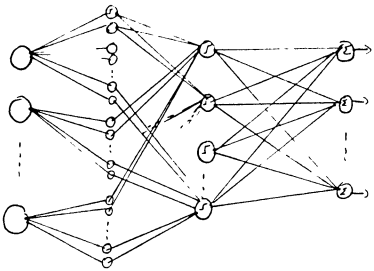


Figure 3. 28: Réseau universel pour l'approximation de fonctions vectorielles

de fonctions pyramidales. Ce réseau possède les même propriétés que son homologue en dimension 1.

3.13 Améliorer la rétro-propagation

Diverses techniques ou idées de techniques vont être présentées dans le but de comprendre les processus de convergence de la rétro-propagation. Certaines de ces idées seront développées afin de proposer des méthodes d'accélération de convergence.

3.13.1 Quelques lois d'équivalence

Lorsque l'on utilise une procédure de gradient, la trajectoire suivie par les poids sur la surface de coût est la ligne de plus grande pente. Il est clair que cette trajectoire n'est ni la plus courte, ni la plus rapide. C'est particulièrement vrai lorsque la fonction de coût comporte de nombreux ravins et plateaux. L'allure de la surface de coût, et par conséquent le temps de convergence dépendent évidemment de l'architecture du réseau et des valeurs des différents paramètres. L'allure de la fonction de coût, bonne ou mauvaise, est souvent acceptée comme une sorte de fatalité, les difficultés de convergence étant attribuées à la difficulté du problème ou à l'inefficacité de l'algorithme. Il semble au contraire qu'elles soient dues, non seulement à ces causes, mais également à des choix implicites liés à l'architecture du réseau, aux poids initiaux ou aux paramètres de la fonction non-linéaire. En effet, le temps de parcours

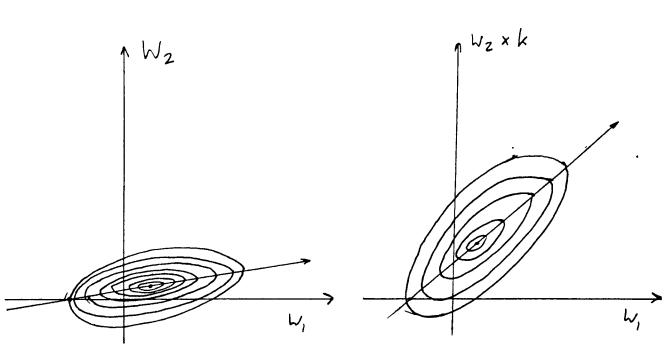


Figure 3. 29: transformation d'un ravin

de la ligne de plus grande pente dépend de la MÉTRIQUE dans laquelle est décrite cette trajectoire. Au chapitre 1, il a été décrit comment un changement de repère (de métrique) permettait d'accélérer substantiellement l'apprentissage. La quantité de calculs associée au changement de métrique optimal est en général assez importante (inversion de la matrice des dérivées secondes). Il existe cependant des situations où le gain en vitesse justifie de telles techniques. Un exemple de telle situation est lorsqu'un ravin est presque aligné avec l'un des axes du repère, ou lorsqu'il est aligné avec un sous-espace colinéaire à un ensemble d'axes. Ceci correspond à la situation où le gradient relatif à un ensemble de poids est grand (ou varie rapidement), et où le gradient relatif à d'autres poids est faible et varie lentement, c'est un cas où la matrice des dérivées secondes contient une sous-matrice diagonale, ou presque diagonale. Il est clair qu'un simple changement d'échelle des axes permet de corriger au moins partiellement la situation (voir figure 3. 29). Or cette situation peut se produire tout-à-fait fortuitement avec un réseau multicouche, comme nous allons le voir sur un exemple.

Changement du gain

Considérons un réseau multicouche constitué de cellules dont la transformation non-linéaire est la sigmoïde habituelle:

$$f(a) = m\frac{\exp ka - 1}{\exp ka + 1}$$

isolons une cellule quelconque du réseau, à laquelle nous attribuerons l'indice i. Le gradient y_i associé à cette cellule est donné par

$$y_i = -\frac{\partial C}{\partial a_i} = f'(a_i)b_i$$

où b_i est la somme des gradients des cellules aval pondérés par les poids $w_{j,i}$. Effectuons la transformation suivante: multiplions le facteur de gain k de la fonction f associée à la cellule i par un facteur r, et multiplions tous les poids de cette cellule par $1/r$. Il est clair que le comportement du réseau (sa relation entrée-sortie) sera inchangée, cette transformation le laisse invariant. Il n'en est pas du tout de même pour le comportement en situation d'apprentissage. En effet, après cette transformation, le gradient associé à la cellule sera multiplié lui aussi par r, et les incréments des poids, les $\Delta w_{i,j}$, le seront également puisqu'ils lui sont proportionnels. Il en résulte que la variation relative des poids $\Delta w_{i,j}/w_{i,j}$ sera multipliée par un facteur r^2. Deux situations sont alors possibles: soit l'algorithme diverge du fait d'une variation trop rapide des poids, soit il ne diverge pas, auquel cas on peut conclure qu'avant la transformation, la vitesse de convergence n'était pas optimale.

Changement du pas de gradient

La transformation décrite au paragraphe précédent est en fait totalement équivalente à une multiplication par r^2 du pas de gradient λ_i associé à la cellule considérée. En effet, ceci ne modifie pas le comportement du réseau, sauf en situation d'apprentissage où la variation relative des poids est multipliée par r^2 comme précédemment.

Changement de l'amplitude des cellules amont

Les deux transformations précédentes sont encore équivalentes à une troisième. Multiplions les paramètres d'amplitude m des cellules fournissant leur sortie à la cellule i par r, multiplions également tous les poids de la cellule i par $1/r$. Nous obtenons une transformation équivalente aux précédentes [23] le comportement est inchangé, excepté la variation relative des poids qui est multipliée par r^2.

Conclusion

Les transformations qui viennent d'être décrites sont équivalentes à un changement d'échelle sur les axes de l'espace des poids. Ainsi des ravins peuvent se trouver plus

[23] Il faudrait également modifier les poids de toutes les cellules "voyant" les cellules affectées par la transformation

escarpés ou, au contraire, atténués après de telles transformations. Dans la plupart des cas, la recherche de simplicité conduit à choisir le même gain, la même amplitude et le même pas de gradient pour toutes les cellules, or, non seulement ce choix ne se justifie pas, mais il peut même se révéler désastreux. Il n'y a aucune chance pour que la métrique associée à ce choix soit bonne. nous devons donc trouver des techniques permettant de ne pas faire reposer les performances du réseau sur des choix aussi arbitraires. Il s'agit de rendre le comportement du réseau indépendant de ces choix qui conditionnent l'allure de la surface de coût. On peut déjà en esquisser les principales caractéristiques. Si l'on considère que c'est la variation relative des poids qui doit rester invariante, et que, des trois paramètres utilisés ci-dessus, nous choisissons de faire varier le pas de gradient λ, nous pouvons dire que, dans une certaine mesure, le λ, associé à une cellule devra être inversement proportionnel au carré de la dérivée de la fonction f, et au carré des entrées de la cellule.

3.13.2 Une approximation de l'algorithme de Newton

L'exemple de la section précédente a bien montré la nécessité "d'égaliser" les vitesses de convergence des différentes cellules. Ces vitesses sont en effet totalement dépendantes de paramètres arbitraires, comme le rapport Fan-Out/Fan-In, le gain associé à chaque cellule ou la norme moyenne des entrées d'une cellule. Dans des situations réelles, les valeurs de ces paramètres ne sont pas aisément contrôlables, et peuvent tout à fait avoir des valeurs interdisant un bon comportement en situation d'apprentissage. Pour s'affranchir de ce problème, il faut modifier la procédure d'apprentissage de manière à la rendre plus indépendante de ceux-ci. L'influence de tout ces paramètres se résume en un seul: l'allure de la surface de coût. Les problèmes de convergence sont principalement causés par des ravins ou des plateaux. Comme nous l'avons vu, dans quelques cas, un simple changement d'échelle sur un groupe de coordonnées suffit à améliorer notablement le l'évolution des poids, c'est exactement ce qui se passe lorsqu'on multiplie le gain (ou le λ) d'une cellule par un facteur bien choisi. Il a été montré au chapitre 1 avec l'analyse de la fonction de coût d'une cellule linéaire qu'il existait une transformation des coordonnées de l'espace (des poids ou des entrées) qui permettait d'assurer une convergence en un coup. Cette transformation est la composée d'une translation, d'une rotation, et d'une homothétie, et rend les calculs relatifs à un repère attaché aux axes principaux de la paraboloïde de coût. Au niveau des calculs, cela se traduit par la multiplication du gradient par l'inverse de la matrice des dérivées secondes de la fonction de coût.

Cette méthode n'est pas directement transposable aux réseaux multi-couches et à

la rétro-propagation, et ceci pour deux raisons principales. Premièrement, le volume des calculs, il faut tout d'abord estimer la matrice Hessienne de la fonction de coût, opération rendue complexe par la non linéarité du réseau. Il faut ensuite l'inverser, ou estimer itérativement son inverse, ceci est totalement hors de portée lorsqu'on réalise que c'est une matrice carrée dont la dimension est égale au nombre de poids. La deuxième raison (s'il était nécessaire d'en donner une deuxième) est que la surface de coût n'est pas convexe. Ceci a pour conséquence que la matrice Hessienne peut être non positive, ou même nulle. Inutile de préciser que les conséquences peuvent être catastrophiques pour la convergence. Le problème vient essentiellement du fait que la procédure de Newton est faite pour trouver les zéros du gradient, que ceux-ci correspondent à des minima ou des maxima.

Une grossière approximation

Contrairement à ce que 'pourrait faire penser ce qui précède, cette démarche n'est quand même pas sans espoir. On peut imaginer une approximation très grossière de la procédure de Newton dont les qualités seraient suffisantes. L'inspiration est la même que lorsqu'au chapitre 1 nous avons présenté les méthodes de calcul d'un λ optimal permettant de s'affranchir des variations d'amplitude du signal d'entrée. La valeur de λ était calculée à l'aide de la formule

$$\lambda = \frac{\mu_1}{\mathcal{M}[X^T X] + \mu_2}$$

où μ_1 et μ_2 sont des constantes, et $\mathcal{M}[X^T X]$ une estimation de la moyenne du carré de la norme du vecteur d'entrée. Nous avons vu ensuite que lorsqu'il existait de fortes disparités entre les amplitudes des composantes du vecteur d'entrée, on pouvait appliquer cette normalisation indépendamment sur chaque composante.

Nous allons opérer le même genre d'approximation dans le cas des réseaux, mais au lieu d'opérer une normalisation sur les poids, nous opérerons sur les gradients par rapport aux a_i, c'est à dire les y_i. Nous ne nous intéressons qu'au calcul des termes diagonaux de la matrice des dérivées secondes, c'est à dire les:

$$\gamma_i = \frac{\partial^2 C}{\partial a_i^2}$$

Nous allons tenter d'établir une relation de récurrence pour les γ_i de la même manière que pour les y_i. Nous omettrons d'écrire l'opérateur de moyennage pour ne pas alourdir les notations, C représente le carré de l'erreur instantanée, et non le critère moyen. Nous faisons l'hypothèse que C ne dépend de a_i que par l'intermédiaire des a_j $j \neq i$,

ce qui nous permet d'écrire

$$\gamma_i = \frac{\partial^2 C}{\partial a_i^2} = \sum_j \frac{\partial^2 C}{\partial a_i \partial a_j} \frac{\partial a_j}{\partial a_i}$$

où la somme est prise sur les j différents de i. Mais nous savons que

$$\frac{\partial C}{\partial a_i} = \sum_k \frac{\partial C}{\partial a_k} \frac{\partial a_k}{\partial a_i}$$

nous tirons

$$\gamma_i = \sum_j \sum_k \left(\frac{\partial^2 C}{\partial a_j \partial a_k} \frac{\partial a_k}{\partial a_i} + \frac{\partial C}{\partial a_k} \frac{\partial^2 a_k}{\partial a_i \partial a_j} \right) \frac{\partial a_j}{\partial a_i} \tag{3.94}$$

Nous allons maintenant faire deux hypothèses outrageusement simplificatrices. La première (qui a déjà été faite implicitement) est la suivante

$$j \neq k \Rightarrow \frac{\partial^2 C}{\partial a_j \partial a_k} = 0$$

nous négligeons les termes non-diagonaux de la matrice des dérivées secondes de C par rapport aux a. La seconde hypothèse est que l'on ne tiendra compte que des dépendances directes des a_i entre eux. Autrement dit, on fait l'hypothèse suivante:

$$i \neq j \Rightarrow \frac{\partial^2 a_k}{\partial a_i \partial a_j} = 0$$

Avec ces deux hypothèse, l'équation se transforme en

$$\gamma_i = \sum_k \left(\frac{\partial^2 C}{\partial a_k^2} \left(\frac{\partial a_k}{\partial a_i} \right)^2 + \frac{\partial C}{\partial a_k} \frac{\partial^2 a_k}{\partial a_i^2} \right)$$

Cette expression est développée à l'aide des identités suivantes

$$\frac{\partial a_k}{\partial a_i} = f'(a_i) w_{ki}$$

$$\frac{\partial^2 a_k}{\partial a_i^2} = f''(a_i) w_{ki}$$

l'expression de γ_i devient

$$\gamma_i = \sum_k \left(\frac{\partial^2 C}{\partial a_k^2} w_{ki}^2 f'(a_i)^2 + \frac{\partial C}{\partial a_k} w_{ki} f''(a_i) \right)$$

soit

$$\gamma_i = f'(a_i)^2 \sum_k w_{ki}^2 \gamma_k - f''(a_i) b_i \tag{3.95}$$

en utilisant les notations habituelles suivantes

$$y_i = -\frac{\partial C}{\partial a_i} \qquad b_i = \sum_k w_{ki} y_k$$

L'équation 3.95 concerne les cellules cachées, il nous reste à calculer les dérivées secondes attachées aux cellules de sortie. Si i est l'indice d'une cellule de sortie

$$\gamma_i = \frac{\partial^2 C}{\partial a_i^2} = 2(f'(a_i)^2 - \epsilon_i f''(a_i))$$

avec la notation usuelle

$$\epsilon_i = d_i - x_i$$

Il faut maintenant calculer les dérivées secondes par rapport aux poids. Elles sont approximativement données par [24]

$$\frac{\partial^2 C}{\partial w_{ij} \partial w_{ik}} = \frac{\partial^2 C}{\partial a_i^2} \frac{\partial a_i}{\partial w_{ij}} \frac{\partial a_i}{\partial w_{ik}}$$

soit

$$\frac{\partial^2 C}{\partial w_{ij} \partial w_{ik}} = \gamma_i x_j x_k$$

Nous pouvons appliquer un algorithme de moindres-carrés récursifs indépendamment sur chaque cellule, en estimant récursivement les inverses des matrices R_i dont le terme d'indices j,k est donné par l'équation précédente. De la même manière qu'au chapitre 1, on peut simplifier de nouveau et approximer ces matrices par les termes de leurs diagonales, en négligeant les termes non-diagonaux. Il est également possible d'adopter l'attitude prudente qui consiste à normaliser par la trace, ce qui, d'une certaine manière, revient au même. Dans ce dernier cas nous disposons d'un terme normalisateur par cellule, qui pour la cellule d'indice i est égal à

$$\gamma_i \sum_j x_j^2$$

où la somme est prise sur les indices des cellules connectées à la cellule i, la somme est donc le carré de la norme de l'entrée de la cellule i. C'est naturellement par une estimation de la moyenne de cette quantité qu'il faudra diviser l'incrément des poids. Au temps h le pas de gradient associé à la cellule i est donné par

$$\lambda_i(h) = \frac{\mu_1}{|\sigma_i(h)| + \mu_2} \tag{3.96}$$

[24] approximativement, eu égard aux hypothèses préalablement formulées

avec $\sigma_i(h)$ estimé à l'aide de

$$\sigma_i(h) = (1-\alpha)\sigma_i(h-1) + \alpha\gamma_i(h)\sum_j x_j^2(h)$$

où la somme est toujours prise sur les cellules d'entrées de la cellule i. α est une constante déterminant la vitesse de réaction du facteur de normalisation à un changement d'environnement. Il est nécessaire de prendre la valeur absolue de σ et d'ajouter une "valeur talon" μ_2 dans l'équation 3.96 de manière à assurer la convergence vers un minimum (et non vers l'extremum le plus proche au cas où la dérivée seconde est négative), et de manière à empêcher λ de prendre des valeurs trop importantes lorsque la dérivée seconde est petite.

Un détecteur de ravins et de plateaux

En plus de procurer une relative indépendance de l'algorithme vis à vis des choix architecturaux, cette technique est susceptible de permettre une amélioration de la convergence dans certaines zones de la surface de coût, en particulier dans les ravins et les plateaux, qui sont typiquement des régions où la dérivée seconde est faible. On doit néanmoins se prévenir, par des solutions ad-hoc, des divergences que peuvent entraîner des zones à dérivée seconde faible.

L'approximation de la dérivée seconde semble apporter une réponse partielle à plusieurs problèmes de convergence dont nous avons pu constater l'importance au cours des travaux précédents. Parmi ceux-ci, le problème de l'amplitude relative des gradients dans les couches, les problèmes liés à un niveau d'activité ou des poids trop faibles.

Il reste cependant à confirmer ces intuitions par des simulations systématiques.

3.13.3 Une autre manière de propager les erreurs: diriger la sélectivité

L'algorithme de rétro-propagation possède l'avantage d'être très simple, mais il est des situations où l'on préfère utiliser un algorithme un peu plus compliqué, si la contrepartie est une amélioration de sa contrôlabilité.

Un des problèmes que doit résoudre l'algorithme d'apprentissage est la division du travail parmi les cellules. Il s'agit d'affecter chaque cellule à une tâche particulière en évitant les redondances et les omissions. Parfois, effectuer cette affectation revient à choisir de quelle(s) sortie(s) globale(s) une cellule doit "s'occuper". Nous allons

décrire une technique permettant de contrôler ce type d'affectation manuellement ou automatiquement.

Les gradients des sorties

Au lieu de calculer les gradients de la fonction de coût par rapport aux poids, ou par rapport aux a_n il est possible de calculer les dérivées partielles de CHAQUE SORTIE du réseau indépendamment. En effet, la dérivée partielle d'une sortie particulière par rapport aux a_i obéit à la même relation de récurrence que la dérivée partielle du coût, c'est à dire

$$\frac{\partial s_k}{\partial a_i} = \sum_j w_{ji} \frac{\partial s_k}{\partial a_j}$$

où s_k est la sortie d'indice k. Nous pouvons donc affecter à chaque cellule un vecteur de dérivées partielles G_i dont les composantes sont définies par

$$g_{ik} = \frac{\partial s_k}{\partial a_i} = \sum_j w_{ji} g_{jk}$$

Le calcul de ces variables nécessite autant de rétro-propagations qu'il y a de sorties. Les gradients du coût sont donné par

$$y_i = -\frac{\partial C}{\partial a_i} = -\sum_k \frac{\partial C}{\partial s_k} g_{ik} = \sum_k \epsilon_k g_{ik} \tag{3.97}$$

où ϵ_k est l'erreur associée à la sortie k. Pour calculer les gradients du coût, il faut donc faire le produit scalaire du vecteur d'erreur et du vecteur G_i pour chaque cellule.

Diriger la sélectivité

Il est possible d'utiliser cet artifice pour orienter l'utilisation d'une cellule vers l'une ou l'autre des sorties. Il suffit en effet de ne conserver dans la somme 3.97 que les termes correspondant aux sorties sélectionnées. Une procédure automatique peut même être utiliser pour éliminer de la somme les termes trop petits, accentuant ainsi la différenciation des fonctions réalisées par les cellules.

3.13.4 Plonger le problème dans un espace plus grand: les réseaux virtuels

Tout problème d'optimisation dans un espace de dimension donnée peut être résolu dans un espace de dimension plus grande dans lequel les solutions sont soumises à des contraintes. Il est possible d'appliquer ce principe à la RP, en espérant que les dimensions supplémentaires du nouvel espace permettront de gommer certains défauts de

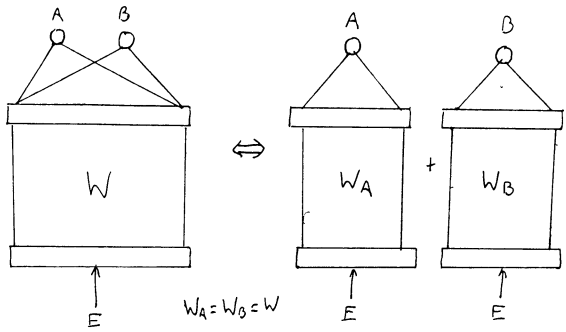


Figure 3. 30: Transformation d'un réseau en deux réseaux virtuels

l'espace d'origine. Raisonnons sur un exemple. Nous disposons d'un réseau contenant une couche cachée et deux cellules de sortie. Le problème principal de l'apprentissage sera de choisir les fonctions des cellules cachées de manière à satisfaire les deux cellules de sortie. Les contraintes à respecter, et par conséquent la difficulté à trouver une solution, sont plus importantes lorsqu'il y a deux cellules à satisfaire que lorsqu'il n'y en a qu'une. L'idée est de considérer les problèmes relatifs aux deux cellules comme deux problèmes indépendants. Avec le réseau original, on construit deux "réseaux virtuels". Le premier (appelé réseau A) est identique au réseau original sans la deuxième cellule de sortie, le second (réseau B) est identique au réseau original sans la première cellule de sortie (voir figure 3. 30). Appelons W_A et W_B les matrices de poids des cellules cachées des réseaux A et B respectivement. Nous pouvons calculer les poids du réseau original en calculant tout d'abord les poids des deux réseaux A et B indépendamment, puis en imposant la contrainte $W_A = W_B$. Ceci peut être effectué à l'aide d'une technique de pénalisation consistant à rajouter dans les fonctions de coût des deux réseaux virtuels un terme du type

$$\sum_{ij} (w_{Aij} - w_{Bij})^2$$

affecté d'un coefficient que l'on fait croître au cours de l'apprentissage.

Cette technique qui consiste à ajouter une contrainte d'égalité sur les poids en cours d'apprentissage peut être utilisée dans d'autres circonstances, en particulier pour les problèmes où l'on désire réduire le nombre de paramètres du système. C'est le cas lorsque l'on veut améliorer la généralisation. C'est également le cas lorsqu'on dispose

d'un réseau pseudo-itératif, où l'on peut envisager de doter chaque pseudo-couche d'une matrice de poids différente, et de leur imposer la contrainte d'égalité après une première phase d'apprentissage.

On peut espérer que le nombre de minima locaux de la fonction de coût se trouvera réduit par cette technique du découplage, l'addition de dimensions supplémentaires permettant de contourner les obstacles. Par ailleurs il est bien connu que l'apprentissage est d'autant plus rapide (et la généralisation d'autant plus mauvaise) qu'il y a de poids dans le réseau. On peut donc envisager l'utilisation de cette technique lorsqu'il est nécessaire d'effectuer un apprentissage rapide, même au prix d'une mauvaise généralisation, susceptible d'être raffiné par la suite au cours du processus d'égalisation des poids. On peut assimiler ce type de technique à une sorte "d'apprentissage par sélection" où le nombre de paramètres libres décroît au cours du temps sans que l'architecture du système en soit modifiée.

3.14 Les problèmes ouverts

Les diverses techniques explorées au cours de ce chapitre ont permis de développer des idées nouvelles et de tenter de résoudre des problèmes pratiques. Néanmoins de très nombreuses questions restent encore en suspens. Certaines sont d'ordre technique, spécifique à la rétro-propagation ou aux méthodes connexionnistes, d'autres présentent un caractère plus fondamental.

3.14.1 Quelles sont les lois d'échelle

La plus importante des questions spécifiques concerne les lois d'échelle. Nous n'avons aucun résultat théorique ou expérimental (et systématique) sur la loi de croissance du temps d'apprentissage en fonction de la taille du réseau et de la "complexité" du problème. Les moyens théoriques permettant de répondre à cette question sont pour l'instant hors d'atteinte. S'il s'avérait que le temps d'apprentissage était exponentiel, ou même polynômial de degré élevé en fonction de la taille du réseau, il ne resterait qu'à abandonner la R.P. La réponse est certainement plus complexe que cela. Il est possible d'imaginer des structures permettant un apprentissage très rapide (et très coûteux en matériel) sans aucune généralisation, équivalent connexionniste de la R.A.M. Les simulations semblent confirmer qu'à mesure que la taille du réseau s'approche de la taille minimum nécessaire à la résolution du problème, le temps d'apprentissage croît, et la généralisation s'améliore. Il y aurait donc un compromis entre la qualité de la représentation interne générée et le temps de son élaboration. D'autre part, il

est clair que dès que la tâche est complexe, l'ordre et la fréquence de présentation des exemples deviennent primordiaux pour la qualité de la généralisation. Il devient nécessaire d'utiliser des rudiments de "pédagogie" [25]. La signification du mot "pédagogie" est ici très particulière, il s'agit tout au plus de construire une suite d'ensembles d'apprentissage dont les surfaces de coût successives conduiront à la solution recherchée. Il faut commencer par une surface de coût très lisse, et comportant peu de minima locaux, correspondant à un ensemble avec peu d'exemples. Puis, à mesure que l'on approche de la solution, il faut ajouter des exemples de plus en plus atypiques ou bruités, dont l'effet sera de rendre la surface de coût plus irrégulière, et les minima mieux localisés.

3.14.2 Existe-t-il une méthode d'apprentissage universelle

Ce problème des lois d'échelle pose une question plus fondamentale: existe-t-il une méthode d'apprentissage universelle? Tout porte à croire que la réponse à cette question est non.

Il semble que l'apprentissage et la généralisation soient des problèmes "mal posés", pour lesquels il n'existe pas de critère permettant de déterminer si une solution est meilleure qu'une autre. D'un point de vue objectif, toute généralisation d'une collection d'exemples est valable, seules des hypothèses a priori sur la nature de la fonction permettent de juger de la qualité d'une généralisation. Prenons un exemple: considérons les deux séries d'objets représentés sur la figure 3. 31. La question à laquelle il faut répondre pour être capable de généraliser correctement est: qu'est-ce qui distingue la classe A de la classe B. En tentant de répondre à cette question, nous faisons implicitement des hypothèses, nous ignorons les irrégularités et la couleur des traits, ainsi que la position des objets et bien d'autres primitives qui pourraient être significatives et permettre de discriminer la classe A de la classe B [26]. Nous ne retenons en fait que leurs formes en tant que polygônes. Nous faisons donc une hypothèse très forte sur l'espace des possibilités à explorer.

Il est d'autant plus facile d'apprendre que l'espace des possibilités est restreint. Lorsqu'il s'agit de choisir entre quelques possibilités de représentations, une méthode exponentielle (par exemple exhaustive) peut être suffisante. Mais en général, l'espace des possibilités est énorme. Dans un réseau, l'espace des possibilités est la zone de

[25] avec tous les guillemets dont doit être entouré ce mot dans ce contexte

[26] l'épaisseur d'encre sur la feuille, la surface totale du trait, la courbure maximale des segments, la longueur du plus grand segment, la masse d'encre utilisée, etc

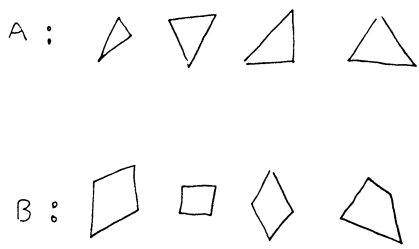


Figure 3. 31: Deux classes d'objets A et B

l'espace des poids accessible [27].

Les capacités de généralisation sont directement reliées à la taille de l'espace des possibles que l'on se donne a priori. Cover [28] donne un théorème à ce sujet qui relie la généralisation (mesurée par les gains d'une martingale) à la taille de l'espace des fonctions potentiellement calculables par l'apprentis. Il montre que lorsqu'il s'agit d'apprendre une fonction booléenne définie sur n exemples, et que l'espace des possibles est l'espace des 2^n fonctions possibles sur ces n exemples, l'espérance des gains de la martingale est nulle, en d'autres termes, aucune généralisation ne peut être espérée. Le principe de la martingale est de parier que la réponse correcte associée à un nouvel exemple est "1" (ou "0") en proportion des nombres de fonctions disponibles dans l'espace des possibles qui produisent "1" (ou "0") sur cet exemple. On conçoit aisément que l'espérance des gains sera d'autant plus élevée que le stock de fonctions initial est moins important.

Plus les hypothèses sur la nature du problème sont fortes (plus l'espace des possibles est réduit), plus celui-ci est simple à résoudre. Ce théorème semble être en faveur de l'hypothèse selon laquelle il n'existerait pas de règle d'apprentissage universelle.

Ceci confirme nos intuitions concernant la nécessité de concevoir des réseaux adaptés à chaque problème, et de mettre dans ces réseaux le maximum de con-

[27] c'est à dire toutes les configurations de poids de coût inférieur au coût actuel et situées dans le même bassin.

[28] "Generalization on patterns using Kolmogorov complexity", référence inconnue, copie de l'article disponible auprès de l'auteur.

naissances disponibles sur le domaine, afin de contraindre l'espace de recherche le plus possible. Des efforts doivent être concentrés sur ce problème.

3.15 Conclusion

L'algorithme de rétro-propagation a ouvert la voie à de nombreux travaux de par le monde. Le nombre d'équipes travaillant sur le sujet est en croissance rapide. Néanmoins, bien que de nombreuses applications intéressantes soient en cours d'étude, aucune n'a encore donné de résultats définitivement meilleurs que d'autres techniques.

L'une des applications les plus prometteuses que nous n'avons malheureusement pas abordée, est le traitement adaptatif du signal et l'identification de systèmes non-linéaires. D'autres applications sont en cours de développement, citons la reconnaissance de la parole, le traitement de signaux sonar ou radar, la compression d'images (en utilisant des "diabolos") ou le contrôle moteur en robotique.

Une meilleure compréhension théorique et pratique des processus de convergence seront nécessaires pour envisager l'utilisation de cet algorithme sur de très grands réseaux (de quelques dizaines ou centaines de milliers de poids). Ces réflexions devront aller de pair avec l'implantation d'un simulateur sur une machine vectorielle ou cellulaire.

L'étude systématique de l'algorithme avec des réseaux de taille raisonnable passe nécessairement par son implantation sur une machine parallèle. De telles réalisations sont déjà en fonctionnement aux USA sur des machines vectorielles et sur une Connection Machine. Un processeur optique réalisant la rétro-propagation est également à l'étude à CalTech [29].

Il semble nécessaire de développer des outils permettant la définition et la manipulation de réseaux, permettant d'intégrer de la connaissance dans le système avant apprentissage, un peu à la manière des outils de développement de systèmes experts.

Il est clair que des applications intéressantes ne pourront être réalisées sans l'intégration des modèles connexionnistes à des systèmes plus vastes, mettant en jeu des techniques très diverses.

[29] Communication de Wagner et Psaltis à "Neural Network for computing", Snowbird, Utah, Avril 1987

Chapitre 4

Manuel du simulateur HLM

4.1 présentation générale

HLM est un ensemble logiciel permettant la création et la simulation de réseaux adaptatifs utilisant l'algorithme de rétro-propagation et ses variantes. Il est écrit en Pascal et existe actuellement en trois versions tournant sur 3 machines:

- sur PRIME en Pascal PRIME
- sur IBM-PC en Turbo-Pascal
- sur AMIGA (68000) en Pascal ISO

L'ensemble se compose:

- du simulateur
- d'un ensemble d'utilitaires de génération et de manipulation de réseaux
- du langage de définition hiérarchique de réseaux GRL (version PRIME uniquement)

L'interface utilisateur d'HLM est très rudimentaire mais elle a le mérite d'exister. On apréciera en particulier la présence de l'aide en ligne. Cependant, HLM n'a pas été conçu comme un produit commercial, tout au plus pour être utilisable par quelqu'un d'autre que son concepteur.

Une session typique se déroule de la manière suivante: tout d'abord il faut disposer d'une base d'exemples de formes d'apprentissage qui pourra selon le cas, soit être construite manuellement à l'aide de votre éditeur préféré, soit être constituée de données expérimentales ou encore provenir d'un autre programme. Dans le premier cas il suffira d'écrire une série d'exemples et de réponses associées en se conformant au format des fichier .IMA décrit plus loin. Dans le second cas il existe deux possibilités. La première est, bien sûr, de transformer le ou les fichiers pour se conformer au format .IMA, la deuxième est d'écrire une procédure de lecture spécifiquement adaptée au problème, et de l'intégrer à HLM. Cette dernière situation ne devrait se présenter que rarement.

Dans un second temps il faudra générer un réseau adapté à l'application. Là encore plusieurs solutions. Si le réseau possède une architecture classique (ex: réseau en couches totalement connecté, poids initiaux aléatoires) il suffira d'utiliser l'un des programmes de génération de réseau. Si ce n'est pas le cas il faudra écrire un programme de génération en utilisant la bibliothèque NETMAC. Un autre possibilité est d'écrire (à la main ou par programme) un fichier de description de réseaux ascii au

format .NET et de la convertir au format utilisable par HLM à l'aide de l'utilitaire de conversion prévu à cet effet.

Ensuite viendra la simulation proprement dite. Elle peut se dérouler soit en mode interactif soit en mode non-interactif. Dans les deux cas il s'agira d'activer une séquence de commandes HLM pour successivement charger un réseau, charger un ensemble d'apprentissage, initialiser les paramètres du modèle, lancer des itérations d'apprentissage, etc.... Le réseau résultant de l'apprentissage pourra être sauvegardé et réutilisé pour d'autres simulations et/ou comme partie d'un réseau plus grand.

dans sa version actuelle, HLM permet la simulation de réseaux quelconques, y compris avec boucles. Cependant certaines fonctionnalités (en faible nombre) supposent l'absence de boucle dans le réseau. Ce n'est pas trop restrictif dans la mesure où les réseaux à boucles se rencontrent rarement. Il existe sur PRIME une version appelée LHLM qui permet la simulation de réseau itératifs utilisant le "dépliement spatial" (voir chapitre 3) Cette version est, en fait plus générale que cela, puisqu'elle permet d'imposer à deux arcs quelconques du réseau d'avoir le même poids. Ce peut être utile pour générer des réseaux dont la structure est invariante par translation où par toute autre transformation géométrique. HLM et LHLM ont les mêmes fonctionnalités, mais ils diffèrent sur deux points importants: Le format des fichiers de définition de réseau sont différents, et, surtout, HLM est plus rapide que LHLM à l'exécution.

Un réseau est décrit par 3 fichiers dont les noms sont constitués du nom du réseau suivi d'une des extensions ".STR", ".IO" ou ".POI" Par exemple, le réseau BZZT sera décrit par les trois fichiers BZZT.STR, BZZT.IO, BZZT.POI . Ces fichier ne sont pas en ASCII, et par conséquent, pas directement lisibles. Il existe cependant deux utilitaires (appelé "HLMToNet" et "NetToHLM" dans la version Turbo-Pascal) qui permettent la conversion de fichiers réseaux en un format ASCII "en clair", ainsi que l'opération inverse. Le fichier BZZT.STR contient la définition du mode d'itération utilisé pour la simulation. Il spécifie l'appartenance de chaque cellule aux blocs pour une itération séquentielle par bloc. C'est le cas dans un réseau en couches où chaque bloc contient une couche ou une partie de couche. Le fichier BZZT.IO contient la liste des cellules possédant une entrée externe et la liste des cellules possédant une sortie externe. Une même cellule peut éventuellement être à la fois entrée et sortie. Le fichier BZZT.POI contient une liste des poids. chaque poids est représenté par un triplet indiquant la cellule amont, la cellule aval, et la valeur du poids. Le format précis de chacun de ces fichiers est décrit au paragraphe ??. Pour les descriptions de réseau au format LHLM, le fichier .POI est remplacé par deux fichiers .ARC et .WEI.

4.2 Manuel d'utilisation

Ce manuel décrit la version Turbo-Pascal pour IBM-PC et compatibles. Il existe quelques différences mineures d'une version à l'autre dues aux différences des systèmes d'exploitation ainsi qu'aux possibilités des différentes implémentations du langage PASCAL.

4.2.1 Fonctionnalités

A son lancement, HLM affiche un message puis une invite (un "prompt"): "HLM >" invitant l'utilisateur à taper une commande. Les commandes sont de plusieurs types:

- les commandes de contrôle
- les commandes de chargement/sauvegarde
- les commandes de positionnement/visualisation de paramètres

Une description détaillée de chacune des commandes est donnée au paragraphe 4.2.3 L'appel d'HLM s'effectue par la commande suivante (les arguments entre [ ] sont optionnels):

HLM [fichier_apprentissage [fichier_startup]]

Le premier argument est le nom du fichier dans lequel seront écrits les résultats d'évaluation de performances du réseau. Son nom par défaut est "DEFAULT.APP". Ce fichier pourra ensuite être utilisé pour tracer des courbes d'apprentissage. Le deuxième argument est le nom d'un fichier de commandes qui sera éxécuté au lancement d'HLM. Si cet argument est absent, HLM recherchera dans le répertoire courant le fichier de commandes STARTUP.HLM, et l'éxécutera s'il le trouve. Dans le cas contraire, un message d'erreur sera affiché dont il n'est pas nécessaire de tenir compte.

4.2.2 Syntaxe des commandes

Les commandes obéïssent à la syntaxe suivante:

commande [[[option1]option2]...]

Les commandes peuvent être tapées en majuscules ou minuscules et les espaces en début de ligne sont ignorés. Elles sont parfois longues à taper, c'est pourquoi il est possible de les abréger. Pour invoquer un commande, il suffit de frapper le nombre de caractères juste nécessaire à sa discrimination des autres commandes. Par exemple, si l'utilisateur tape RESE, la commande RESEAU sera effectuée car aucune autre commande ne commence par RESE.Par contre s'il tape RE, la situation est ambiguë

car il existe également la commande REMARQUE. Dans ce cas, le système demande de confirmer que la commande désirée est REMARQUE, si la réponse de l'utilisateur est positive REMARQUE est effectuée, dans le cas contraire le système rend la main. L'entrée d'une commande non existante provoque l'impression d'un message d'erreur: "commande inconnue".

Les options doivent être séparées de la commande par un ou plusieurs espaces et sont généralement constituées chacunes d'un unique caractère. Plusieurs options peuvent être tapées à la suite avec ou sans espaces entre elles. Les majuscules et minuscules ne sont pas différenciées.

Généralement, une commande a besoin d'un ou plusieurs arguments. Ceux-ci peuvent être de deux types: numériques ou alphabétiques. Lorsque une commande nécessite un argument, celui-ci est demandé à l'utilisateur après l'invocation de la commande. Toutefois lorsque l'argument d'une commande est de type alphabétique (ex: un nom de fichier) il est possible de le taper sur la même ligne que la commande en lieu et place des options.

Les commandes peuvent être lues au terminal ou sur un fichier. Le passage du mode terminal au mode fichier est assuré par la commande FICHIER (ou FIC) suivie du nom du fichier contenant les commandes, le retour au mode terminal est effectué par CONSOLE (voir paragraphe 4.2.3 pour une description plus complète).

ATTENTION: actuellement, seule la version turbo-pascal traite les erreurs d'entrées-sorties

4.2.3 Description des commandes

Voici une liste des commandes de contrôle disponibles. Les caractères discriminants sont en majuscule.

Help Aide en ligne.

REMarque Insertion d'un commentaire

FIChier Redirige le flot d'entrée sur un fichier

Console Redirige le flot d'entrée sur la console

FIN Termine la session

Les autres commandes disponibles sont les suivantes (les options sont entre crochets à la fin des commentaires):

AFfichage Définit les informations affichées à chaque itération [c]

APprend Effectue des cycles d'apprentissage [u][t]

ALphabeta Change la viscosité et l'acceleration et le decay [d][m]

Bruit Mode de bruitage des entrées

Debugger Inverse le mode "déverminage"

ENsemble Modifie l'ensemble de travail [t]

EPsilon Modifie le pas de convergence [t][n]

ETat Visualise l'etat du réseau

EXecute Effectue des cycles d'apprentissage

FOnction Change la fonction et/ou ses paramètres [c]

Oublie Initialise et/ou modifie les poids [f][s][n]

PArametres Visualise les paramètres du reseau

PEDagogie Procédure de choix des formes

PERformance Calcule le taux de reconnaissance [t][c][i][f]

PREsentation Mode de présentation aléatoire ou en séquence

RESEau Charge un réseau

RESTaure Restaure une configuration de poids

SAuve Sauve une configuration de poids

STatistique Statistiques sur l'état du réseau

USer Commande réservée aux extensions

UNivers Charge un ensemble de formes

Nous allons maintenant décrire chacune de ces commandes en détail. Les arguments entre crochets sont optionnels.

HELP

Syntaxe d'appel HELP [commande]

Fonction aide en ligne: imprime la liste des commandes disponibles avec une petite explication. Si une commande est présente en argument, donne une explication détaillée de la commande en question. Cette commande peut être abrégée. Une documentation générale sur HLM est donnée par la commande "HELP HELP" (ou "H H")

REMARQUE

Syntaxe d'appel REMARQUE [message]

Fonction Permet d'insérer des commentaires dans les fichiers de commandes le "message" est imprimé à l'écran à l'execution de cette commande

FICHIER

Syntaxe d'appel FICHIER [fichier[.HLM]]

Fonction Après éxécution de cette commande, les commandes suivantes sont lues sur le fichier dont le nom est donné en argument. L'usage est de donner l'extension .HLM aux fichiers de commandes mais ce n'est pas une obligation. La forme des commandes contenues dans ce fichier doit être exactement identique à ce que l'on taperait au clavier. Un fichier de commande doit impérativement se terminer par une des trois commandes suivantes: FICHIER, CONSOLE ou FIN. CONSOLE rend la main a l'operateur, FIN termine la session. Il est possible de chainer l'execution de plusieurs fichiers de commandes en les terminant par la commande "FICHIER nom du fichier suivant". Le dernier fichier doit se terminer par CONSOLE ou FIN.

CONSOLE

Syntaxe d'appel CONSOLE

Fonction Repasse en mode interactif. Rend la main à l'utilisateur au terminal après l'execution d'un fichier de commandes.

FIN

Syntaxe d'appel FIN

Fonction Termine la session.

AFFICHAGE

Syntaxe d'appel AFFICHAGE [C]

Fonction Détermine le type des informations affichées après chaque itération d'apprentissage et/ou de reconnaissance.

Utilisations

AFFICHAGE
Param

Param peut prendre 4 valeurs

- 0 : ne sont affichés que les taux de réussite après un calcul de performances (voir PERFORMANCE et APPREND)
- 1 : affichage des signes des sorties désirées et produites à chaque iteration
- 2 : comme 1 plus affichage de la forme d'entrée
- 3 : comme 1 plus affichage des statistiques (voir STATISTIQUE)
- 4 : affichage complet de l'état du réseau à chaque itération. ATTENTION: tous les poids sont affichés.

AFFICHAGE C
inf
sup

pour changer les seuils d'affichage des signes des entrées et sorties Toute valeur de sortie inférieure a "inf" sera imprimée avec le 1-er caractère spécifié dans le fichier .IMA, les valeurs supérieures a "sup" sont imprimées avec le 3-eme caractere, les autres avec le 2-eme caractère

APPREND

Syntaxe d'appel APPREND [u] [t]

Fonction execute des cycles d'apprentissage et calcule les performances sur l'ensemble d'apprentissage après chaque série de cycles. Les informations concernant le

taux de réussite sont écrites dans le fichier d'apprentissage dont le nom peut être spécifié par le premier argument au lancement d'HLM. (voir la commande PERFORMANCE pour une description du format de ce fichier)

APPREND [t]

Utilisation nbreIt

nbreCycles

éxécute nbreIt iterations d'apprentissage, calcule le taux de reconnaissance, répète cette séquence nbreCycles fois. l'option T force le calcul du taux de reconnaissance sur tout l'ensemble de formes présent en mémoire.

APPREND u [t]

nbreIt

TauxMin

nbreCyclesMax

éxécute "nbreIt" itérations d'apprentissage, calcule le taux de reconnaissance, répète cette séquence jusqu'a ce que le taux de reconnaissance "TauxMin" soit atteint ou que cette séquence ait été répétée plus de "nbreCyclesMax" fois. l'option T force le calcul du taux de reconnaissance sur tout l'ensemble de formes présent en mémoire.

ALPHABETA

Syntaxe d'appel ALPHABETA [d]

Fonction Détermine la valeur des paramètres de viscosité et de decay

ALPHABETA

Utilisations alpha

beta

Pour changer les valeurs de alpha et beta utilisées dans la formule d'apprentissage:

$$\Delta W \leftarrow \beta \Delta W + \alpha \text{Gradient}$$

$$W \leftarrow W + \Delta W$$

Il est judicieux de respecter la condition

$$\alpha + \beta = 1$$

La simulation est optimisée lorsque $\alpha = 1$ et $\beta = 0$ pour éliminer les opérations inutiles. L'effet de ces deux paramètres est d'introduire une inertie et une

viscosité dans les mouvements du vecteur de poids. $\beta = 1$ correspond à une viscosité nulle, les gradients sont accumulés (intégrés) sans perte.

ALPHABETA d

Decay

Pour changer le Decay utilisé dans la formule d'apprentissage:

$$W \leftarrow W(1 - \text{Decay}) + \Delta W$$

L'apprentissage trouve alors un compromis entre la minimisation du critère de sortie et la norme du vecteur de poids. Une valeur de l'ordre de 0.0001 est raisonnable, mais attention un Decay trop grand entraine la convergence de tous les poids vers 0. ATTENTION: Il est généralement nécessaire de changer la valeur de Decay après avoir changé celle de Epsilon.

BRUIT

Syntaxe d'appel BRUIT

Fonction controle le type de bruit généré sur les formes d'entrée. Un bruit différent est généré à chaque itération.

BRUIT

Utilisations Type

Valeur

Type peut prendre 4 valeurs:

- 0 : Pas de bruit (defaut)
- 1 : Bruit additif, Une quantité aléatoire comprise entre -Valeur et +Valeur (distribution uniforme) est ajoutée à chaque composante de l'entrée (valeur aléatoire différente pour chaque composante).
- 2 : Inversion aleatoire de points, "Valeur" donne la probabilité d'inversion d'une composante de la forme d'entrée.
- 3 : Inversion aleatoire de points, à chaque présentation d'une forme, exactement "Valeur" composantes de la forme d'entrée choisies au hasard sont multipliées par -1. "Valeur" est la distance de hamming entre la forme initiale et la forme bruitée si celles-ci sont binaires (en -1, +1).

DEBUGGER

Syntaxe d'appel DEBUGGER

Fonction Place en mode "déverminage" si celui-ci était inactif. Passe en mode normal si le mode "déverminage" était actif. en mode "déverminage" des information supplémentaires sur l'état interne du simulateur sont affichées au cours de certaines opérations.

ENSEMBLE

Syntaxe d'appel ENSEMBLE [t]

Fonction fixe le sous-ensemble des formes sur lesquelles le programme travaille.

ENSEMBLE

Utilisation BorneInf

BorneSup

Au moment de la lecture d'un fichier de formes contenant N formes au moyen de la commande UNIVERS, les formes sont numérotées de 0 à N-1. La commande ENSEMBLE permet de définir un sous ensemble de ces N formes que le simulateur utilisera comme ensemble de travail. Les formes dont les numeros sont compris entre BorneInf et BorneSup inclus constituent l'ensemble de travail. Le choix d'une forme sera effectué dans ce sous-ensemble au cours de toute les opérations suivantes (jusqu'au prochain appel de ENSEMBLE). Toutefois, l'option T de certaines commandes (PERFORMANCE, APPREND) permet temporairement de prendre en compte tout l'ensemble présent en mémoire sans changer "BorneInf" et "BorneSup". La commande ENSEMBLE est utile pour découper un ensemble de formes en un ensemble d'apprentissage et un ensemble de test.

ENSEMBLE t

L'ensemble de travail est constitue de toutes les formes présentes en memoire. C'est à dire BorneInf est positionné à 0 et BorneSup à N-1.

EPSILON

Syntaxe d'appel EPSILON [t] [n]

Fonction Change la valeur du pas d'apprentissage. permet de changer selectivement le Epsilon de chaque couche. Le Epsilon effectif d'une cellule est égal à

la valeur spécifiée par la commande divisée par le nombre d'entrées de la cellule. ex: si epsi=0.5 le epsilon effectif d'une cellule à 10 entrées sera égal à epsi/10=0.05. NB: Les couches sont numérotées à partir de 0, mais la couche 0 n'a généralement pas de poids.

Utilisations

EPSILON
epsi

Fixe le pas de gradient de chaque cellule a epsi/FanIn ou FanIn est le nombre d'entrées de la cellule considérée. L'information imprimée à l'écran est la valeur moyenne des epsilon(s) associés a chaque couche.

EPSILON t
epsi1
...
epsiN

Fixe le pas de gradient de chaque couche indépendament. A l'intérieur de chaque couche la valeur effective est epsiI/FanIn.

EPSILON n
N
epsiN

Permet de modifier le pas de gradient associé à la couche N sans modifier les autres. La valeur effective est toujours dépendante du FanIn.

ETAT

Syntaxe d'appel ETAT

Fonction Imprime l'état complet du réseau, y compris les poids et les gradients. ATTENTION: utiliser cette commande avec parcimonie pour les gros réseaux.

EXECUTE

Syntaxe d'appel EXECUTE

Fonction Execute des cycles d'apprentissage

Utilisation

EXECUTE
Nbre

Execute "Nbre" iterations d'apprentissage. C'est a dire, presente "Nbre" formes d'entrée et effectue une itération d'apprentissage après chaque présentation.

FONCTION

Syntaxe d'appel FONCTION [c]

Fonction Change le type de la fonction non-linéaire, et ses paramètres.

Utilisations

FONCTION	FONCTION
kt	Pente
ka	Coeff
km	.

Deux cas d'utilisation selon le type courant de la fonction non-linéaire:

- 1-er cas: Pour une fonction type Fermi-Dirac modifiée. Fonction utilisée:

$$F(x) = \mathrm{ka}\frac{\exp \mathrm{kt}x - 1}{\exp \mathrm{kt}x + 1}$$

  ka: amplitude; kt: gain; Les valeurs par défaut sont ka=1.715; kt=1.333; ce qui assure que $F(1) = 1$. La constante km est rajoutée à la dérivée de F, en d'autres termes, la dérivée effectivement utilisée est égale à F'+km. Ceci permet de s'affranchir des effets des erreurs de calculs dus à la formule (optimisée) de calcul de la dérivée.

- 2-eme cas: Fonction linéaire par morceaux. Fonction utilisée:
  - si $|x| < 1$ alors $F(x) = x$
  - si $x < -1$ alors $F(x) = \mathrm{Pente}(x + 1) - 1$
  - si $x > +1$ alors $F(x) = \mathrm{Pente}(x - 1) + 1$

  La dérivée utilisée pour cette fonction peut être contrôlée par le paramètre "Coeff". La dérivée, avant d'être utilisée est multipliée par "Coeff". Si Coeff=1 c'est la dérivée exacte qui est utilisée. La modification de "Coeff" permet de contrôler la propagation des gradients dans les couches intermédiaires et d'échapper à certaines situations de blocage.

FONCTION c

Type

Change le type de fonction non-lineaire utilisée:

- Type=1: Fermi-Dirac modifiée
- Type=2: Linéaire par morceaux

OUBLIE

Syntaxe d'appel OUBLIE [s][n]

Fonction Initialise les poids, ou multiplie les poids par un facteur.

Utilisation
OUBLIE
Vmax

Initialise tous les poids à une valeur aléatoire comprise entre -Vmax et +Vmax avec une distrbution uniforme. La valeur aléatoire est évidemment différente pour chaque poids.

OUBLIE s	OUBLIE n
Coeff	NumCouche
.	Coeff

La première syntaxe permet de multiplier tous les poids du réseau par "Coeff". La seconde permet de ne multiplier par "Coeff" que les poids de la couche "NumCouche". Cette option est particulièrement utile pour sortir de certains blocages du gradient. NB: Les couches sont numérotées à partir de 0.

PARAMETRE

Syntaxe d'appel PARAMETRE

Fonction Visualise les principaux paramètres du réseau: Epsilon, Alpha, Beta, Decay, Type de fonction, etc...

PEDAGOGIE

Syntaxe d'appel PEDAGOGIE

Fonction Spécifie la stratégie de présentation des formes.

Utilisation
PEDAGOGIE
Type

L'entier "Type" prend les valeurs 0 ou 1. Si "Type" vaut 1, la stratégie utilisée est la suivante: Lors d'une session d'apprentissage, une formeprésentée est répétée jusqu'à l'obtention d'une réponse correcte. Si "Type" vaut 0, chaque forme est présentée une fois. A l'itération suivante une nouvelle forme est choisie (Voir la commande PRESENTATION pour une description de la stratégie de choix de la forme suivante). L'activation de la "pédagogie" permet d'échapper à certains minima locaux de la fonction de coût.

PERFORMANCE

Syntaxe d'appel PERFORMANCE [t][c][i][f]

Fonction Calcule le taux de reconnaissance sur l'ensemble courant ou sur tous l'ensemble présent en mémoire. Toute les formes de l'ensemble courant sont testées, aucun apprentissage n'est effectué. Le résultat est écrit dans le fichier apprentissage. Des options permettent de tester un réseau rebouclé sur lui-même pour les applications de type mémoires associatives. A chaque appel de PERFORMANCE (implicitement appelé par APPREND) une ligne est écrite dans le fichier apprentissage. Cette ligne contient dans l'ordre:

- Le nombre d'itérations d'apprentissage effectuées depuis la dernière initialisation des poids.
- Le pourcentage de formes bien reconnues selon le critère choisi.
- Le taux de bruit courant.

PERFORMANCE

calcule le taux de reconnaissance (pourcentage de formes bien reconnues dans l'ensemble courant)

PERFORMANCE t

calcule le taux de reconnaissance sur tout l'ensemble de formes présent en mémoire.

PERFORMANCE c
NbrePres

Change le nombre de passage de la base d'exemple pour le calcul du taux de réussite. Cette option est intéressante lorsque les formes d'entrée sont bruitées Le taux de réussite calculé est alors une moyenne sur "NbrePres" échantillons de la même forme bruitée différemment. Le paramètre "NbrePres" est automatiquement remis a 1 lorsque le bruitage est supprimé (option 0 dans la commande BRUIT).

PERFORMANCE i
Niter

Fixe le nombre d'itérations pour l'auto-association. Lorsque le nombre d'entrées et le nombre de sorties sont égaux il est possible de réinjecter la sortie sur l'entrée, et d'itérer ce processus un nombre de fois fixé par "Niter". Cette

réinjection n'est effectuée que pendant la reconnaissance mais pas pendant l'apprentissage. Si Niter=1 le mode auto-association est inactif.

PERFORMANCE f
TypeFeedBack
param1
param2

Change la fonction utilisée pour la reinjection des sorties sur les entrées en mode auto-association. "Param1" et "Param2" ont un sens différent (et peuvent même ne pas être présents) selon la valeur de "TypeFeedBack"

Utilisation

- TypeFeedBack=1: Feed-back avec seuillage, la sortie est seuillée avec la fonction seuil en -1,+1, et réinjectée sur l'entrée. Pas de Param1 ni Param2.
- TypeFeedBack=2: FeedBack "mou", la sortie est multipliee par Param1 (le gain de boucle) et reinjectee sur l'entree. C'est un bouclage non linéaire puisque la sortie est déjà transformée par la fonction F. Pas de Param2.
- TypeFeedBack=3: Feed-Back type "Brain-state-in-the-box", la sortie est multipliée par Param1 (le alpha du modele BSB), l'entrée est multipliée par Param2 (le beta du modele BSB), ces deux quantités sont additionnées et constituent la nouvelle entrée

PRESENTATION

Syntaxe d'appel PRESENTATION

Fonction Fixe la procédure de choix des formes.

Utilisation

PRESENTATION
Type

Si "Type" vaut 0, les formes sont simplement présentées dans l'ordre de leur numérotation (leur ordre dans le fichier .IMA). Cette séquence est répétée autant de fois que désiré. Par exemple, lorsque les bornes de l'ensemble de travail sont égales à "Binf" et "Bsup", HLM commence par présenter la forme "Binf", suivie de "Binf"+1, ... "Bsup", "Binf", ... Si "Type" vaut 1, les formes sont présentées dans un ordre aléatoire. A chaque itération, la forme courante est choisie au hasard dans l'ensemble de travail courant.

N.B.: C'est un tirage aléatoire avec remise.

RESEAU

Syntaxe d'appel RESEAU [nom_de_réseau]

Fonction Charge un réseau en mémoire. Ce réseau doit être décrit par trois fichiers au format HLM dont les noms sont: "Nom_de_reseau.STR", "Nom_de_reseau.IO", "Nom_de_reseau.POI". Si aucun nom de réseau n'est donné sur la ligne de commande, il en est demandé un à l'utilisateur.

RESTAURE

Syntaxe d'appel RESTAURE [nom_de_fichier_de_poids]

Fonction Charge un fichier de connexions (nom_de_fichier_de_poids.POI). Cette commande permet de restaurer le résultat d'un apprentissage antérieur sans avoir à recharger un réseau complet. On peut ainsi disposer de plusieurs fichiers de poids correspondant à un même réseau. (voir commande SAUVE)

SAUVE

Syntaxe d'appel SAUVE [nom_de_fichier_de_poids]

Fonction Sauve une configuration de poids dans le fichier "nom_de_fichier_de_poids.POI". Ce fichier pourra ensuite être réutilisé avec la commande RESTAURE.

STATISTIQUE

Syntaxe d'appel STATISTIQUE

Fonction Imprime les valeurs moyennes des états internes du réseau. Les informations affichées sont, pour chaque couche du réseau, la moyenne des valeurs absolues (MVA) des sorties des cellules, la MVA des ΔW, la MVA des poids.

USER

Syntaxe d'appel USER

Fonction Permet l'extension d'HLM à peu de frais.

UNIVERS

Syntaxe d'appel UNIVERS [nom_de_fichier_de_formes]

Fonction Charge un ensemble de formes à partir du fichier "nom_de_fichier_de_formes.IMA". Le fichier doit respecter un format précis décrit plus loin. Il convient d'appeler cette commande APRES avoir chargé un réseau. En cas de non-compatibilité du fichier de formes avec le réseau déjà chargé (nombre d'entrées ou de sorties différents), HLM signale l'erreur et rend la main.

4.2.4 format des fichiers de formes .IMA

Les fichiers de description de formes ont l'extension .IMA et doivent respecter un format ASCII décrit ci-après. Il est constitué d'une entête, suivie de la description des formes d'entrée et des réponses associées.

Le fichier se compose ainsi (ligne par ligne):

- ¡commentaire descriptif de la base d'exemple¿
- ¡nombre de colonnes des formes d'entree¿
- ¡nombre de lignes des formes d'entree¿
- ¡nombre de colonnes des formes de sortie¿
- ¡nombre de lignes des formes de sortie¿
- ¡3 caracteres utilises pour representer les valeurs -1, 0 et +1¿
- ¡critere de classification 1,2,3 ou 4, voir ci-dessous¿
- ¡ligne blanche¿
- Forme 0
- ¡ligne blanche¿
- Reponse 0
- ¡ligne blanche¿
- Forme 1
- ¡ligne blanche¿

- Reponse 1
- ...
- ¡ligne blanche¿
- Forme N
- ¡ligne blanche¿
- Reponse N

ATTENTION: Pas de ligne blanche a la fin du fichier.

Description de chaque ligne

Ligne 1 un message de 80 caractères maximum, qui sera affiché lors du chargement des formes.

Lignes 2,3,4,5 les formes (d'entrée et de sortie) sont décrites sous formes de tableaux bidimensionnels de caractères. Les lignes 2,3,4 et 5 spécifient respectivement les nombres de colonnes et de lignes des formes d'entrée et de colonnes et de lignes de formes de sortie désirées. Dans le cas de formes unidimensionnelles, le nombre de lignes (lignes 3 et/ou 5) doit être égal à 1. Si l'un des nombres des lignes 4 et 5 est égal à 0, le simulateur se place en mode auto-association, les sorties sont rebouclées sur les entrées. Il n'est plus nécessaire de spécifier les formes de sortie désirées, elles sont égales aux entrées. Les paramètres du rebouclage (nombre d'itérations, type de rebouclage) sont modifiables par la commande PERFORMANCE. Si le réseau déjà chargé en mémoire possède un nombre de sorties différent du nombre d'entrées, le mode rebouclage n'est pas activé, et le fichier de formes n'est pas chargé.

Ligne 6 Les trois premiers caractères de la ligne 6 sont utilisés pour désigner les valeurs d'entrée et de sortie -1, 0 et +1. Par exemple, si ces trois caractères sont "-","0", et "+", la forme décrite par "–0+-+" correspondra au vecteur (-1,-1,0,+1,-1,+1). Ces trois caractères sont utilisé par la fonction d'affichage d'HLM, les seuils d'affichage des trois caractères sont modifiables avec la commande AFFICHAGE et l'option "C".

Ligne 7 Un entier spécifiant le critère de classification utilisé par le simulateur pour décider si une configuration de sortie est correcte ou non. Les 4 types de classification possibles sont:

- 1 : Une forme est considérée comme correctement classée si la cellule de sortie la plus active (par rapport aux autres cellules de sorties) est la même dans la réponse produite et dans la réponse désirée.
- 2 : Une forme est considérée comme correctement classée si les signes des sorties sont les mêmes dans la réponse produite et dans la réponse désirée.
- 3 : Une forme est considérée comme correctement classée si les valeurs absolues des différences entre les sorties désirées et produites sont toutes inférieures a une marge donnée. Cette marge peut être modifiée par la commande PERFORMANCE avec l'option "E".
- 4 : Une forme est considérée comme correctement classée si la distance Euclidienne entre les vecteurs de sortie désirée et produite est inférieure à une marge donnée.

Les lignes suivantes sont les descriptions des formes d'entrée et de sortie séparées par des lignes blanches (vides).

4.2.5 les utilitaires de génération et manipulation de réseaux

GenRes

GenRes permet la génération de réseaux en couches à connexion globale. Le programme demande à l'utilisateur d'entrer successivement: le nom du réseau à générer, le nombre de couches, la présence ou l'absence de cellule seuil, le nombre de cellules dans chaque couche ainsi que le type de connexion d'une couche avec la précédente.

Le réseau généré comporte $N+1$ couches numérotées de 0 à N. Toutes les cellules de la couche 0 sont des cellules d'entrée, et toutes les cellules de la couche N des cellules de sortie. Les connexions relient les cellules de la couche $i-1$ à celles de la couche i (pour $0 < i \leqslant N$). La connexion est globale, toute les cellules de la couche $i-1$ sont connectées à celles de la couche i. On a cependant ajouté un option permettant d'établir des connexions partielles. Dans ce cas, un schéma de connexion locale unidimensionnelle est établi selon le principe décrit ci-après. Si le nombre de cellules de la couche i N_i est inférieur au nombre de cellules de la couche $i-1$ N_{i-1}, on spécifie le nombre f de connexions par cellule de la couche i. Ces connexions seront réparties à peu près uniformément sur la couche $i-1$. Il y aura donc au total fN_i connexions générées. La répartition des connexions se fait selon un voisinnage unidimensionnel. Les connexions arrivant sur une cellule de la couche i proviennent de

cellules de la couche $i-1$ dont les indices sont contigüs. Si le nombre de cellules de la couche i N_i est supérieur au nombre de cellules de la couche $i-1$ N_{i-1}, on spécifie le nombre f de connexions par cellule de la couche $i-1$. Ces connexions sont projetées sur les cellules de la couche i selon le même principe que ci-dessus.

Lorsque l'option "cellule seuil" est choisie, GenRes crée crée une cellule supplémentaire dans la couche 0, (ainsi qu'une entrée supplémentaire: la première) qui sera connectée à toute les autres cellules exceptées celles de la couche 0. Les poids correspondant seront les seuils des cellules. L'entrée assignée à cette cellule (la première entrée) devra avoir toujours la même valeur dans le fichier de formes .IMA.

Le réseau généré est sauvé dans les trois fichiers dont les noms sont formés par la concaténation du nom du réseau et des trois extensions .STR, .IO, .POI.

HLMToNet

HlmToNet permet la traduction de fichiers de description de réseau en ASCII. Cet utilitaire s'appelle avec la commande

HLMToNet Reseau_source [Reseau_objet]

En cas d'absence du deuxième argument, le nom du réseau objet est le même que celui du réseau source. HlmToNet utilise les trois fichiers Reseau_source.STR, Reseau_source.IO, Reseau_source.POI et génère un fichier Reseau_objet.NET dont le format est décrit au paragraphe 4.2.5. Ce programme est utilisable conjointement avec NetToHLM (voir paragraphe suivant) pour examiner et/ou modifier la structure d'un réseau.

NetToHLM

Comme son nom l'indique, NetToHLM effectue l'opération inverse de HLMToNet décrit au paragraphe précédent. Il s'appelle avec la commande

NetToHLM Reseau_source [Reseau_objet]

En cas d'absence du deuxième argument, le nom du réseau objet est le même que celui du réseau source. NetToHLM Utilise un fichier au format .NET et génère trois fichiers aux extensions .STR, .IO, .POI. utilisables par HLM. Cet utilitaire permet la création de réseaux spéciaux "à la main", ou la modification de réseaux lorsqu'il est utilisé avec HLMToNet.

ATTENTION: Aucune vérification de validité des données du fichier .NET n'est effectuée.

Format des fichiers .NET

Les fichiers .NET en ASCII permettent la spécification de réseau à peu de frais, ainsi que le transfert de réseaux entre machines différentes. Un fichier .NET est composé de trois parties dont l'ordre est indifférent. Ces trois parties correspondent aux informations contenues dans les trois fichiers réseaux au format HLM. Le début de chaque partie est signalé par l'une des trois lignes suivantes:

- $STR pour la partie structure
- $IO pour la partie entrées/sorties
- $POI pour la partie connexions

le symbole $ est impérativement le premier caractère de la ligne. La fin d'une partie est signalée par la fin du fichier ou le début d'une autre partie. Toute ligne dont le premier caractère est un point d'exclamation :"!" est considérée comme une ligne de commentaires.

La partie structure ($STR) contient les informations sur les blocs du réseau (les couches). Chaque ligne est composée de 4 entiers séparés par des espaces. Le premier nombre est le numéro du bloc (de la couche) décrite, le second est l'indice de la première cellule appartenant à ce bloc, le troisième est l'indice de la dernière cellule du bloc. Le quatrième nombre est l'attribut du bloc qui dans la plupart des cas est simplement son numéro. Les couches sont supposées être numérotées à partir de 0. Les indices des cellules d'un couche doivent être supérieurs aux indices des cellules de toutes les couches d'indice inférieur.

La partie entrées/sorties ($IO) contient la correspondance entre les entrées/sorties (numérotées) et les cellules. Chaque ligne de cette partie contient deux entiers. Le premier est l'indice d'une cellule à laquelle est reliée l'e/s considérée. Le second est le type de l'e/s: il vaut 1 si c'est une entrée, 2 si c'est une sortie, 3 si c'est une entrée et une sortie à la fois.

La partie connexions ($POI) contient la liste des connexions. Chaque connexion est décrite par une ligne composée de deux entiers et d'un réel. Le premier entier est l'indice de la cellule amont (présynaptique),le second l'indice de la cellule aval (postsynaptique), le réel est la valeur du poids. Il est préférable que l'ordre des arcs respecte l'ordre des couches des cellules aval (par exemple: les poids dont la cellule aval est dans la couche 2 doivent être avant ceux dont la cellules aval est dans la couche 3). Ce n'est pas une obligation, mais cela permet d'accélérer le chargement du réseau par HLM qui dans ce cas n'a pas besoin de les réordonner.

4.3 Structure interne

Ce paragraphe est destiné aux personnes désireuses de modifier HLM en vue d'une application particulière, de l'implantation de nouvelles procédures d'apprentissage, ou tout simplement de son extension.

4.3.1 Architecture générale du simulateur

Les versions Turbo-PASCAL et ISO diffèrent de la version PRIME, car cette dernière accepte la compilation séparée, ce qui n'est pas le cas des deux premières. L'architecture logique reste néanmoins la même quelle que soit la version. Nous décrirons cependant la version Turbo-PASCAL. Par ailleurs, du à l'absence de chaines dynamique en Pascal ISO, les fonctionnalités de cette version sont légèrement différentes.

Les fichiers sources

Le source d'HLM Turbo-Pascal est constitué par cinq fichiers réunis en un seul à la compilation. Ces fichiers sont:

- HLM.DCL: fichier contenant toutes les déclarations de types et de variables globales.
- HLM.PAS: contient l'essentiel du code d'HLM, toutes les procédures de calcul et les procédures interactives propres au simulateur.
- DIALOGUE.PAS: contient un petit interpréteur de commandes. Ce module est totalement indépendant d'HLM et peut être réutilisé à d'autres fins comme interface utilisateur.
- HLMIO.TUR: c'est l'ensemble des procédures d'entrée/sortie propres à HLM (lecture/sauvegarde de réseau...).
- STRING.TUR: Ensemble de fonctions de manipulation de chaines de caractères.

Lors du portage d'une machine à une autre, il suffit généralement de ne modifier que les deux derniers fichiers. Ils regroupent les fonctions qui sont "non-standard" et qui, par conséquent, risquent d'être différentes d'une implémentation de PASCAL à l'autre.

4.3.2 Structure des données

La représentation des données dans HLM est conçue pour être à la fois portable, rapide et générale. Nous avons donc banni les structures dynamiques dont l'accès,

et surtout le chargement/sauvegarde sur disque, posent des problèmes de portabilité et/ou d'efficacité. Tout est donc représenté dans des tableaux. La simulation d'un réseau nécéssite la connaissance du graphe de connexion et des valuations des connexions, ainsi que du mode d'itération retenu. Tous les modes d'itérations déterministes sont réalisables dans HLM, séquentiel, parallèle ou séquentiel par blocs. Les deux premiers types d'itérations sont, en réalité, des cas particuliers du troisième, nous ne considérerons donc que celui-ci. Dans une itération séquentielle par blocs, nous devons définir les blocs, et l'ordre dans lequel ils seront mis à jour. Un bloc est un groupe de cellules dont la mise à jour est effectuée "en parallèle". L'information concernant les blocs est stockée dans les fichier .STR et chargée dans un tableau de RECORDs dont le nom est "Layer". A chaque élément du tableau "Layer" correspond un bloc. Un élément du tableau "Layer" contient un pointeur sur la première cellule du bloc, un pointeur sur la dernière cellule du bloc, un pointeur sur le premier poids du bloc,un pointeur sur le dernier poids du bloc. Ces "pointeurs" sont en fait des entiers qui représentent des indices dans un tableau. Chaque cellule est représentée par un RECORD élément du tableau "Cell" qui contient les éléments suivants:

- la somme pondérée des entrées (lx)
- la sortie (x)
- la somme pondérée rétrograde des gradients (ly)
- les gradients (y)
- le Epsilon (le pas de gradient) attachés à la cellule considérée (epsi)

Chaque poids est représenté par un RECORD élément du tableau "Arc" qui contient quatre champs:

- un pointeur sur (indice de) la cellule présynaptique (pre)
- un pointeur sur la cellule postsynaptique (post)
- la valeur du poids (w)
- la valeur de l'incrément du poids (deltaw)

Pour calculer le nouvel état d'un bloc, il suffit de mettre toutes les sommes pondérées des cellules de ce bloc à 0, puis de parcourir tous les poids de ce bloc, c'est à dire tous les poids dont la cellule postsynaptique appartient à ce bloc, et d'accumuler les produit des poids par les états des cellules présynaptiques dans les somme pondérées

des cellules postsynaptiques (les "lx"), cette opération est effectuée par la procédure "itlx" qui prend en argument un numéro de bloc. Il faut ensuite transformer ces somme pondérées à l'aide de la fonction non-linéaire et les stocker (dans les "x"), ceci est réalisé par la procédure "itx". Ces états ne seront pas effacés avant l'itération suivante, il est donc tout à fait possible de tenir compte de l'état à l'instant précédent pour calculer le nouvel état. Ceci est utile dans les réseaux comportant des boucles. Pour calculer l'état du réseau, on met à jour les bloc dans l'ordre défini dans "Layer". Un même bloc peut apparaitre plusieurs fois dans "Layer" lorsque l'on veut utiliser des modes d'itération complexes de réseaux à boucles avec des échelles de temps différentes pour chaque bloc.

La rétro-propagation des gradients est effectué de la même manière, en parcourant le tableau "Layer" en sens inverse, les procédures utilisées sont "itly" et "ity".

4.3.3 Ajouter des commandes à HLM

Rajouter une commande à HLM est extrèmement simple. Il suffit d'effectuer les cinq opérations suivantes (la dernière est optionnelle).

- rajouter la commande dans le dictionnaire des mot-clés, dans le fichier HLMDICO

  HLM.MOT. éventuellement en ajoutant sur la même ligne un commentaire qui sera imprimé lors de l'éxécution du HELP.

- écrire une procédure appelée par cette commande. L'entête de cette procédure doit être de la forme:

  PROCEDURE NomDeLaProcedure(st: stringue);

  Au moment de l'appel, la variable st contient les arguments de la ligne de commande.

- insérer l'appel de cette procédure dans l'instruction CASE située à la fin du fichier HLM.PAS dans la procédure "Biblio". L'endroit de l'insertion doit évidemment respecter l'ordre du fichier dictionnaire. Renuméroter les constantes de sélection des procédures suivante.

- recompiler HLM

- ajouter un fichier de HELP long dans le répertoire HLMDICO dont le nom est formé des 8 huit premiers caractères de la commande avec l'extension .HLP.

4.3.4 Quelques idées pour un futur simulateur

A l'usage, il s'avère que la qualité de l'interface utilisateur est de première importance. Il serait bon de pouvoir agir directement sur les paramètres du réseau, et de pouvoir redéfinir interactivement la séquence d'opérations effectuée à chaque itération d'apprentissage. Il semble également indispensable de disposer d'outils puissants et interactifs pour la visualisation graphique de l'état du réseau.

Il s'avère que la plupart des limitations d'HLM sont dues aux contraintes du langage PASCAL dont la portabilité est une chimère [1] et dont l'efficacité très relative. L'emploi du langage C parait tout indiqué pour une réalisatione future, en particulier pour résoudre les problèmes d'efficacité et de portabilité.

En ce qui concerne l'interface utilisateur, l'idéal serait de disposer d'un interpréteur LISP servant d'interface à un ensemble de routines de calculs écrites en C. Malheureusement, quoique réalisable sur plusieurs machines, (notamment avec Le_Lisp) cette solution est peu portable. La solution à ce problème de portabilité est d'écrire le mini-interpréteur LISP en C. La mise au point d'un système de ce type est en cours. Une grande attention a été apportée à la portabilité, la rapidité, et l'interactivité du système.

4.3.5 GRL: Description

GRL est un outil implanté sur PRIME permettant la description hiérarchique de graphes valués. Nous l'avons spécifiquement développé en vue de son utilisation pour la conception et la manipulation de réseaux connexionnistes.

GRL génère des fichiers de description de réseaux directement lisibles par HLM.

Il est constitué d'un compilateur (ou plus exactement d'un pré-processeur) du langage GRL. Le langage généré par ce compilateur est du PASCAL.

La syntaxe de GRL est, en fait, celle de PASCAL, à laquelle on a rajouté quelques fonctions et procédures prédéfinies, ainsi qu'un type de données particulier: le type "Cell". Une cellule est créée par la fonction "Cree_Cell" (sans jeu de mot) dont les arguments sont le nom de la cellule et un attribut (par exemple son numéro de couche). Le nom d'une cellule est composé d'une suite de caractères alphanumériques optionnellement suivie d'un crochet ouvrant, d'une suite de variables ou constantes entières PASCAL séparées par des virgules, et d'un crochet fermant. Par exemple:

```
VAR i,j: integer;
  BEGIN
```

[1] bien que des efforts aient été entrepris pour assurer la portabilité d'HLM

```
i=2;  j:=6;
Cree_Cell( Bzzt[3,i,j+4], 2);
END;
```

crée la cellule dont le nom est "Bzzt[3,2,9]" dans la couche 2. Nous pouvons connecter deux objets à l'aide de la procédure Arc dont les arguments sont les noms des deux objets et la valeur de la connexion. Ainsi l'instruction:

```
Arc( Bzzt[3,2,2*4+1], Pouet[23], -2.3);
```

connecte la cellule créée par l'exemple précédent à la cellule "Pouet[23]".

Il est donc possible de créer des objets élémentaires indicés dans un espace de dimension quelconque, et de les connecter entre eux.

GRL permet la définition hiérarchique, c'est à dire l'utilisation de structures comme parties de structures plus grandes. Il est possible de créer un sous réseau, appelé par exemple SouReso. Il suffit pour cela d'effectuer toutes les opérations de création et de connexions d'objets à l'intérieur d'une procédure dont le nom est "Cree_SouReso(...);" (ou "Cree_xxx(...);") et de lui définir des entrées/sorties, c'est à dire d'associer des "pattes de sortie" à certaines cellules de ce sous-réseau. La définition d'un sous-réseau de type "SouReso" peut-être faite à l'aide de la procédure suivante:

```
PROCEDURE Cree_SouReso( nom: ... );
  BEGIN
  Cree_Cell( bzzt[5], 2);
  Cree_Cell( pouet, 1);
  Arc( bzzt[5], pouet, 12);
  ....
  END;
```

A l'appel de cette procédure par une instruction du type

```
Cree_SouReso( MonReso[2,3], ... );
```

seront créées deux cellules dont les noms seront formés de la concaténation du nom du sous-réseau et du nom de la cellule. Dans notre cas, les cellules créées auront pour noms:

```
MonReso[2,3].bzzt[5]  et  MonReso[2,3].pouet
```

elles seront reliées par une connexion de valeur 12.

Comme cela a été indiqué plus haut, il est possible de créer des entrées/sorties associées à un sous-réseau. Ceci permet ensuite de considérer ce sous-réseau comme

une "boite noire", un module, dont il n'est pas nécessaire de connaitre la structure interne. Ceci est effectué à l'aide de la procédure "Cree_IO" dont les paramètres sont le nom de la "patte" d'entrée/sortie à créer, le nom de la cellule à laquelle cette patte est associée, et le type de la patte: entrée, sortie, ou les deux à la fois. Si l'instruction

```
Cree_IO( e[1], bzzt[2+3], 1);
```

est insérée à la place des trois petits points dans l'exemple précédent, L'appel de la procédure SouReso comme ci-dessus provoquera la création d'un objet de type entrée dont le nom sera

```
MonReso[2,3].e[1]
```

Effectuer une connexion sur cette entrée/sortie par une instruction du type

```
Arc( xxx, MonReso[2,3].e[1], -14);
```

sera totalement équivalent à l'effectuer avec l'instruction:

```
Arc( xxx, MonReso[2,3].bzzt[5], -14);
```

L'intéret de ce type de notation est clair pour des réseaux très modulaires.

Ce type d'emboitement peut être répété pluseurs fois. La seule contrainte concerne la longueur des noms d'objets générés qui doit rester inférieure à 40 caractères.

Diverses autres procédures standard existent comme la sauvegarde d'un réseau, ou l'inclusion d'un réseau précédemment créé et stocké sur disque comme sous-partie du réseau courant.

Le résultat de la compilation et de l'expansion est sauvé dans quatre fichiers au format HLM (.STR, .IO, .POI, .CEL). Le fichier .CEL contient les noms des cellules, il n'est pas utilisé par HLM, c'est pourquoi les utilitaires de génération de réseaux autres que GRL ne le génèrent pas. Par ailleurs les fichier .IO contiennent les noms des entrées/sorties, bien que ces noms ne soient pas utilisés par HLM, ils sont conservés pour des raisons de compatibilité avec le format GRL.

Le principal avantage de GRL est sa souplesse, et le fait qu'il donne accès à toute la puissance du langage PASCAL: c'est, en fait, un sur-ensemble de PASCAL. Son principal défaut est sa lenteur, en effet pour chaque description en GRL, il faut compiler de GRL vers PASCAL, ajouter les bibliothèques propres à GRL, compiler et faire l'édition de liens du programme pascal ainsi obtenu, puis l'éxécuter. L'enchainement de ces opérations est évidemment automatique, mais elles n'en sont pas moins gourmandes en temps machine.

Ceci ne constitue pas un manuel d'utilisation mais une simple description des fonctionnalités de GRL.

Appendice A

Apprentissage Linéaire

Extrait de
"Automata Networks and Artificial Intelligence"

F. Fogelman-Soulié, P. Gallinari, Y. le Cun, S. Thiria

A paraitre dans "Automata networks: theory and applications"
Manchester University Press.

3. LINEAR LEARNING

Work on perceptrons had shown that it was possible to design a machine which was capable of learning to automatically categorize, i.e. to associate with various input patterns $\mathbf{X_i}$ their class $\mathbf{Y_i}$. The use of threshold functions by the decision units was central in this approach. More recently, LINEAR models have been studied and the theory of such systems is now well developped (see Kohonen [47,48,49]) and has been used for a number of applications. In this paragraph, we will give an outline of this theory and refer the reader to Kohonen [49] for more details.

3.1. The model.

We have defined elsewhere [67] an automata network as a mapping $F: \mathbb{E}^n \rightarrow \mathbb{E}^n$. This model can be represented as a "black box" type model (see fig 5): we replicate $\mathbb{E}^n$ twice, each version is interpreted as a network or "layer" and mapping F allows to define the state of the second layer in terms of the state of the first layer. More precisely, if $\mathbf{x}(t)$, t=1,2, is the state vector of layer t and j is a cell in the second layer, then j will have state:

$$x_j(2) = F_j[\mathbf{x}(1)]$$

Then the dynamics on F can be defined as usual, by feeding-back the output of the second layer to the first one.

This representation allows us to extend the notion of automata network to mappings $F\ \mathbb{E}^n \rightarrow \mathbb{E}^p$, with n≠p (see fig 5). The connections between the two layers defining F will be endowed with a WEIGHT $W_{ij} \in \mathbb{R}$. Let $\mathbf{W}=(W_{ij})$ be the weight matrix (if there is no connection between i and j, then by convention $W_{ij}=0$) (fig 5)

We want to MEMORIZE into the network the ASSOCIATIONS between m input patterns $\mathbf{X_i} \in \mathbb{E}^n$, i=1 m, and m output patterns $\mathbf{Y_i} \in \mathbb{E}^p$ Usually we will take $\mathbb{E} = \mathbb{R}$ or a finite space, [-1,1] for example $\mathbf{X}$ will denote the nxm matrix with columns $\mathbf{X_i}$ and $\mathbf{Y}$ the pxm matrix with columns $\mathbf{Y_i}$ X_{ji} is thus

FIGURE 5-1

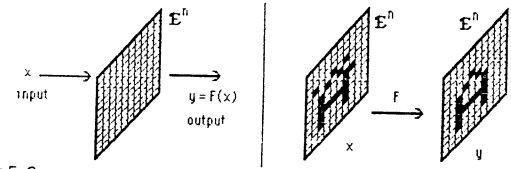


FIGURE 5-2

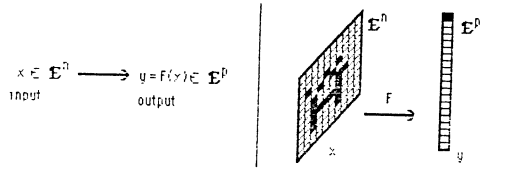


Figure 5: Automata network.
The figure shows an automata network (left) as a mapping F. $\mathfrak{E}^n \rightarrow \mathfrak{E}^n$ (Figure 5-1) or F. $\mathfrak{E}^n \rightarrow \mathfrak{E}^p$ (figure 5-2) and its equivalent (right) as a two-layered network. In the layered network representation, the input x to network F is the state of the input layer (we represented here this input as a noisy "A") and the output y is the state of the output layer (here y is a "pure" A in case of fig 5-1 and a code -1 for the first letter of the alphabet- in fig.5-2).

the state of automaton j in pattern $\mathbf{X_i}$ In the case where $\mathbf{X_i}=\mathbf{Y_i}$, for all i, we will say that the network is performing AUTO-ASSOCIATION (for example in pattern recognition, index searching ...), HETERO-ASSOCIATION otherwise (as in automatic classification).

The linear learning scheme assumes that these associations are realized by a LINEAR automata network: when presented with an input pattern $\mathbf{x}$, the network F computes an output pattern $\mathbf{y} = \mathbf{W}\mathbf{x}$ The desired associations will be encoded in the weight matrix $\mathbf{W}$ In order to store the desired associations, one thus has to make sure that. $\mathbf{W}\,\mathbf{X_i} = \mathbf{Y_i}$ i=1...m

i e to solve equation $\mathbf{W}\,\mathbf{X} = \mathbf{Y}$ (13)

or in the case of auto-association $\mathbf{W}\,\mathbf{X} = \mathbf{X}$ (14)

where $\mathbf{X}$ and $\mathbf{Y}$ are given

Remark. in general the state space $\mathfrak{E}$ is finite, and thus equation (13) must be replaced by $\mathbf{I}[\mathbf{W}\,\mathbf{X}] = \mathbf{Y}$ (13')

where function $\mathbf{1}$ denotes thresholding for example, if $\mathbf{\Sigma} = \{0,1\}$, $\mathbf{1}[u]=1$ if $u \geq 0$, 0 otherwise, if $\mathbf{\Sigma} = \{-1,+1\}$, $\mathbf{1}[u]=1$ if $u \geq 0$, -1 otherwise

3.2. The generalized inverse matrix.

A formal solution to equation (13) or (14) is given by the generalized inverse matrix theory [13,65,78] In the following, we will use only the pseudo-inverse [33] By definition, $\mathbf{X}^+$ is a PSEUDO INVERSE of matrix $\mathbf{X}$ iff

1• $\mathbf{X}\mathbf{X}^+\mathbf{X} = \mathbf{X}$
2• $\mathbf{X}^+\mathbf{X}\mathbf{X}^+ = \mathbf{X}^+$
3• $\mathbf{X}\mathbf{X}^+$ and $\mathbf{X}^+\mathbf{X}$ are hermitian (if $\mathbf{X}$ is a real matrix, this just means that $\mathbf{X}\mathbf{X}^+$ and $\mathbf{X}^+\mathbf{X}$ are symmetric)

It has been shown [33,65] that for any matrix $\mathbf{X}$, there exists one and only one pseudo-inverse $\mathbf{X}^+$ $\mathbf{X}^+$ can be computed by

$$\mathbf{X}^+ = \lim_{d \to 0} (\mathbf{X}^t\mathbf{X} + d^2\,\mathbf{I})^{-1}\mathbf{X}^t = \lim_{d \to 0} \mathbf{X}^t(\mathbf{X}\mathbf{X}^t + d^2\,\mathbf{I})^{-1}$$

where $\mathbf{X}^t$ denotes the transpose of matrix $\mathbf{X}$, $\mathbf{I}$ the identity matrix, $\mathbf{Y}^{-1}$ the inverse of matrix $\mathbf{Y}$ and $d \in \mathbb{R}$.

Pseudo inverses can be used to solve matrix equations such as. $\mathbf{W}\mathbf{X}=\mathbf{Y}$ (13)
where $\mathbf{X}$ and $\mathbf{Y}$ are given and $\mathbf{W}$ is unknown It has been shown that:

Theorem equation (13) has an exact solution iff·

$$\mathbf{Y}\mathbf{X}^+\mathbf{X} = \mathbf{Y} \tag{15}$$

and in this case, there exists an infinite number of solutions given by

$$\mathbf{W} = \mathbf{Y}\mathbf{X}^+ + Z(\mathbf{I} - \mathbf{X}\mathbf{X}^+) \tag{16}$$

where Z is an arbitrary matrix with the same dimensions as $\mathbf{W}$

Remarks

• condition (15) is clearly necessary, since

$$\mathbf{Y} = \mathbf{W}\mathbf{X} \Rightarrow \mathbf{Y}\mathbf{X}^+\mathbf{X} = \mathbf{W}\mathbf{X}\mathbf{X}^+\mathbf{X} \Rightarrow \mathbf{Y}\mathbf{X}^+\mathbf{X} = \mathbf{W}\mathbf{X} = \mathbf{Y}$$

• if the columns $\mathbf{X}_i$ of $\mathbf{X}$ are linearly independent, then $\mathbf{X}^+ = (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t \Rightarrow \mathbf{X}^+\mathbf{X} = \mathbf{I} \Rightarrow \mathbf{Y}\mathbf{X}^+\mathbf{X} = \mathbf{Y}$ and condition (15) is satisfied for all $\mathbf{Y}$ In this case the solution $\mathbf{W} = \mathbf{Y}\mathbf{X}^+$ can be computed easily by computing only $(\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t$

Theorem among all solutions to equation (13), $\mathbf{W} = \mathbf{YX}^+$ is of minimum quadratic norm

PROOF $\| \mathbf{YX}^+ + \mathbf{Z}(\mathbf{I} - \mathbf{XX}^+)\|^2 = \mathrm{tr}[\mathbf{YX}^+ + \mathbf{Z}(\mathbf{I} - \mathbf{XX}^+)]^t [\mathbf{YX}^+ + \mathbf{Z}(\mathbf{I} - \mathbf{XX}^+)]$

$$= \| \mathbf{YX}^+\|^2 + \| \mathbf{Z}(\mathbf{I} - \mathbf{XX}^+) \|^2 + 2\,\mathrm{tr}[\,\mathbf{Z}^t\mathbf{YX}^+] - 2\,\mathrm{tr}[\,(\mathbf{X}^+)^t\,\mathbf{X}^t\,\mathbf{Z}^t\mathbf{YX}^+]$$

where tr($\mathbf{A}$) is the trace of matrix $\mathbf{A}$

By using the classical properties. $\mathrm{tr}(\mathbf{PQ}) = \mathrm{tr}(\mathbf{QP})$, $(\mathbf{X}^+)^+ = \mathbf{X}$ and $\mathbf{XX}^t(\mathbf{X}^+)^t = \mathbf{X}$, we show that

$$\mathrm{tr}[\,(\mathbf{X}^+)^t\,\mathbf{X}^t\,\mathbf{Z}^t\mathbf{YX}^+] = \mathrm{tr}[\mathbf{Z}^t\mathbf{YX}^+(\mathbf{X}^+)^t\,\mathbf{X}^t] = \mathrm{tr}[\mathbf{Z}^t\mathbf{YX}^+]$$

and thus $\| \mathbf{YX}^+ + \mathbf{Z}(\mathbf{I} - \mathbf{XX}^+)\|^2 = \| \mathbf{YX}^+ \|^2 + \| \mathbf{Z}(\mathbf{I} - \mathbf{XX}^+) \|^2$

Matrix $\mathbf{W} = \mathbf{YX}^+$ (17)

is thus the solution to equation (13) of minimum quadratic norm □

In the case of auto association, the solution to equation (14) of minimum quadratic norm is: $\mathbf{W} = \mathbf{XX}^+$ (17')

If condition (15) is not satisfied, then equation (13) has no solution, but it can be shown that matrices $\mathbf{W}$ for which the quadratic norm $\| \mathbf{WX} - \mathbf{Y} \|$ is minimum are just those given by equation (16), and of course, among those, matrix $\mathbf{W} = \mathbf{YX}^+$ is of minimum quadratic norm Hence, when (15) is not satisfied, (16) gives the general expression of approximate solutions to (13) and (17) the best approximate solution.

Usually, the dimension of the state space $\mathcal{S}^n$ (i.e the number of automata in the network) is large (for example, in an image [49], n=3024 -number of pixels- and $|S| = 8$ -number of gray levels-), and thus presumably $m \ll n$ and the assumption of linear independence of the m patterns $\mathbf{X}_i$ is thus not very restrictive In the case $m<n$, the operator $\mathbf{XX}^+$ is the orthogonal projection on the vector space $\mathcal{L}$ spanned by patterns $\mathbf{X}_1, .., \mathbf{X}_m$ and $\mathbf{I}-\mathbf{XX}^+$ the orthogonal projection on $\mathcal{L}^\perp$ In the case of auto association, $\mathbf{W}$ being just $\mathbf{XX}^+$, mapping F is then nothing more than the projection on $\mathcal{L}$

3.3. Computational techniques.

Exact methods to compute $\mathbf{X}^+$ have been derived . they usually are very time consuming, since they involve computations of inverses of matrices However, every computer math-library always includes its efficient

program to compute the pseudo-inverse We will present here various methods

The first one is a recursive algorithm, due to Greville [33], which does not need to invert matrices and may be implemented by successively presenting the examples $X_1, ..X_m$ let us denote $\mathcal{X}_k = (\mathcal{X}_{k-1} \quad X_k)$ a matrix with k columns, where $\mathcal{X}_{k-1}$ is the submatrix of X formed with its k-1 first columns and X_k is the k-th column of X We thus have $X = \mathcal{X}_m$ Greville's theorem states that X^+ can be computed in m steps by

$$\mathcal{X}_k^+ = \begin{bmatrix} \mathcal{X}_{k-1}^+ (I - X_k p_k^t) \\ p_k^t \end{bmatrix} \qquad (18)$$

$$\text{and } p_k = \frac{(I - \mathcal{X}_{k-1}\mathcal{X}_{k-1}^+)X_k}{\|(I - \mathcal{X}_{k-1}\mathcal{X}_{k-1}^+)X_k\|^2} \qquad \text{if the numerator is non zero}$$

$$= \frac{(\mathcal{X}_{k-1}^+)^t \mathcal{X}_{k-1}^+ X_k}{1 + \|\mathcal{X}_{k-1}^+ X_k\|^2} \qquad \text{otherwise}$$

As $\mathcal{X}_1$ is just the first column X_1 of X we have.

$$\mathcal{X}_1^+ = \begin{cases} (X_1^t X_1)^{-1} X_1^t & \text{if } X_1 \text{ is non zero} \\ 0 & \text{otherwise} \end{cases}$$

and of course $X^+ = \mathcal{X}_m^+$

The complexity of Greville's algorithm is of order n^2m^2, which may be a problem in practical situations We will thus try to compute W directly instead of computing X^+ A slight adaptation of Greville [33] allows to recursively compute W By conserving the same notations for $\mathcal{X}_k$ and evident notations for $\mathcal{Y}_k$ and denoting $W_k = \mathcal{Y}_k \mathcal{X}_k^+$, we have

$$W_k = W_{k-1} - (W_{k-1}X_k - Y_k)p_k^t \qquad (19)$$

where p_k is the same vector as in Greville

At iteration step k, W_k is the solution to equation $WX_k = Y_k$ with minimum norm This matrix thus encodes the associations between input patterns $X_1,...,X_k$ and output patterns $Y_1,...,Y_k$.

If the columns of X are independent, the computation of W becomes simpler We then have

$$W_k = W_{k-1} - (W_{k-1}X_k - Y_k)[X_k^t / \|X_k\|^2] \quad \text{if } X_k \neq 0 \qquad (20)$$
$$W_{k-1} \quad \text{otherwise}$$

with

$$X_k = S_{k-1} X_k \qquad X_1 = X_1 \qquad (20')$$
$$S_k = S_{k-1} - X_k X_k^t / \|X_k\|^2 \qquad S_0 = I$$

The X_k can also be computed through the Gram-Schmidt orthogonalization process

$$X_1 = X_1 \qquad (20'')$$
$$X_k = X_k - \sum_{i=1..k-1} \langle X_k, X_i \rangle . X_i / \|X_i\|^2 \quad k=2,..,m$$

where the summation runs on all non zero X_i This result is similar to Pyle [64] More details can be found in Kohonen [49]

3.4. Approximated computation

The preceding algorithms may imply a huge computational burden if the dimension n of the patterns is large Moreover, simulations show that the linear learning scheme is very fault tolerant (see below) even approximate values of matrix W would do Therefore, approximation methods are used to estimate W instead of computing the exact matrix YX^+, we compute a matrix W which minimizes the error E between the output value WX and the desired value Y Of course many error functions can be used, the most classical being the quadratic norm which requires the minimization of

$$E = \sum_{k=1..m} \sum_{i=1..p} (\sum_{j=1..n} W_{ij} X_{jk} - Y_{ik})^2 \qquad (21)$$

This is a classical minimization problem, which can be solved by using gradient methods $\text{grad}_W E = 2(WX - Y)X^t$ is the gradient of E with respect to W We have presented various methods in §2 2 for example, the gradient method with increment (λ_k) is given by

W_1 arbitrary

$$W_k = W_{k-1} - \lambda_k(W_{k-1} X - Y)X^t \qquad (22)$$

This procedure has been shown to converge provided the λ_k are chosen such that

$$\sum_{k=1}^{\infty}\lambda_k = +\infty \text{ and } \sum_{k=1}^{\infty}\lambda_k^2 < \infty$$

This is satisfied in particular for $\lambda_k = \lambda_1/k$, but convergence may be very slow

One weakness, for our purpose, of these gradient methods is that they require the full knowledge of matrices X and Y We would prefer instead methods which allow the "knowledge" of the network to gradually evolve when new data are presented, without of course destroying previous memories In other words, we want to design a method which would allow to compute W_k in terms of W_{k-1}, by making the correction needed to take into account the observed error when retrieving the new pattern X_k by W_{k-1} This can be done by using a correction term proportional to $(W_{k-1} X_k - Y_k)$, which is the discrepancy between the desired output and the output computed with W_{k-1} This gives the following algorithm.

$$W_k = W_{k-1} - (W_{k-1} X_k - Y_k)c_k^t \qquad (23)$$

where c_k^t is a vector which will allow to adapt the weights to the new incoming pattern X_k If the columns X_k are linearly independent, then one possible choice is $c_k = p_k$, and then, at each iteration step, matrix W_k is optimal for patterns $X_1, ,X_k$ and adding one more pattern just requires to iterate formula (23) This is just the case discussed in (19)

In the general case, going in the direction computed with $c_k = \lambda X_k$ decreases the error, since $(W_{k-1} X_k - Y_k)X_k^t$ is the gradient of the error E_k on the k-th pattern X_k

There also exist more methods, which do not require the optimality of matrix W_k at each iteration step In "stochastic" gradient methods, each step minimizes the error on one pattern only One of the most famous of these stochastic methods is the Widrow-Hoff procedure [81] patterns $X_1, ,X_m$ are introduced sequentially and repeatedly, until convergence To simplify notation, we will denote this sequence (X_1,Y_1), (X_2,Y_2), . .,(X_m,Y_m), (X_1,Y_1), . by (X^1,Y^1),(X^2,Y^2), ,(X^m,Y^m),(X^{m+1},Y^{m+1}), .. At each step, a new pattern is presented and the error on this pattern is minimized it is necessary to go in the opposite direction of the gradient of the error E_{i+1} on pattern x^{i+1} This gives the following rule

$$\mathbf{W^0} \text{ arbitrary}$$
$$\mathbf{W^{i+1}} = \mathbf{W^i} - \lambda_i(\mathbf{W^i X^{i+1}} - \mathbf{Y^{i+1}})[\mathbf{X^{i+1}}]^t \qquad (24)$$

There is no existing proof of the convergence of this algorithm However, in practice, it has been observed that with $\lambda_i=\lambda_1/i$, the algorithm usually converges to a solution The problem of adding one pattern to the memory is quite easy, it is sufficient to start with $\mathbf{W^0} = \mathbf{W}$ and iterate with the new pattern until convergence As the starting matrix is probably close to the solution, the computing time will not be long before convergence.

3.5. Other models

CONSTRAINED GRADIENTS

It is sometimes useful to impose on the connection matrix some constraints which reflect properties of the information flows in the network For example, the connection weight of an element with itself, W_{ii}, could be bounded, or even cancelled, so that its influence would not overkill the others In this case, matrix $\mathbf{W}$ is constrained to have a bounded -or zero- diagonal and can be computed by gradient projection methods In the case of $W_{ii}=0$, there exist solutions to equation (13) of the form. $\mathbf{W=YX^*+Z(I-XX^*)}$

HEBB LAW

In 1949, D Hebb [34] proposed his "reinforcement law" based on biological evidence any two neurons firing at the same time, when "shown" a given pattern, will have their connection strength reinforced This is also related to the theory of learning by evolution of the synapse structure (see Changeux [14]) Stated in unformal fashion by Hebb, this rule has been repeatedly used to compute weights W_{ij}, through algorithms such as.

$$W_{ij} = W_{ij} + X_{ik}X_{jk} \qquad (25)$$

if pattern $\mathbf{X_k}$ is shown to the network $X_{ik}=1$ if automaton i is on in pattern $\mathbf{X_k}$, -1 otherwise In the case of auto-association between patterns $\mathbf{X_1}$, $\mathbf{X_2}$, .,$\mathbf{X_m}$, algorithm (25) is successively applied to all patterns and the corresponding weights added Matrix $\mathbf{W}$ is thus given by $\mathbf{W = XX}^t$ (26) In the case of hetero-association between patterns ($\mathbf{X_1}$,$\mathbf{Y_1}$), ($\mathbf{X_2}$,$\mathbf{Y_2}$), .., ($\mathbf{X_m}$,$\mathbf{Y_m}$), this rule can be extended $W_{ij} = W_{ij} + Y_{ik}X_{jk}$ and $\mathbf{W = YX}^t$

Note that, when patterns $\mathbf{X_k}$ are linearly independent and orthonormal, i e $\langle \mathbf{X_i},\mathbf{X_j} \rangle=0$, $\forall$ i≠j, and $\|\mathbf{X_k}\| =1$, then $\mathbf{X^*} = \mathbf{X}^t$ and $\mathbf{W} = \mathbf{YX}^t$ is the solution of minimum norm of equation (13) Hence Hebb rule, based on biological evidence, can be considered as an approximated version of the theoretical results in section 3 2

4. RETRIEVAL

4.1. Introduction

After the learning session, during which the network connection matrix **W** has been computed, we can use the network to perform recognition tasks. The main interest of these learning techniques is that they lead to efficient retrieval even if the data are noisy or partial. This will thus allow to

- classify patterns close to memorized items
- retrieve memorized items from noisy or incomplete patterns.

that is to say, implement a function of ASSOCIATIVE MEMORY.

Various procedures have been designed for retrieval. remember that our learning procedure was based on a linear network, but, as can be expected, non linear retrieval rules may be used for improved performances.

4.2. Linear retrieval.

We will now discuss, in the case of a linear retrieval process, the properties of resistance to noise depending on the model of noise which is applied to the inputs. we will derive the "best" choice of matrix **W** for various models of noise.

We have seen that the output of a linear network, with connection matrix **W**, when the input is some vector **x**, is $\mathbf{y} = \mathbf{Wx}$. If $\mathbf{x}=\mathbf{X_k}$, then **y** will be exactly $\mathbf{Y_k}$. If **x** is only close to some $\mathbf{X_k}$, then **y** will be close to $\mathbf{Y_k}$.

suppose $\mathbf{W}=\mathbf{YX}^+$

then $\mathbf{y} = \mathbf{YX}^+\mathbf{x} = \mathbf{YX}^+[(\mathbf{XX}^+)\mathbf{x}]$ (because $\mathbf{X}^+\mathbf{XX}^+ = \mathbf{X}^+$, by the definition of the pseudo-inverse)

$$\Rightarrow \quad \mathbf{y} = \mathbf{YX}^+\mathbf{x}_{\mathcal{L}}$$

where $\mathbf{x}_{\mathcal{L}}$ denotes the projection of **x** onto the vector space $\mathcal{L}$ spanned by the $\mathbf{X_k}$ (see §3.1)

$$\mathbf{x}_{\mathcal{L}} = \sum_{i=1 \ldots m} \alpha_i \mathbf{X_i}$$

$$\Rightarrow \quad \mathbf{y} = \mathbf{YX}^+ \sum_{i=1 \ldots m} \alpha_i \mathbf{X_i} = \sum_{i=1 \ldots m} \alpha_i \mathbf{YX}^+\mathbf{X_i} = \sum_{i=1 \ldots m} \alpha_i \mathbf{Y_i}$$

But if **x** is close to $\mathbf{X_k}$, then α_k is large compared to the other α_i, and thus **y** is close to $\mathbf{Y_k}$

Simulations (see Kohonen [49]) show that results are better if the memorized patterns are close to mutually orthogonal, thus any preprocessing which tends to orthogonalize the patterns would lead to increased efficiency

The signal-to-noise ratio is improved after retrieval

PROOF suppose $\mathbf{x} = \mathbf{x_k} + \mathbf{v}$, where $\mathbf{v}$ is some noise vector Then

$$\mathbf{y} = \mathbf{YX^+x} = \mathbf{YX^+}[(\mathbf{XX^+})(\mathbf{x_k} + \mathbf{v})] \quad \text{(because } \mathbf{X^+XX^+} = \mathbf{X^+}\text{)}$$

$$\Rightarrow \quad \mathbf{y} = \mathbf{YX^+}[\mathbf{x_{\mathfrak{L}}} + \mathbf{v_1}]$$

where $\mathbf{v_1}$ denotes the orthogonal projection of $\mathbf{v}$ on $\mathfrak{L}$

It has been shown [49], for auto-association, that if $\mathbf{v}$ is of constant length and has a direction uniformly distributed in $\mathbb{R}^n$ then the variance

$$\mathrm{Var}(\|\mathbf{Wx} - \mathbf{x_k}\|) = (m/n)\|\mathbf{x} - \mathbf{x_k}\|$$

Hence, if $m<n$, the noise is damped in the orthogonal projection. In the case of hetero association, performances are usually quite good, but the output noise-to-signal ratio may eventually become large even for small input noise-to-signal ratio, when m/n increases and the determinant of $\mathbf{XX^t}$ goes to 0 [78]

More generally, suppose $\mathbf{x} = \mathbf{x_k} + \mathbf{v}$, then $\mathbf{y} = \mathbf{Wx} = \mathbf{W}(\mathbf{x_k} + \mathbf{v}) = \mathbf{Wx_k} + \mathbf{Wv}$. Hence, to improve the efficiency in terms of signal-to-noise ratio, we can try to find the matrix $\mathbf{W}$ which minimizes the dispersion of the remaining noise at the output $\mathbf{Wv}$

If $\mathbf{W}$ is of the form of equation (16) and $\mathbf{v}$ is of zero mean, then the dispersion of the remaining noise at the output is

$$\mathrm{Var}(\mathbf{Wv}) = E[\mathbf{Wv}\,(\mathbf{Wv})^t] = E[\mathbf{Wv}\,\mathbf{v}^t\mathbf{W}^t] = \mathbf{W}\,E[\mathbf{v}\,\mathbf{v}^t]\,\mathbf{W} = \mathbf{W}\,\boldsymbol{\Sigma}\,\mathbf{W}^t$$

where $\boldsymbol{\Sigma}$ is the dispersion matrix of noise $\mathbf{v}$

$$\Rightarrow \quad \mathrm{Var}(\mathbf{Wv}) = [\mathbf{M_0} + \mathbf{Z}\,\mathbf{M_1}]\,\boldsymbol{\Sigma}\,[\mathbf{M_0} + \mathbf{Z}\,\mathbf{M_1}]^t \text{ with } \mathbf{M_0} = \mathbf{YX^+}$$
$$= [\mathbf{M_0} + \mathbf{Z}\,\mathbf{M_1}]\,\boldsymbol{\Sigma}\,[\mathbf{M_0}^t + \mathbf{M_1}^t\mathbf{Z}^t] \text{ and } \mathbf{M_1} = \mathbf{I} - \mathbf{XX^+}$$
$$= \mathbf{M_0}\boldsymbol{\Sigma}\,\mathbf{M_0}^t + \mathbf{M_0}\boldsymbol{\Sigma}\mathbf{M_1}\mathbf{Z}^t + \mathbf{Z}\,\mathbf{M_1}\,\boldsymbol{\Sigma}\,\mathbf{M_0}^t + \mathbf{Z}\,\mathbf{M_1}\,\boldsymbol{\Sigma}\,\mathbf{M_1}\mathbf{Z}^t$$
$$= \mathbf{M_0}\boldsymbol{\Sigma}\,\mathbf{M_0}^t + \mathbf{AZ}^t + \mathbf{ZA}^t + \mathbf{ZBZ}^t$$

with evident notations for $\mathbf{A}$ and $\mathbf{B}$ Note that $\mathbf{M_1}$ and $\mathbf{B}$ are symmetric. Let V_{ij} be the coefficient in line i and column j of matrix Var$(\mathbf{Wv})$ We have

$$V_{ij} = (\mathbf{M_0 \Sigma M_0}^t)_{ij} + \sum_{k=1..n} A_{ik}(Z^t)_{kj} + \sum_{k=1..n} Z_{ik}A_{jk} + \sum_{k=1...n} Z_{ik} \sum_{\ell=1...n} B_{k\ell}(Z^t)_{\ell j}$$

$$= (\mathbf{M_0 \Sigma M_0}^t)_{ij} + \sum_{k=1..n} A_{ik}\, Z_{jk} + \sum_{k=1...n} A_{jk}\, Z_{ik} + \sum_{k,\ell=1..n} B_{k\ell}\, Z_{ik}\, Z_{j\ell}$$

To minimize Var$(\mathbf{Wv})$, it is thus sufficient to choose $\mathbf{Z}$ such that

$$\partial V_{ij} / \partial Z_{\alpha\beta} = 0 \qquad \forall \alpha, \beta$$

with.

$$\partial V_{ij} / \partial Z_{\alpha\beta} = 0 \qquad \forall \alpha \neq j,i$$

$$\partial V_{ij} / \partial Z_{i\beta} = \delta_{ij} A_{i\beta} + A_{j\beta} + \sum_{\ell=1..n} B_{\beta\ell} Z_{j\ell} + \delta_{ij} \sum_{k=1..n} B_{k\beta} Z_{ik}$$

$$\partial V_{ij} / \partial Z_{j\beta} = A_{i\beta} + \delta_{ij} A_{j\beta} + \delta_{ij} \sum_{\ell=1..n} B_{\beta\ell} Z_{j\ell} + \sum_{k=1...n} B_{k\beta} Z_{ik}$$

where δ_{ij} is 1 iff i=j, 0 otherwise. Thus $\mathbf{Z}$ satisfies the equation

$$\mathbf{ZB = -A} \text{ or } \mathbf{Z M_1 \Sigma M_1 = -M_0 \Sigma M_1}$$

which is of the same kind as equation (13) Thus there exists a matrix $\mathbf{Z}$ of minimum quadratic norm which minimizes the dispersion of the noise at the output, it is given by $\mathbf{Z = -AB^+}$

If noise $\mathbf{v}$ has a diagonal dispersion matrix $\mathbf{\Sigma} = \sigma^2 \mathbf{I}$, then $\mathbf{Z}$ is such that.

$$\mathbf{Z M_1 \Sigma M_1 + M_0 \Sigma M_1 = 0} \Rightarrow \mathbf{Z[I - XX^+][I - XX^+]}\sigma^2 + \mathbf{YX^+[I - XX^+]}\sigma^2 = \mathbf{0}$$

$$\Rightarrow \mathbf{Z[I - XX^+] = 0} \text{ (if } \sigma^2 \neq 0) \Rightarrow \mathbf{Z = 0} \text{ or } \mathbf{I - XX^+ = 0}$$

Hence, for a noise $\mathbf{v}$ with 0 mean and dispersion $\sigma^2 \mathbf{I}$, the solution to (13) $\mathbf{W = YX^+}$(with Z=0) minimizes the remaining noise at the output □

In many practical situations, the state space $\mathbf{S}$ is not $\mathbb{R}$, but a discrete space, for example |S|=r for an image in r gray levels Then, the input patterns $\mathbf{X_k}$ are in this discrete state space, but $\mathbf{W}$ has its elements in $\mathbb{R}$ and thus the output $\mathbf{y}$ is in $\mathbb{R}^p$ The output must then be thresholded so as to be interpreted in the same terms as the inputs (e g image) Usually, performances will be improved, in the case of auto association, if the thresholded

output is fed-back to the network and the whole process of projection, thresholding reiterated (fig 6) this can be seen when the linear operator is used With input $\mathbf{x}$, the projection $\mathbf{XX^+}$ builds vector $\mathbf{x}\mathfrak{L}$ which may lie in between the memorized items $\mathbf{X_i}$, very close -but not identical- to some $\mathbf{X_k}$ As further projections will keep $\mathbf{x}\mathfrak{L}$ fixed, there will be no way to move $\mathbf{x}\mathfrak{L}$ towards $\mathbf{X_k}$ repeated thresholding, by taking $\mathbf{x}\mathfrak{L}$ out of $\mathfrak{L}$ will allow further approximations to take place, thus improving performances This leads us to retrieval processes which are in fact far from linear and there does not exist any more theoretical reason why matrix $\mathbf{W = YX^+}$ should be optimal for these retrieval processes.

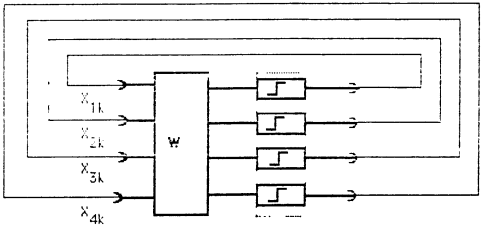


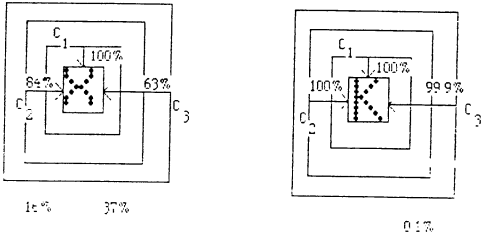
Figure 6: Iterative network.
The figure shows a network, with connection matrix $\mathbf{W}$ The retrieval process consists in sending the input, $\mathbf{X_k}$ in the figure, through the network which computes $\mathbf{WX_k}$, then thresholding this result, component by component, then feeding-back the thresholded value into the network. The process can be re-iterated until a fixed point is reached, or stopped after a certain number of iterations

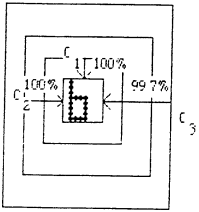
4.3. Threshold networks.

The model of threshold networks has been largely studied [27,31,45,52, 61,80] almost independently from the work presented so far in the previous paragraphs However, when those networks are proposed to implement associative memories [27,45], it is easy to see, as we mentioned before (§3.5), that they are very close to the linear model, with thresholded output when patterns $\mathbf{X_k}$ are linearly independent and orthonormal, then the optimal associative memory, i.e the solution of minimum quadratic norm of equation (13) $\mathbf{WX=Y}$ is given by (26) $\mathbf{W = XX^t}$ This rule, inspired from Hebb reinforcement law [34], was first introduced in Little [52] and further elaborated by Hopfield [45] who proposed to use an ENERGY function.

FIGURE 7-1

FIGURE 7-2





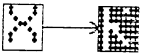


FIGURE 7-3

Figure 7: Threshold network to retrieve the 3 letters "X", "b","K"
The learning rule is Hebb law with 0-diagonal The memorized patterns (Fig 7-1) are clearly not independent, which leads to fault tolerance properties rather weak The figure 7-2 shows the attractivity of each memorized item in its three first neighborhoods C_i (i bits different) Spurious attractors appear to be very close to the memorized items, since, for example, 37% of the items in neighborhood C_3 of "X" are attracted to a spurious attractor
Note also that such networks cannot achieve pattern invariance as is shown in the bottom part of the figure (fig 7-3). an "X" translated leads to a spurious attractor Going beyond this limitation requires networks with more complex structure (§5)

One very distinctive features of the approach developed below is the use of ITERATIONS in the linear retrieval model, a pattern $\mathbf{x}$ is projected (in case of auto-association) onto $\mathfrak{E}$, and this gives the final output We will present here models where the output is the result of an iteration process run on the network Various kinds of iteration modes have been defined on networks [67] which might eventually lead to different results

In this section, we will fully develop this model and show how tools introduced in other chapters [32,67] can be used to give theoretical results on the retrieval performances. The material presented here can be found in more details in [23,31]

Using the definition of a threshold automaton (1), we have

Definition We will say that $F : \mathfrak{E}^n \to \mathfrak{E}^n$ is a THRESHOLD NETWORK iff all its components $(f_1, \ldots, f_n)$ are threshold automata (see (1), (2)).

$$\forall i(1, \ldots, n), \forall \mathbf{x} \in \mathfrak{E}^n, \quad f_i(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum_j W_{ij} x_j - \theta_i \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

where $\mathbf{W}=(W_{ij})$ is a real nxn matrix and $\boldsymbol{\theta}=(\theta_i)$ is a real n-vector, denoting, as usual, the weights and thresholds of the network.

In the following, we will use, unless specifically mentioned, the case where $\mathfrak{E} = \{0,1\}$ We have seen before that this case could easily be rewritten into the $\mathfrak{E} = \{-1,1\}$ case

f_i is called a STRICT threshold automaton (see chapter 4) iff

$$f_i(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum_j W_{ij} x_j - \theta_i > 0 \\ -1 & \text{if } \sum_j W_{ij} x_j - \theta_i < 0 \end{cases}$$

It is easy to show that any threshold automaton is equivalent to a strict threshold automaton [31] by changing the thresholds. We will thus, in the following, make the assumption of strict threshold, which is necessary in some proofs

It was shown [66,67] that different modes of iteration could be defined on a network We will use the following notations

Notation: Let $F : \mathfrak{E}^n \to \mathfrak{E}^n$ be an automata network of n elements and $\mathbf{x}^0 \in \mathfrak{E}^n$ The different ITERATION MODES on F with initial condition $\mathbf{x}^0$ are defined by $\mathbf{x}^0=\mathbf{x}(0)$ and $\mathbf{x}(t) \in \mathfrak{E}^n$, the state of F at time t, where $\mathbf{x}(t)$ is

- PARALLEL ITERATION $\forall t \geq 0, \mathbf{x}(t+1) = F[\mathbf{x}(t)]$

- SEQUENTIAL ITERATION associated to the permutation π of $(1, ..,n)$

$$\forall t \geq 0,\ \mathbf{x}(t+1) = F_{\pi}[\mathbf{x}(t)]$$

Elements i of the network change state one at a time in the order prescribed by π

- BLOCK-SEQUENTIAL ITERATION associated to a partition $(I_k)_{k=1...p}$ of $\{1,...,n\}$, ordered (which means that $\forall \alpha \in I_i, \forall \beta \in I_j, i<j \Rightarrow \alpha < \beta$) as:

$$\mathbf{x}(t+1) = F_{(I_k)}[\mathbf{x}(t)]$$

Elements i in I_k change state in parallel, and the blocks I_k are iterated serially one after the other

- RANDOM ITERATION, with visiting strategy $(i_t)_{t\geq 0}$ (where $(i_t)_{t\geq 0}$ is a random process with values in $\{1,..,n\}$)

$$\forall t \geq 0,\ \mathbf{x}(t+1) = F_{(i_t)}[\mathbf{x}(t)]$$

Elements i of the network change state one at a time in the order prescribed by i_t

Definition : Let $F : \mathbb{E}^n \to \mathbb{E}^n$ be a threshold network. We define the ENERGY of the network for the different iteration modes as follows:

- sequential, block sequential or random iterations. $\forall \mathbf{x} \in \mathbb{E}^n$,

$$E(\mathbf{x}) = -1/2 \sum_{i=1}^{n} x_i \sum_{j\neq i} W_{ij} x_j + \sum_{i=1}^{n} (\theta_i - W_{ii})\, x_i$$

- parallel iteration $\forall \mathbf{x} \in \mathbb{E}^n$, $E[\mathbf{x}(t)] = \mathfrak{E}[\mathbf{x}(t),\mathbf{x}(t-1)]$

with $$\mathfrak{E}[\mathbf{u},\mathbf{v}] = \sum_{i=1}^{n} u_i \sum_{j=1}^{n} W_{ij} v_j + \sum_{i=1}^{n} \theta_i\, [u_i + v_i]$$

Remark: this energy coincides with the energy of a spin glass with field θ, as defined for example in [45], only in the case of sequential, block sequential and random iterations. The energy for the parallel iteration does not have any physical meaning in this context (see also [32])

The importance of the concept of energy is due to the following theorem (see [32] for proofs)

Theorem: Let $F : \mathbb{E}^n \to \mathbb{E}^n$ be a threshold network.

Then $$\forall t \geq 0,\ \mathbf{x}(t+1) \neq \mathbf{x}(t) \Rightarrow E[\mathbf{x}(t+1)] < E[\mathbf{x}(t)]$$

Appendice C

Automata Networks as a Tool for Knowledge Acquisition

Automata Networks as a Tool for Knowledge Acquisition

Y. le Cun, P. Gallinari, F. Fogelman-Soulié, S. Thiria

Décembre 1985.

AUTOMATA NETWORKS AS A TOOL FOR KNOWLEDGE ACQUISITION

Y. LE CUN *, P. GALLINARI **, F. FOGELMAN SOULIE *, S. THIRIA ****

* Laboratoire IAAI
Dép. d'Informatique
ESIEE
89 rue Falguière
75 015 PARIS,FRANCE

** CNAM
292 rue Saint Martin
75 003 PARIS
FRANCE

*** Laboratoire LIA
Ecole des Hautes Etudes en Informatique
Université de Paris 5
45 rue des Saints Pères
75 006 PARIS, FRANCE

and

Laboratoire de Dynamique des Réseaux
CESTA
1 rue Descartes
75 005 PARIS, FRANCE

TRACK: SCIENCE
FULL PAPER
TOPIC. KNOWLEDGE ACQUISITION AND LEARNING

KEYWORDS: LEARNING, CONNECTIONISM, AUTOMATA NETWORK, GRADIENT PROPAGATION, DISTRIBUTED REPRESENTATION OF KNOWLEDGE, ASSOCIATIVE MEMORY, FAULT TOLERANCE

ABSTRACT

In this paper, we discuss the paradigm of LEARNING ON AUTOMATA NETWORKS and its connections with classical AI machine learning. This approach is actually developing under the name of CONNECTIONISM or PARALLEL DISTRIBUTED PROCESSING [20] and is used in numerous fields of computer science, in particular AI (vision, natural language, games,) or pattern recognition, associative memory,....

We first present the framework used for learning-from-examples on automata networks: an automata network is a pool of processing elements linked through weighted connections. A learning session consists in presenting a set of examples, together with their associated responses. The weights are progressively adapted so as to decrease the discrepancy between the output computed by the network and the desired output (the response).
We then give an algorithm to modify these weights, the GRADIENT BACK PROPAGATION ALGORITHM, for the case of MULTI LAYERED NETWORKS. Its performances are illustrated on two applications: associative memories and medical diagnosis.

1- Introduction

Learning has recently become a major concern in AI. As the complexity of the problems at hand increased, it has become necessary to include more knowledge, domain dependent in the programs. The problem of designing these knowledge bases then raised the question of knowledge acquisition, which now proves to be one of the major bottlenecks of many AI applications.

In the classical «machine learning» approach [10, 15], knowledge is encoded in a rule space, which is initially fed into the machine and progressively updated to allow for generalization and thus increasing performance. But many problems arise in this approach:

- The learning procedures are highly problem-dependent, and it is difficult to extend the basic concepts to other applications.
- The learning procedures usually rely, in a way or another, on the exploration of a huge parameter space. As is classical in many AI problems, heuristics have to been designed to correctly canalize the learning process to a reduced parameter sub-space, thus avoiding exponentially growing exploration time.
- Initial representations must be provided to the system -through rules for example- and this step may be critical for the final performances. It is clear that the generalization capabilities of the system, i.e. its behaviour when faced with an unknown situation, heavily relies on the choice of the initial representations used. It is clear too that it is usually impossible to define the best representation: this is an ill posed problem. From a purely pragmatic standpoint, no criterion exists that can decide which solution is best.

In this paper, we propose an algorithm which performs a learning-from-examples task, through a trial-and-error procedure [12, 20]. The main property of this algorithm is that it provides automatic generation of «good» internal representations, by extracting the regularities from the data. This process, if correctly completed, allows for GENERALIZATION, i.e. good response to unknown examples.

We discuss in §2 the general idea of our learning algorithm, then present the technical tools: threshold automaton, in §3, and back-propagation algorithm, in §4. Two typical applications are then described to illustrate the learning capabilities of the model: associative memories, §5, and medical diagnosis, §6. In conclusion, we discuss some technical problems and further improvements of this approach.

2- Learning and the credit assignment problem

Our algorithm is based on the formalism of AUTOMATA NETWORKS, i.e. sets of interconnected computing elements that interact by exchanging information. It is possible to make a rough comparison between an automata network and a set of rules (in the proposition calculus formalism). Consider a set of rules that solves a particular problem, and the interaction graph associated to it. A rule produces a result (e.g. a truth value) and sends it to the rules which take it

as a premice The basic idea is to view each rule as an automaton in a network whose connections are defined by the interaction graph The learning takes place by modifying the transition functions of the automata, and eventually the connection graph In general this problem is too complex and some additional assumptions are necessary to solve it

We choose to use a very restricted class of automata, namely the THRESHOLD AUTOMATA (FIG 1) The output S of a threshold automaton is -1 if the weighted sum of its inputs x_i is negative, and +1 otherwise The values of the weights W_i define the precise function computed by the threshold element.

$$S = \begin{cases} 1 & \text{if } \sum_i W_i x_i \geq 0 \\ -1 & \text{otherwise} \end{cases} \qquad (1)$$

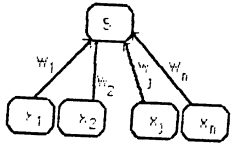


FIGURE 1 THRESHOLD AUTOMATON

A threshold network can then be seen as a pool of processing elements linked by a set of valued (weighted) connections

The learning algorithm works as follows the network is shown with a set of input descriptors for which it produces an output This output is compared with the desired output, and the structure of the knowledge base is changed accordingly The major problem with this approach is how to identify, when the final output is wrong, which step led to the error, and thus how to update the knowledge base This is classically referred to as the CREDIT ASSIGNMENT PROBLEM This question is strongly related to the problem of extracting good internal representations since the learning process is supposed to find adequate intermediate predicates that take into account the underlying regularities of the task

The learning rule thus merely consists in a procedure which iteratively modifies the weights so as to compute the desired input-output relationship A number of such learning rules are known and used for classifier synthesis The study of threshold automata and their application to learning systems began with the early works on adaptive systems [18, 26], but soon led to a dead end The main reason for that was found in the intrinsic limitations of systems made from a single threshold element as is shown in the following section [16]. These limitations can be overcome by using networks of many threshold elements instead of a single one But in this case, the credit assignment problem must be solved since the final output of the system depends on several elementary inference steps produced by each of the automata involved in the

decision These models are called CONNECTIONNIST because their "knowledge" is stored in the connections (the weight values)
These models have been widely developed in the past few years, and involve a growing number of research teams, with applications to a variety of fields vision [3, 4, 22] and perception [7], pattern recognition [8], natural language [25], cognitive science [2, 20], optimization [9],

The learning rule we propose can be used with any network structure, allowing complex reasoning to take place, thus overcoming the limitations of the classical models Yet, in the following, we will describe the learning rule in the particular case of feed-forward networks, i.e. networks composed of several layers of automata where the connections can only go from lower layers forward to higher layers This restriction is made only for sake of simplicity and can be compared to the prescription of non circularity in rule based systems
In this framework, the credit assignment problem is equivalent to the problem of finding the functions that cells in intermediate layers must compute. These cells are called hidden units because they have no direct interaction with the outside world (see below)

Two classes of algorithms have been recently proposed for learning with hidden units The first one -the so-called Boltzmann Machine algorithm [1], - used hidden units and non deterministic automata it proves the feasibility of solving the credit assignment problem, thus opening the door for further research on «hard» problems The second algorithm is the Back-Propagation algorithm proposed independently by several groups [12, 13, 17, 20]. These groups have recently produced some interesting applications [6, 19, 24] and it seems possible today to envision the application of these ideas to machine learning

The principal characteristics of these systems is that knowledge is represented in a distributed fashion among the connections Given a learning rule, the network will encode in its connections a knowledge progressively extracted from the learning set In fact the learning process consists in exploring the weight space so as to construct the network that will optimally perform the desired task This learning process allows the network to find internal representations that enable generalization The search process is guided by restricting the exploration to a small part of the space through the use of a METRIC which measures the performances of the network. Learning thus amounts to finding the weights which maximize the performance function We postulate that the nature of this function as well as the structure of the network lead to a solution that has some good properties, specifically concerning the quality of the generalization

Eventually, after the learning phase has been completed, one can examine the state of the network and deduce the «rules» it has built Here, the word «rule» has not the classical meaning, since all of the automata are activated and produce an output during the reasoning process This is not the case with classical rule-based systems where a rule is activated only if the values of every premices are known The descriptor value "unknown" does not play a special role in our approach Looking for rules discovered by the network may

be quite tricky because it uses a distributed representation of knowledge embedded in the strength of connections However, in some cases [19], weights have been shown to code for rules meaningful in the domain of the examples (such as nationality, generation, in the framework of a genealogy tree [19]) Extracting explicit rules is not the main goal of the approach However, it could be a result of this line of research to provide classical AI programs with knowledge automatically extracted by networks

Our main goal is to provide a general framework for learning without a priori knowledge Ideally the Back Propagation algorithm should be able to perform on every set of input-output pairs However this is clearly not realistic and this goal still remains out of reach To be efficient, networks must be given some a-priori knowledge by choosing an appropriate architecture and a good set of initial connection strengths The behaviour of the network is also strongly related to the coding of the input and output data

3 The threshold automaton

The threshold automaton -or formal neuron- was first introduced by McCulloch and Pitts [14] and was further used as the prototype of adaptive or learning machines
It can be interpreted as a unit which receives inputs $x_1, x_2, \ldots, x_n$ from other units and sends an output S, computed as $S = f(x_1, x_2, \ldots, x_n)$ where f is defined by (1) (see FIG 1)

$$S = \begin{cases} 1 & \text{if } \sum_i W_i x_i \geq 0 \\ -1 & \text{otherwise} \end{cases} \qquad (1)$$

Coefficients W_i are called the WEIGHTS of the automaton

One of the first algorithms proposed for learning correct weights was the PERCEPTRON LEARNING RULE [16, 18] Other learning rules, based on the minimization of a cost function, were introduced later on [11, 26]
They are based on the following idea
Assume that we want to teach a threshold automaton to classify a set of vector-patterns X_k into two classes C^+ and C^-. We label Y_k the class of pattern X_k $Y_k = +1$ if X_k is in C^+, $Y_k = -1$ if X_k is in C^-.
We thus want to realize the association of input-output pairs $(X_1, Y_1), (X_2, Y_2), \ldots, (X_m, Y_m)$
According to equation (1), the threshold automaton, when its input is X_k, outputs S_k computed by

$$S_k = 1[A(X_k)] \qquad (2)$$

where

$$1[u] = \begin{cases} +1 \text{ if } u \geq 0 \\ -1 \text{ otherwise} \end{cases}$$

and

$$A(x) = \sum_i W_i x_i \qquad (3)$$

A(x) denotes the total weighted input to the automaton

A cost function measuring the discrepancy between the computed output S_k and the desired output Y_k can be chosen as, for example

$$C(W) = \| S_k - Y_k \|$$

or
$$C(W) = \sum_k [A(X_k) - Y_k]^2 \qquad (4)$$

This criterion can be minimized by various techniques (see [6, 26] for more details) such as for example

- gradient descent method

$$W_{t+1} = W_t - \varepsilon(t)\, \mathrm{grad}_W [C(W_t)] \qquad (5)$$

which, by equation (4), gives

$$W_{t+1} = W_t - \varepsilon(t) \sum_k [A(X_k) - Y_k] X_k \qquad (6)$$

This procedure requires the knowledge of the total cost C for all patterns X_k. It is usually more convenient to introduce the patterns one at a time and progressively modify the weights. This procedure is known as the

- stochastic gradient method or Widrow Hoff rule [26].

Pairs (X_1, Y_1), (X_2, Y_2), .. , (X_m, Y_m), (X_1, Y_1), .. are presented indifinetely. At time t, let us denote (X^t, Y^t) the pair which is presented. Then weights are updated by

$$W_{t+1} = W_t - \varepsilon(t) [A(X^t) - Y^t] X^t \qquad (7)$$

which only requires the knowledge of one input-output pair at a time.

Note that this learning procedure is fully supervised: the desired output must be given for each input. Limitations of these procedures are well known [6, 16, 23]: they can only learn to classify LINEARLY SEPARABLE inputs or, stated in Minsky-Papert's words, they can only learn first order predicates. Going further these limitations requires the use of networks of automata with hidden units, i.e. units which are not input or output units [23]: these units are used by the network to decompose a «hard» problem into successive elementary sub-problems. The question is then to generalize the previous learning procedures: this requires a way by which a desired state can be assigned to these hidden units. As they cannot «see» the inputs or output, there is no direct error signal available for them. This credit assignment problem is nothing more here than the problem of generating the «good» hidden units, i.e. good internal representations.

In the following section, we will present the back-propagation algorithm which provides a means to propagate error signals to these hidden units in multi-layers networks.

4- THE BACK-PROPAGATION ALGORITHM

The idea underlying the introduction of MULTI-LAYERED NETWORKS is to use successive decoding layers to extract relevant features in the data. These layers are made with threshold automata whose weights will be modified by the training process. In fact we will use here a modified version of threshold automata: «smooth» threshold automata. Their output is computed by

application of a smooth function f (a sigmoid function in this paper FIG 2) instead of thresholding 1

$$S = f[A(x)] \tag{8}$$

where A(x) is the total weighted input to the automaton

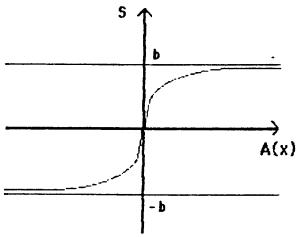


FIGURE 2 SMOOTH THRESHOLD AUTOMATON

A multi-layered network is made from successive layers, labelled 0,1, .,N (FIG 3) Connections may only go from one layer forward to layers with larger labels More complicated architectures may be designed, but we will not use them in this paper Layer 0 receives the inputs $\mathbf{x^1}$, $\mathbf{x^2}$, ...,$\mathbf{x^m}$ and layer N the associated desired output $\mathbf{y^1}$, $\mathbf{y^2}$, ...,$\mathbf{y^m}$.

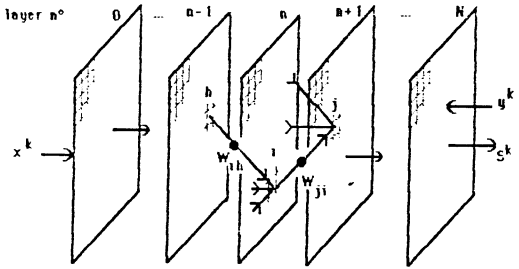


FIGURE 3 MULTI-LAYERED NETWORK

The output $\mathbf{S^k}$ associated to $\mathbf{x^k}$ is computed by propagating $\mathbf{x^k}$ from the input to the output layer (FIG 4) Then $\mathbf{S^k}$ is compared to the desired vector $\mathbf{y^k}$ The "error" function, which measures the discrepancy between $\mathbf{S^k}$ and $\mathbf{y^k}$, is:

$$C[\mathbf{W}(t)] = \sum_j [S^k_j - y^k_j]^2 \tag{9}$$

The weight vector will be modified so as to decrease this cost, using the stochastic gradient method (7)

$$W_{ij}(t+1) = W_{ij}(t) - \varepsilon(t)\, \partial C / \partial W_{ij} \qquad (10)$$

where $\varepsilon(t)$ is the iteration step at time t

The problem is to compute the $\partial C / \partial W_{ij}$ Chain rule gives

$$\partial C / \partial W_{ij} = \partial C / \partial A_i \cdot \partial A_i / \partial W_{ij} = \partial C / \partial A_i \cdot x_j$$

- for a cell i on the last layer and pattern $\mathbf{x}^k$, we denote S^k_i the state of the cell in the associated output $\mathbf{S}$ we then have

$$\partial C / \partial A_i = 2[S^k_i - y^k_i]\, \partial S^k_i / \partial A_i$$
$$= 2[S^k_i - y^k_i]\, f'(A_i)$$

Let us denote $g_i = \partial C / \partial A_i$. We thus have

$$g_i = 2[S^k_i - y^k_i]\, f'(A_i) \qquad (11)$$

- for the other layers

$$\partial C / \partial A_i = \sum_h \partial C / \partial A_h \cdot \partial A_h / \partial A_i$$

where h indexes the cells which receive their input from i Hence, from (8) and (9)

$$\partial C / \partial A_i = \sum_h g_h \cdot \partial A_h / \partial x_i \cdot \partial x_i / \partial A_i$$
$$= \sum_h g_h \cdot W_{hi} \cdot f'(A_i)$$

i e

$$g_i = [\sum_h g_h \cdot W_{hi}]\, f'(A_i) \qquad (12)$$

Equation (10) thus becomes

$$W_{ij}(t+1) = W_{ij}(t) - \varepsilon(t)\, g_i \cdot x_j \qquad (13)$$

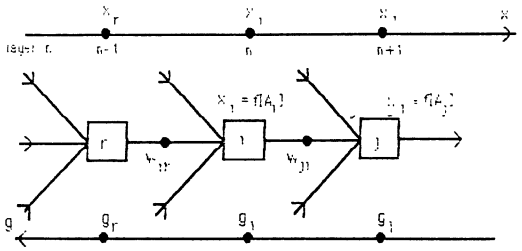


FIGURE 4 GRADIENT BACK PROPAGATION

This algorithm has been proposed independently by several authors [12, 17, 20] and is known as the GRADIENT-BACK-PROPAGATION algorithm One step of the algorithm (FIG 4) thus consists in

- presenting an input-output pair $(\mathbf{x}^k, \mathbf{y}^k)$
- propagating forward the states from layer 0 to layer N

- propagating backward the error signal $\mathbf{g}$ from the last layer N to the first
- modifying the weights according to equation (13)

Note that this algorithm solves the problem of assigning a «desired value» to the hidden units by back-propagating the error signal $\mathbf{g}$. As the cost function C is not convex, there might be various minima, hence various solutions for a given problem. Usually, the initial distribution of weights, the order and frequency of presentation of input-output pairs strongly influence the resulting solution.

We now present two applications of this algorithm and illustrate on these examples the characteristics of the method.

5- Associative memories

The general problem of supervised learning-from-examples can be viewed in our model as follows: given a set of input vectors $\mathbf{x}^k$ (the examples), how to associate them with given output vectors $\mathbf{y}^k$? If these vectors are coded as binary vectors, this problem thus amounts to «completing» a truth table given by some of its lines (Fig 5). The desired property of generalization means that those lines which do not correspond to presented examples, but do «ressemble» them in some way must be completed so as to ensure the same corresponding output. Thus the completion process must be «guided» so as to ensure this generalization property.

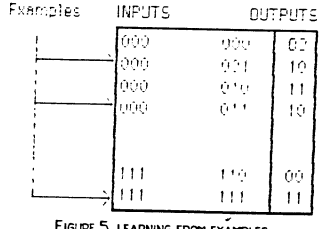


Figure 5 Learning from examples

This generalization problem has been widely studied in the context of associative memories [5, 11]. The problem there is to associate with each input vector $\mathbf{x}^k$ (a letter, a word, ...) an output $\mathbf{y}^k$ which might be either itself (auto association) or a code (hetero-association). The generalization property which is looked for here is mainly fault-tolerance: when the input is close to one of the memorized inputs $\mathbf{x}^k$, the output should still be $\mathbf{y}^k$. This means that the completion of the truth table of figure 5 is done in such a way as to ensure that lines which input part is close (in Hamming distance terms) to an example have the same output part. This generalization property thus allows retrieval of stored data from noisy or partial items.

Many models have been proposed which lead to this property their performances can only be compared in the framework of a unified model and a unified test set of patterns We have given elsewhere [6] a comparison of classical linear models of learning and multi-layered models for various learning algorithms and a test set made from letters of the alphabet The problem with this kind of data is that the regularities of the patterns may bias the results, since the architecture of the multi-layered network may take advantage of these regularities to perform better
We thus propose here to test the associative memory capacity of various multi-layered networks for a test set made of RANDOM patterns: this problem is more difficult to solve, since no a-priori knowledge can be used to help designing the most efficient architecture This task can thus be considered a good «benchmark» to compare the performances of various architectures for the back propagation algorithm

The input patterns are random 16-bits vectors We have compared the performances of networks where 4, 8 and 16 such vectors were memorized

The learning algorithm is a modified gradient-back propagation algorithm

$$W_{ij}(t+1) = W_{ij}(t) - \Delta W(t) \qquad (14)$$

where

$$\Delta W(t) = \alpha\, \varepsilon(t)\, g_i \,.\, x_j + \beta\, \Delta W(t-1) \qquad (15)$$

The «pure» gradient-back propagation algorithm corresponds to $\alpha=1$ and $\beta=0$ Wae take here $\alpha=.5$ and $\beta=.5$ This formulation allows to keep information from the past (the β term) and tends to dampen the oscillations due to the α term

The learning process proceeds by repeating 4 times the following sequence for each of the m input vectors $\mathbf{x}^k$ (m=4, 8, or 16)

- choose a «noise level» n n will vary from 3 down to 0
- for each noise level n, set at random 10 patterns at (Hamming) distance n of the input
- present those 10 noisy versions of input $\mathbf{x}^k$, with the associated «pure» output $\mathbf{y}^k$

This procedure allows to give the network a model of noise (i e those patterns which are to be considered «close» to the input) instead of giving such information through a probability distribution (e g an additive gaussian noise), examples of this noise are presented Such a procedure has been shown to lead to better results [6] without a model of noise, multi-layered networks cannot «invent» what noise should be

In figure 6, the multi-layered networks used for the test are shown. network n° 1 has 2 layers of 16 automata (it is the structure of a perceptron), network n°2 has 3 layers, with the input and output layers as before and 8 hidden units, network n°3 has 4 layers, with the input and output layers as before and 2 layers of hidden units (with 6 and 7 units respectively). All networks have full connections between successive layers this makes 256 weights for network:

and 2, and 250 for network 3, i e approximately the same number of parameters to solve the problem

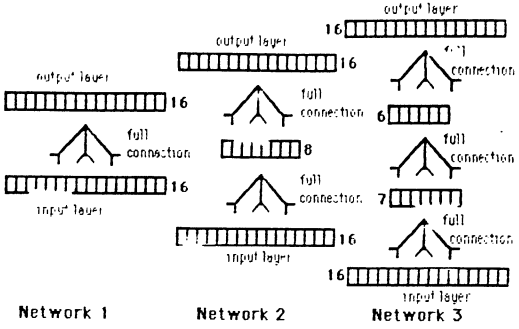


FIGURE 6 MULTI-LAYERED NETWORKS FOR ASSOCIATIVE MEMORY

In figure 7, we compare the performances of networks 1 and 2 on an auto-association task after the training session, for each noise level n (from 0 to 5), we compute the output generated by the network for an input at distance n of a memorized pattern We average the number of correct answers over all m memorized patterns and 5 different choices of input patterns at distance n this gives the recognition rate, which is plotted against the noise level n
Network 2 is always better than network 1 at learning this task, especially for large m
In figure 8, we compare the performances of the 3 networks, for a given value of m m=8 Network 3 with 4 layers is the best
Figure 9 is as figure 7 but on a hetero-association task for each (random) input $\mathbf{x}^k$, its associated output $\mathbf{y}^k$ is set at random the task is more difficult to learn, the number of presentations of the learning set has to be increased In the figure, we show the results for 5 presentations (instead of 4 in the previous figures) Full memorization in the case of m=16 is not even reached in this case (it would take longer, but can be obtained) In this case, performances of networks 1 and 2 are similar more learning would be necessary to discriminate
In figure 10 finally, we compare the performances of network 2 for the auto-association and the hetero-association tasks hetero-association is obviously harder

6- AN AUTOMATA NETWORK FOR MEDICAL DIAGNOSIS

We have used a data-base containing 5700 files of patients suffering from abdominal pains Each file may contain up to 130 symptoms, qualitative or semi-quantitative they are coded on 200 bits 22 diagnosis are possible (cancer, appendicitis, peritonitis, obstruction of the bowels... or non specific pain!)

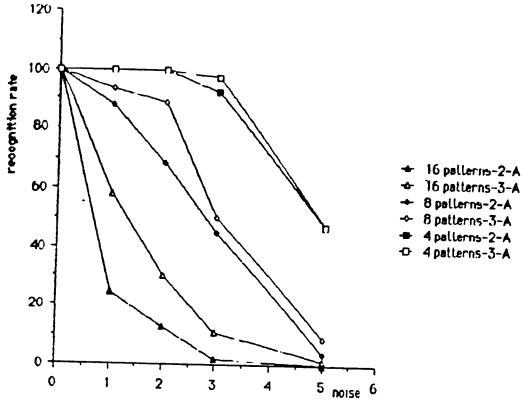


FIGURE 7 AUTO-ASSOCIATION FOR NETWORKS WITH 2 LAYERS (2-A) AND 3 LAYERS (3-A)

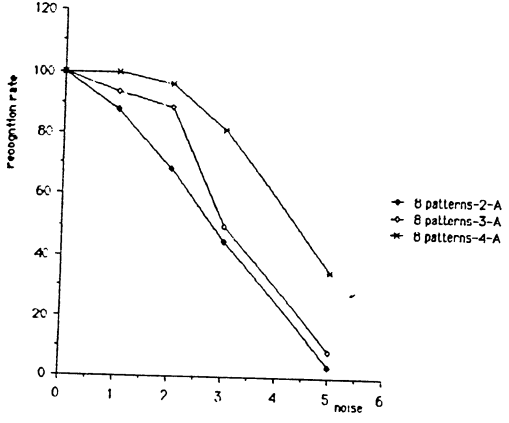


FIGURE 8 AUTO-ASSOCIATION FOR NETWORKS WITH 2 LAYERS (2-A), 3 LAYERS (3-A) AND 4 LAYERS (4-A)

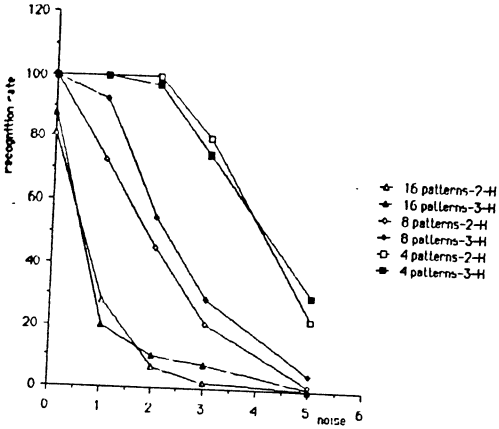


FIGURE 9 HETERO-ASSOCIATION FOR NETWORKS WITH 2 LAYERS (2-H) AND 3 LAYERS (3-H)

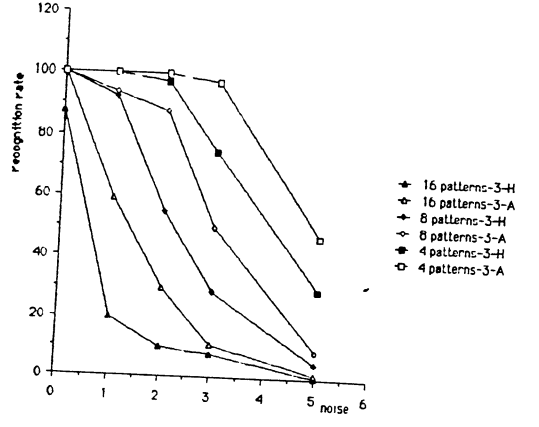


FIGURE 10 AUTO-ASSOCIATION (3-A) VS HETERO-ASSOCIATION (3-H) FOR NETWORKS WITH 3 LAYERS

Figure 11 shows the network used to learn the association symptoms-diagnosis 4000 files were used for learning and the remaining 1700 as test. The results lead to 65% success on the learning set and 60% on the test set These results are comparable to the 60% rate performed by an expert (in the situation of the emergency room of the Hospital) or by a MYCIN type expert system

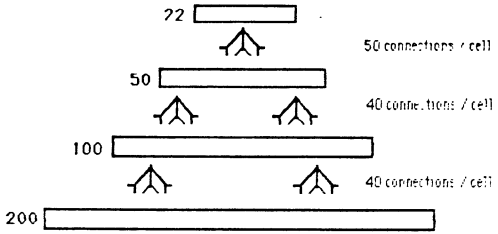


FIGURE 11. NETWORK FOR MEDICAL DIAGNOSIS

These results were obtained by a first «guess» on the architecture of the network and «brute» application of the gradient back propagation algorithm (domain independent!)
Work in progress involves designing architectures taking into account available medical knowledge This is the main difference between this problem and the associative memory problem with random patterns In a real AI application, such as this medical diagnosis problem, a priori knowledge domain dependent may be used to increase performances

7- CONCLUDING REMARKS

Many problems remain to be solved to fully exploit the possibilities of learning on automata networks

- the convergence problems learning may be long when the cost function is ill-conditionned, for example with almost flat regions above minima The metrics which guide the steepest descent might be appropriately chosen to cope with such convergence problems
- the architecture problem performances are not systematically improved when the number of weights is increased When it is not enough constrained by a limited number of weights, the network just performs rote learning (over-parametrization).
  Rules to generate an appropriate architecture must be designed, possibly by making use of existing algorithms on the decomposition of boolean formulas
- links with other AI tools in machine learning these systems could be used as a preliminary step to extract logical predicates from data, thus performing «low level» learning

If these problems can be solved, we will have a powerful and universal tool to

8- References

[1] D H ACKLEY, G.E HINTON, T.J SEJNOWSKI A learning algorithm for Boltzmann machines **Cognitive Science**, 9, pp 147-169, 1985

[2] J.A ANDERSON Cognitive capabilities of a parallel system In « Disordered Systems and Biological Organization», E Bienenstock, F. Fogelman Soulié, G Weisbuch Ed, **Springer Verlag**, NATO ASI Series in Systems and Computer Science, F20, pp 209-226, 1986

[3] D H BALLARD Parameter nets **Artificial Intelligence**, 22, pp 235-267, 1984.

[4] J.A FELDMAN, D.H BALLARD. Connectionnist models and their properties **Cognitive Science**, 6, pp 205-254, 1982

[5] F. FOGELMAN SOULIE, P. GALLINARI, S THIRIA Learning and associative memory. In «Pattern Recognition, Theory and Applications », P.A. Devijver Ed, NATO ASI Series in Computer Science, **Springer Verlag**, to appear.

[6] F FOGELMAN SOULIE, P GALLINARI, Y LE CUN, S THIRIA Automata Networks and Artificial Intelligence In «Computing on Automata Networks», F. Fogelman Soulié, Y. Robert, M. Tchuente Eds, **Manchester Univ. Press**, to appear

[7] G.E. HINTON, T. SEJNOWSKI. Optimal perceptual inference **IEEE Conf. on Computer Vision and Pattern recognition**, pp 448-453, 1983

[8] G.E HINTON Learning to recognize shapes in a parallel network. Fyssen meeting on vision, Paris, march 1986

[9] J.J.HOPFIELD, D W. TANK: simple neural optimization networks **Biol. Cybern.** vol 52, pp 141-152, 1985

[10] Y. KODRATOFF. Generalizing and particularizing as the techniques of learning. **Computers and Artificial Intelligence**, 2, pp 417-441, 1983

[11] T. KOHONEN Self-Organization and Associative Memory. Springer Series in Information Sciences, vol 8, **Springer Verlag**, 1984

[12] Y LE CUN A learning scheme for assymetric threshold network. In «Cognitiva 85», **CESTA-AFCET Ed.**, pp 599-604, 1985

[13] Y. LE CUN. Learning process in an asymmetric threshold network In «Disordered systems and biological organization», E Bienenstock, F. Fogelman Soulié, G Weisbuch Eds, **Springer Verlag**, NATO Asi series in systems and computer science, n°20, pp 233-240, 1986.

[14] W.S MAC CULLOCH, W. PITTS A logical calculus of the ideas immanent in nervous activity. **Bull. Math. Biophysics**, 5, pp 115-133, 1943

[15] R.S MICHALSKI, L.G CARBONELL, T.M MITCHELL· Machine learning **Tioga**, 1983

[16] M MINSKY, S PAPERT. Perceptrons, an Introduction to Computational Geometry. Cambridge, **MIT Press**, 1969

[17] D.B. PARKER Learning logic TR-47, Center for Computational research in economics and Management Science, MIT, 1985

[18] F. ROSENBLATT. Principles of neurodynamics **Spartan**, 1962

[19] D E RUMELHART, G E HINTON, R.J WILLIAMS Learning representations by back propageting errors **Nature**, vol 323, 9, pp 533-536, 1986

[20] D E RUMELHART, J.L MAC CLELLAND Eds Parallel Distributed Processing: Explorations in the Microstructure of Cognition **MIT Press**, 1986

[21] D E RUMELHART, J.L MAC CLELLAND On learning the past tenses of english [illegible], pp 216-271

[22] D SABBAH Computing with connections in visual recognition of Origami objects **Cognitive Science**, 9, pp 25-50, 1985.

[23] T.J SEJNOWSKI, P.K KIENKER, G E HINTON Learning symmetry groups with hidden units beyond the perceptron **Physica D**, to appear

[24] T J SEJNOWSKI, C H ROSENBERG NETtalk a parallel network that learns to read aloud Johns Hopkins Technical Report, **JHU/EECS-86/01**, 1986

[25] D L WALTZ, J B POLLACK Massively parallel parsing a strongly interactive model of natural language interpretation **Cognitive Science**, 9, pp 51-74, 1985.

[26] B. WIDROW, M.E HOFF: Adaptive switching circuits **IRE WESCON Conv. Record**, part 4, pp 96-104, 1960

Bibliographie

[Abu-Mostafa & Saint Jacques 85] Y. S. ABU-MOSTAFA, J-M SAINT JACQUES: "Information capacity of the hopfield model", *IEEE Trans. Inf. Theory*, Vol IT-31, No 4, July 1985.

[Ackley & al. 85] D.H. ACKLEY, G.E. HINTON, T.J. SEJNOWSKI: "A learning algorithm for Boltzmann Machines", *Cognitive Science*, 9, 1985, pp147-169.

[Agmon 54] S. AGMON: "The relaxation method for linear inequalities", *Canadian Journal of Mathematics*, 6, 382-392, 1954. cité dans R.O. Duda, P.E. Hart: "Pattern classification and scene analysis", Wiley and Son, 1973.

[Amari 71] S. I. AMARI: "Characteristics of randomly connected threshold-elements networks and network systems", *Proc. of the IEEE*, Vol 59, No 1, January 1971.

[Amari 72] S. I. AMARI: "Learning patterns and pattern sequences by self-organizing net of threshold elements", *IEEE Trans. Computer*, Vol C-21, n 11, Nov 1972.

[Anderson 83a] J.A. ANDERSON: "The architecture of cognition", Harvard Univ. Press, 1983.

[Anderson 83b] J.A. ANDERSON: "Cognitive and psychological computation with neural models", *IEEE Trans on System, Man and Cybernetics*, SMC-13, 1983, pp799-815.

[Anderson 83c] J.A. ANDERSON: "Cognitive capabilities of a parallel system", in E. Bienenstock, F. Fogelman-Soulie, G. Weisbuch (eds): "Disordered systems and biological organization", Proc. of a NATO workshop, Les Houches, Mars 1985, Springer-Verlag, NATO ASI series in systems and computer science, n F20, pp 109-206, 1986.

[Andrew 65] A. M. ANDREW: "Significance feedback in neural nets", Report of the Biological Computer Laboratory, Univ of Illinois, 1965. Cité dans un article d'Andrew de référence illisible.

[Athans & Falb 66] M. ATHANS, P.L. FALB: "Optimal control theory", McGraw Hill, 1966.

[Ballard 84] D.H. BALLARD: "Parameter nets", *Artificial Intelligence*, 22, 1984, pp235-267.

[Ballard & Brown 82] D.H. BALLARD, C.M. BROWN: "Computer Vision", Prentice Hall, 1982.

[Ballard & al. 83] D.H. BALLARD, G.E. HINTON, T.J. SEJNOWSKI: "Parallel visual computation", *Nature*, Vol 306, No 3, 1983, pp21-26.

[Barr & al. 82] A. BARR, P.R. COHEN, E.A. FEIGENBAUM: "The handbook of artificial intelligence", Pitman, 1982.

[Barto & al. 83] A. G. BARTO, R. S. SUTTON, C. W. ANDERSON: "Neuronlike adaptive elements that can solve difficult learning control problems", *IEEE: Trans. Syst. Man, Cyb*, Vol SMC-13, No 5, Sept/Oct 1983.

[Bienenstock & al. 86] E. BIENENSTOCK, F. FOGELMAN-SOULIE, G. WEISBUCH (eds): "Disordered systems and biological organization", Proc. of a NATO workshop, Les Houches, Mars 1985, Springer-Verlag, NATO ASI series in systems and computer science, n F20, 1986.

[Block 62] H. D. BLOCK: "The perceptron: a model for brain functionning", *Review of modern physics*, Vol 34, pp 123-135, Jan 1962.

[Block & al. 62] H. D. BLOCK, B. W. KNIGHT, F. ROSENBLATT: "Analysis of a four-layer series-coupled perceptron" *Review of modern physics*, Vol 34, pp 135, Jan 1962.

[Bobrowski 78] L. BOBROWSKI: "Learning processes in multilayer threshold nets", *Biological Cybernetics*, 31, pp 1-6, 1978.

[Bobrowski 81] L. BOBROWSKI: "Formation of the optimal decision rule in the multilayer threshold nets during the learning process", *Biocybernetics and biomedical engineering*, Vol1, No 3/4, pp 5-18, 1981.

[Cadzow 68] J. A. CADZOW: "Synthesis of non-linear decision boundaries by cascaded threshold gates", *IEEE Trans. Computers*, Vol C-17, No 12, December 1968.

[Cameron 60] S. H. CAMERON: "An estimate of the complexity requisite in a universal decision network", Proc of the 1960 Bionics Symposium, Wright Air Development Division Tech Report 60-600, pp 197-211, Dec 1960, Cité dans N.J. Nilsson: "Learning Machines, the foundations of trainable pattern-classifying systems", Mc Graw-Hill, 1965.

[Campbell & Meyer 79] S.L. CAMPBELL, C.D. MEYER: "Generalized inverses of linear transformations", Pitman 1979.

[Chaitin 70] G. J. CHAITIN: "On the difficulty of computation", *IEEE Trans. Inf Theory*, Vol IT-16, No 1, Jan 70.

[Chaitin 74a] G. J. CHAITIN: "Information theoretic computational complexity", *IEEE Trans. Inf. Theory*, Vol IT-20, No 1, Jan 74.

[Chaitin 74b] G. J. CHAITIN: "Information theoretic limitations of formal systems", *J. ACM*, Vol 21, No 3, pp 403-424, July 1974.

[Chaitin 75a] G. J. CHAITIN: "A theory of program size formally identical to information theory", *J. ACM*, Vol 22, No 3, pp 329-340, July 1975.

[Chaitin 75b] G. J. CHAITIN: "Randomness and mathematical proof", *Scientific American*, May 1975.

[Changeux 83] J.P. CHANGEUX: "L'homme neuronal", Fayard, 1983.

[Cherry 61] C. CHERRY (ed): "Information Theory, Proc. of the 4th London Symposium", Butterworth 1961

[Cover 65] T.M. COVER: "Geometrical and statistical properties of linear inequalities with application to pattern recognition", *IEEE Trans Electronic Computer*, Vol EC-14(3), 1965, pp326-334.

[Cover 75] T. M. COVER: "Generalization on patterns using Kolmogorov complexity", 1975.

[Demongeot & al. 85] J. DEMONGEOT, E. GOLES, M. TCHUENTE (eds): "Dynamical systems and cellular automata", Academic Press, 1985.

[Duda & Hart 73] R.O. DUDA, P.E. HART: "Pattern classification and scene analysis", Wiley and Son, 1973.

[Fahlman & Hintom 87] S. E. FAHLMAN, G. E. HINTON: "Connectionnist architectures for artificial intelligence", *Computer*, 1987.

[Farley & Clark 54] B. FARLEY, W. CLARK: "Simulation of self-organizing systems by digital computer", *Trans. IRE Information theory*, PGIT-4, pp 76-84, Sept 1954, cité dans: N. Nilsson: "Learning Machines, the foundations of trainable pattern-classifying systems", Mc Graw-Hill, 1965.

[Farley 61] B. G. FARLEY, W. A. CLARK: "Activity in networks of neuron-like elements", in C. Cherry (ed): "Information Theory, Proc of the 4th London Symposium", Butterworth 1961.

[Farmer & al. 84] D. FARMER, T. TOFFOLI, S. WOLFRAM (eds): "Cellular automata", *Physica 10D*, North-Holland 1984.

[Fogelman 85a] F. FOGELMAN-SOULIE: "Contribution à une théorie du calcul sur réseau", Thèse d'état, Grenoble 1985.

[Fogelman 85b] F. FOGELMAN-SOULIE: "Cerveau et machines: des architectures pour demain", Actes de COGNITIVA 85, CESTA-AFCET, 1985.

[Fogelman & Goles 86a] F. FOGELMAN-SOULIE, E. GOLES-CHACC: "Knowledge representation by automata networks", in P. Chenin, C. di Crescenzo, F. Robert (eds): "computer and computing", Masson-Wiley, pp 175-180, 1986

[Fogelman & Weisbuch 86b] F. FOGELMAN-SOULIE, G. WEISBUCH: "Random iterations of threshold networks and associative memories", SIAM Journal on computing 1986.

[Fogelman & al. 87a] F. FOGELMAN-SOULIE, P. GALLINARI, S. THIRIA: " Linear learning and associative memory", in *Pattern Recognition, theory and applications*, J. Devijver (ed), NATO ASI Series in computer science, Springer-Verlag, 1987.

[Fogelman & al. 87b] F. FOGELMAN-SOULIE, P. GALLINARI, Y. LE CUN, S. THIRIA: "Automata networks and artificial intelligence", in F. Fogelman, Y. Robert, M. Tchuente (eds): "Automata networks in computer science, theory and applications", Manchester University Press, 1987.

[Fukushima 75] K. FUKUSHIMA: "Cognitron: a self-organizing multilayered neural network", *Biological Cybernetics*, 20, pp 121-136, 1975.

[Fukushima & Miyake 78] K. FUKUSHIMA, S. MIYAKE: "A self-organizing neural network with a function of associative memory: feedback-type cognitron", *Biological cybernetics*, 28, pp 201-208, 1978.

[Gallinari & al. 87] P. GALLINARI, Y. LE CUN, S. THIRIA, F. FOGELMAN-SOULIE: "Mémoires associatives distribuées: une comparaison", in COGNITIVA 87, Cesta-Afcet (eds), Paris, Mai 1987.

[Gardner 84] W. A. GARDNER: "Learning characteristics of stochastic gradient descent algorithms: a general study, analysis, and critique", *Signal Processing*, No 6, pp 113-133, 1984.

[Geman & Geman 84] S. GEMAN, D. GEMAN: "Stochastic relaxation, Gibbs distribution and the Bayesian restoration of images", *IEEE Trans. PAMI*, Vol 6, no 6, pp721-741, 1984.

[Goles 85] E. GOLES-CHACC: "Comportement dynamique de réseaux d'automates", Thèse d'état, Grenoble 1985.

[Greville 60] T. N. E. GREVILLE: "Some applications of the pseudo-inverse of a matrix", SIAM Rev. 2, pp 15-22, 1960.

[Hebb 49] D. HEBB: "Organization of behavior", Science Edition 1961.

[Hillis 82] W. D. HILLIS: "New computer architectures and their relationship to physics or why computer science is no good" *Int J. of Theor. Physics*, Vol 21, no 3/4, pp 255-262, 1982

[Hillis 86] W. D. HILLIS: "The Connection Machine", MIT press, 1986.

[Hinton 81a] HINTON G. F.: "A parallel computation that assigns canonical object-based frames of reference", Proc. of the seventh IJCAI, Vancouver, 1981.

[Hinton 81b] HINTON G. F.: "Shape representation in parallel" Proc. of the seventh IJCAI, Vancouver, 1981.

[Hinton & Anderson 81] G. E. HINTON, J. A. ANDERSON (eds): "Parallel models of associative memory", Hillsdale, Erlbaum, 1981.

[Hinton & Sejnowski 83] G. E. HINTON, T. SEJNOWSKI: "Optimal perceptual inference", *IEEE conf. on Computer vision and Pattern recognition*, pp 448-453, 1983.

[Hinton 84a] G. E. HINTON: "Distributed representations", Working paper, CMU-CS-84-157, 1984.

[Hinton 84b] G. E. HINTON: "Parallel computation for controlling an arm", J. of Motor Behavior, Vol-16, No 2, 1984.

[Hinton & Al. 84c] G. E. HINTON, T. SEJNOWSKI D. H. ACKLEY: "Boltzman Machines: constraint satisfaction networks that learn", Tecnical Report, CMU-CS-84-119, Carnegie-Mellon university, Dept of Computer Science, May 1984.

[Hinton 85] G. E. HINTON: "Learning in parallel networks", Byte, pp 265-273, Avril 1985.

[Hinton 86] G. E. HINTON: "Learning to recognize shapes in a parallel network", in Proceding of the Fyssen meeting on vision, Paris, Mars 1986

[Honig & Messerschmitt 84] M. L. HONIG, D. G. MESSERSCHMITT: "Adaptive filters, structures algorithms and applications", Kluwer academic publishers, 1984.

[Hopfield 82] J. J. HOPFIELD: "Neural networks and physical systems with emergent collective computational abilities", P.N.A.S. USA, Vol 79, pp 2554-2558, 1982.

[Hopfield 84] J. J. HOPFIELD: "Neurons with graded response have collective computational properties like those of two-state neurons", P.N.A.S. USA, Vol 81, pp 3088-3092, 1984.

[Joseph 60] R. D. JOSEPH: "Contribution to perceptron theory", Cornell aeronautical labs report, VG-1196-G-7, Buffalo, June 1960, Cité dans N.J. Nilsson: "Learning Machines, the foundations of trainable pattern-classifying systems", Mc Graw-Hill, 1965.

[Kandel] E. KANDEL: "Cellular basis of behavior", Freeman, .

[Keller & Hunt 85] J. M. KELLER, D. J. HUNT: "Incorporating fuzzy membership functions into the perceptron algorithm", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol PAMI-7, No 6, November 1985.

[Kirkpatrick & al. 83] S. KIRKPATRICK, C. D. GELATT, M. P. VECCHI: "Optimization by simulated annealing", Science, Vol 220, no 4598, pp 671-680, 1983.

[Kobylarz 80] T. J. KOBYLARZ: "The application of threshold logic to I2L programmable logic arrays", Proc. ISSCC, 1980.

[Kohonen 72] T. KOHONEN: "Correlation matrix memories", *IEEE Trans. Computers*, Vol C-21, no 4, April 1972.

[Kohonen & Ruohonen 73] T. KOHONEN, M. RUOHONEN: "Representation of associated data by matrix operators", *IEEE Trans. Computers*, July 1973.

[Kohonen 74] T. KOHONEN: "An adaptive associative memory principle", *IEEE Trans Computers*, July 1974.

[Kohonen 77] T. KOHONEN: "associative memory: a system theoretic approach", Springer-Verlag, 1977.

[Kohonen 81] T. KOHONEN: "Content addressable memories" Springer-Verlag, 1981.

[Kohonen 82] T. KOHONEN: "Self-organized formation of topologically correct feature maps", *Biological Cybernetics*, 43, pp 59-69, 1982.

[Kohonen 84a] T. KOHONEN: "Self-organization and associative memory", Springer series in information sciences, Vol 8, Springer-Verlag, 1984.

[Kohonen 84b] T. KOHONEN: "Phonotopic maps: insightful representations of phonological features for speech recognition", Proc. of the ICPR, Montreal, 1984.

[Kolmogorov 68] A. N. KOLMOGOROV: "Logical basis for information theory and probability theory", *IEEE Trans. Inf. Theory*, Vol IT-14, No 5, Sept 1968.

[Landau 81] I. D. LANDAU (ed): "Outils et modèles mathématiques pour l'automatique, l'analyse des systèmes et le traitement du signal", Editions du CNRS, 1981.

[le Cun & Metsu 83] Y. LE CUN, P. METSU: "Contribution à la conception d'une famille de circuits VLSI de traitement du signal", Rapport de fin d'études Thomson-CSF, ESIEE, Juin 1983.

[le Cun 84] Y. LE CUN: "Propriétés d'apprentissage et d'auto-réparation d'un réseau d'automates à seuil", Rapport de DEA, Paris VI, 1984.

[le Cun 85] Y. LE CUN: "Une procédure d'apprentissage pour réseau à seuil asymmétrique", Proc. of COGNITIVA 85, Cesta-Afcet, pp 599-604, Juin 1985.

[le Cun 86] Y. LE CUN: "Learning processes in an asymmetric threshold network", in E. Bienenstock, F. Fogelman-Soulie, G. Weisbuch (eds): "Disordered systems and biological organization", Proc. of a NATO workshop, Les Houches, Mars 1985, Springer-Verlag, NATO ASI series in systems and computer science, n F20, pp 233-240, 1986. également dans "Connectionnist models: a summer school", Carnegie-Mellon Univ., Juin 1986.

[le Cun & Fogelman 87a] Y. LE CUN, F. FOGELMAN-SOULIE: "Modèles connexionnistes de l'apprentissage", in J. Sallantin, J. Quinqueton (eds) "Intellectica, numéro spécial apprentissage et machines", Association pour la Recherche Cognitive, Mars 1987.

[Little 74] W. A. LITTLE: "The existence of persistent states in the brain", *Math Biosc*, 19, pp 101-120, 1974.

[Macchi 84] O. MACCHI (ed): "IEEE Trans. Information Theory, Special issue on linear adaptive filtering", *IEEE Trans IT*, Vol *IT-30*, No 2, March 1984.

[McClelland & Rumelhart 85] J. L. McCLELLAND, D. E. RUMELHART: "Distributed memory and the representation of general and specific information", *J. of Exp Psychology*, Vol 114, no 2, pp 159-188, 1985.

[McCulloch & Pitts 43] W. McCULLOCH, W. PITTS: "A logical calculus for the ideas immanent in nervous activity", *Bull. Math. Biophysics*, 5, pp 115-133, 1943

[Marr & Poggio 76] D. MARR, T. POGGIO: "Cooperative computation of stereo disparity", *Science*, vol 194, pp 283-287, 1976.

[Michalski & al. 83] R. MICHALSKI, J. CARBONELL, T. MITCHELL (eds): "Machine Learning", Tioga 1983.

[Michalski & al. 86] R. MICHALSKI, J. CARBONELL, T. MITCHELL (eds): "Machine Learning: an artificial intelligence approach", Tioga 1986.

[Miclet 84] L. MICLET: "Méthodes structurelles pour le reconnaissance des formes", Eyrolles, Paris 1984.

[Minsky 67] M. MINSKY: "finite and infinite machines", Prentice Hall, 1967.

[Minsky & Papert 68] M. MINSKY, S. PAPERT: "Perceptron", MIT Press 1968.

[Miyake & Fukushima 84] S. MIYAKE, K. FUKUSHIMA: "A neural network for the mechanism of feature extraction", *Biol. Cybernetics*, 50, pp 377-384, 1984.

[Mozer 86] MOZER M. C.: "Early parallel processing in reading: A connectionnist approach", Tech. Report, Inst. for Cognitive Science, C-015, UCSD, June 1986.

[Murakami & Aibara 81] K. MURAKAMI, T. AIBARA: "Construction of a distributed associative memory on the basis of the bayes discriminant rule", *IEEE Trans PAMI*, Vol-3, No-2, March 81.

[Nakano 71] K. NAKANO: "Learning process in a model of associative memory", in K. S. Fu (ed) "Pattern recognition and machine learning, Proc of Japan-US Seminar on learning process in control systems, Nagoya, Japon, Aout 1970", Plenum press, 1971.

[Nakano 72] K. NAKANO: "Associatron: a model of associative memory", *IEEE Trans. Syst, Man Cybernetics*, Vol SMC-2, No 3, Juillet 1972.

[von Neumann 66] J. VON NEUMANN: "Theory of self-reproducing automata", A. W. Burks (ed), Univ. Illinois Press, 1966.

[Nilsson 65] N.J. NILSSON: "Learning Machines, the foundations of trainable pattern-classifying systems", Mc Graw-Hill, 1965.

[Parker 85] PARKER D. B.: "Learning logic", Tech Report TR-47, Center for Computational research in economics and management science, MIT, April 1985.

[Peretto 84] P. PERETTO: "Collective properties of neural networks, a statistical physics approach", *Biol. Cybernetics*, 50, pp 51-62, 1984.

[Pearlmutter & Hinton 86] B. A. PEARLMUTTER, G. E. HINTON: "G-Maximisation: an unsupervised learning procedure for discovering regularities", in J. Denker (eds) "Neural Networks for computing", American Institute of Physics Proceedings 151.

[Piatelli-Palmarini 80] M. PIATELLI-PALMARINI (ed): "Théorie du langage, théorie de l'apprentissage: le débat entre Jean Piaget et Noam Chomsky", Seuil, Paris, 1980.

[Poggio & al. 85] T. POGGIO, V. TORRE, C. KOCH: "Computationnal vision and the regularization theory", *Nature*, Vol 317, 26, pp 314-319, 1985.

[Plaut & al. 86] D. C. PLAUT, S. J. NOLAN, G. E. HINTON: "Experiments on learning by back-propagation", Technical report CMU-CS-86-126, 1986.

[Pyle 64] L. D. PYLE: "Generalized inverses of matrices and its applications", Wiley, 1971.

[Quinqueton % Sallantin 87] J. QUINQUETON, J. SALLANTIN (eds): *Intellectica*, Numéro spécial "Apprentissage et Machines", Vol 1, No 2/3, A.R.C., 1987.

[Rao & Mitra 71] C. R. RAO, S. K. MITRA: "Generalized inverse of matrices and its applications", Wiley, 1971.

[Reid & Sutherland Frame 75] R. J. REID, J. SUTHERLAND FRAME: "Convergence in iteratively formed correlation matrix memories", *IEEE Trans. Computer*, pp 827-830, August 1975.

[Ribeiro 67] S. T. RIBEIRO: "Random-pulse machines", *IEEE Trans. Electronic Computer*, Vol EC-16, No 3, June 1967.

[Ridgway 62] W. C. RIDGWAY: "An adaptive logic system with generalizing properties" Tech Report 1556-1, Stanford Electronic Labs, Stanford University. cité dans N.J. Nilsson: "Learning Machines, the foundations of trainable pattern-classifying systems", Mc Graw-Hill, 1965.

[Rosenberg & Sejnowski 86] C. R. ROSENBERG, T. J. SEJNOWSKI: "Practice on NetTalk, a massively parallel network that learns to read aloud", in "Connectionnist models: a summer school", Carnegie-Mellon Univ., Juin 1986.

[Rosenblatt 57] "The Perceptron: a perceiving and recognizing automaton", Project PARA, Cornell Aeronautical Lab. Report No 85-460-1, Janvier 1957.

[Rosenblatt 60] F. ROSENBLATT: "On the convergence of reinforcement procedures in simple perceptrons", Cornell Aeronautical Lab. Report No VG-1196-G-4, Fevrier 1960, cité dans N.J. Nilsson: "Learning Machines, the foundations of trainable pattern-classifying systems", Mc Graw-Hill, 1965.

[Rosenblatt 61] F. ROSENBLATT: "Principles of Neurodynamics: Perceptrons and the theory of brain mechanism", Spartan Books, Washington DC, 1961.

[Rumelhart & Zipser 85] D. RUMELHART, D. ZIPSER: "Competitive learning", *Cognitive Science*, 9, pp 75-112, 1985.

[Rumelhart & al. 86a] D. RUMELHART, G. HINTON, R. WILLIAMS: "Learning internal representations by error propagation", in *Parallel distributed processing.*

exploring the micro-structure of cognition, D. Rumelhart, J. McLelland (eds), MIT Press 1986.

[Rumelhart & al. 86b] D. RUMELHART, G. HINTON, R. WILLIAMS: "Learning internal representations by back-propagating errors", *Nature*, 323, pp 533-536, 1986.

[Rumelhart & al. 86c] D. RUMELHART, J. McCLELLAND and the PDP research group (eds): "Parallel distributed processing, exploring the micro-structure" of cognition, MIT Press 1986.

[Sejnowski & Hinton 85] T. J. SEJNOWSKI, G. E. HINTON: "Separating figure from ground with a Boltzmann machine", in M. A. Arbib, A. R. Hanson (eds): "Vision, brain and cooperative computation", MIT Press, Cambridge, 1985.

[Sejnowski & al. 85] T. J. SEJNOWSKI, P. K. KIENKER, G. E. HINTON: "Learning symmetry groups with hidden units: beyond the perceptron", Preprint, soumis à Physica D, 1985.

[Sejnowski & Rosenberg 86] T. J. SEJNOWSKI, C. R. ROSENBERG: "NETtalk, a parallel network that learns to read aloud", Tech Report 86-01, Dept of EECS, Johns-Hopkins University, Baltimore, MD, 1986.

[Solomonoff 64] R. J. SOLOMONOFF: "A formal theory of inductive inference", *Information and Control*, 7, pp 1-22, 1964.

[Solomonoff 66] R. J. SOLOMONOFF: "Some recent works in artificial intelligence", *Proc. of the IEEE*, Vol 54, No 12, pp 1687-1697, Dec. 1966.

[Shiozaki 84] A. SHIOZAKI: "Recollection ability of three-dimensional correlation matrix associative memory", *Biological Cybernetics*, 50, pp 337-342, 1984.

[Stiles & Denq 85] G. S. STILES, DONG-LIH DENQ: "On the effect of noise on the More-Penrose generalized inverse associative memory", *IEEE Trans PAMI*, Vol-7, No 3, May 1985.

[Stonham & Shaw 75] T. J. STONHAM, M. A. SHAW: "Automatic classification of mass spectra by means of digital learning nets - existence of characteristic features of chemical class in mass spectra", *Pattern Recognition*, Vol 7, pp 235-241, 1975.

[Taniguchi & al. 82] K. TANIGUCHI, T. INOUE, F. UENO: "Analysis and applications of new I2L multivalued logic circuits", *Electronics and Communications in Japan*, Vol 65-C, 1, 1982.

[Taylor 83] G. N. TAYLOR (ed): "IEE Proc. Communication, Radar and Signal Processing, Special Issue on adaptive arrays", *IEE Proc. CMRSP, Part F*, Volume 130, No 1, February 1983.

[Yovits & Cameron 60] M. C. YOVITS, S. CAMERON (eds): "Self-organizing systems", Pergamon, New-York, 1960.

[Yovits & al. 62] M. C. YOVITS, JACOBI, GOLDSTEIN (eds): "Self-organizing systems", Spartan books, Washington, 1962.

[Von Foerster & Zopf 62] H. VON FOERSTER, H. ZOPF (eds): "Principles of self-organization", Pergamon, New-York, 1962.

[Watanabe 80] WATANABE S.: "Pattern recognition as conceptual morphogenesis", IEEE Trans Pattern Analysis and Machine Intelligence, Vol PAMI-2, No 2, March 1980.

[Weisbuch & Fogelman 85] G. WEISBUCH, F. FOGELMAN-SOULIE: "scaling laws for the attractors of Hopfield networks", *J. Phys. lett*, **46, 1985**, pp623-630

[Widrow & Stearn 85] B. WIDROW, S.D. STEARNS: "Adaptive signal processing", Prentice-Hall 1985.

[Widrow & Hoff 60] WIDROW B. WIDROW, M.E, HOFF: "Adaptive switching circuits". *IRE WESCON Conv. record*, part 4 1960, pp96-104

[Willshaw 81] D. WILLSHAW: "Holography, associative memory and inductive generalization", in Hinton & Anderson (eds) "Parallel models of associative memory", Hillsdale, Erlbaum, 1981.

[Winder 62] R. O. WINDER: "Threshold logic", PhD dissertation, Princeton Univ. NJ 1962, Cité dans N.J. Nilsson: "Learning Machines, the foundations of trainable pattern-classifying systems", Mc Graw-Hill, 1965.