MASTER EN SCIENCES INFORMATIQUES

Les Réseaux Neuronaux : Approche théorique et Applications

Thiry, Vincent; Nivelles, Philippe

Award date:
1990

Awarding institution:
Universite de Namur

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX

NAMUR

INSTITUT D'INFORMATIQUE

Les Réseaux Neuronaux : Approche théorique et Applications

Philippe Nivelles
Vincent Thiry

Promoteur : Professeur M. Noirhomme

Mémoire présenté en vue de l'obtention
du titre de :
Licencié et Maître en Informatique

Année Académique 1989 - 1990

"In order to understand the brain we have used the computer as a model for it. Perhaps it is time to reverse this reasoning : to understand where we should go with the computer we should look to the brain for some clues. If we do so, we may find that today's computer is indeed the fossil of our age".

Bob Noyce, Intel.

"Nous construirons une machine qui sera fière de nous".

Thinking Machine Corp.

RESUME

Les Réseaux Neuronaux artificiels constituent une approche nouvelle et non algorithmique du traitement de l'information. Constitués d'une multitude de neurones artificiels, ceux-ci sont bons là où les ordinateurs conventionnelss ne le sont pas. La vision , la robotique, la reconnaissance de la parole, le traitement de signaux, l'optimisation et la reconnaissance de pattern en sont les principaux champs d'investigation.

Le premier objectif de ce mémoire est de poser les bases théoriques nécessaires à la compréhension du fonctionnement des réseaux neuronaux ainsi que de présenter l'état de l'art pour les applications, le software et le hardware. Ensuite, afin d'illustrer notre approche théorique, nous avons implémenté deux petites applications : d'une part une correspondance non linéaire utilisée en robotique; d'autre part une implémentation permettant de reconnaître des caractères imprimés.

Les résultats sont impressionnants mais encore loin des attentes les plus optimistes. La théorie est nouvelle et de nombreuses recherches doivent encore être faites aussi bien dans le domaine de la neurobiologie que dans le domaine de la modélisation neuronale.

ABSTRACT

Artificial Neural Networks are a new, non algorithmic approach to information processing. Constituded with a multitude of artificial neurons, neural nets are good at doing things that computers can not. Vision, robotic, speech recognition, signal processing, optimization and pattern recognition are the main fields investigated.

Our first goal in this dissertation is to give the theoretical context that is necessary to understand the working of neural networks, and, to present the state of the art in applications, software and hardware. Secondly, to illustrate our theoretical approach, we have implemented two short applications : a non linear mapping used in robotics and an automatic recognizer of printed characters.

Results are impressive but still far from the most optimistic expectations. The theory is new and a lot of research still has to be made, both in neurobiology and neurocomputing.

REMERCIEMENTS

Nous tenons ici à exprimer nos plus vifs remerciements à nos promoteur et co-promoteur madame Monique Noirhomme et monsieur Jorge Barreto pour nous avoir donné l'occasion de travailler dans cette "nouvelle" branche de l'intelligence artificielle que sont les réseaux neuronaux artificiels;

Nous remercions également les membres du groupe de recherche en inginiérie biomédicale (GPEB) de l'Université Fédérale de Santa Catarina au Brésil et plus particulièrement les professeur Walter Celso de Lima et doctorant Renato Garcia Olida pour leur encadrement durant ce stage.

Notre travail n'aurait vraiment pu débuter sans l'implémentation Fortran de Laurent Honet, laquelle nous a éclairé sur la méthodologie exacte à employer. A cet effet nous le remercions très chaleureusement.

Nous tenons à remercier de façon tout à fait particulière monsieur Jean-Pierre Peters pour avoir su nous prendre en charge, à plusieurs reprises, "au pied-levé" et pour nous avoir fourni des outils logiciels, clé du succès de plusieurs de nos applications.

Nous souhaitons également remercier toutes les personnes qui nous ont permis de travailler sur l'Olivetti M380, que nous avons souvent monopolisé, ainsi que toutes celles qui nous ont aidés de près ou de loin dans notre travail.

Nous ne saurions terminer ces remerciements sans citer nos parents qui nous ont donné la possibilité de réaliser nos études et notre stage au Brésil.

TABLE DES MATIERES

LISTE DES TABLEAUX ET FIGURES

PARTIE I :

ASPECTS THEORIQUES

DES RESEAUX NEURONAUX

I Introduction

Il ne se passe plus un mois sans qu'une revue ou un journal ne parle des réseaux neuronaux. Il semble que le sujet suscite à nouveau l'intérêt des chercheurs et du public. Après une mise à l'index de deux décennies (60-82), les plus grands laboratoires mettent leurs meilleurs neurones sur le coup.

"Pattern recognition" ou reconnaissance des formes constitue l'apanage de la neuro-informatique. Encore dénommée neuronique ou cogniscience, la modélisation neuronale étudie le comportement de centaines, voire de milliers, de neurones artificiels largement interconnectés. Leur domaine d'investigation est l'ensemble des fonctions mentales humaines ou animales : vision, reconnaissance de formes, mémorisation, apprentissage, généralisation, abstraction, ...

La théorie neuronale qui a permis la résurgence de la discipline est fraîchement émoulue des plus grandes universités (MIT, CIT,...). Elle s'étend rapidement mais est encore inadaptée pour supporter les prévisions les plus optimistes. Cependant, les applications dans le domaine existent et se comptent déjà par centaines. Nous mêmes avons simulé certains aspects du pilotage d'un bras de robot ainsi qu'une reconnaissance automatique de caractères imprimés (IIième partie).

La première partie de notre travail a quant-à-elle pour objectif d'introduire le lecteur à ce qu'est la modélisation neuronale, cette branche de l'intelligence artificielle encore mal connue en Belgique. Nous l'envisagerons d'abord sous forme de boîte noire (II. Aperçu général des réseaux neuronaux), montrant tout d'abord en quoi elle se différencie de la programmation conventionnelle.

Nous expliquerons ensuite d'où elle vient, passant en revue ses années fastes et moins fastes, et sa résurgence au cours des années '80 (III. Historique). Après avoir exposé le chemin parcouru par les réseaux neuronaux nous pourrons passer en revue les concepts théoriques de base qui sous-tendent les réseaux neuronaux artificiels (IV. Concepts fondamentaux des réseaux neuronaux).

Dans le cinquième point, nous avons tenté de faire une brève comparaison avec d'autres systèmes de traitement de l'information (V. Comparaison avec d'autres systèmes de traitement de l'information).

Tous les réseaux ne sont pas destinés à résoudre les mêmes problèmes. Il existe une classification basée sur la structure du réseau et sur la manière utilisée pour y "engrammer" les informations. C'est ce que nous rencontrerons au point VI. (Taxonomie des réseaux neuronaux).

Cette classification ne regroupe pas un ensemble de modèles simplement théoriques; il en existe, à l'heure actuelle, bon nombre d'utilisations concrètes dans des domaines aussi variés que le contrôle

des bagages dans les aéroports, la reconnaissance des formes ou encore que la compréhension du langage naturel. C'est ce que nous montrerons au point VII. : "Applications actuelles des réseaux neuronaux".

Ces implémentations ont été permises grâce aux développements logiciels (langages de haut niveaux, programmes intégrés) mais aussi, et surtout, technologiques qui ont permis un plus grand parallélisme dans le matériel (VII. Implémentations software et hardware).

Enfin, une réflexion critique sur la nature même des modèles connexionistes, sur leurs relations avec les technologies conventionnelles ainsi que sur les problèmes qu'ils posent pour le programmeur sera entamée au point IX, "Quid des réseaux neuronaux ?".

Finalement, le domaine des réseaux neuronaux étant encore peu connu , nous avons pensé bon de dresser un glossaire des termes les plus courants. Celui-ci peut être consulté dans l'annexe A. L'annexe B reprend une liste de hardware spécifique aux réseaux neuronaux et l'annexe C les résultats d'une enquête menée parmi 55 chercheurs expérimentés dans le domaine de la modélisation neuronale.

La 2ième partie de ce mémoire est consacrée à l'utilisation pratique des réseaux neuronaux. Notre première application a été d'apprendre à un réseau neuronal à déduire la courbe de commutation pour un bras de robot dans le cas de la commande optimale en inertie pure. L'implémentation a été réalisée en Pascal et avec un logiciel spécialisé appelé ANSim V. 2.30 (II. Parabole de commutation en robotique).

La deuxième application, développée cette fois uniquement avec le logiciel ANSim, consiste en une reconnaissance automatique de caractères. Nous montrerons les résultats obtenus avec différents réseaux et les capacités de rappel propres à chaque réseau (III. Reconnaissance automatique de caractères).

Finalement, nous présentons deux outils logiciels de simulation de réseaux neuronaux : NeuralWorks Professional I de NeuralWare Inc. et ANSim de SAIC (Annexes A et B).

Le code pascal afférent à la première application est présent à l'annexe C et le rapport de notre stage au Brésil à l'annexe D.

II Aperçu général des réseaux neuronaux

II.1. INTRODUCTION

Dans ce chapitre, nous allons brièvement expliquer ce que sont les réseaux neuronaux, et également donner une définition du "**neurocomputing**".

II.2. LES RESEAUX NEURONAUX

Une façon de comprendre comment fonctionnent les réseaux neuronaux est de considérer les principes du système nerveux sur lesquels ils sont basés.

Une caractéristique importante que les réseaux neuronaux partagent avec le cerveau est qu'ils sont composés de nombreuses unités qui opèrent en parallèle (figure II.1). Ces unités sont cinq à six fois plus rapides que les neurones biologiques.

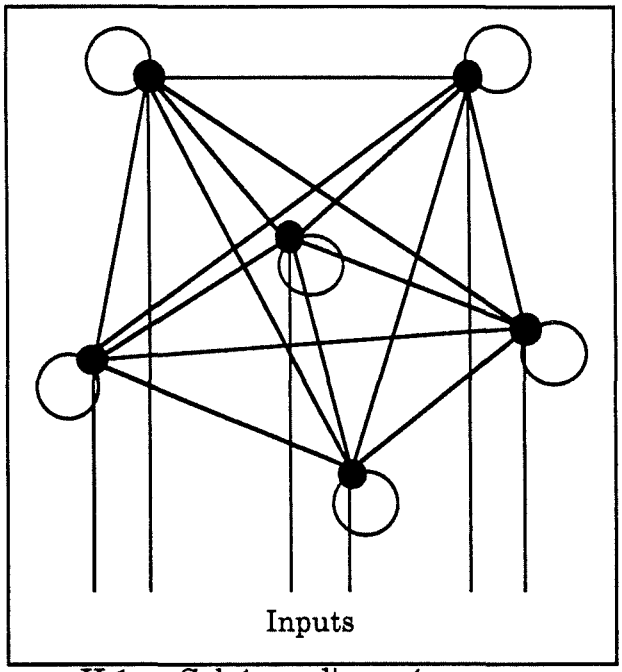


Figure II.1. : Schéma d'un réseau neuronal.

Cette organisation parallèle permet d'effectuer certaines opérations beaucoup plus rapidement que le plus puissant ordinateur digital, actuellement disponible. Le cerveau obtient ces résultats en utilisant un parallélisme massif entre neurones, c'est-à-dire que beaucoup d'entre eux travaillent en même temps sur le même problème. Ainsi des éléments de faible puissance sont combinés de manière à obtenir un système puissant.

Cette technique des réseaux neuronaux peut mener à la solution de traitements de l'information complexes, actuellement non résolus par les techniques conventionnelles que sont les systèmes experts et les langages procéduraux.

Contrairement aux systèmes experts classiques (n'utilisant pas les réseaux neuronaux) qui demandent une description symbolique de la solution d'un problème, et aux langages procéduraux qui en demandent une description procédurale, les réseaux neuronaux demandent une représentation statistiquement valide du problème pour produire une solution. Cela signifie qu'il faut choisir un ensemble représentatif d'exemples et de solutions du problème.

Presque tous les traitements de l'information actuels sont basés sur la machine de Von Neumann, et des algorithmes plus ou moins complexes sont nécessaires. Certaines tâches cependant non pas d'algorithme ou pas d'algorithme efficient. C'est le cas notamment pour le pilotage automatique de véhicule, les traducteurs automatiques, la reconnaissance de caractères manuscrits, la compréhension continue de la parole, la reconnaissance de formes sous différents angles, ... Les réseaux neuronaux sont bons là où des machines, des langages, des programmes conventionnels le sont moins.

Ces différentes tâches, qui ne peuvent être implémentées avec des techniques conventionnelles, ont trois caractéristiques en commun :

- les humains savent comment effectuer ces tâches,
- de nombreux exemples de solution peuvent être générés,
- chaque tâche comprend des associations entre objets d'ensembles différents. Ces associations sont appelées transformations ou "mappings".

Les réseaux neuronaux traitent l'information d'une façon dynamique auto-organisée typique des systèmes vivants. Ces réseaux peuvent apprendre à associer un modèle avec un autre, distinguer deux modèles, générer un modèle commun à partir de plusieurs exemples, ... Ils possèdent aussi quelques propriétés souvent associées aux modèles vivants tels que l'optimisation et la tolérance aux fautes.

Le "**neurocomputing**" est "la discipline technique concernée par les systèmes adaptatifs non-programmés de traitement de l'information (réseaux neuronaux) qui développent des associations (transformations ou "mappings") entre différents objets en réponse à leur environnement" ([HECHT,88]). Ce n'est pas une procédure pas-à-pas. Le réseau neuronal génère lui-même ses propres règles internes. Grâce à l'entraînement et l'erreur, le réseau neuronal apprend comment effectuer une tâche.

Etant donné que les réseaux neuronaux sont basés sur la structure du système nerveux biologique, on peut s'attendre à ce qu'ils héritent de certaines propriétés importantes de celui-ci :

- faire face rapidement et efficacement à des processus de traitement de l'information complexes, pouvant varier dans l'espace et dans le temps,
- être capable de résoudre un grand nombre de processus en temps réel,
- être résistant à un défaut de fonctionnement de certains neurones ou aux pannes de composants hardware individuels (pour les réseaux hardware).

Les réseaux neuronaux de part leur structure sont efficaces pour :

- l'interprétation d'ensembles de données imparfaits ou incomplets,
- fournir une réponse appropriée à un stimulus inconnu,
- s'adapter aux conditions variables lors du traitement de l'information.

Les réseaux neuronaux permettent d'éviter le développement long et difficile de certains algorithmes ou la "capture" compliquée de la connaissance des experts. Le principe sous-jacent aux réseaux neuronaux est le "learning by examples". [TRELE,88], [JOSIN,88], [SCHWA,88-2], [REMY,87], [YOON,89], [HECHT,88], [HAMME,86], [KLIMA,87-1], [PERSO,88].

III Historique

III.1. LES JALONS

III.1.1. Le modèle de McCulloch & Pitts

C'est en 1943 que l'on trouve les fondations de ce qui allait être plus tard le calcul neuronal. C'est en effet à cette époque que McCulloch et Pitts publient leur article -"A Logical Calculus of The Ideas Immanent in Nervous Activity" [McCUL,43]- sur le modèle du neurone formel ou comment les neurones du cerveau pourraient fonctionner. Ils démontrent, sous forme de propositions logiques, que des réseaux formés de tels neurones abstraits vérifient la définition de Turing pour les machines à calculer à utilisation générale.

En 1949, D.O. Hebb développe une règle d'apprentissage pour les réseaux de neurones. Cette règle stipule que lorsque deux neurones A et B interconnectés sont simultanément activés, il faut incrémenter la force de la connexion qui les relie.

III.1.2. Le début des années '50

Au début des années '50, les chercheurs ont commencé à développer des modèles hardware et software pour simuler les fonctions de bas niveau qui avaient été découvertes dans le système nerveux. Ces simulations se basèrent sur le modèle du neurone formel inventé par McCulloch et Pitts. Dès lors, les simulations firent intervenir des éléments de traitements et des connexions dont l'efficacité est pondérée par des poids et qui sont -vaguement- analogues aux neurones et synapses du cerveau. Les éléments de traitement ont été largement interconnectés entre eux pour former des réseaux. Ces modèles d'inspiration neurobiologique furent alors appelés "réseaux neuronaux". Parallèlement à ce travail de modélisation des fonctions du cerveau, un autre courant s'est attaché à modéliser les processus de pensée comme une boîte noire. Cette voie de recherche a conduit au développement des systèmes experts.

III.1.3. Le "Darmouth Summer Research Project on Artificial Intelligence"

Les chercheurs des deux groupes se sont rencontrés pendant l'été 1956 au "Dartmouth Summer Research Project on Artificial Intelligence". Ils se sont penchés sur l'utilisation potentielle des ordinateurs dans la simulation de chaque aspect de l'apprentissage ou toute autre caractéristique de l'intelligence. Cette conférence fut la rampe de lancement aussi bien de l'intelligence artificielle que du calcul

neuronal. Il y fut présenté, par Nathanial Rochester, la première simulation de réseau neuronal.

III.1.4. Les années d'or

C'est ainsi que l'on peut qualifier la décennie et demie qui a suivi la conférence de Dartmouth. Les points de repère de cette période sont tout d'abord l'élément linéaire adaptatif, appelé **Adaline**, de G. Widrow en 1959. Ce réseau fut utilisé pour supprimer les échos sur une ligne téléphonique. C'est par lui que le calcul neuronal est appliqué pour la première fois à un problème du monde réel.

Un autre événement clé est l'invention d'un élément de calcul, appelé **Perceptron** (figure III.1.), par Frank Rosenblatt en 1957. Il déclencha un grand intérêt pour le calcul neuronal. Le Perceptron était un système de classification de modèles qui pouvait à la fois apprendre à identifier des modèles abstraits et géométriques. Il était robuste car ses réponses ne se dégradaient que graduellement suite à un endommagement du réseau. En d'autre terme, le Perceptron était caractérisé par une grande plasticité car il pouvait être réentrainé et réapprendre sans erreur après que certaines de ses cellules aient été détruites. De plus, il était capable de généraliser ses réponses à des inputs déformés par le bruit.

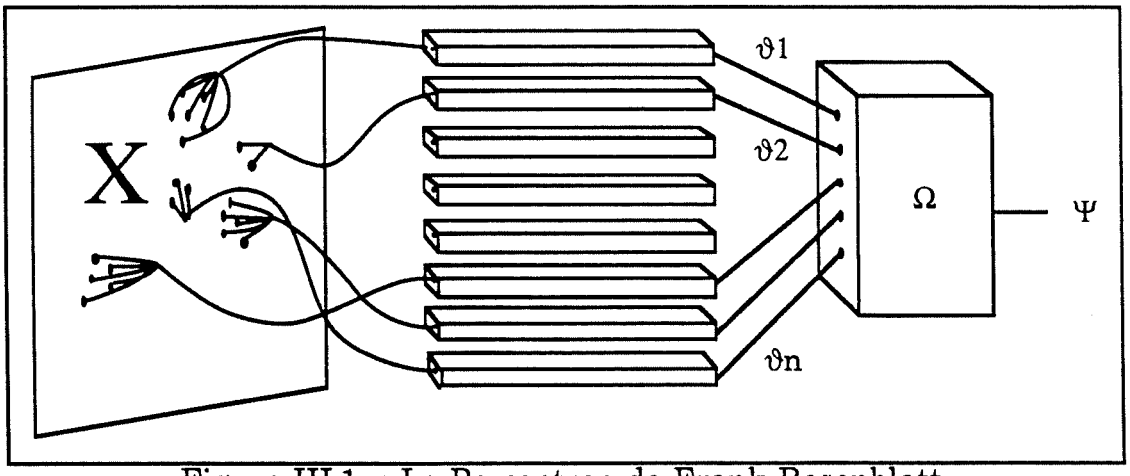


Figure III.1. : Le Perceptron de Frank Rosenblatt.

Le modèle du Perceptron était représenté par une grille de 400 cellules photo-électriques correspondant aux neurones sensibles à la lumière de la rétine. Ces cellules recevaient les stimuli optiques et étaient connectées aléatoirement à des unités d'association, imitant ainsi les connexions aléatoires entre les neurones du cerveau. Les éléments d'association étaient les éléments de traitement du Perceptron. Ils

émettaient un output si leurs inputs dépassaient un certain seuil. Ces outputs étaient ensuite additionnés pour former l'output global.

Le Perceptron provoqua une intensification des recherches sur le calcul neuronal jusqu'à ce que, quelque 12 ans plus tard, Minsky et Papert publient une critique des réseaux neuronaux remettant en cause les capacités du Perceptron, et par-là même l'avenir des réseaux neuronaux.

Pendant cette même période, l'approche "boîte noire" connaît également un grand succès.

III.1.5. La critique de Minsky et Papert

C'est en effet en 1969, 12 ans après l'invention du Perceptron, que les initiateurs du "Dartmouth Research Project on Artificial Intelligence", Minsky et Papert, publient leur critique des réseaux neuronaux. Ils démontrent entre autre que, puisque le Perceptron ne marche que pour un ensemble d'inputs linéairement séparables, il est incapable de représenter un problème aussi simple que le ou-exclusif. Les conséquences sur le monde du travail des conclusions de Minsky et Papert furent que le calcul neuronal, le Perceptron en particulier, n'étaient pas des sujets d'étude intéressants. L'approche des réseaux neuronaux fut alors abandonnée au profit de l'approche "boîte noire" qui donna naissance aux systèmes experts dans les années 70.

Malgré ce retour de flamme, certains chercheurs résistèrent dans leur étude des systèmes neuronaux. James Anderson fut l'un d'eux. Son travail se focalisa sur le développement d'un modèle linéaire, "l'associateur linéaire". Ce modèle était basé sur le principe (semblable à celui de Hebb) que des connexions entre "neurones" étaient renforcées chaque fois qu'elles étaient activées.

Anderson inventa par la suite une puissante extension de son modèle, appelée Brain-State-in-a-Box (BSB). Le principe était d'amplifier les éléments d'un vecteur d'entrée et de fixer les limites inférieures et supérieures que les états des neurones ne pouvaient dépasser. D'où le nom de **Brain-State-in-a-Box** (BSB). Ce modèle sera décrit de manière plus approfondie dans la taxonomie (point VII.). Notons dès à présent qu'il fut surtout utilisé pour simuler une grande variété de phénomènes psychologiques : perception catégorique, perception des mots, catégorisation, etc.

Un autre chercheur qui continua à poursuivre ses travaux fut Stephen Grossberg. Il passa une grande partie de son temps à concevoir des mécanismes basés sur le modèle neurobiologique pour reproduire la perception et la mémoire. Il traduisit une règle associative pour un

modèle synaptique en des équations explicites. A côté de cela, il formula bon nombre de règles d'apprentissage pour les modèles qu'il avait inventés, dont une qui porte le nom de théorème de Cohen-Grossberg [KLIMA,87-4].

III.2. LA RENAISSANCE DU CALCUL NEURONAL

III.2.1. Le modèle de Hopfield

Il faut attendre 1982 et John Hopfield pour que la modélisation neuronale regagne ses lettres de noblesse. Celui-ci présente à l'époque, à la "National Academy of Sciences", un modèle appelé "**Crossbar Associative Network**" et rebaptisé plus tard du nom de son inventeur. Ce modèle était une variation de l'associateur linéaire d'Anderson, évoqué ci-dessus, et avait beaucoup de caractéristiques en commun avec le modèle BSB développé par après. Cependant, Hopfield alla au-delà de la simple interprétation psychologique du comportement de son réseau pour en donner une interprétation physique. Il conceptualisa la dynamique de son réseau en terme d'une fonction énergie et montra qu'il était isomorphique à un modèle particulier de la "Spin Glass physics". Ensuite, il montra que des éléments de traitement avec des outputs bistables convergeaient vers un minimum local et stable de cette fonction énergie.

La combinaison de ces facteurs et du charisme personnel de J. Hopfield résultèrent en une nouvelle respectabilité du calcul neuronal et un regain d'intérêt pour ce domaine. Certains observateurs estiment qu'entre 50 et 80 % des travaux de recherche actuels sur la modélisation neuronale sont une conséquence directe de la publication du modèle de Hopfield.

En ce qui concerne l'autre approche de l'intelligence artificielle, les systèmes experts, les années '80 représentent pour elle à une phase de maturité.

III.2.2. Les événements récents

Le regain d'intérêt suscité par les travaux de J. Hopfield a trouvé sa résultante dans bon nombre de nouvelles architectures neuronales mais également dans des conférences internationales. On notera à ce propos la "First US-Japan Joint Conference on Cooperative/Competitive Neural Networks" tenue à Kyoto en 1982. Egalement, la première conférence sur le calcul neuronal, "Neural Networks for Computing", à Santa Barbara en avril 1985. Plus récemment, en juin 1987, s'est tenue à San Diego la - peut être- plus grande conférence sur le sujet : la "First International

Conférence on Neural Networks". Depuis, les conférences se succèdent en France, Angleterre, etc.

Parallèlement à ces événements, on assiste à l'apparition d'ordinateurs et de composants électroniques beaucoup moins chers et beaucoup plus puissants, ainsi qu'à un flux continu d'informations en provenance des neurosciences. Tous les ingrédients sont donc réunis pour maintenir un haut niveau d'activité dans la recherche et le développement des réseaux neuronaux.

On assiste d'ailleurs à la création de nouvelles entreprises, en réponse à la demande pour la nouvelle technologie neuronale, à la fois aux Etats-Unis et à l'étranger. Bon nombre de scientifiques se sont également installés à leur compte pour développer les fruits de leurs recherches. La technologie du calcul neuronal a encore inspiré certaines grandes compagnies qui se sont intéressées aux applications possibles. Parmi toutes ces compagnies, on remarquera Hecht-Nielsen Neurocomputer corp. (HNC), Nestor, TRW inc., NHK Science and Technology Research Laboratories, IBM corp. , Texas Instruments corp. (TI), NEC et Fujitsu, pour n'en citer que quelques unes [KLIMA,87-4].

IV Concepts fondamentaux

IV.1. LES FONDEMENTS BIOLOGIQUES

L'élément fondamental du système nerveux est le neurone ou la cellule nerveuse (figure IV.1.). Le neurone "classique" peut être défini de la manière suivante :

"Ensemble constituant la cellule nerveuse et comprenant : une masse protoplasmique qui entoure le noyau (péricaryone), de nombreuses arborisations protoplasmiques ou **dendrites** (dont l'ensemble constitue le dendrone) et un long prolongement cylindrique, le cylindraxe ou **axone**"[1]. [GARNI,89].

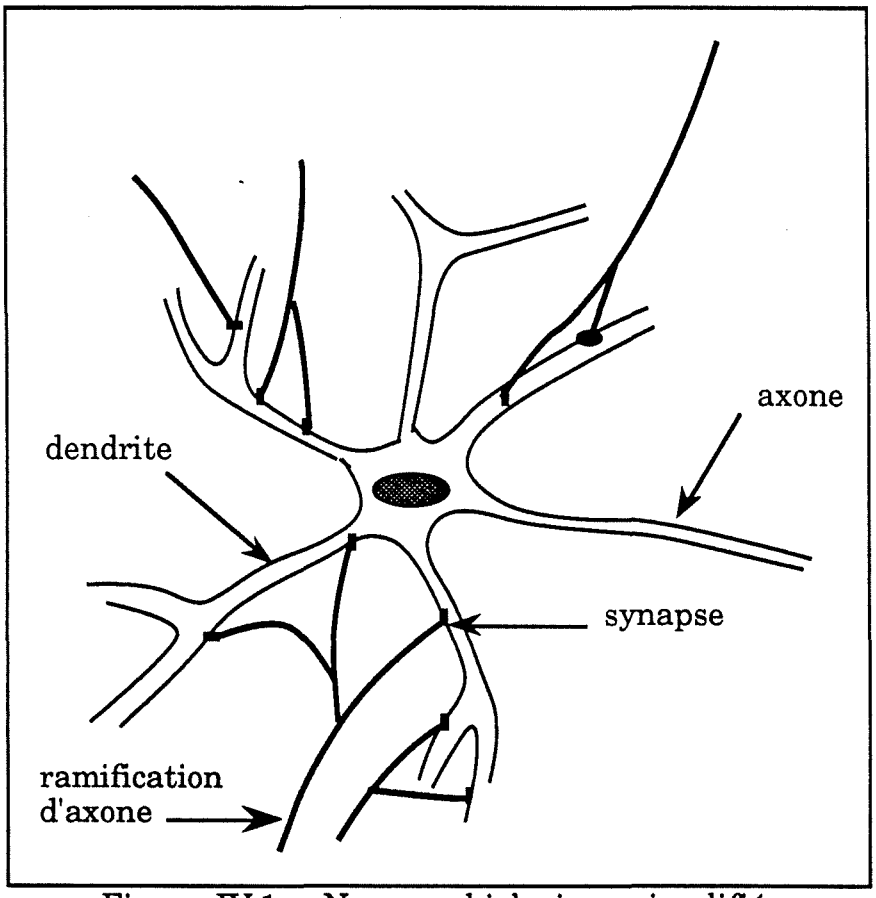


Figure IV.1. : Neurone biologique simplifié.

A partir de cette définition un peu formelle, on peut voir le système nerveux comme un ensemble de neurones interconnectés. Les **ramifications d'un axone** sont reliées à des dendrites d'autres neurones

[1] Neurone "classique" car il existe des variantes : neurones sans axone, axones qui forment des synapses sur d'autres axones, dendrites qui forment des synapses sur d'autres dendrites,... Ces différentes formes sont liées à des localisations différentes dans le cerveau.

et/ou aux siennes propres. Les points de jonction entre axones et dendrites sont appelés **synapses** et sont caractérisés par une **efficacité synaptique**. L'efficacité synaptique peut être vue comme un amplificateur de l'impulsion qui traverse la synapse.

Chaque noyau possède quant à lui un **additionneur** qui somme les impulsions électriques entrant dans le neurone et **un seuil de stimulation**, que l'excitation du neurone doit excéder pour initier une **impulsion**.

La dynamique d'un neurone est la suivante. Premièrement, des potentiels d'action envoyés par un premier neurone via son axone entrent en contact avec les synapses des neurones suivants, ce qui a pour effet de relâcher des paquets de neurotransmetteurs qui vont exciter ou inhiber les dendrites de ces neurones [TRELE,88]. La première partie du corps du neurone à être active est l'additionneur qui somme les impulsions véhiculées par les dendrites. Dans un second temps, cette somme est comparée au seuil de stimulation et si elle est supérieure à ce seuil, le neurone tirera c'est-à-dire émettra une impulsion via son axone qui sera à son tour envoyée aux neurones auxquels il est connecté.

Cette impulsion peut prendre la forme d'une **excitation** ou d'une **inhibition**. Si le courant émis par le neurone et pondéré par l'efficacité synaptique du neurone cible vise à réduire la somme des signaux entrant dans ce neurone, la stimulation est inhibitrice. De manière plus générale, l'inhibition est "la terminaison ou la prévention de l'activité d'un groupe de neurones par l'activité concurrente ou antécédente d'un second groupe" [McCUL,43]. Le corollaire est que l'excitation d'un groupe de neurones par un second groupe vise à renforcer l'activité du premier.

On mesure l'activité d'un neurone par sa fréquence de tir. La fréquence de tir est le nombre de potentiels d'action par seconde. Les neurones biologiques ne sont pas binaires, c'est-à-dire ayant seulement un état de sortie ouvert ou fermé. Les sorties sont des valeurs continues et le neurone agit fréquemment comme un convertisseur voltage-fréquence, transformant un potentiel de membrane en un taux de tir [ANDER,87].

IV.2. LE NEURONE ARTIFICIEL

Sur base de ces considérations médicales, il n'est pas étonnant que la brique de base des réseaux neuronaux soit un modèle du neurone biologique, appelé de façon équivalente **élément** ou **unité de traitement**, **cellule**, **noeud**, **neurone artificiel** ou simplement "**neurone**". Nous utiliserons cette dernière appellation seulement lorsqu'il n'y aura pas d'ambiguité et dans le sens de "neurone artificiel". Ce dernier ne ressemble en effet que vaguement au neurone biologique du point de vue

de son fonctionnement. La figure IV.2. donne une représentation du neurone formel et le tableau IV.1. la correspondance entre les termes médicaux et ceux de la modélisation.

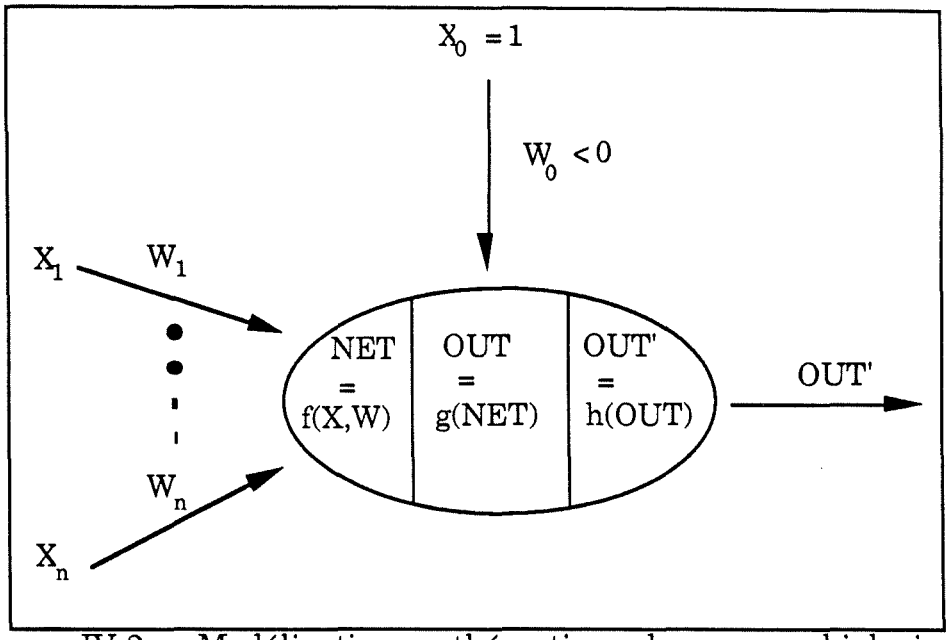


Figure IV.2. : Modélisation mathématique du neurone biologique.

NEURONE BIOLOGIQUE	NEURONE ARTIFICIEL
dendrites	inputs/entrées
axone	output/sortie
synapse	connexion pondérée
efficacité synaptique	poids de connexion, poids synaptiques
impulsion, potentiel d'action	signal
additionneur	fonction d'addition
seuil (d'activation)	seuil, déplacement, bias + fonction d'activation, de transfert
corps, soma, neurone	neurone artificiel,...
excitation	connexion positive
inhibition	connexion négative

Tableau IV.1. : Correspondance entre les termes de la biologie et ceux de la modélisation.

On observe que lors de la modélisation, les notions de dendrites et axones ont fait place à des notions plus abstraites d'entrées et sorties. Le concept de connexion n'est plus présent que par la modélisation de la synapse qui représente à elle seule la connexion et son poids entre neurones artificiels.

Dans la représentation mathématique du neurone formel (figure IV.2.), on distingue trois parties logiques : les entrées, le

traitement et la sortie. Pour chaque partie, nous allons présenter son rôle dans le fonctionnement du neurone ainsi que les notations introduites pour la représenter[2].

1) la première, celle qui correspond aux synapses et dendrites du neurone biologique, reçoit une ou plusieurs entrées et pondère chaque entrée par un poids synaptique, ce qui a pour effet d'amplifier ou de diminuer la valeur d'entrée. Les poids synaptiques constituent l'information stockée, la **mémoire du réseau** mais également **l'algorithme** qui permet de recréer ce qui a été stocké. Changer les poids, c'est changer ce que le réseau **connaît**. Les pondérations sont ensuite passées à la **fonction d'entrée totale f** qui les combine et transmet le résultat à la section logique suivante.

Nos conventions :

X: vecteur d'entrée;

W : matrice des poids du réseau;

f : fonction d'entrée totale;

$\mathbf{NET}_i(t) = f(X,W)$: le résultat de la fonction d'entrée totale.

La fonction d'entrée totale f peut prendre différentes formes :

a) *affine* : $f(X,W) = W\,X = \sum_{i=1}^{n} w_i x_i + w_0 x_0$;

$W = (w_0, w_1, \ldots, w_n)$: $x_0 = 1$ et $w_0 < 0$. L'input x_0 modulé par le poids synaptique négatif w_0 représente le **seuil** du neurone artificiel. En effet, le produit $x_0 w_0$ étant en principe[3] toujours négatif, cela peut être vu comme la connexion à un neurone qui réduirait la somme des entrées du neurone;

---

[2] Remarquons qu'il n'existe pas encore de standards en matière de termes, notations et représentations graphiques des réseaux de neurones artificiels.

[3] Nous verrons par la suite qu'il est possible de changer W_0 sous l'effet de l'apprentissage. W_0 peut alors devenir positif sans que le comportement du réseau n'en souffre.

b) *linéaire* : $f(X,W) = W X = \sum_{i=1}^{n} w_i x_i$;

$W = (w_1,...,w_n)$: on utilise pas w_0 comme seuil mais une fonction de sortie h (cfr 3.) de type seuil.

c) *polynomiale de degré supérieur à 2*;

2) Cette section calcule un signal d'output en appliquant une **fonction d'activation g** au résultat fourni par la section logique précédente et transmet le résultat à la section logique suivante.

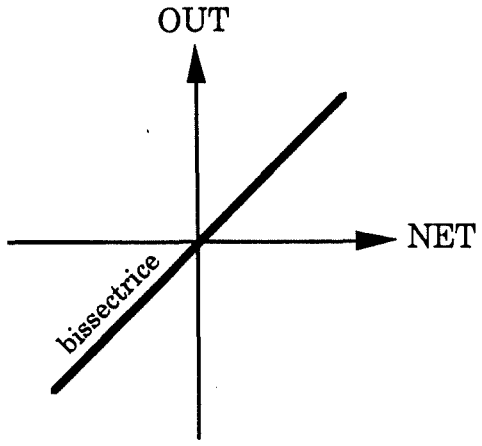
Nos conventions :

g : fonction d'activation;

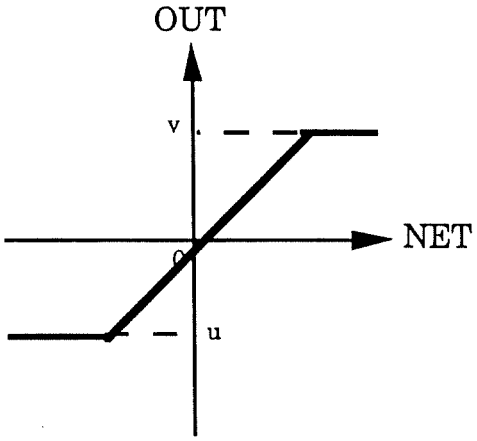
$\mathbf{OUT}_i(t) = g(NET)$: résultat de g pour le neurone i au temps t.

La fonction d'activation est en général attachée à un modèle particulier de réseau neuronal. Les différentes formes de cette fonction sont :

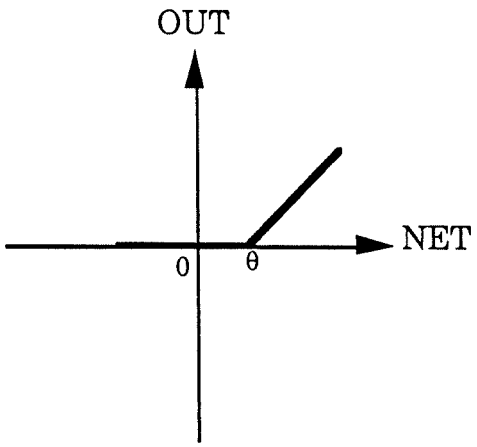
a) une *fonction linéaire simple* : OUT=NET;



b) une *fonction linéaire à seuil* :

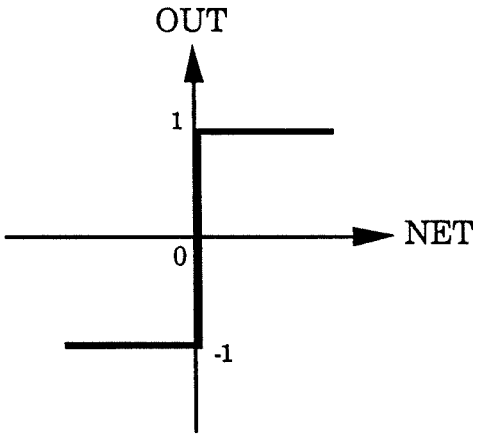


Satur(x) = x si x ∈ [u,v]
u si x ≤ u
v si x ≥ v

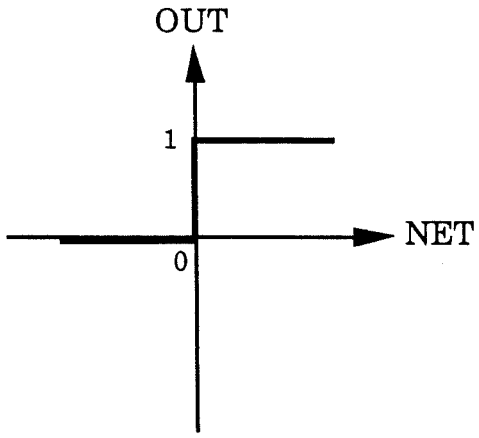


Perceptron

c) une *fonction binaire à seuil* :

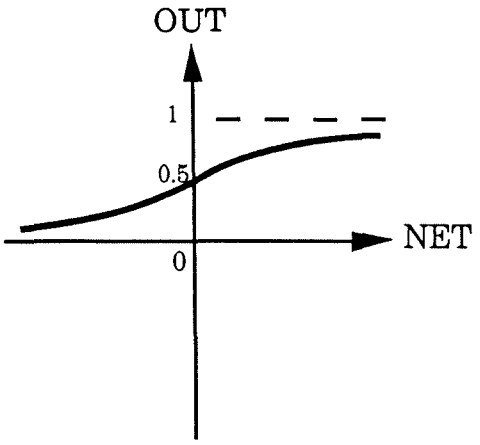


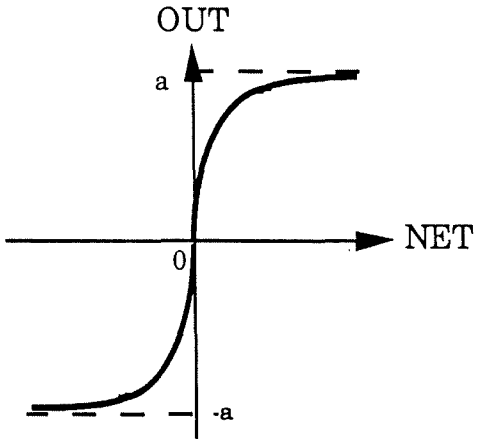
fonction signe



fonction de Heaviside

d) une *fonction sigmoïde* :



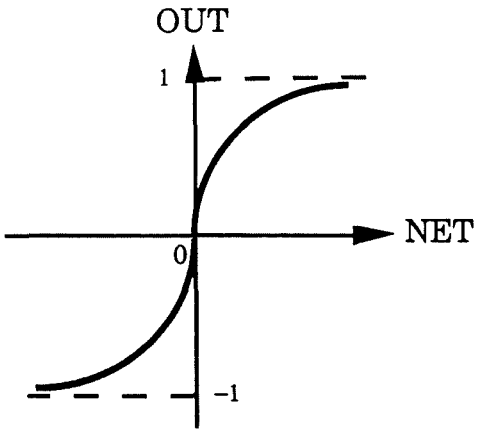


fonction logistique
(qui simule le mieux la non-linéarité du neurone biologique)

$OUT = 1 / (1 + e^{-NET})$

$OUT = a (e^{kx} - 1) (e^{kx} + 1)$

ou encore :



OUT = tanh(x)

3) La dernière section calcule par la **fonction de sortie** le signal qui sera éventuellement passé à un ou plusieurs autres neurones, au neurone lui-même ou au monde extérieur. Ce signal d'output constitue l'**état d'activité** du neurone. Il est le **codage** de ce que le neurone représente au niveau conceptuel.

Nos conventions :

h : fonction de sortie;

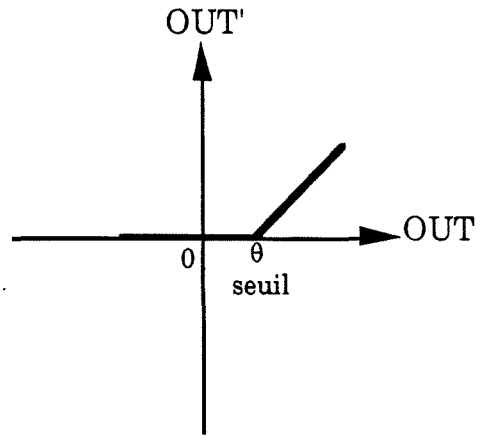
$\mathbf{OUT'}_i(t)$ = h(OUT) : l'état d'activité du neurone i au temps t.

En général, cette fonction de sortie est considérée comme la fonction identité (notre hypothèse) :

OUT' = h(OUT) = OUT = g(NET) .

Dans ce cas, l'état d'activité du neurone est fourni par la fonction g.

Certains réseaux utilisent cependant une fonction de sortie différente. On désire par exemple que seul le neurone ayant l'état d'activité le plus grand émette son résultat. Dans ce cas, la fonction de sortie h permet d'annuler la sortie de tous les neurones ayant une activation supérieure à ce seuil (réseau Adaline ou réseau de Kohonen). Parfois encore, on utilise une fonction d'entrée linéaire et une fonction de sortie à seuil pour simuler le seuil du neurone biologique :



Ce modèle simple ne tient pas compte de certaines caractéristiques de sa contrepartie biologique et notamment des délais temporels qui affectent la dynamique d'un système biologique. En effet, pour chaque neurone, chaque tir est séparé d'une période de 0.75 msec, correspondant à un temps d'addition latente des entrées et un temps de propagation de l'impulsion dans le corps du neurone [McCUL,43]. Dans le modèle mathématique initial, au contraire, chaque neurone tire tout de suite. C'est pourquoi des unités plus complexes ont été développées. Celles-ci intègrent le temps ou d'autres types de dépendances vis à vis du temps, ainsi que des opérations mathématiques plus complexes que l'addition.

Les neurones artificiels les plus utilisés sont les suivants :

- neurone booléen :

* E(ntrées)/S(ortie) : booléennes;
* h : fonction booléennes des entrées;

- neurone à seuil :

* E : binaires ou réelles;
* S : binaire;
* f : affine;
* g : signe ou Heaviside;

- neurone linéaire :

* E/S : réelles;
* f : linéaire;
* g : linéaire simple;

- neurone à saturation :

* E/S : appartiennent à un intervalle [u,v] de valeurs réelles;
* f : linéaire;
* g : Satur;

- neurones continus :

* E/S : réelles;
* f : linéaire ou affine;
* g : sigmoïde;

- neurones probabilistes :

* E : quelconques;
* S : binaires;
* f : linéaire ou affine;
* g : stochastique.

IV.3. ARCHITECTURES D'INTERCONNEXION

Très rare sont les cas où seul un neurone est utilisé pour résoudre un problème donné. On préfère plutôt utiliser plusieurs neurones travaillant en parallèle pour bénéficier d'une plus grande puissance de traitement. A notre connaissance, seul l'égaliseur adaptatif de canal utilisé pour stabiliser la voix dans les communications téléphoniques, fonctionne avec un seul neurone.

IV.3.1. Définition d'un réseau neuronal

Des neurones artificiels opérant en parallèle et largement interconnectés forment un réseau neuronal artificiel. La fonction du réseau est déterminée par la **structure de ses interconnexions**, les **poids synaptiques** et le **traitement** effectué dans les cellules (fonctions f, g et h). Le réseau est entièrement spécifié lorsqu'on a en plus déterminé sa **règle d'apprentissage** c'est-à-dire la manière dont le réseau va modifier ses poids.

D'un point de vue fonctionnel, le réseau peut exécuter des tâches de haut niveau, telles l'adaptation ou l'apprentissage ou des tâches de bas niveau comme le traitement d'inputs sensoriels dans les tâches de vision ou de reconnaissance de la parole.

Nous utiliserons de façon équivalente les vocables **système**, **réseau**, **réseau de neurones** et **réseau neuronal artificiel**.

IV.3.2. Réseaux multi-couches

Si l'on s'en réfère aux études biologiques du cerveau, on constate que le cortex cérébral est divisé en plusieurs couches. A l'intérieur d'une couche, les interactions entre neurones sont très grandes et entre les couches les neurones sont également interconnectés, le tout formant un système d'une complexité gigantesque. L'architecture des réseaux neuronaux reprend cette découpe en couches et peut aller d'une **connectivité totale** à une **connectivité locale** où les neurones ne sont reliés que dans un certain voisinage.

Vu la difficulté que représente la modélisation d'un tel système hyper-connecté, on préfère la structure de réseau à couches où les neurones d'une couche ne sont pas connectés entre eux mais seulement aux couches adjacentes (cfr figure IV.3.). Chaque couche reçoit des inputs de la couche précédente et transmet ses résultats à la couche suivante.

Les couches sont de deux types :

1) **couche d'entrée** : le seul rôle des neurones de cette couche est de transmettre les inputs provenant du monde extérieur à la couche suivante (fonction d'activation linéaire et un seul input par neurone d'entrée). Il existe une et une seule couche d'entrée par réseau. Ce rôle simpliste des neurones d'entrée incite certains auteurs à ne même pas la représenter. Dans ce cas le réseau commence donc directement avec des connexions comme indiqué dans le bas de la figure IV.3.;

2) **couches de traitement** : c'est dans ces couches que s'effectue tout le calcul du réseau. On peut les décomposer en deux catégories :

a) la **couche de sortie** : couche située après la couche d'entrée. Elle permet au réseau de communiquer ses résultats au monde extérieur;

b) les **couches cachées** : situées entre la couche d'entrée (ou la première couche de connexion si la couche d'entrée n'est pas représentée) et la couche de sortie, elles n'ont aucun contact avec le monde extérieur.

Seul le nombre de couches cachées (nbr_c_cachées) et la couche de sortie interviennent dans le calcul du nombre de couches d'un réseau (nbr_couches) :

nbr_couches = nbr_c_cachées + 1

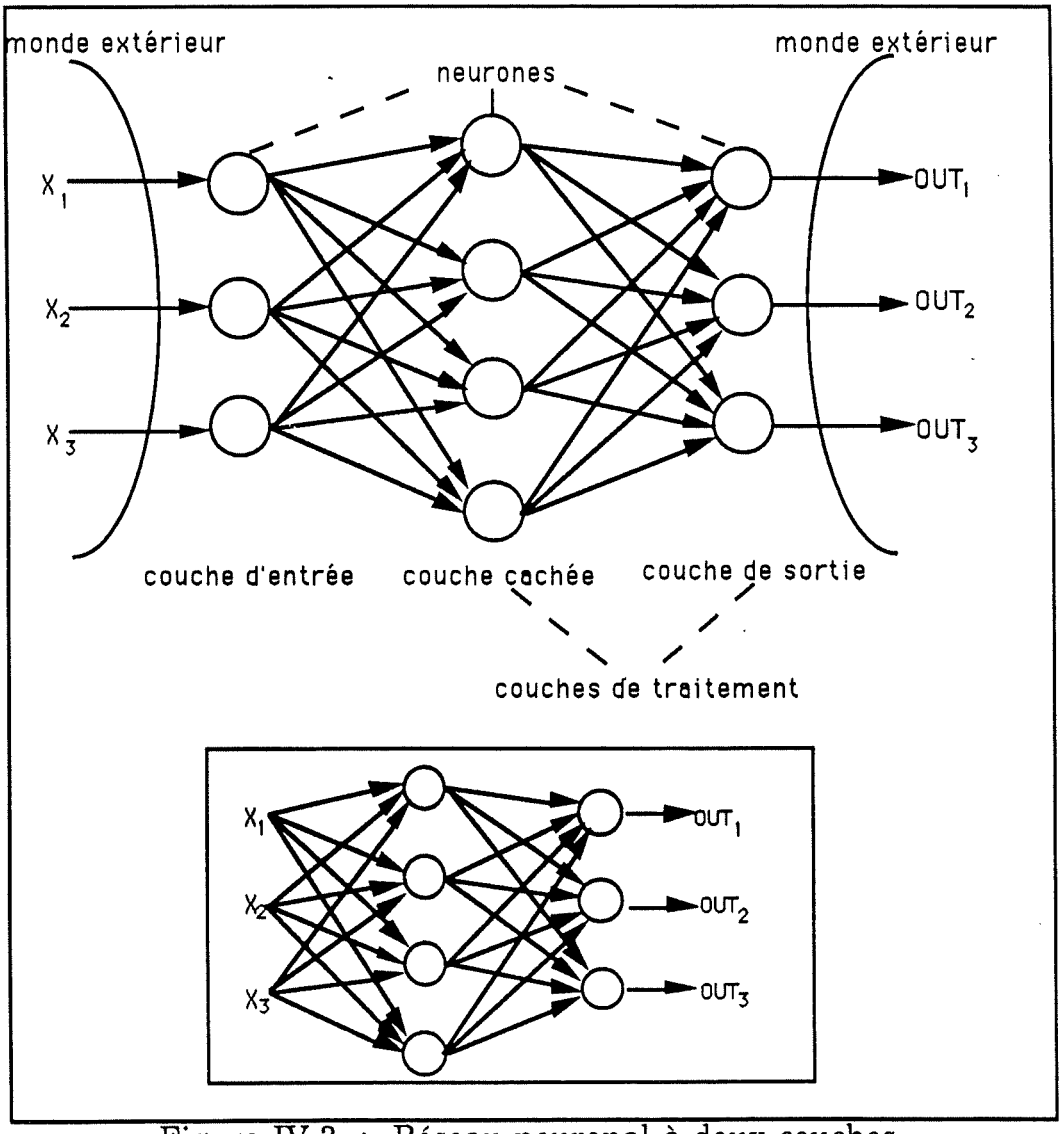


Figure IV.3. : Réseau neuronal à deux couches.

Remarque : Un tel réseau à plusieurs couches n'offre pas une puissance de traitement supérieure à un réseau à couche unique si la fonction d'activation est linéaire. En effet, dans ce cas, pour un réseau a deux couches, si X représente le vecteur d'entrée et W_1 et W_2 les matrices des poids des connexions des couches cachées et de sortie, l'output est exprimé par :

$$OUT = NET = (X W_1) W_2 = X (W_1 W_2);$$

par associativité du produit matriciel; ce qui est équivalent à l'output d'un réseau monocouche dont la matrice des poids serait le produit $W_1 W_2$.

IV.3.3. Réseaux entièrement connectés

Ces réseaux sont constitués **d'une seule** couche de traitement et d'une couche d'entrée et sont entièrement connectés. Chaque neurone envoie ses résultats vers tous les autres neurones ainsi qu'à lui-même (figure IV.4.)

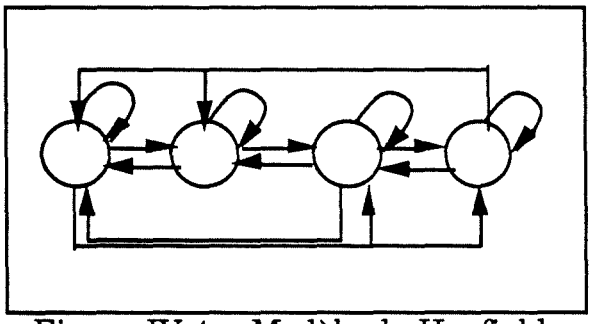


Figure IV.4. : Modèle de Hopfield.

L'importance de ces réseaux n'est pas due à une origine biologique, comme c'était le cas pour les réseaux à couches, mais au fait qu'on a pu établir une analogie entre ce type de réseau constitués de neurones à seuil et le comportement d'un verre de spin (cfr réseau de Hopfield dans la taxonomie au point VI.).

IV.3.4. Topologie

Les formes d'interconnexion entre noeuds varient d'un modèle à l'autre. Les noeuds peuvent être **localement connectés** aux voisins, **complètement interconnectés** entre eux (cfr figure IV.4.) ou **faiblement**

connectés à quelques noeuds distants (par ex. : le réseau multi-couches figure IV.3.).

De plus, les réseaux peuvent posséder des **connexions feed-forward**[4] d'une couche inférieure vers une couche supérieure (par ex. : réseau multi-couches) ou des **connexions feed-back** d'une couche vers elle-même ou vers des couches inférieures s'il y en a. Ces derniers sont appelés **réseaux récurrents** et présentent des propriétés semblables à la mémoire à court terme chez les humains -du fait que les outputs sont réinjectés dans le réseau.

Les réseaux récurrents doivent itérer de nombreuses fois avant de fournir un output alors que les réseaux feed-forward doivent seulement exécuter le traitement jusqu'à ce que les inputs se propagent vers la couche de sortie et ensuite vers le monde extérieur (pas de feed-back).

IV.3.5. La non linéarité

Le problème de la linéarité a déjà été abordée dans les réseaux multi-couches (point IV.3.2). Un des changements fondamentaux de la modélisation neuronale a été de remplacer des systèmes initialement linéaires par des systèmes non-linéaires. A la fin des années 60, c'est justement la nature linéaire du Perceptron de Rosenblatt qui a permis à Minsky et Papert d'en souligner les faiblesses. Un tel réseau ne pouvait même pas résoudre un "ou-exclusif" parce qu'incapable de déterminer, dans l'espace des données, une région de décision suffisamment précise (cfr. [MINSK,69] ou [HINTO,87]).

Dès lors, la non-linéarité a été introduite dans les réseaux neuronaux. Elle peut prendre différentes formes :

* *feed-back* : c'est le cas du réseau de Hopfield; la non-linéarité est introduite via le feed-back dans une même couche;

* *feed-back + plusieurs couches* : généralisation du modèle de Hopfield à un modèle multi-couches; l'inventeur est Bart Kosko et son réseau s'appelle BAM ("Bi-directional Associative Memory");

* utiliser la *compétition lors de l'apprentissage* : un élément qui produit un résultat plus grand que les autres pendant l'apprentissage peut adapter ses poids, contrairement à ses congénères; un des premiers à utiliser cette technique fut Kohonen;

[4] Les mots "feed-back", "pattern", "output" et "input" sont repris dans la langue française. Nous avons considéré qu'il en allait de même pour "feed-forward" vu le sens évident du terme et pour la facilité d'écriture.

* *réseaux multi-couches feed-forward* avec une *fonction de transfert non linéaire* : ces systèmes ont été appelé "réseaux à rétro-propagation" et ont été notamment mis en oeuvre par Rumelhart.

IV.4. ETATS D'UN RESEAU NEURONAL

L'état du neurone i au temps t est représenté par le signal de sortie $OUT_i'(t)$, posé comme égal à $OUT_i(t)$ (cfr IV.2). Dès lors, **l'état d'un réseau** au temps t est spécifié par un vecteur de N nombres réels (où N est le nombre d'unités contenues dans le réseau), OUT(t), représentant le **vecteur d'état** ou **pattern d'activation** de l'ensemble des unités de traitement au temps t.

L'état initial du système est :

* soit aléatoire pour tous les neurones;

* soit forcé à une certaine valeur pour les neurones d'entrée et aléatoire ou forcé pour les autres neurones.

L'état courant du système peut être calculé selon trois techniques :

1) mode synchrone

On suppose que le temps est discrétisé et que l'évolution du réseau se fait suivant cette discrétisation. A chaque pas d'horloge, *tous* les neurones recalculent leur nouvelle activation à partir des entrées obtenues au top précédent. Cette mise à jour continue jusqu'à obtenir un **état stable** c'est-à-dire un vecteur d'état qui ne change plus au cours du temps.

2) mode asynchrone

Chaque neurone, à tout moment, a une certaine probabilité d'évaluer ses entrées et de recalculer son état d'activation. A chaque instant, il y a donc un nombre aléatoire de neurones qui réévaluent leur état. Cependant, si on regarde à des intervalles de temps suffisamment petits, il y a un seul neurone à la fois qui est mis à jour.

Le mode asynchrone permet d'améliorer la stabilité en évitant au réseau d'osciller autour de la réponse, ce qui arrive fréquemment dans le mode synchrone.

3) mode mixte

C'est une combinaison des modes synchrones et asynchrones.

David Tank et John Hopfield ont représenté l'état de leur réseau neuronal par une quantité mathématique appelée **énergie quantitative** (application du concept de stabilité globale de Liapunov -2ième théorème) [TANK,88]. Cette énergie est fonction des caractéristiques des éléments de traitement, des poids des connexions et des données externes. Elle permet de visualiser les différents états d'un réseau comme un paysage constitué de collines et de vallées (cfr figure IV.5.).

Le réseau calcule un chemin qui décroît l'énergie quantitative jusqu'à ce que le chemin atteigne un point stable d'énergie c'est-à-dire un minimum local (fond d'une vallée). Les vallées correspondent à des données préalablement enregistrées et les collines à un état de départ lors de l'utilisation du réseau avec de nouvelles données. Le but étant pour le réseau de se rapprocher de l'état correct préenregistré. Nous reviendrons en détail sur la fonction énergie lors de la présentation du modèle de Hopfield.

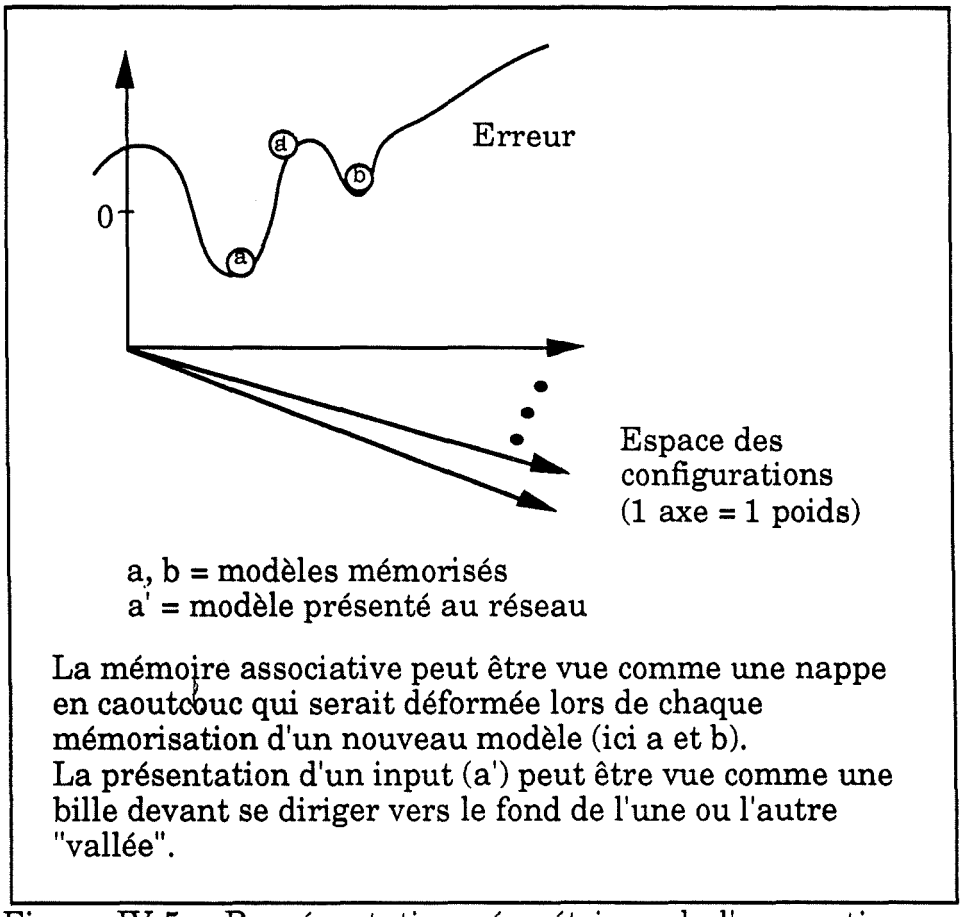


Figure IV.5. : Représentation géométrique de l'apprentissage.

Nous allons maintenant voir comment un réseau peut enregistrer initialement des connaissances via le mécanisme de l'apprentissage.

IV.5. L'APPRENTISSAGE D'UN RESEAU NEURONAL

IV.5.1. Un exemple

Illustrons d'abord par un exemple simple en quoi consiste l'apprentissage d'un réseau.

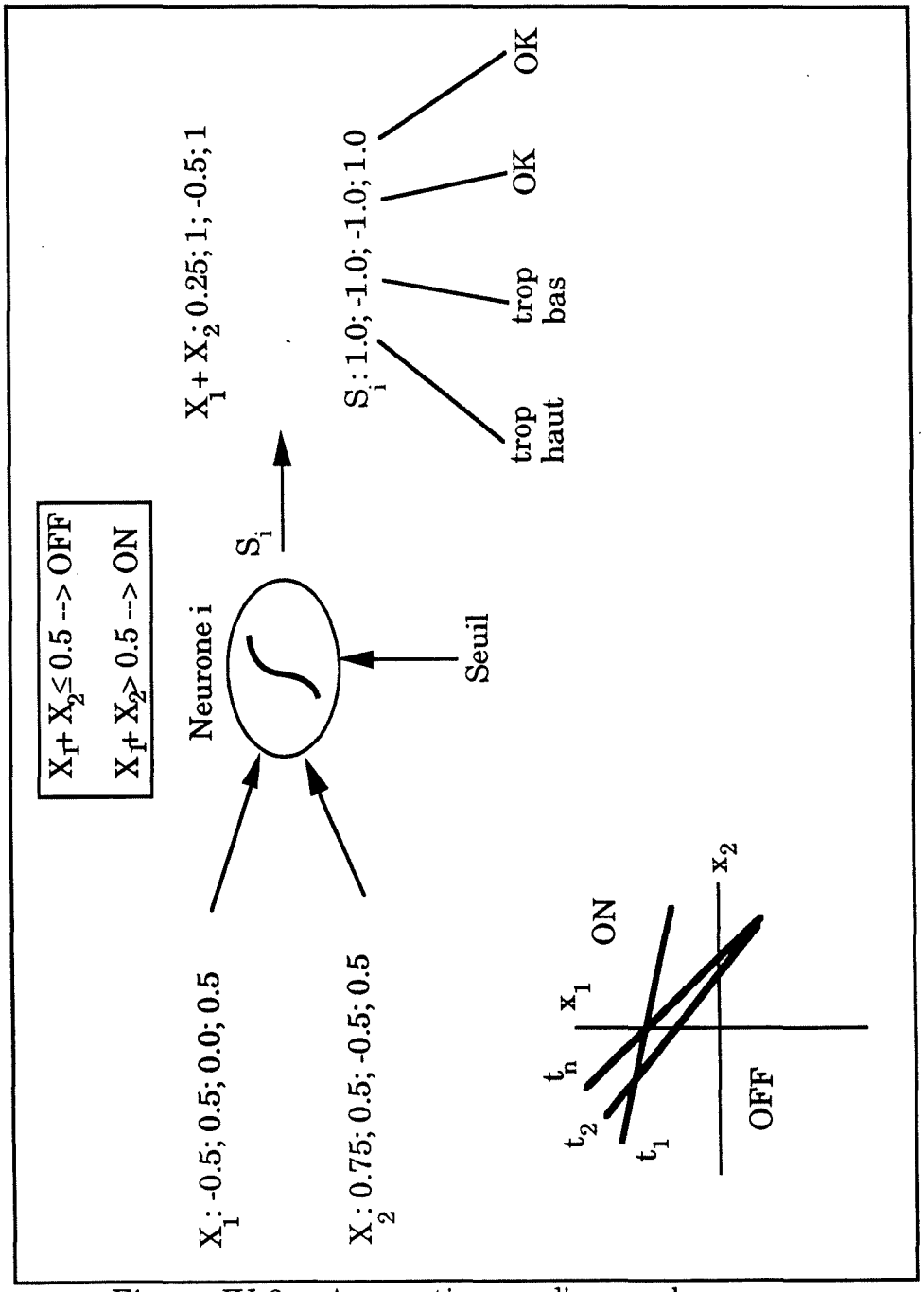


Figure IV.6. : Apprentissage d'un seul neurone.

Le réseau est constitué d'un seul neurone qui est entraîné à allumer une lumière lorsque la somme de ses voltages d'entrées x_1 et x_2 est supérieure à 0.5 et à éteindre cette lumière lorsque la somme est inférieure (figure IV.6.).

Le neurone sera entrainé via l'ajustement manuel de ses poids w_1 et w_2 -correspondants aux voltages respectifs de x_1 et x_2- et de son seuil θ_1.

Il s'agit ici d'un apprentissage supervisé c'est-à-dire dans lequel un professeur connaît à tout moment la réponse que le réseau doit fournir. Le processus d'apprentissage se réalise en plusieurs étapes :

1) au premier essai, la somme des inputs est inférieure à 0.5, la lumière devrait donc être éteinte. Or, les poids et le seuil sont tels que la lumière est allumée. Le professeur va donc ajuster les poids et le seuil pour l'éteindre;

2) au deuxième essai, la somme est supérieure à 0.5 et la lumière doit donc être allumée. Il se fait que cette lumière est éteinte du fait que le professeur a trop corrigé les poids et le seuil à l'essai précédent. Ces derniers sont donc réajustés par le professeur pour allumer la lumière;

3) la somme est négative et la lumière s'éteint, ce qui est correct. Aucun n'ajustement n'est donc fait;

4) la quatrième somme est supérieure à 0.5 et la lumière s'allume. On peut considérer que le réseau a bien appris c'est-à-dire que ses poids et seuil sont correctement réglés.

Cet exemple montre en fait que le réseau a appris à identifier une ligne dans un espace à deux dimensions, comme indiqué dans le coin gauche de la fig. IV.6.. Cette droite est : $x_1+x_2 = 0.5$.

De la même façon, avec trois inputs, le réseau pourrait être entraîné à identifier un plan dans un espace à trois dimensions. Avec n inputs, le réseau pourra identifier un hyperplan dans un espace à n dimensions. L'analogie géométrique peut être étendue à plusieurs couches de neurones (figure IV.7.). Avec une couche de neurones supplémentaire (2), chaque neurone de la première couche peut identifier un demi-plan et la deuxième couche permet d'en réaliser l'intersection.

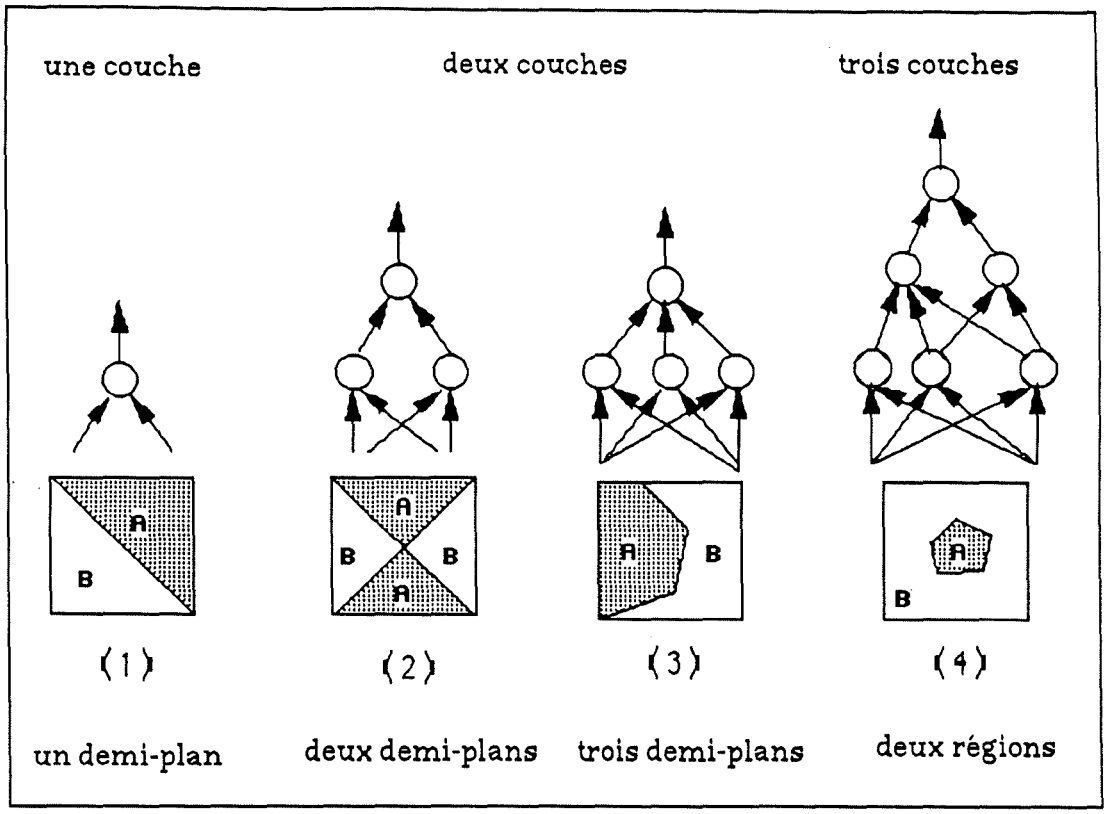


Figure IV.7. : Régions formées par différents réseaux.

Ceci permet de résoudre le problème du ou-exclusif (XOR) sur lequel butait le perceptron de Rosenblatt :

	0	1
0	A	B
1	B	A

L'addition d'un troisième neurone dans la première couche (3) permet de diviser la région en trois demi-plans, ce qui donne une région convexe. Plus on rajoute de neurone et plus on obtient de précision, mieux on peut cerner une région. Si l'on rajoute une troisième couche de neurone (4), on obtient comme résultat de l'apprentissage une région complexe, intersection de plusieurs convexes. Dans ce cas, on peut isoler deux régions concentriques A et B.

IV.5.2. Définition

Après cet exemple du neurone qui allume et éteint une lumière, on peut dire qu'on attend d'un neurone en phase d'apprentissage un

comportement qui n'est pas celui qu'il produit initialement. Pour amener ce comportement à tendre vers celui qui est désiré, il faut modifier les poids synaptiques du réseau. Les autres paramètres, fonctions d'entrée, d'activation et de sortie, étant fixés lors de la création du réseau, ne sont plus changés par la suite. On peut définir l'apprentissage de la manière suivante.

La **règle, procédure** ou **loi d'apprentissage** ou **d'adaptation** est un algorithme dont l'application par le neurone qui en fait l'objet permet la modification de tout ou partie des poids de ses connexions et ce, en vue d'augmenter sa contribution à la performance du réseau.

Par performance, on entend un rapprochement des solutions fournies par le réseau vers la réponse correcte (un classement correct, une reconnaissance d'objet exacte,...). On peut remarquer que chacun des éléments du réseau contient la règle d'apprentissage qui lui permet de s'auto-ajuster pendant l'entraînement.

La modification de la dynamique du réseau engendrée par l'adaptation d'un ou plusieurs neurones en vue d'accomplir un but particulier s'appelle **auto-organisation**. L'auto-organisation est donc au réseau ce que l'adaptation est au neurone

L'ensemble des données utilisées lors de l'apprentissage -ou entraînement- constitue **l'ensemble d'entraînement.**

Après plusieurs présentations de l'ensemble d'entraînement et d'ajustements des poids consécutifs, l'écart entre résultat effectif et résultat désiré est inférieur à un certain seuil. On dit alors que le réseau est entraîné, prêt pour la phase d'utilisation ou de rappel (figure IV.8.)

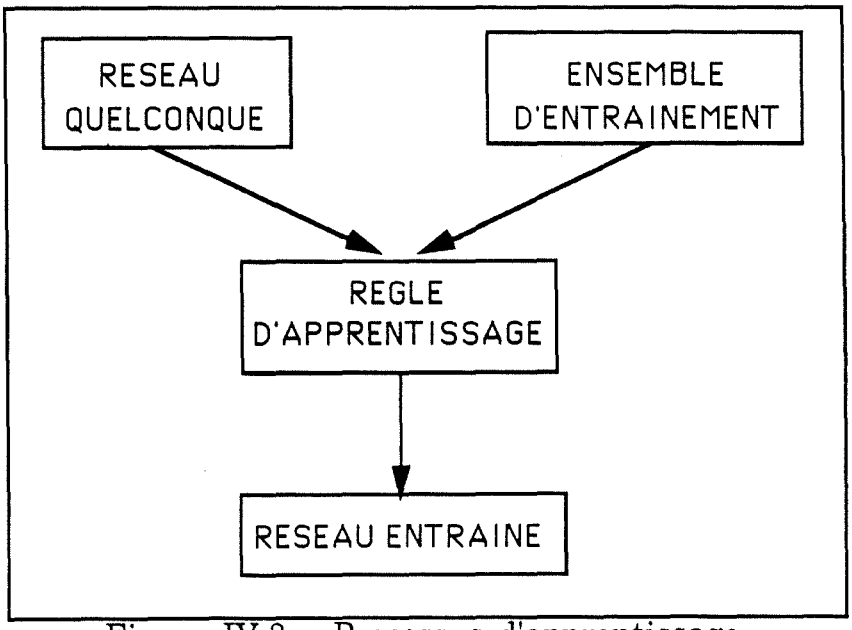


Figure IV.8. : Processus d'apprentissage.

Un réseau peut contenir différentes règles d'apprentissage. Si la structure du réseau est multi-couches, chaque couche peut posséder sa propre règle. Cependant, pour une loi donnée, l'unité minimale d'application reste la couche. Il n'est donc pas possible d'avoir différents neurones d'une même couche avec des lois d'adaptation différentes. De plus, ces lois sont invariantes pendant toute la durée de l'apprentissage. Ces limitations proviennent, à notre sens, de considérations d'ordre biologique : on observe dans le cerveau des structures régulières et on fait dès lors l'hypothèse qu'à l'intérieur d'une structure tous les éléments fonctionnent de la même façon.

IV.5.3. Types d'apprentissage

Avant de passer en revue différentes règles d'apprentissage, nous allons distinguer les trois grandes catégories dans lesquelles elles peuvent être regroupées.

1) Apprentissage supervisé

C'est l'apprentissage utilisé dans l'exemple du neurone allumant et éteignant une lumière.

L'apprentissage supervisé requiert l'étiquettage des données et un professeur externe. Chaque élément de l'ensemble d'entraînement est un couple **(input, output désiré)** et la différence ou l'erreur entre l'output fourni et l'output désiré est rétro-propagée à travers le réseau. Les poids sont alors changés selon un algorithme qui vise à minimiser cette erreur (cfr Perceptron et Perceptron multi-couche au point VI.4.4.). Les couples sont présentés séquentiellement et les erreurs et les poids sont calculés pour chaque couple jusqu'à ce que l'erreur pour tout l'ensemble d'entraînement soit à un niveau suffisamment bas. Cet apprentissage est parfois appelé **apprentissage de renforcement** ou **apprentissage avec un critique** lorsque le professeur indique seulement si la réponse est correcte ou incorrecte sans donner d'information plus détaillée.

2) Apprentissage non supervisé

L'apprentissage non supervisé développé par Kohonen en 1984 ne requiert ni d'étiquettage des données ni de professeur externe. Le processus d'apprentissage extrait les propriétés statistiques de l'ensemble d'entraînement (cfr VI.4.6. modèle de Kohonen) et groupe des inputs similaires en classes. En appliquant lors de la phase de rappel un input d'une classe donnée à l'entrée du réseau, ce dernier produira un vecteur d'output spécifique à la classe. Il n'y a aucune façon de déterminer avant l'entraînement quel output spécifique sera produit pour un vecteur d'entrée donné, pas plus que de dire quelles classes seront formées. Il faut dès lors transformer les outputs du réseau en une forme

compréhensible pour mieux identifier les relations d'entrées/sorties établies par le réseau.

On peut remarquer que l'apprentissage non supervisé est une modélisation beaucoup plus plausible de l'apprentissage biologique. En effet, sous l'hypothèse d'un entraînement supervisé dans le système biologique, d'où viendrait l'output désiré ?

3) Apprentissage auto-supervisé

L'apprentissage auto-supervisé est utilisé par des automates qui règlent leurs performances de façon interne. Il ne requiert pas de professeur externe. Il est encore appelé **apprentissage par expérimentation.**

Par exemple, un robot qui est entraîné à suivre un point lumineux peut générer un signal d'erreur calculé d'après la distance séparant l'impact du point lumineux sur la rétine simulée et le centre de cette rétine [DARPA,88].

Bien que les trois types d'algorithmes aient été développés pour des applications différentes, les plus populaires et les plus puissants sont supervisés [ANDER,87].

IV.5.4. Règle de Hebb

La **règle de D. O. Hebb** (1949) est le premier mécanisme d'évolution proposé pour les synapses :

> Si deux neurones connectés entre eux sont activés au même moment, la connexion qui les relie doit être **renforcée**. Dans le cas contraire, elle n'est pas modifiée (figure IV.9.).

On explique de cette façon le phénomène d'habitude et d'apprentissage. Si w_{ij} représente le poids de la connexion reliant le neurone j au neurone i (et pas l'inverse), la modification de poids proposée par Hebb est :

$$\Delta w_{ij} = \eta \; OUT_i \; OUT_j$$

avec :

- OUT_i et OUT_j les outputs effectifs des neurones i et j respectivement;

- η, une constante de proportionalité représentant la rapidité ou le taux d'apprentissage.

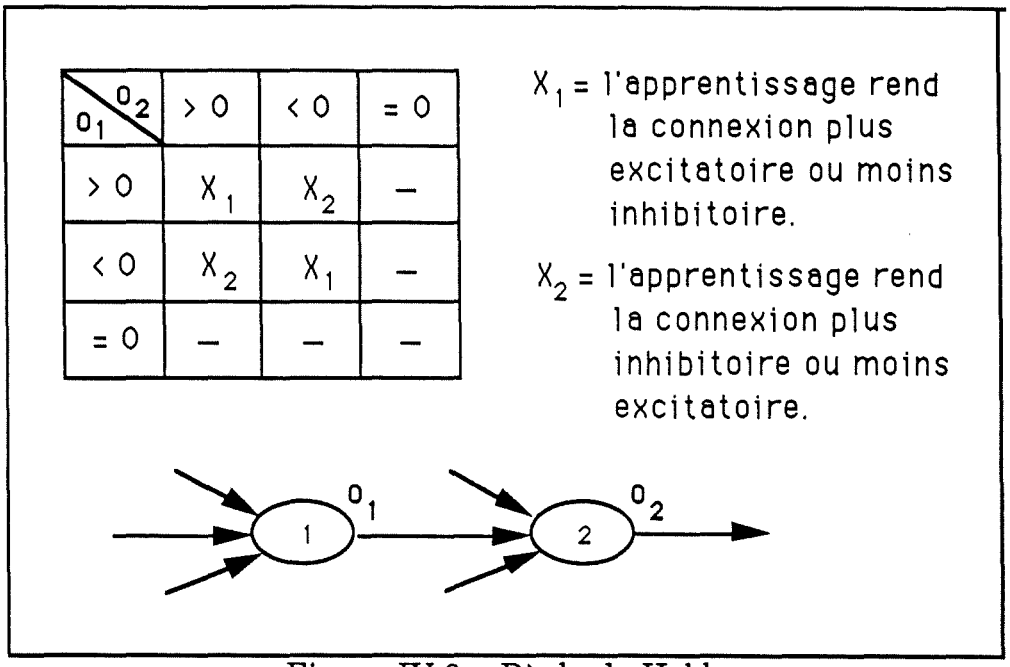


Figure IV.9. : Règle de Hebb.

Comme on le voit dans la formule, l'information nécessaire pour utiliser la règle de Hebb est disponible localement. On peut donc implémenter la modulation de poids localement dans chaque connexion, sans avoir besoin d'un professeur externe qui irait dans chaque connexion ajuster le poids à sa bonne valeur.

La plus grande limitation de la règle de Hebb est la nécessité d'utiliser des vecteurs d'entraînement non corrélés c'est-à-dire linéairement indépendants. La démonstration de cela peut être trouvée dans [RUMEL,88].

A l'heure actuelle, toutes les règles d'apprentissages peuvent être considérées comme une variante de la règle de Hebb. Cependant, alors que la règle de Hebb était d'inspiration biologique, la plus grande partie des variantes (Perceptron, Rétro-propagation) sont d'inspiration mathématique. On part d'un réseau de neurones formels et on cherche à lui imposer une certaine fonctionnalité, reconnaître des caractères ou stocker des informations par exemple. La règle d'apprentissage la plus connue est la **règle Delta** développée par Widrow et Hoff :

$$\Delta w_{ij} = \eta \, (T_i(t) - OUT_i(t)) \, OUT\,(t)$$

où :

- $T_i(t)$ est l'output désiré pour le neurone i au temps t;

- w_{ij}, $OUT_i(t)$ et $OUT_j(t)$ ont la même signification que ci-dessus.

Le nom de cette règle vient de ce que l'apprentissage est proportionnel à la différence (ou delta) entre l'output effectif et l'output désiré fourni par le professeur.

IV.5.5. Mise au point de l'apprentissage

Une fois que l'apprentissage du réseau est terminé, il faut tester si celui-ci a été réalisé correctement. Nous nous limiterons ici à expliquer la mise au point ("débugging") des réseaux supervisés.

Tout d'abord, il faut retirer aux neurones de la couche de sortie la connaissance qu'ils avaient de la bonne réponse. Une façon de faire est de fixer leurs signaux de sortie égaux à 0. On donne ensuite aux neurones de la couche d'entrée des modèles semblables à ceux utilisés pendant l'apprentissage mais différents . Autrement, cela reviendrait à donner à un étudiant une copie de la feuille de questions avant l'examen. Comme nous connaissons à priori la réponse que le réseau doit fournir pour ces entrées, il est facile de vérifier sa correction.

Que se passe-t-il si le réseau n'a pas appris correctement et que la couche de sortie fournit un output incorrect ? Plusieurs possibilités s'offrent au programmeur. Il peut tout d'abord donner au neurone une **nouvelle session d'apprentissage.** Si les performances ne sont pas meilleures après cette session, on peut alors **modifier les constantes** intervenant dans la règle d'apprentissage.

On peut aussi **s'assurer que les modèles d'apprentissage et de test sont bien semblables.** Dans le cas de neurones interconnectés il faut s'assurer de la **correction des connexions** ou encore, vérifier que la structure du réseau est bien appropriée au problème.

Finalement, il faut s'interroger sur la **pertinence du choix de représentation des modèles d'entrée.** A ce propos, il est important de choisir un nombre suffisant de neurones d'entrée et un nombre adéquat de neurones de la/des couche(s) cachée(s).

Pratiquement, lorsque le réseau comporte un nombre important de neurones, il est très difficile, voire impossible de réaliser la mise au point neurone par neurone. Les modèles d'entrée ont été transformés en une représentation interne éclatée sur l'ensemble des poids des connexions. Autrement dit, il est peu probable d'arriver à supprimer des effets indésirables sans modifier ce qui, justement, produisait les effets souhaités. C'est pourquoi on s'en tient à réexaminer la règle d'apprentissage, les modèles de test ou encore le nombre de neurones, c'est-à-dire ce qui est expliqué ci-dessus.

IV.6. PROPRIETES DES RESEAUX NEURONAUX

Nous allons ici décrire les principales propriétés des réseaux neuronaux : le parallélisme massif, une mémoire non limitée, une vitesse de classification indépendante du nombre d'états mémorisés, un adressage par le contenu, une dégradation gracieuse avec des inputs erronés, une assignation par défaut de certains neurones et une généralisation spontanée.

1) Parallélisme massif

Il s'agit là d'une des propriétés les plus marquantes des réseaux neuronaux. La figure IV.10. nous montre un réseau en couches de 64*64 éléments complètement interconnecté. Le réseau peut ainsi exécuter des millions d'opérations en parallèle.

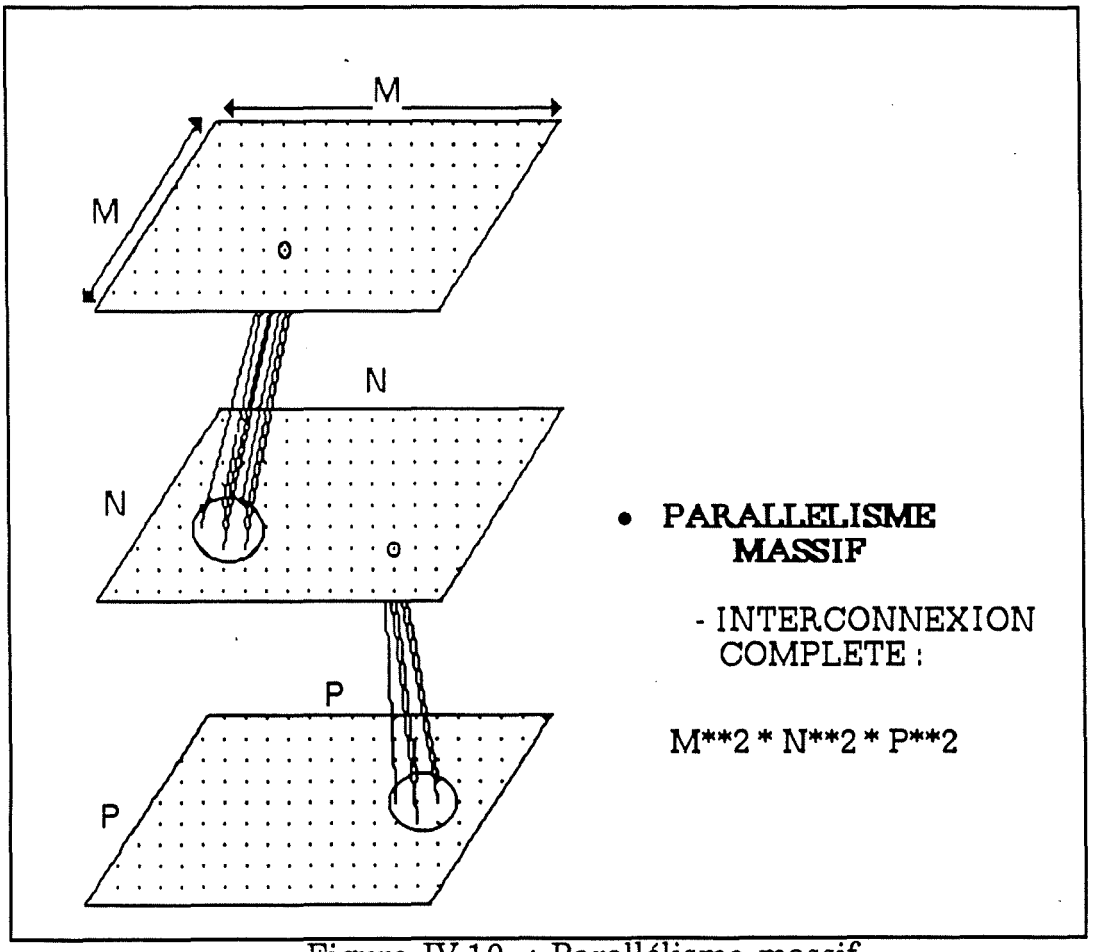


Figure IV.10. : Parallélisme massif.

2) Mémoire non limitée

La mémoire d'un réseau, comme indiqué dans la formalisation (point IV.2) est représentée par **l'ensemble des poids synaptiques** de ce

réseau. La capacité mémoire est donc directement dépendante du nombre de connexions dans le réseau mais également de la quantité d'ajustements que l'on peut faire sur l'ensemble des poids. En théorie, le nombre de réglages peut être illimité. En pratique, en revanche, on observe certains effets de bords au delà d'un certain seuil de mémorisation :

- soit par le refus d'acquérir de nouvelles informations tout en restant capable de restituer celles acquises;
- soit par une confusion provoquant une perturbation dans ce qui a été appris;
- soit par une surimpression des nouvelles informations sur les anciennes induisant l'effacement de celles-ci [Remy,87].

Si l'on reprend l'analogie des vallées, on peut dire que le nombre de vallées n'est pas limité, mais si ce dernier devient trop important, la classification du réseau n'est alors plus déterministe.

3) Classification indépendante du nombre d'états mémorisés

Chaque fois que le réseau mémorise un nouvel état, on procède à l'ajustement de ces poids et rien de plus. Le temps de classification est donc indépendant du nombre de réglages c'est-à-dire du nombre d'états mémorisés. Pour cette raison et aussi parce que le traitement effectué par le réseau est relativement court, on affirme souvent que les réseaux neuronaux travaillent en temps réel.

4) Adressage par le contenu

Une caractéristique marquante de la mémoire humaine est qu'elle est adressable par le contenu. On peut accéder à l'information mémorisée en se basant sur presque n'importe quel attribut de la représentation que l'on essaye de retrouver. De toute évidence, certains attributs sont meilleurs que d'autres parce que plus spécifiques de la représentation. Un attribut partagé par beaucoup de représentations ne nous aide pas beaucoup mais une conjonction de tels attributs nous permet d'arriver à nos fins.

Par exemple, si je demande à quelqu'un qui sort avec plusieurs filles quelle est la fille avec laquelle je l'ai vu la semaine dernière, il ne pourra probablement pas me répondre. Par contre, si je spécifie quelque chose de plus à son égard, la couleur des cheveux ou des yeux par exemple, il pourra probablement me répondre.

McClelland développa en 1981 un modèle de simulation pour illustrer cette propriété ainsi que d'autres (voir points 5,6 et 7 ci-après) [McClelland,81]. Considérons la figure IV.11., une adaptation du modèle de McClelland à des membres hypothétiques du cercle des étudiants en informatique ainsi qu'à leurs caractéristiques (prénom, filière choisie,

âge, année suivie, cours suivis avant la licence et maîtrise en informatique et fonction dans le cercle).

PRENOM	FILIERE	AGE	ANNEE	AVANT INFO	FONCT.
Bernard	Moyens logiciels. (M.L.)	22	2e	cand. éco.	reproduct. syllabus
Marc	Système d'inform. (S.I.)	22	2e	IESN	guindaille
Isabelle	S.I.	21	2e	cand. éco.	guindaille
Julien	M.L.	22	2e	cand. éco.	président
Jacques	M.L.	22	2e	cand.éco.	vice-prés.
André	M.L.	23	2e	cand. math.	trésorier
Carine	S.I.	22	3e	IESN	commiss. de contact
Sylvie	S.I.	23	3e	cand. éco.	A.G.E.

Figure IV.11. : Caractéristiques des membres du cercle informatique.

Un sous-ensemble des unités nécessaires pour représenter ces informations est présenté à la figure IV.12. Dans ce réseau, il y a une *unité d'instanciation* pour chaque membre du cercle, cette unité étant reliée par des connexions excitatrices à chacune des *unités de propriétés* représentant les caractéristiques de l'individu. Remarquons que ces unités ont été introduites aussi bien pour le nom que pour les autres propriétés. Supposons maintenant que l'on veuille retrouver les propriétés d'un individu particulier, Marc par exemple. S'il n'y a qu'un individu qui s'appelle Marc, ce qui est le cas, il n'y aura qu'une unité d'instanciation qui sera activée. Cette unité va à son tour activer les autres unités auxquelles elle est reliée par une connexion excitatrice, à savoir les propriétés de Marc.

On peut évidemment retrouver un individu en partant de plusieurs caractéristiques. Par exemple, qui a fait un graduat à l'IESN et est en 3ième licence ? En activant les unités IESN et 3ième licence, il ressort qu'il n'y a qu'une personne Carine qui répond à la description. Son unité d'instanciation sera donc activée, ce qui aura à son tour pour effet d'activer les autres propriétés de Carine.

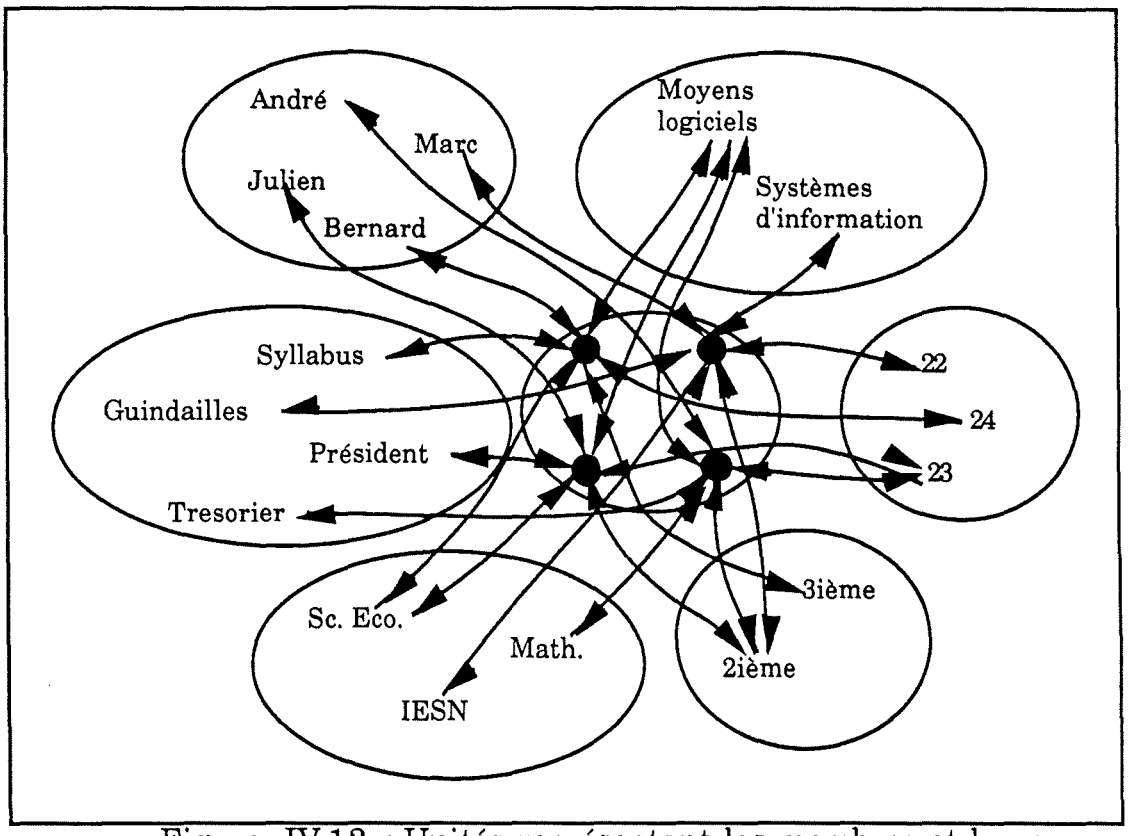


Figure. IV.12. : Unités représentant les membres et leurs caractéristiques.

5) Dégradation gracieuse

N'importe quel ensemble de caractéristiques qui est suffisant pour identifier un membre du cercle informatique activera plus fortement l'unité d'instance correspondante que toute autre unité. Si maintenant des données contiennent des caractéristiques inexactes, celles-ci vont provoquer l'activation de l'unité d'instance qui correspond le mieux. Le signal d'activation sera moins fort que s'il n'y avait pas d'erreur dans les caractéristiques de départ, mais le réseau fournira quand même la bonne réponse. Pour autant bien sûr que les caractéristiques erronées ne soient pas celles d'un autre membre du cercle, auquel cas ce serait l'unité d'instance de cet autre membre qui serait activée.

Cette sorte de modèle permettant de traiter des données incomplètes ou partielles ne requiert pas de plan de recouvrement d'erreur. Celui-ci est un dérivé naturel du réseau qui est capable de dégradation gracieuse.

6) Assignation par défaut

Le pattern d'activation ne se limite pas aux propriétés de l'individu que l'on essaye de retrouver. Une fois ses propriétés activées, elles vont à leur tour activer les unités d'instance auxquelles elles sont connectées

(les autres membres du cercle) qui vont à leur tour activer leurs unités de propriétés. Le modèle peut donc remplir les propriétés de certains individus en se basant sur ce qu'il connaît à propos d'autres individus.

Pour illustrer cela, prenons dans notre exemple du cercle informatique le cas de Bernard et supposons que l'on veuille connaître son âge. Comme il y a assez bien d'autres membres du cercle qui partagent les mêmes propriétés que Bernard, lorsque ses propriétés sont actives les autres unités d'instance sont partiellement activées et commencent elles aussi à activer leurs unités de propriétés, celle d'âge notamment. Comme les membres du cercle qui partagent le plus de propriétés avec Bernard ont en majorité un âge égal à 22 ans, c'est l'unité correspondant à cet âge qui sera la plus activée, ce qui dans notre exemple est correct. Il n'y a évidemment aucune raison que cela soit la bonne réponse mais, en général, si deux choses sont similaires sur ce que nous connaissons d'elles, il est probable qu'elles soient similaires sur ce que nous ne connaissons pas d'elles. Le modèle implémente cette heuristique.

7) Généralisation spontanée

Le modèle proposé a la propriété de retrouver ce qui est commun à des unités qui correspondent à une caractéristique donnée. Par exemple, on pourrait tester le modèle en activant l'unité correspondant à "la filière moyens logiciels". Cette unité va activer partiellement toutes les instances de la filière, provoquant elles-mêmes l'activation partielle de chacune de leurs propriétés. De cette manière, le modèle peut retrouver les valeurs typiques de chacun des membres de la filière moyens logiciels. Sur les quatre membres recensés, tous sont en 2ième licence, 3 viennent de candidatures en sciences économiques et sociales et 3 ont 22 ans. Lorsque l'on sonde le réseau en activant l'unité "moyens logiciels", les unités correspondant à ces trois propriétés seront très activées alors que celles correspondant à la fonction dans le cercle ou le nom du membre le seront très faiblement. Remarquons de nouveau que cette généralisation spontanée est un sous-produit du réseau et qu'il n'est nul besoin de la stocker explicitement.

IV.7. REPRESENTATION DES DONNEES

Nous savons que les informations sont stockées dans les poids des connexions entre neurones mais nous ne savons pas encore la forme que peut prendre la représentation interne d'un concept dans un réseau. Deux modélisations opposées sont envisageables.

La première, celle du **modèle ponctuel** prône la **représentation locale** ou **compacte** : à chaque concept est associé un neurone.

Cette approche est celle adoptée dans l'exemple des membres du cercle informatique. On a associé à chaque individu et chaque caractéristique d'individu une unité de traitement.

La seconde considère que chaque concept est représenté par l'activité de beaucoup d'unités et que chaque unité contribue à la représentation de beaucoup de concepts. Ce modèle est connu sous le nom de **modèle holographique** de la mémoire ou encore sous le nom de **représentation entièrement distribuée.**

Jérome Feldman a montré qu'aucune de ces deux approches ne reflétaient la réalité biologique [FELDM,86]. La réalité se situerait plutôt à mi-chemin entre le modèle ponctuel et le modèle holographique; ce serait plutôt une représentation des concepts largement distribuée mais pas entièrement.

Quelque soit l'optique choisie, l'objectif reste de faire apprendre au réseau les poids synaptiques adéquats de sorte que le bon pattern d'activation soit produit dans les bonnes circonstances.

IV.8. TACHES EXECUTEES PAR UN RESEAU NEURONAL

Les réseaux neuronaux sont conçus pour résoudre un certain nombre de tâches dans des domaines d'application tels que la vision, le traitement de la parole ou la robotique.

Deux classes distinctes de réseaux neuronaux ont été explorées :

a) **Les modèles neurobiologiques.**

Ils modélisent certains aspects du cerveau humain ou du comportement humain ou animal. Ces modèles sont évalués sur base de leur fidélité dans la reproduction de caractéristiques neurologiques et psychologiques ainsi que sur base de la précision de leurs prédictions.

b) **Les modèles de calcul.**

Ils réalisent certaines fonctions techniques importantes. Ces modèles sont évalués sur base de leurs performances et de leur efficience d'implémentation. Nous avons choisi de ne considérer que ce deuxième type de réseau suite à leur importance technique potentielle.

La plupart des chercheurs concentrent donc leur attention sur les modèles de la catégorie b, qui peuvent exécuter les tâches suivantes [DAPRPA,88] :

(1) Classification, mémoire hétéro-associative;

(2) Formation de catégories;
(3) Mémoire auto-associative;
(4) Traitement des données sensorielles;
(5) Problème de calcul;
(6) Correspondance non-linéaire;
(7) Automate multi-détecteurs.

Nous allons maintenant montrer en quoi consiste ces tâches.

1) Classification ou mémoire hétéro-associative.

Les classifieurs sont entraînés avec supervision et utilisent des données étiquetées pour partitionner l'ensemble des inputs en un nombre pré-spécifié de classes. A chaque classe formée est associé un élément appelé **exemplaire de classe** (cfr figure IV.13.) et considéré comme l'élément le plus représentatif de la classe. Les classes peuvent représenter différents mots pour un appareil de reconnaissance de la parole ou différents objets pour un classifieur visuel d'images. Par exemple, si le réseau reconnaît des caractères, on aura une classe pour chaque lettre de l'alphabet. Chaque classe représentant par exemple les différentes façons d'écrire la lettre et l'exemplaire de classe la lettre majuscule en imprimé. Si un input est déformé par le bruit, le réseau reste capable de lui associer un exemplaire de classe proche de celui mémorisé. On parle alors d'"**interpolative network**".

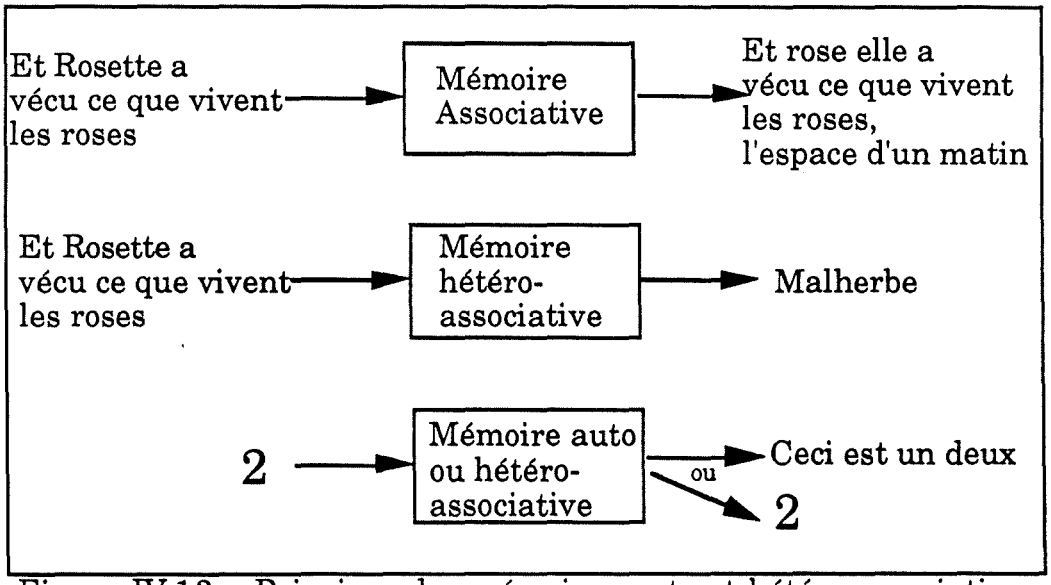


Figure IV.13. : Principes des mémoires auto et hétéro-associatives

2) Formation de catégories.

Ce type de réseau partitionne les inputs en groupes ou grappes en utilisant des données d'entraînement non-étiquetées. Le nombre de grappes formées peut être pré-spécifié ou déterminé à partir des exemples. Ces réseaux constituent une technique efficace pour réduire

l'information qui doit être traitée à des niveaux supérieurs avec peu de pertes au niveau performances. Ils font aussi un bon usage des données non étiquetées qui sont typiques de la parole et de la vision.

3) Une mémoire associative.

La mémoire associative ou **mémoire adressable par le contenu** fournit en output une représentation mémorisée complète à partir de certains éléments de cette représentation en inputs. Dans l'exemple des membres du cercle informatique (point IV.6. § 4), on pouvait retrouver toutes les propriétés d'un membre à partir de deux ou plus de ses caractéristiques. Dans un autre exemple, le réseau pourrait retourner une citation d'auteur d'après quelques mots seulement de cette citation (figure IV.13.). On parle alors d'"**accreditive network**".

4) Traitement de données sensorielles.

Une grande quantité de prétraitement en temps réel est exécutée dans les centres sensoriels périphériques de vision et d'audition. Les réseaux neuronaux peuvent réaliser cette fonction en temps réel en utilisant leur parallélisme massif.

5) Problèmes de calcul.

Des architectures de réseaux neuronaux sur mesures peuvent être conçues pour résoudre des problèmes d'optimisation, tel le problème du voyageur de commerce.

6) Correspondance non-linéaire.

Beaucoup de réseaux peuvent faire correspondre un vecteur d'inputs analogiques à un vecteur d'output en utilisant une fonction de correspondance non linéaire. Cette dernière peut être déduite d'après l'ensemble d'entraînement. Ces types de correspondances sont utiles dans beaucoup de domaines et plus particulièrement en robotique (pour déduire une courbe de commutation par exemple) et dans le traitement des signaux non linéaires. Dans notre application (cfr partie II), par exemple, nous avons appris à un bras de robot très simplifié à réaliser des flexions et extensions selon une courbe d'inertie non linéaire dans l'espace à deux dimensions. Lorsque les coordonnées cartésiennes du bras se trouvent en dessous de cette courbe, le réseau envoie un certain signal au robot et lorsque les coordonnées sont au-dessus, le réseau envoie le signal opposé.

7) Automate multi-détecteurs.

Des automates constitués de plusieurs modules complexes de réseaux neuronaux ont été construits avec une capacité visuelle et un bras de robot pour manipuler des objets dans l'espace. Ces automates démontrent comment un oeil ou une caméra peuvent apprendre à

scanner une scène en utilisant la supervision, comment la coordination d'un oeil à un bras et une main permet d'exécuter des tâches simples. Les données utilisées proviennent de l'utilisation de plusieurs détecteurs.

Différents réseaux qui permettent de traiter les problèmes énoncés ci-dessus ont été construits. Certains de ces réseaux sont présentées aux figures IV.14. et IV.15. La figure IV.14. souligne l'inspiration biologique et le type d'apprentissage utilisé et la figure IV.15. classe les réseaux d'après le type d'application.

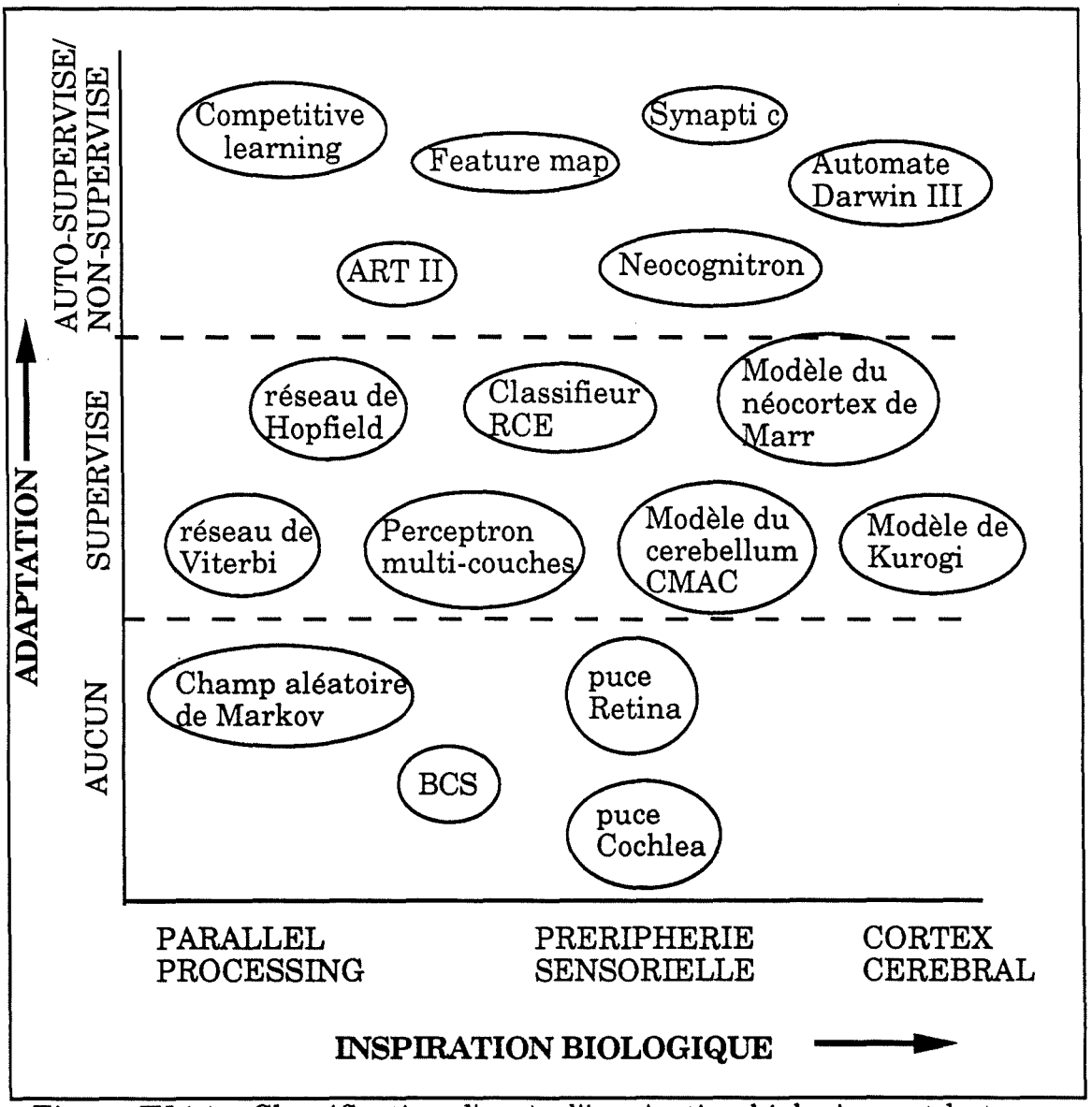


Figure IV.14. : Classification d'après l'inspiration biologique et le type d'apprentissage.

Les réseaux se trouvant dans la partie gauche de la figure IV.14. n'ont pas d'autre inspiration biologique que le parallélisme massif. Les

réseaux se trouvant au milieu ont une architecture modélisée d'après les zones de vision et d'audition, et les modèles sur la droite tentent de modéliser le cortex cérébral. Les réseaux de la partie inférieure ne subissent pas d'apprentissage, ils sont utilisés avec des poids fixes. Les réseaux se trouvant dans la partie immédiatement supérieure sont supervisés pendant l'apprentissage et ceux situés tout au dessus sont non supervisés ou auto-supervisés.

Les réseaux sans apprentissage renseignés ici sont utilisés pour le traitement des premières phases de la vision (*champ aléatoire de Markov*, "*Boundary Contour System*" ou BCS, *puce "Retina"*) et pour le traitement de la parole (*puce "Cochlea"*).

Les réseaux dans le deuxième tiers sont entraînés avec un professeur. Le *réseau de Hopfield* peut être utilisé comme mémoire associative ou pour résoudre certains problèmes d'optimisation tel le problème du voyageur de commerce.

Le *perceptron* multi-couches est entraîné avec un nouvel algorithme appelé rétro-propagation. Il a été utilisé avec succès comme classifieur dans des problèmes de vision et de reconnaissance de la parole, pour classifier des cibles repérées à l'aide d'un sonar (cfr VII. Applications), pour réaliser des correspondances non linéaires utiles dans le traitement du signal et en robotique.

Le *réseau Viterbi* est une architecture de réseau qui implémente avec succès un algorithme de décodage intégrant le temps pour la reconnaissance de la parole. Le "*reduced Couloumb energy* "(RCE) est un classifieur qui a la capacité de former rapidement des régions de décision complexes.

Le *modèle du cerebellum de Marr/Albus* , appelé *modèle CMAC*, était originellement un modèle de cerebellum, la partie du cerveau dédiée à la coordination fine des mouvements. Il est maintenant utilisé dans des applications robotiques.

Le *modèle du néocortex de Marr*, tout comme le *modèle spatio-temporel de reconnaissance de formes de Kurogi*, tente de déterminer les fonctions des couches de cellules du néocortex. Le modèle de Kurogi est un véritable modèle neurobiologique en ce sens qu'il modélise la génération de potentiels d'action d'un neurone au coup par coup. D'habitude, les modèles incluent des variables qui représentent le taux moyen de tir.

PATTERN CLASSIFICATION	RECONNAISSANCE DE LA PAROLE	VISION ARTIFICIELLE	ROBOTIQUE	TRAITEMENT DU SIGNAL	OPTIMISATION/ CALCUL
Adaline	préprocesseur de la parole Martin	BCS/FCS	modèle du cerebellum CMAC	Adaline	
ART	Masking field	Automate cellulaire	Darwin III	Perceptron multi-couches	Cellulaire
machine de Boltzmann	Perceptron multi-couches	modèles connexionnistes	Infant		Markov Random Field
apprentissage compétitif	Silicon Cochlea	Darwin II	Perceptron multi-couches		réseau de Hopfield
réseau de Kohonen	Synaptic Triad	réseau High-Order	réseaux Tensor		réseau Winner-Tak-All
réseau de Hamming	réseau Time-Concentration	Markov Random Field	Topographic Maps		
réseau Perceptron multi-couches	réseau Time-Delay	Neocognitron			
	TRACE	réseau paramètre			
	réseau de Vertibi	Silicon Retina			

Figure IV.15. : Classification d'après le domaine d'application.

La troisième partie de la figure IV.14. représente les modèles qui sont entraînés sans supervision ou avec auto-supervision. Les trois réseaux de gauche forment des grappes à partir des inputs qui leur sont présentés. Le "*competitive learning network*" travaille avec des inputs binaires alors que le "*Kohonen's feature map*" utilise des inputs continus. Ce dernier réseau utilise une grille de neurones à deux dimensions. Les neurones dans un même voisinage répondent à des stimuli ayant les mêmes caractéristiques. L'"*adaptive resonance theory clustering algorithm*" forme une nouvelle grappe lorsqu'un input est trop loin d'une grappe existante.

Le "*synaptic triad network*" est le seul réseau autre que le réseau Vertibi à intégrer le temps pour traiter des séquences de patterns. Il peut être utilisé dans les domaines de vision ou de reconnaissance de la parole. Le *Neocognitron* est un réseau multi-couches qui reconnaît les caractères écrits à la main. Il présente la propriété d'invariance après une translation des caractères dans le plan. Le *Darwin III* est un automate complexe qui permet de suivre une cible mobile et de la toucher via un bras articulé. Ce modèle est la première instanciation d'une nouvelle théorie du cerveau appelée "Darwinisme Neuronal".

IV.9. AUTRES ASPECTS DE L'APPRENTISSAGE

Avant d'explorer en détails les principaux modèles exposés ci-dessus, nous allons revenir sur l'apprentissage et tenter de voir de quelle stratégie il procède. Il existe une taxonomie des stratégies d'apprentissage développée par Jaime G. Carbonell, Ryzard S. Michalski et Tom M. Mitchell ([MICHA,83], [MICHA,86]). Nous allons passer brièvement en revue les différentes composantes de cette classification et tenter de situer les réseaux neuronaux par rapport à chacune d'elles.

Les stratégies d'apprentissages sont ordonnées selon le degré d'inférence réalisé par le système sur les informations disponibles. Plus la quantité d'inférence que le système est capable de fournir augmente et plus le fardeau placé sur le professeur chargé de l'apprentissage diminue.

IV.9.1. L'apprentissage par coeur et implémentation directe des nouvelles connaissances.

Il n'y a ici aucune inférence ou d'autre transformation des connaissances acquises par le système. Les autres formes de cette méthode d'acquisition des connaissances sont : apprendre en étant programmé (le style habituel de la programmation sur ordinateur), apprendre en stockant des données (principe des bases de données).

Les systèmes neuronaux ne sont en rien concernés par ce type d'apprentissage. En effet, d'une part ces derniers transforment toujours leurs connaissances en une représentation interne et réalisent des inférences à partir des données apprises. Supposons que le système ait appris à reconnaître des caractères écrits à la main. Si on lui présente, par la suite, des caractères un peu différents de ceux faisant partie de l'ensemble d'apprentissage, le réseau va être à même d'induire quels caractères mémorisés se rapprochent le plus de ceux fournis en entrée.

D'autre part, l'apprentissage de réseau neuronal ne correspond pas au style habituel de la programmation sur ordinateur. Dans la mesure où c'est le réseau lui-même qui se crée son propre algorithme de travail pendant la phase d'apprentissage, le rôle essentiel du programmeur consiste à choisir le type de réseau adéquat, la règle d'apprentissage à utiliser et à organiser les connaissances qu'il faut présenter.

IV.9.2. L'apprentissage par instruction.

Il consiste en l'acquisition de connaissances provenant d'un livre ou d'un professeur. Le système accepte des connaissances qu'il transforme en une représentation interne et qu'il stocke ensuite. Il combine ces nouvelles connaissances avec celles antérieures pour pouvoir les utiliser de manière effective. Le système réalise donc quelque inférence mais le gros du travail est supporté par le professeur qui présente et organise les connaissances de façon à augmenter progressivement les connaissances du système.

La méthode d'apprentissage par instruction fait référence aux méthodes formelles d'éducation. Elle suggère qu'un professeur organise un ensemble de connaissances et qu'il les présente à l'élève d'une manière qui augmente graduellement ses connaissances. La notion d'instruction est intimement liée à un ordonnancement des connaissances à transmettre ou à acquérir. Il est possible de concevoir l'apprentissage d'un réseau neuronal conformément à une instruction; leur utilisation en tant que système expert en est la preuve. Dans cette optique, le réseau apprend au fur et à mesure les différents faits et règles, organisés par le programmeur, et spécialise ainsi ses connaissances dans le domaine d'expertise considéré.

IV.9.3. L'apprentissage déductif.

Par ce type d'apprentissage, le système tire des inférences à partir des informations qu'il possède et stocke les conclusions utiles.

L'utilisation des réseaux neuronaux en tant que systèmes experts nous permet d'affirmer qu'ils réalisent des déductions à partir des connaissances stockées. Comme ces déductions sont réparties sur une

multitude de poids de connexion, au même titre que les connaissances dont elles proviennent, on peut affirmer qu'il s'agit d'un apprentissage déductif.

IV.9.4. L'apprentissage par analogie.

Il consiste à acquérir de nouveaux faits ou savoir-faire en transformant et augmentant les connaissances existantes. Cet apprentissage exige plus d'inférence que les trois précédents : le système doit retrouver dans sa mémoire un fait ou un savoir-faire analogue, le transformer, l'appliquer à la nouvelle situation et le stocker.

En ce qui concerne l'apprentissage par analogie, il est implicite dans la modélisation neuronale car la règle utilisée lors de l'apprentissage est fixée dès le départ et reste la même durant toute cette période. Le choix de cette règle est du ressort du programmeur et non du réseau. On peut affirmer que le système apprend par analogie mais indépendamment de sa "volonté". Une façon de remédier à cela serait de laisser au réseau le choix de sa règle d'apprentissage, ce qui, à l'heure actuelle, n'est pas possible.

IV.9.5. L'apprentissage par induction.

L'apprentissage par induction est subdivisé en l'apprentissage à partir d'exemples et l'apprentissage par observation et découverte.

IV.9.5.1. L'apprentissage à partir d'exemples.

Etant donné un ensemble d'exemples et de contre-exemples d'un concept, le système induit la description générale du concept qui reprend tous les exemples positifs et aucun contre-exemple. La source de ces exemples peut être un professeur, le système lui-même ou l'environnement extérieur. L'apprentissage stimulus-réponse peut aussi être classé dans cette catégorie.

C'est dans l'apprentissage à partir d'exemples que l'on trouve la meilleure définition de l'apprentissage neuronal. Lorsque le programmeur veut instruire son réseau, il le fait généralement en lui fournissant une grande quantité de d'exemples. Les exemples sont généralement présentés de façon progressive de telle sorte que l'apprentissage soit incrémental. De cette manière, le réseau pourra construire la meilleure abstraction possible du concept et ainsi tenter de répondre à des situations non rencontrées durant l'apprentissage.

IV.9.5.2. L'apprentissage par observation et découverte, encore appelé apprentissage non supervisé.

Le système cherche sans l'aide d'un professeur les régularités et les règles générales qui expliquent la majorité des observations. Cette forme d'apprentissage requiert de la part de celui qui en fait l'objet plus d'inférence que toutes les autres formes précédemment évoquées. En effet, celui qui apprend ne reçoit pas un ensemble d'instances relatives à un concept particulier mais plutôt des observations incluant plusieurs concepts à la fois.

L'apprentissage par observation et découverte ou apprentissage non supervisé ne doit pas être confondu avec l'apprentissage non supervisé qui a été développé pour les réseaux neuronaux. Dans le dernier cas, la non supervision fait référence au fait que le réseau n'a pas la connaissance du résultat qu'il doit produire, plutôt qu'au fait qu'il n'y a personne pour organiser des observations et qui plus est peuvent être relatives à différents concepts. Autrement dit, la différence réside dans le fait que, dans le cas des réseaux de neurones, le professeur ne porte pas à la connaissance du réseau le résultat désiré, alors que dans le cas de l'observation proprement dite, il n'y a pas de professeur du tout.

V Comparaison avec d'autres systèmes de traitement de l'information

Nous avons vu que les réseaux neuronaux sont des outils surtout orientés vers les tâches de vision, de reconnaissance de la parole et de robotique. Mais ce ne sont pas là les seuls systèmes de traitement de l'information, loin s'en faut. Nous avons repris une étude comparative concise [DARPA,88] qui compare les architectures des ordinateurs parallèles, des ordinateurs analogiques et des réseaux neuronaux, ainsi que leur aptitude à implémenter le mieux possible l'algorithme pour lequel ils sont conçus (figure V.1.).

	Ordinateur parallèle	Ordinateur analogique	Réseau neuronal
Correspondance entre architecture et algorithme	Pose problèmes	Bonne	Parfaite
Représentation des données	Digitale	Analogique	Analogique
Type d'élément de traitement	UAL	Analogique	Analogique simple
Elément de traitement simple ou multi-fonctions	Multi-fonctions	Simple	simple
Temps : continu ou discret	Discontinu	Continu	Disc. ou cont
Parallélisme : échelle	Des milliers	Centaines	1 à 10**6
Stockage des données	"Bank"	Analogique	ds chq élém.
Adaptabilité de l'élément	Non	Non	Oui
Communication : liens	Msg	Valeurs	Valeurs
Forme de l'algorithme	Programme	Cablé	Cablé + "training"

Figure. V.1. : Comparaison de différents systèmes de traitement de l'information.

Sous le critère "type de l'élément de traitement", U.A.L. signifie que le processeur est de type "Unité Arithmétique et Logique". En ce qui concerne l'évolution des systèmes, elle peut se faire à chaque top d'horloge et le temps est alors discret, ou bien il se fait en permanence et il est continu.

Les principales différences qui ressortent de cette comparaison sont :

1- tout comme un ordinateur analogique, le réseau neuronal est une implémentation directe, parfaite, c'est-à-dire la plus efficiente qui soit de l'algorithme qu'il veut réaliser;

2- le réseau neuronal est basé sur un petit nombre d'éléments de traitement simples et de types différents;

3- il est composé d'éléments adaptatifs qui peuvent chacun stocker de l'information;

4- il possède un parallélisme massif qui, dans certains cas (10^6 interconnexions) est 1000 fois supérieur aux meilleures architectures existantes (plusieurs milliers de processeurs) pour un algorithme donné;

5- le réseau neuronal implémente l'algorithme via un cablage et un entraînement, ce qui permet d'ajuster les paramètres du réseau pour donner les réponses désirées et compenser les imprécisions (et les fautes) qui se produisent pendant la fabrication du hardware.

VI Taxonomie des réseaux neuronaux

VI.1. INTRODUCTION

Ce chapitre a pour but de donner différents modèles de classification des réseaux neuronaux, ainsi que de passer en revue les réseaux les plus courants. On y trouvera tout d'abord quelques définitions suivies des principaux modes de classification. Enfin, les principaux réseaux existants seront développés plus en détails.

VI.2. DEFINITIONS

Classeur

Un classeur permet de déterminer laquelle parmi M classes est la plus représentative d'un modèle d'entrée contenant N éléments. Il peut être utilisé soit comme mémoire associative, soit comme "vector quantizer" ou "cluster". Dans le premier cas, deux fonctions sont à distinguer: la fonction auto-associative et la fonction hétéro-associative. En tant que "cluster", il permet de regrouper les N éléments d'entrée en M grappes (classe ou "cluster"). Cette fonction est intéressante pour les systèmes de transmission d'images ou de sons. [LIPPM,88].

Score de correspondance

Le score de correspondance est la valeur qui indique la différence entre chacune des M classes et le modèle d'entrée.

VI.3. TAXONOMIES

Une première taxonomie consiste à classer les différents modèles de réseaux neuronaux suivant les règles d'apprentissage qu'ils utilisent et les architectures dans lesquelles ces règles sont utilisées.

Quatre classes de règles d'apprentissage peuvent être définies :

1. corrélationnelle

Les poids des synapses des neurones sont ajustés suivant la règle de Hebb qui stipule que le poids entre deux neurones doit être renforcé si ceux-ci sont activés simultanément. Les règles de cette classe sont utilisées dans beaucoup d'architectures et peuvent être adaptées pour des procédures d'apprentissage supervisées ou non.

2. compétitive

Cette classe concerne les réseaux à deux couches de neurones. D'une part la couche d'entrée et d'autre part la couche de sortie. Les neurones de sortie concourent jusqu'à ce que l'un d'eux domine les

autres. Dès lors, les poids du dominant sont modifiés suivant la règle de Hebb ou une règle dérivée de celle-ci. Cette procédure, non supervisée, a pour effet de grouper les données d'entrée en différentes classes dont les éléments ont à peu près les mêmes propriétés. Elle est utilisée dans les modèles de Carpenter/Grossberg (Adaptive Resonance Theory).

3. correction d'erreur

Cette méthode de correction d'erreur calcule la différence entre les sorties du système et les sorties désirées. Elle minimise ensuite ces erreurs en respectant, pour chacun des poids, un ensemble de règles de changement. Cette méthode est souvent utilisée, notamment dans les modèles Perceptron, Madaline et Rétro-propagation.

4. stochastique

Cette méthode utilise une approche stochastique de l'apprentissage. Les poids sont ajustés pour minimiser une quantité stochastique similaire à la fonction entropie en thermodynamique. L'apprentissage stochastique peut découvrir par lui-même des relations entre des ensembles arbitraires de modèles. Ce type d'apprentissage est utilisé dans les modèles des machines de Boltzmann et de Cauchy. [Yoon 89].

Une seconde taxonomie consiste à classer les différents réseaux en fonction de leurs applications principales. On peut ainsi distinguer six types d'applications principales. Celles-ci sont : la **classification de modèle** ("pattern classification"), la **reconnaissance de la parole** ("speech recognition"), la **vision** ("machine vision"), la **robotique** ("robotics"), l'**analyse de signaux** ("signal processing") et l'**optimisation** ("optimization/computation") (cfr. IV.8. Tâches exécutées par un réseau neuronal).

Une troisième taxonomie peut se faire en fonction du type des connexions des réseaux étudiés. En effet, on a vu que ces connexions pouvaient être soit "**feed-forward**" soit "**feed-back**". Cette taxonomie est cependant moins usitée, car certains réseaux peuvent combiner les deux types de connexions (réseau Counterpropagation par exemple). En générale ce type de classification se retrouvera comme sous-type du dernier mode de classification développé dans ce point.

Une dernière taxonomie pourrait consister en une classification des différents modèles de réseaux neuronaux suivant leurs valeurs d'entrée (binaires ou continues) et/ou suivant le type d'apprentissage de ces réseaux (supervisés ou non-supervisés). Comme écrit ci-dessus, cette classification pourra, éventuellement, être complétée par le type des connexions présentes dans le réseau. Le figure ci-dessous (figure VI.1.) reprend ce quatrième et dernier type de classification.

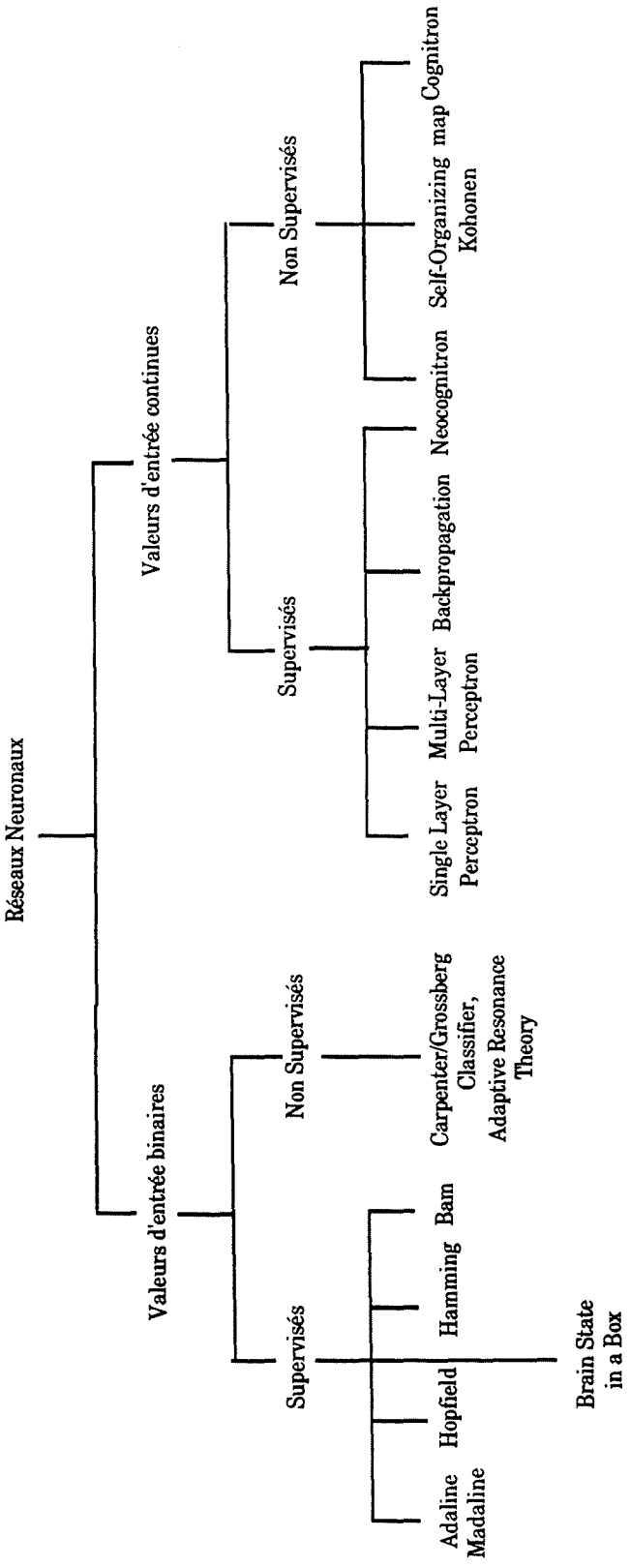


Figure VI.1. : Taxonomie de réseaux neuronaux.

VI.4. ETUDE DES PRINCIPAUX TYPES D'ARCHITECTURES

Le tableau ci-dessous (tableau VI.1.) reprend les modèles de réseaux neuronaux actuellement les plus utilisés, avec pour chacun d'eux les applications possibles, les limitations et quelques commentaires. Les réseaux marqués d'un astérisque seront développés plus en détail.

nom du réseau	auteur	dates	applicatoins premières	limitations	commentaires	(1)	(2)
Adaptive Resonance Theory *	G. Carpenter S. Grossberg	1978-86	reconnaissance de modèles, spécialement pour des modèles compliqués ou non familiés	sensible aux translations, distortions et changements d'échelle	très sophistiqué pas encore applicable à beaucoup de problèmes	b	ns
Avalanche	S. Gossberg	1967	reconnaissance continue de la parole, commande de bras de robots	pas de façon simple pour modifier la vitesse ou interpoler les mouvements	classes de réseaux; un réseau simple ne peut effectuer toutes les tâches		
Back-propagation *	P. Werbos D. Parker D. Rumelhart	1974-85	synthèse de la parole à partir de textes; contrôle adapté de bras de robots	apprentissage supervisé seulement; nécéssité d'avoir des exemples corrects d'entrées et sorties	simple à apprendre; le plus populaire	c	s
Mémoire Associative Bidirectionnelle (BAM) *	B. Kosko	1985	mémoire associative	faible densité de stockage; les données doivent être correctement codées	apprentissage facile	b	s
Machines de Boltzmann et Cauchy	J. Hinton T. Sejnowsky J. Hopkins H. Szu	1985-86	reconnaissance de modèles pour l'image, le son et les radars	Machines de Boltzmann: apprentissage long. Machine de Cauchy: génération de bruits	réseaux simples dans lesquels une fonction bruit est utilisée pour trouver un minimum global		
Brain State in a Box (BSB) *	J. Andeson	1977	extraction de données de bases de données	décisions immédiates, sans raisonnement itératif	similaire au réseau BAM	b	s
Cerebellatror	D. Mar J. Albus A. Pellionez	1969-82	contrôle moteur de bras de robots	demande des entrées de contrôle compliquées	similaire au réseau Avalanche		
Counterpropagation *	R. Hecht-Nielser	1986	compression d'images; analyses statistiques	demande un grand nombre d'éléments et de connexions	fonctionne comme une table de vérificaton auto-programmable	c	s
Hopfield *	J. Hopfield	1982	recherche de données complètes ou d'images à partie de fragments	une structure par application	peut être implémenté à grande échelle	b (c)	s
Adaline Madaline *	B. Widrow	1960-62	modems adaptatifs; équalizeurs adaptatifs dans les lignes téléphoniques	suppose une relation linéaire entre entrée et sortie	loi d'apprentissage puissante; utilisation commerciale depuis plus de 20 ans	b	s
Cognitron Neocognitror *	K. Fukushima	1978-84	reconnaisance de caractères manuscrits	demande un grand nombre d'éléments de connexions	réseau compliqué; insensible aux différences d'échelle, translations. recon. caract. compliqués (chinois)		
Perceptron *	F. Rosenblatt	1957	reconnaissance de caractères types	ne peut reconnaitre des caractères compliqués (chinois); sensible aux différences d'échelle, aux translations et distortions	le plus vieux modèle; rarement utilisé aujourd'hui	c	s
Self-Organizing Map Kohonen *	T. Kohonen	1980	transforme une région géométrique en une autre région	demande un apprentissage très long	très efficace pour le calcul de flux aérodynamiques	c	ns
Hamming *	Baum, Moody, Wilczek, Gold, Domany, Lippmann, Maipass	1986	reconnaissance de caractères; recouvrement d'entrées aléatoires	pas très utilisé; pas général	demande moins de connexions qu'un réseau de Hopfield	b	s

(1) : type des valeurs; b = valeurs binaires; c = valeurs continues.
(2) : type d'apprentissage; s= apprentissage supervisé; ns = apprentissage non supervisé.

Tableau VI.1. : Modèle de réseaux neuronaux

VI.4.1. Une règle de base : La Delta Rule

La Delta Rule inventée par B. Widrow, ou une de ses dérivées, est souvent utilisée comme règle d'apprentissage pour des réseaux

supervisés feed-forward. Cette règle minimise l'erreur du réseau entre la sortie désirée et la sortie calculée c'est-à-dire la différence.

Les poids des différentes connexions sont modifiés de la façon suivante :

$$\Delta W_{ij} = \eta (T_{pj} - OUT_{pj}) X_{pi} = \eta \delta_{pj} X_{pi};$$

avec :

- η = taux d'apprentissage; plus ce taux d'apprentissage est élevé, plus la convergence sera rapide;
- T_{pj} = sortie désirée pour le $j^{ème}$ élément du modèle de sortie du modèle p;
- OUT_{pj} = $j^{ème}$ élément de la sortie actuelle produit par la présentation du modèle p;
- X_{pi} = valeur du $i^{ème}$ élément du modèle d'entrée;
- $\delta_{pj} = T_{pj} - OUT_{pj}$;
- ΔW_{ij} = changement à faire au poids de la connexion du noeud i au noeud j suivant la présentation du modèle p.

La "Delta Rule" est efficace pour des réseaux à un niveau d'interconnexion, c'est-à-dire ayant une couche d'entrée et une couche de sortie. Par contre, elle ne permet pas de déterminer le poids des connexions d'un réseau comprenant une ou plusieurs couches cachées.

Cette règle a été légèrement modifiée ou plus exactement complétée pour devenir la "**Generalized Delta Rule**". C'est en fait une phase de retour dans le réseau. L'erreur est rétro-propagée de couche en couche et les poids sont modifiés en conséquence. [JONE,87], [RUMEL,88], [CAUDI,88-1].

VI.4.2. Hopfield

Les réseaux de Hopfield sont des réseaux récurrents, c'est-à-dire qu'il y a un feed-back de l'output vers l'input. Le réseau ci-dessous (figure VI.2.) représente un tel réseau avec une couche de neurones. La couche 0 n'effectue aucune opération. Elle ne fait que repasser les outputs en input. Les neurones de la couche 1 calculent la somme pondérée des entrées par les poids et produisent un signal NET. Celui-ci est ensuite transformé par une fonction non linéaire pour finalement donner l'output OUT.

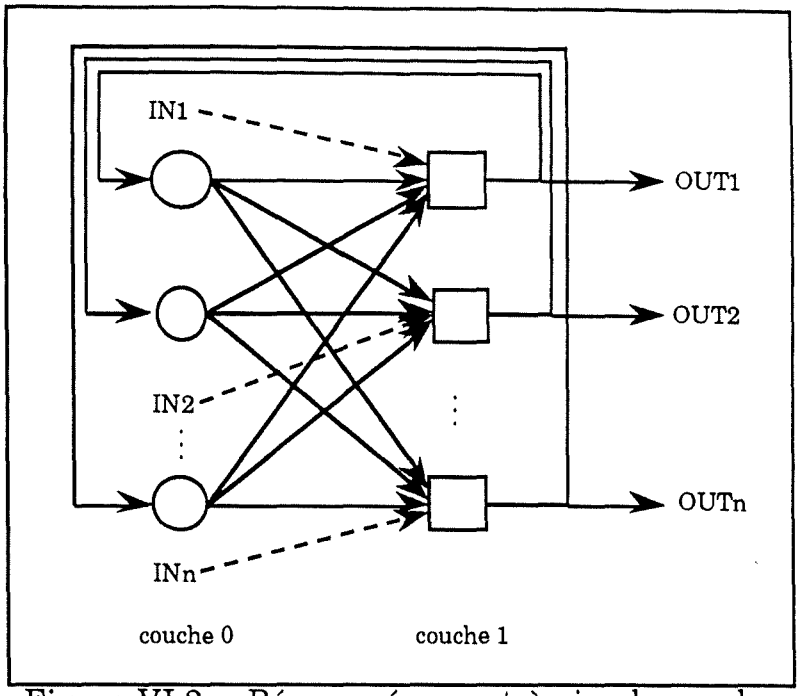


Figure VI.2. : Réseau récurrent à simple couche.

VI.4.2.1. Systèmes binaires

Les réseaux de Hopfield sont bien appropriés lorsqu'une représentation binaire exacte du problème est possible, comme pour des images noir et blanc ou des textes ASCII. Ils le sont moins pour des valeurs d'entrées continues et ce à cause du problème de conversion des valeurs continues en valeurs binaires. C'est pourquoi ces réseaux sont généralement utilisés avec des valeurs d'entrée binaires. Ils peuvent être utilisés comme mémoire associative ou pour résoudre des problèmes d'optimisation.

Dans les systèmes binaires, la fonction de transfert est une simple fonction à seuil (fonction signe ou parfois heaviside). L'output d'un neurone sera égal à un si la somme pondérée des autres outputs par les poids est plus grande que le seuil. Dans le cas contraire, cet output vaudra zéro.

Il est à remarquer ici que l'état du réseau correspond simplement à l'ensemble des valeurs d'output de chaque neurone.

VI.4.2.2. Stabilité du réseau

Cohen et Grossberg ont démontré que les réseaux récurrents sont stables, c'est-à-dire qu'ils convergent vers une solution si les poids entre les couches de neurones sont symétriques (les neurones n'étant pas reliés à eux-mêmes).

La stabilité du réseau peut être prouvée mathématiquement. Pour cela il faut trouver une fonction qui décroît chaque fois que le réseau change d'état, pour éventuellement atteindre un minimum et s'arrêter. Une telle fonction est appelée "fonction de Liapunov". La fonction ci-dessous est valable pour le réseau récurrent présenté plus haut.

$$E = \left(-\frac{1}{2}\right) \sum_i \sum_j W_{ij}\, OUT_i\, OUT_j - \sum_j IN_j\, OUT_j + \sum_j T_j\, OUT_j;$$

avec :

- E = énergie du réseau;
- W_{ij} = poids reliant le neurone i de l'output au neurone j de l'input;
- OUT_j = output du neurone j;
- IN_j = input (externe) du neurone j;
- T_j = seuil du neurone j.

Le changement d'énergie dû à un changement de l'état du neurone j est :

$$\delta E = -\left(\sum_{i \neq j} (W_{ij}\, OUT_i) + IN_j - T_j \right) \delta\, OUT_j;$$

$$= -\left(NET_j - T_j \right) \delta\, OUT_j;$$

avec :

- $\delta\, OUT_i$ = changement de l'output du neurone j;
- $NET_j = \sum_{i \neq j} W_{ij}\, OUT_i + IN_j$ (cfr. réseau présenté plus haut).

Si NET_j est supérieur au seuil, alors l'output du neurone est égal à 1 et $\delta\ OUT_j$ est donc positif ou nul. Le δE sera alors négatif ou nul et l'énergie du réseau sera décroissante ou restera constante.

Avec $NET_j \leq$ seuil, l'output du neurone sera égal à 0 ou -1. $\delta\ OUT_j$ sera donc négatif ou nul et dès lors, l'énergie du réseau sera décroissante ou constante.

A chaque changement d'état d'un neurone, l'énergie du réseau décroît ou reste constante, pour finalement arriver à un minimum et s'arrêter. Un tel réseau est dit "**stable**".

VI.4.2.3. Réseau utilisé comme mémoire associative

Présentation du réseau

Le réseau présenté ci-dessous (figure VI.3.) peut être utilisé comme mémoire associative.

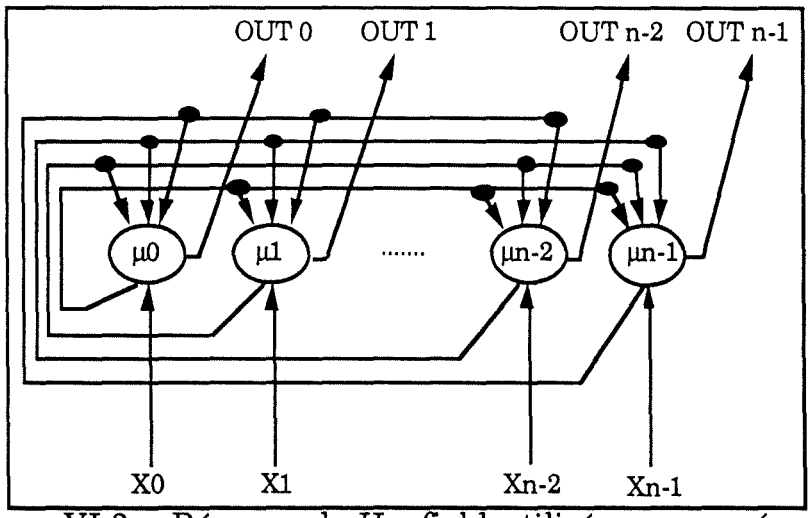


Figure VI.3. : Réseaux de Hopfield utilisé comme mémoire associative.

Ce réseau possède N neurones ($\mu_0, \mu_1, \mu_2, ..., \mu_{n-1}$) à entrées et sorties linéaires (respectivement $X_0, X_1, ..., X_{n-1}$ et $OUT_0, OUT_1, ..., OUT_{n-1}$) prenant les valeurs +1 ou -1. La sortie de chaque neurone est renvoyée à chacun des autres neurones via des poids dénotés W_{ij}. Les différentes opérations sont les suivantes :

- les poids des connexions sont fixés comme suit :

$$W_{ij} = \sum_{s=0}^{M-1} X_i^s X_j^s, \quad i \neq j;$$

$$= 0, \qquad i = j,\ 0 \leq i, j \leq n-1;$$

avec :

- M = nombre de classes;
- W_{ij} = poids de la connexion du neurone i au neurone j;
- X_i^s = $i^{ème}$ élément de l'exemplaire de la classe s;
- X_i^s = +1 ou -1;

- présentation d'un modèle inconnu :

$$\mu_i(0) = X_i, \quad 0 \leq i \leq n-1;$$

avec :

- X_i = $i^{ème}$ élément du modèle d'entrée;
- $\mu_i(t)$ = sortie du neurone i au temps t;

- itération jusqu'à convergence :

$$\mu_j(t+1) = g\left(\sum_{i=0}^{N-1} W_{ij}\,\mu i(t)\right) \qquad 0 \leq j \leq M-1;$$

avec :

- g = fonction de transfert.

Le réseau considère qu'il y a convergence lorsque les sorties ne changent plus lors d'itérations successives.

Hopfield a démontré que ce réseau converge lorsque les poids sont symétriques ($W_{ij} = W_{ji}$) ou lorsque la fonction de transfert est sigmoïde.

Lorsque le réseau de Hopfield est utilisé comme classeur, la sortie, après convergence, doit être comparée aux exemplaires des M différentes classes afin de déterminer si cette sortie correspond exactement avec un des exemplaires de classe. Si c'est le cas, la sortie sera cet exemplaire de classe. Dans le cas contraire, il n'y aura pas de sortie mais un résultat de non correspondance ("no match").

Limitations du réseau

Deux limitations importantes de ce modèle de Hopfield sont à relever. D'une part, le nombre de modèles qui peuvent être stockés et rappelés avec précision est très limité (le nombre de classes doit être

inférieur de 15 fois au nombre d'entrées). Si un trop grand nombre de modèle est stocké, le réseau risque de converger vers un modèle contrefait et différent de tous les modèles stockés. Cela conduira a un résultat de non correspondance ("no match"). D'autre part, un exemplaire de classe sera instable s'il est proche d'un autre exemplaire. Le réseau risque alors de converger vers le second exemplaire lorsque le premier lui est présenté.

VI.4.2.4. Systèmes continus

Il est possible de trouver des réseaux de Hopfield ayant une fonction de transfert continue. Dans ce cas, la fonction sera la fonction sigmoïde :

$$G(x) = \frac{1}{1 + e^{-\lambda NET}};$$

avec :

- λ = coefficient déterminant la pente de la fonction sigmoïde.

Comme pour les systèmes binaires, ce type de réseau sera stable si les poids sont symétriques, c'est-à-dire si $W_{ij} = W_{ji}$.

[DARPA,88], [DAVAL,89], [LIPPM,88], [HECHT,88], [KOLON,87], [KLIMA,88-2], [WASSE,89].

VI.4.3. Perceptron

Une distinction doit être faite entre le "Single Layer Perceptron" et le "Multi-Layer Perceptron". Ces deux types de réseaux peuvent être utilisés avec des valeurs d'entrée soit binaires soit continues.

VI.4.3.1. Single layer Perceptron

Un réseau de ce type permet de décider si une entrée appartient à une classe A ou une classe B (figure VI.4. ci-dessous).

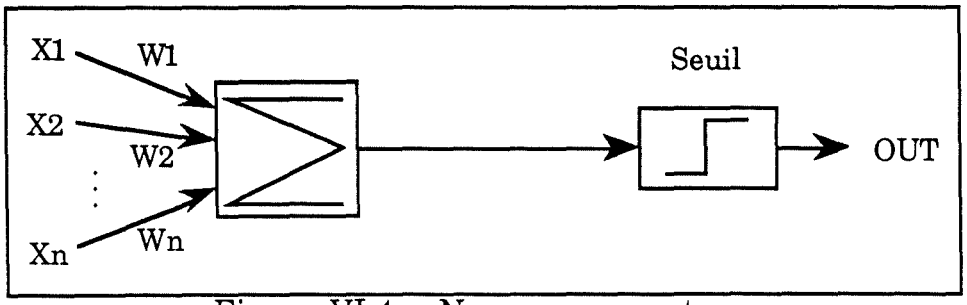


Figure VI.4. : Neurone perceptron.

$$OUT = g\left(\sum_{i=0}^{N-1} W_i X_i - \theta\right);$$

avec :

- g = fonction de transfert (cfr. IV.2.);
- θ = seuil;
- W_i = poids de la connexion i;
- X_i = i ème élément d'entrée;
- OUT = +1 => classe A
  -1 => classe B.

Ce réseau calcule la somme des entrées pondérée par les poids des connexions. Ensuite, il retire un seuil (θ) et passe le résultat à la fonction de transfert g. Le résultat est égal à +1 ou -1 et la règle de décision consiste alors à répondre "classe A" si le résultat est +1 et "classe B" s'il est égal à -1. Les poids et le seuil peuvent être fixés ou adaptés en utilisant différents algorithmes.

La procédure originelle de convergence pour le Perceptron a été développée par Rosenblatt [LIPPM,88]. Elle est fort semblable à la Delta Rule (LMS) qui peut d'ailleurs en être dérivée.

Les différentes opérations sont :

- fixer les poids des connexions et le seuil à de petites valeurs aléatoires différentes de zéro; ($W_i(t)$ et θ avec $0 \leq i \leq n-1$);
- présenter une nouvelle entrée (X_0, X_1, ..., X_{n-1}) à N éléments et la sortie désirée au réseau; une sortie est alors calculée par ce réseau :

$$OUT(t) = g\left(\sum_{i=0}^{n-1} W_i(t)\, X_i(t) - \theta\right);$$

avec :

- g = fonction de transfert;

- les poids sont adaptés si la sortie calculée ne correspond pas à la sortie désirée :

$$W_i(t+1) = W_i(t) + \eta\,\big(T(t) - Y(t)\, X_i(t)\big);$$

avec :

- η = gain, $\eta < 1$,
le gain définit l'ampleur des pas de convergence;
- T(t) = sortie désirée

Le problème avec la procédure de convergence de Rosenblatt, est que les limites de décision (séparant les deux classes) peuvent osciller continuellement. C'est notamment le cas lorsque les entrées ne sont pas linéairement séparables ou que les distributions se recouvrent (figure VI.5.).

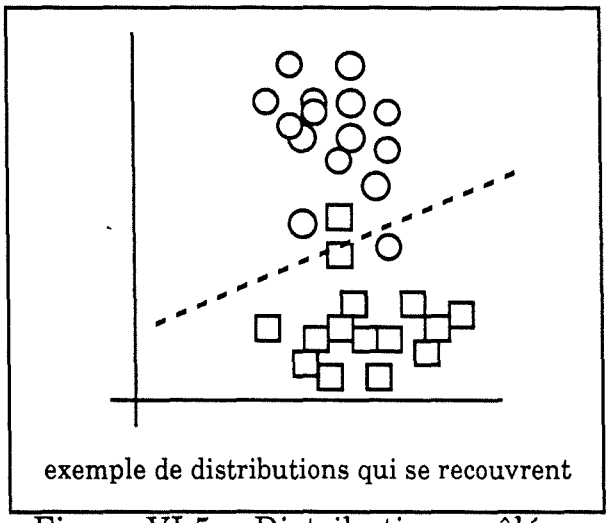


Figure VI.5. : Distributions mêlées.

Remarque :

Les régions de décision formées par la procédure de convergence de Rosenblatt sont similaires à celles formées par le "Gaussian Classifier".

L'utilisation du "Gaussian Classifier" ou de la procédure de convergence de Rosenblatt dépend de l'application. La procédure de convergence ne fait pas de supposition en ce qui concerne la distribution des classes et est plus simple que le "Gaussian Classifier". Toutefois, ni la procédure de convergence, ni le "Gaussian Classifier" ne sont appropriés lorsque les classes ne peuvent être séparées par un hyperplan. L'exemple le plus simple est celui du OU-Exclusif (figure VI.6.) qui ne peut être implémenté par un réseau Perceptron à couche unique (Single Layer Perceptron) [DAVAL,89], [LIPPM,88], [WASSE,89].

Minsky a démontré qu'un ensemble de poids permettant de converger vers les sorties désirées, pour des entrées données, n'existait que si l'ensemble des entrées était linéairement séparable. Cela signifie que dans l'espace de toutes les entrées, toutes celles qui appartiennent à un même ensemble doivent se trouver d'un même côté d'un plan, et toutes les entrées de l'autre ensemble doivent se trouver de l'autre côté de ce plan (pour rappel, le résultat sera égal à +1, -1 ou +1, 0).

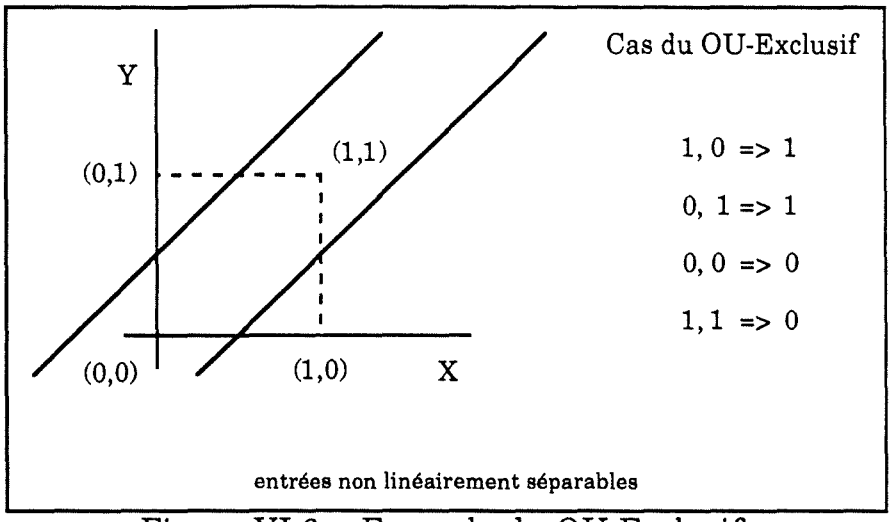


Figure VI.6. : Exemple du OU-Exclusif.

Soit un réseau Perceptron à couche unique avec deux entrées et dont le résultat de la fonction de transfert est :

- 0 si la somme des poids < seuil;
- 1 si la somme des poids > seuil;

soit NET :

$$NET = W_1X_1 + W_2X_2; \qquad (1.1)$$

avec :

- W_i = poids de la i ème connexion
- X_i = i ème élément d'entrée;

soit OUT l'output :

OUT sera égal à 0 si NET ≤ seuil
1 si NET > seuil.

Afin de déterminer les deux régions il faut égaler le seuil (S) à l'équation (1.1). On obtient alors :

$$W_1X_1 + W_2X_2 = S; \qquad (1.2)$$

ce qui devient :

$$W_1X_1 + W_2X_2 - S = 0; \qquad (1.3)$$

équation d'une droite dans le plan.

Application au cas du OU exclusif.

En remplaçant les valeurs de l'équation (1.1) par les valeurs de la table de décision ci-dessous, et en supposant le seuil égal à zéro, on obtient :

X1	X2	sorties désirées
0	0	0
0	1	1
1	0	1
1	1	0

Tableau VI.2. : Table de décision du OU-exclusif

$0 + 0 \leq 0$	sortie = 0	(1.4.)
$0 + W1 > 0$	sortie = 1	(1.5.)
$W1 + 0 > 0$	sortie = 1	(1.6.)
$W1 + W2 \leq 0$	sortie = 0	(1.7.)

D'après la table de décision les équations (1.4) et (1.7) devraient être inférieures ou égales à zéro et les équations (1.5) et (1.6) devraient être strictement supérieures à zéro. Il y a contradiction car W_1 et $W2 > 0$ et $W1 + W2 \leq 0$. Le problème du OU exclusif ne peut donc être résolu par un réseau Perceptron à couche unique. Ce problème de la séparabilité linéaire peut être résolu en introduisant des couches de neurones intermédiaires. [DAVAL,89], [LIPPM,88], [WASSE,89].

VI.4.3.2. Multi-Layer Perceptron

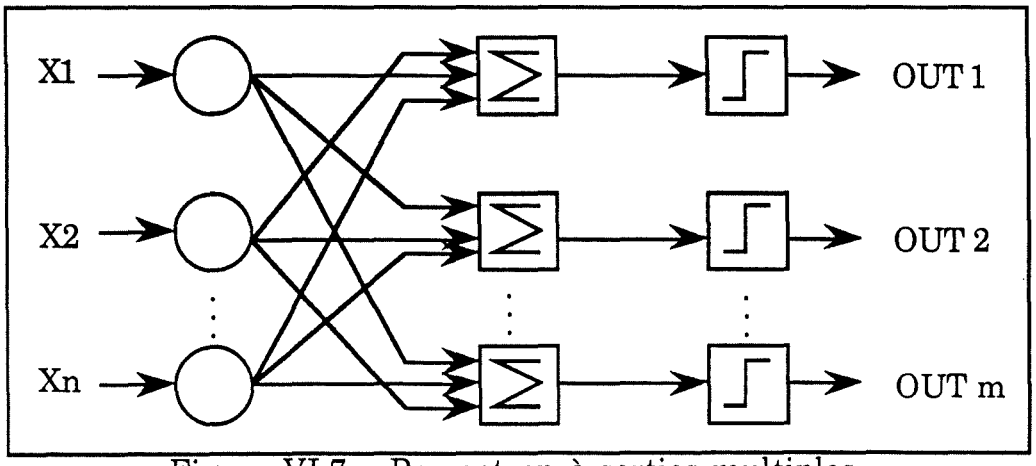


Figure VI.7. : Perceptron à sorties multiples.

Les réseaux Perceptrons multi-couches sont des réseaux avec une ou plusieures couches de noeuds entre les noeuds d'entrée et de sortie. Ces réseaux viennent à bout de la plupart des problèmes non résolus par les Perceptrons à couche unique.

La figure ci-dessous (figure VI.8.) montre les différents types de régions pouvant être formées par des réseaux Perceptron (simple ou multi-couche).

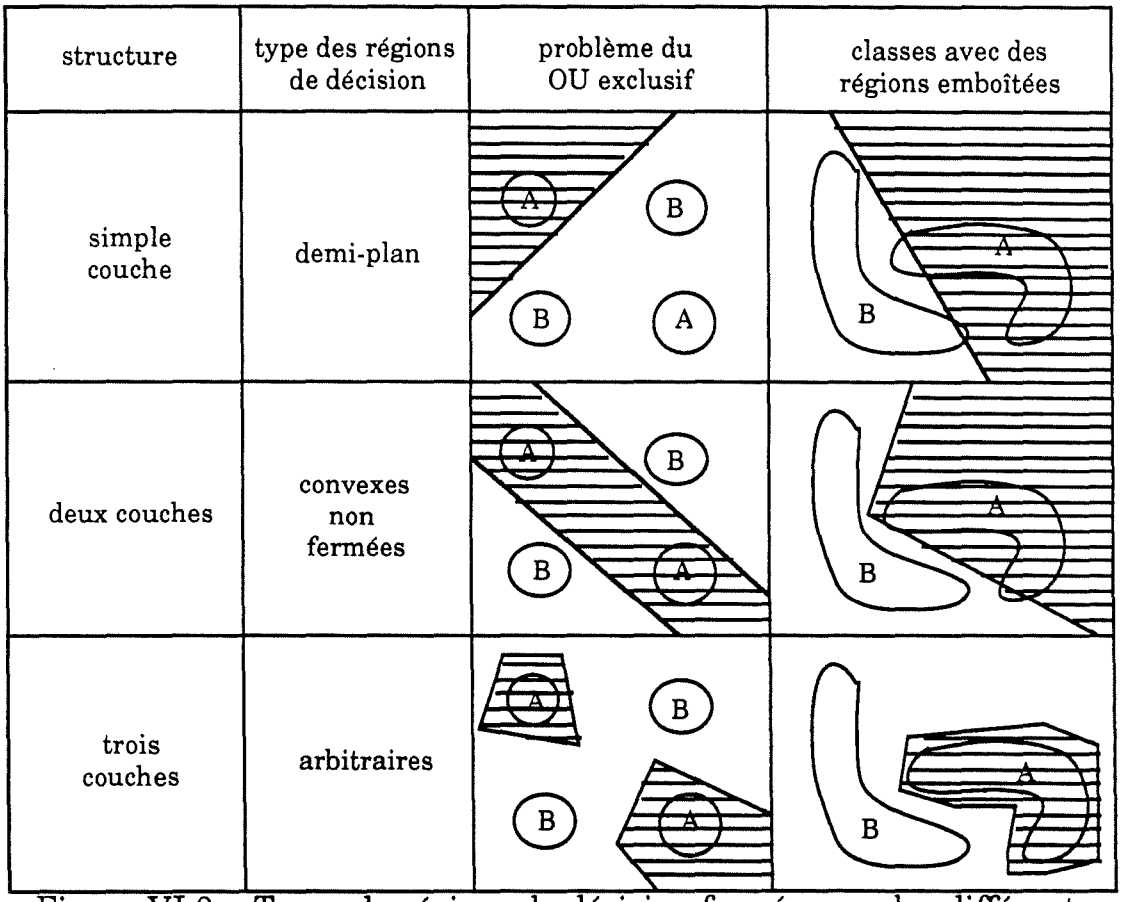


Figure VI.8. : Types de régions de décision formées par les différents réseaux Perceptron.

Un réseau Perceptron à couche unique forme des demi-plans comme régions de décision. Un réseau à deux couches forme des régions de décision convexes non fermées (convexe signifiant ici que toute ligne reliant deux points en bordure de la région passe par des points de cette région).

Les régions convexes sont formées par l'intersection des régions de décision (demi-plan) formées par chaque neurones du niveau intermédiaire du réseau multi-couche. Chacun de ces neurones se comporte comme un réseau Perceptron à couche unique. Si les poids

reliant un neurone de sortie aux N neurones du niveau intermédiaire sont égaux à 1, et si le seuil de la fonction de transfert de ce neurone est égal à N - ε avec $0 < \varepsilon < 1$, alors le neurone de sortie n'aura une sortie "élevée" (+1) que si toutes les sorties des neurones du niveau intermédiaire sont "élevées". Cela revient à effectuer une opération logique ET dans le neurone de sortie, et la région de décision résultante correspond à l'intersection de toutes les régions demi-plan, formées par les neurones du niveau intermédiaire. Cette fonction logique ET n'est pas exclusive. Elle pourrait être remplacée par une autre fonction logique à l'exception du OR-Exclusif ou du NOR-Exclusif.

Un réseau à trois couches ne dessine pas des régions convexes car les neurones de la troisième couche reçoivent comme entrée un groupe de polygones convexes. Donc, quelque soit la complexité de la région de décision, il n'est pas nécessaire d'avoir un réseau de plus de trois couches, car un tel réseau permet de dessiner des régions de toutes les formes. [DAVAL,89], [HECHT,88], [KLIMA,87-2], [LIPPM,88], [WASSE,89].

VI.4.3.3. Quelques problèmes

D'un point de vue pratique, il n'est pas toujours aisé de travailler avec des réseaux Perceptron. Plusieurs problèmes non négligeables sont à relever :

- il est difficile de voir si l'ensemble d'apprentissage choisi est satisfaisant, et s'il permet de résoudre le problème de la séparabilité linéaire;
- rien ne nous permet de savoir le nombre de pas nécessaires pour l'apprentissage du réseau;
- rien ne prouve, que l'algorithme d'apprentissage du Perceptron soit plus rapide que le simple essai de tous les ajustements possibles des poids du réseau. [DARPA,88], [DAVAL,89], [WASSE,89].

VI.4.4. Rétro-propagation

La figure ci-dessous (figure VI.9.) présente un neurone artificiel avec fonction d'activation.

La fonction d'activation la plus souvent utilisée, dans les réseaux backpropagation, est la fonction sigmoide produisant le signal :

$$OUT = \left(\frac{1}{1 + e\text{-}NET} \right);$$

Cette fonction est dérivable, ce qui donne : $G' = \frac{\delta Out}{\delta Net} = Out\ (1 - Out)$.
Cela permet de minimiser l'erreur de l'output lors de l'apprentissage.

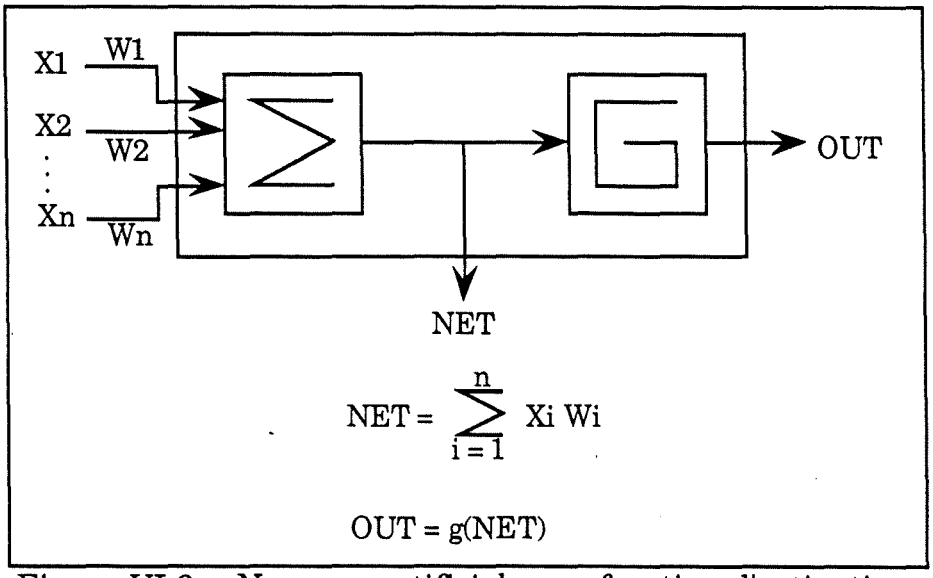


Figure VI.9. : Neurone artificiel avec fonction d'activation.

L'algorithme d'apprentissage est le suivant :

- 1. sélectionner la prochaine entrée et la sortie désirée correspondante et la présenter au réseau;
- 2. calculer l'output du réseau;
- 3. calculer l'erreur entre la sortie du réseau et la sortie désirée;
- 4. ajuster les poids du réseau de façon à minimiser l'erreur;
- 5. répéter les pas 1 à 4 pour chaque entrée de l'ensemble d'apprentissage, et ce jusqu'à ce que l'erreur soit suffisamment faible pour tout l'ensemble.

Les calculs sont effectués couche par couche. Lorsque le réseau a appris, il est utilisé sans que les poids ne changent.

Les étapes 1 et 2 correspondent à ce que l'on peut appeler la "forward pass", et les étapes 3 et 4 à la "reverse pass".

VI.4.4.1. Forward pass

La sortie de chaque neurone est calculée en faisant la somme des inputs pondérée par les poids des entrées. Cette somme (NET) est ensuite transformée à l'aide de la fonction d'activation g, ce qui donne l'output d'un neurone. L'ensemble des outputs des neurones d'une couche constitue l'ensemble des entrées de la couche supérieure.

En terme de vecteurs cela donne :

- X : vecteur d'entrée;
- W : vecteur des poids du réseau;
- le vecteur NET pour une couche N peut être exprimée comme le produit de X et W, ce qui donne N = XW;
- le vecteur d'output est obtenu en appliquant la fonction d'activation au vecteur NET composant par composant. OUT = g(XW).

VI.4.4.2. Reverse pass

Ajuster les poids de la couche d'output

L'apprentissage de la couche output est relativement aisée car il existe un output désiré.

La différence entre l'output désiré et l'output produit par le réseau est calculée, pour chacun des neurones de la couche output. Cela produit un signal d'erreur, pour chaque neurone de la couche, qui est alors multiplié par la dérivée de la fonction d'activation, soit OUT(1 - OUT), calculée pour cette couche. On obtient donc un δ pour chaque neurone :

$$\delta = \text{OUT}(1-\text{OUT}) \text{ (output désiré - OUT).}$$

Si nous prenons un neurone q de la couche output, ce δ est ensuite multiplié, pour chacun des poids du neurone, par l'output du neurone de la couche inférieure. Ce produit est à son tour multiplié par un taux d'apprentissage η (choisi habituellement entre 0.01 et 1.0) et le résultat est ajouté au poids. Cette opération est répétée pour chacun des poids des connexions entre la couche d'output et la couche inférieure. Cela donne :

$$\Delta W_{pq,out} = \eta \, \delta_{q,out} \, OUT_{p,j}; \qquad (1)$$

$$W_{pq,out}(n+1) = W_{pq,out}(n) + \Delta W_{pq,out}; \qquad (2)$$

avec :

- $W_{pq,out}(n)$ = valeur du poids de la connexion reliant le neurone p de la couche inférieure, au neurone q de la couche output au pas n (avant ajustement);

- $W_{pq,out}$ (n + 1) = valeur du poids de la connexion au pas n + 1 (après ajustement);

- $\delta_{q,out}$ = valeur du δ pour le neurone q dans la couche output;

- $OUT_{p,j}$ = valeur de l'output du neurone p dans la couche cachée.

Ajuster les poids des couches cachées

Les équations (1) et (2) sont valables pour toutes les couches. Cependant, les couches cachées ne possédant pas d'"output désiré", le δ doit être calculé différemment. L'apprentissage des couches cachées se fait en rétro-propageant l'erreur de la couche de sortie vers la couche d'entrée.

Considérons un neurone p de la couche cachée se trouvant juste avant la couche output. Ce neurone est relié aux neurones de la couche d'output. Lors de la "reverse pass", l'erreur des neurones de la couche output est propagée à la couche inférieure via ces connexions. Chacun des poids, reliant notre neurone à la couche supérieure, sera multiplié par le δ correspondant du neurone output ($\delta_{q,out} W_{pq,out}$). La valeur du δ du neurone sera obtenue prenant la somme de ces produit et en multipliant le résultat par la dérivée de la fonction d'activation. Cela donne :

$$d_{p,j} = OUT_{p,j} (1 - OUT_{p,j}) \left(\sum_{q} \delta_{q,out} W_{pq,out} \right) \qquad (3).$$

Remarque :

Le calcul des erreurs des différentes couches cachées à l'**étape n** tient compte des poids et erreurs des neurones des couches supérieures à l'**étape n**.

Il faut donc : 1) calculer **toutes** les erreurs du réseaux;
2) calculer **ensuite** les **nouveaux poids** des connexions.

Il est erroné de procéder couche par couche, ce que nous faisions pendanr notre stage.

VI.4.4.3. Neurone bias

Ce neurone déplace horizontalement la fonction d'activation par rapport au système d'axe, ce qui a pour effet d'augmenter la vitesse du processus d'apprentissage. Ce neurone est connecté à tous les neurones

du réseau via des poids. Ces poids peuvent être modifiés lors de l'apprentissage, mais la source est toujours égale à +1 (fonction d'entrée affine cfr. IV.2. : Neurone Artificiel).

VI.4.4.4. Momentum

Le "momentum" permet de diminuer le temps d'apprentissage d'un algorithme "rétro-propagation" (rétro-propagation). Cette méthode consiste à ajouter un terme à l'ajustement des poids, proportionnel aux ajustements de poids préalables. Les équations d'ajustement deviennent :

$$- \Delta W_{pq,out}(n+1) = \eta \left(\delta_{q,out} OUT_{p,j}\right) + \alpha\left(\Delta W_{pq,out}(n)\right);$$

$$- W_{pq,out}(n+1) = W_{pq,out}(n) + \Delta W_{pq,out}(n+1);$$

avec :

- α = coefficient du momentum (proche de 0.9).

Cette méthode du momentum, développée par Rumelhart, Hinton et Williams, donne de bon résultat mais pas pour toutes les applications. C'est pourquoi Sejnowski et Rosenberg ont développé une méthode semblable, et meilleure pour la résolution de certains problèmes. Il faut remarquer que **rien** ne permet de dire au préalable quelle sera la méthode optimale. Les équations d'ajustement deviennent dans ce cas :

$$- \Delta W_{pq,out}(n+1) = \alpha \Delta W_{pq,out}(n) + (1 - \alpha)\ \delta_{q,out}\ OUT_{p,j}$$

$$- W_{pq,out}(n+1) = W_{pq,out}(n) + \eta \Delta W_{pq,out}(n+1)$$

avec :

- α = coefficient de lissage; souvent fixé entre 0.0 et 1.0.

VI.4.4.5. Problèmes

Malgré son succès et ses nombreuses utilisations, l'algorithme de rétro-propagation n'est certainement pas **la** solution. L'apprentissage est souvent très long (il peut facilement prendre plusieurs jours) et incertain (il est possible de ne pas obtenir d'apprentissage). [DARPA,88], [DAVAL,89], [RUMEL,88], [WASSE,89].

VI.4.5. Counterpropagation

Le réseau "Counterpropagation" a été développé par Robert Hecht-Nielsen. Comparé au réseau "rétro-propagation", il permet de

réduire le temps d'apprentissage par un facteur proche de cent, mais il n'est malheureusement pas général.

Ce réseau est une combinaison de deux autres types de réseau, à savoir le "Carpenter/Grossberg Classifier" et le "Self-organizing Map" ou "Kohonen". Ensemble, ces deux réseaux fournissent des propriétés qui ne sont pas disponibles autrement.

Afin de pouvoir travailler plus facilement, nous parlerons ici de vecteurs d'input ou d'output, mais cela ne modifie en rien le raisonnement.

Le processus d'apprentissage associe des vecteurs d'inputs avec les vecteurs d'outputs correspondants. Ces vecteurs peuvent être binaires ou continus. Lorsque le réseau a appris, la présentation d'un vecteur d'input produit le vecteur d'output correspondant. Cela est également vrai lorsque le vecteur d'input est incomplet ou partiellement incorrect. Ce type de réseau correspond donc à une mémoire auto ou hétéro-associative et est donc particulièrement destiné pour résoudre des problèmes de "pattern-recognition" ou de "pattern-completion".

VI.4.5.1. Structure du réseau

Chaque neurone de la couche d'input est connecté via des poids $W_{i,j}$ à tous les neurones de la couche supérieure, appelée "**couche de Kohonen**",). La matrice des poids sera représentée par W. Les neurones sont eux-mêmes connectés à tous les neurones de la couche supérieure, appelée quant à elle "**couche de Grossberg**", via des poids $V_{j,k}$.

La différence avec les autres réseaux réside dans le processus exécuté par les neurones de Kohonen et de Grossberg.

VI.4.5.2. Fonctionnement des couches

La couche de Kohonen

La couche de Kohonen fonctionne de la façon suivante : "le vainqueur prend tout". Cela signifie que pour un vecteur d'input donné, un et un seul neurone donne "1" comme output; tous les autres neurones de la couche donnent zéro.

Contrairement à l'hypothèse faite au début du mémoire, le réseau présenté ici utilise donc une fonction de sortie h différente de la fonction de transfert g (cfr. IV.2.). Il ne suffit pas pour un neurone de dépasser un certain seuil pour pouvoir tirer, il faut encore que sa valeur de sortie soit supérieure à celles de **tous** les autres neurones.

Le résultat de la fonction d'entrée pour chaque neurone de Kohonen est simplement la somme pondérée des entrées par les poids. Cela donne :

$$OUT'_j = OUT_j = NET_j = \sum_i X_i W_{ij};$$

avec :

- NET_j = résultat de la fonction d'entrée du neurone de Kohonen j;
- X_i = $i^{ème}$ composant de vecteur d'entrée;
- W_{ij} = valeur du poids reliant le $i^{ème}$ composant du vecteur d'entrée au $j^{ème}$ neurone de la couche de Kohonen.

Le neurone de Kohonen avec la plus grande valeur OUT "gagne". Son output OUT' est alors 1 et les outputs de tous les autres neurones de la couche valent 0.

La couche de Grossberg

Pour la couche de Grossberg, la fonction d'entrée NET est également la somme pondérée, des outputs de la couche de Kohonen, par les poids. Cela donne :

$$OUT_j = NET_j = \sum_i K_i W_{ij};$$

avec :

- OUT_j = fonction d'entrée NET du neurone de Grossberg j;
- K_i = output i de la couche de Kohonen;
- W_{ij} = valeur du poids reliant le $i^{ème}$ neurone de la couche de Kohonen au $j^{ème}$ neurone de la couche de Grossberg.

Dans la couche de Kohonen, il n'y a qu'un neurone dont l'output est différent de zéro. La seule action des neurones de la couche de Grossberg se réduit alors à donner, comme output, les poids les reliant au seul neurone, différent de zéro, de la couche de Kohonen.

VI.4.5.3. Apprentissage de la couche de Kohonen

La couche de Kohonen classe les vecteurs d'input qui sont semblables. Un même neurone de la couche de Kohonen sera activé pour des inputs similaires.

L'apprentissage de Kohonen est un algorithme d'auto-apprentissage en mode non supervisé. Il faut s'assurer que l'apprentissage sépare bien des vecteurs d'input dissemblables.

Normalisation des vecteurs d'input

Il est préférable de normaliser tous les vecteurs d'input avant de les appliquer au réseau. Cela se fait en divisant chaque composant du vecteur par la racine carrée de la somme des carrés de chacun des composants, soit :

$$X_i' = \frac{X_i}{\sqrt[2]{\sum_i X_i^2}}$$

Cela convertit un vecteur en un vecteur unité pointant dans la même direction. Pour des vecteurs d'entrée à deux dimensions, les vecteurs pointent sur un cercle de rayon égal à un. Pour des vecteurs à trois dimensions les vecteurs pointent sur la surface d'une sphère de dimension un. Ce raisonnement est valable quelque soit la dimension du vecteur d'input.

Lors de l'apprentissage de la couche de Kohonen, on calcule le produit du vecteur d'input par le vecteur de poids associé à chaque neurone de Kohonen. L'équation du changement des poids est la suivante :

$$W_{new} = W_{old} + \alpha (X - W_{old});$$

avec :

- X = vecteur d'input;
- W_{new} = nouvelle valeur du poids connectant un composant d'input au neurone vainqueur;
- W_{old} = ancienne valeur du poids;
- α = coefficient d'apprentissage inférieur à 1 et pouvant varier durant l'apprentissage.

Initialisation des vecteurs des poids

Il faut initialiser les vecteurs des poids. Mais le faire aléatoirement risque d'engendrer quelques problèmes. En effet, la répartition sera uniforme sur l'hypersphère, alors que les vecteurs d'input ne le seront très probablement pas. Cela signifie que les vecteurs de poids trop éloignés risquent de ne jamais être modifiés. Il y aura donc gaspillage.

De plus s'il y a une concentration de vecteur d'input en un endroit, il se peut alors qu'il n'y ait pas suffisamment de vecteur de poids pour permettre la séparation des inputs. Il faudrait donc pouvoir initialiser les vecteurs de poids d'après les concentrations des vecteurs d'input. Cette situation peut être approchée de différentes manières. Quatre méthodes sont possibles :

- 1) dans cette méthode appelée "convex combination method", tous les poids ont la même valeur $\left(\frac{1}{\sqrt{n}}\right)$ avec n égal au nombre d'inputs. De plus chaque composant de l'input (X_i) reçoit une valeur $\alpha X_i + \left(\left[\frac{1}{\sqrt{n}}\right][1 - \alpha]\right)$ avec α très petit et augmentant vers 1 lors de l'apprentissage. Cette méthode donne de bons résultats mais elle ralentit fortement le processus d'apprentissage;
- 2) dans cette deuxième méthode, du bruit est ajouté aux vecteurs d'input. Ceux-ci bougent alors aléatoirement. Cette méthode est plus lente que la première;
- 3) tous les vecteurs sont initialisés de façon aléatoire, mais, au début de l'apprentissage, **tous** les poids sont ajustés et non plus seulement ceux associés au neurone de Kohonen vainqueur;
- 4) dans cette dernière méthode, si un neurone gagne trop souvent ($> \frac{1}{K}$ avec K nombre de neurones de Kohonen), il atteint **provisoirement** un seuil qui a pour effet de diminuer ses chance de gagner à nouveau. Les autres neurones ont donc plus de chance de gagner.

Il est difficile, voire impossible, de trouver la meilleure des solutions.

Interpolative mode

Plutôt que de travailler avec un seul neurone de Kohonen comme vainqueur, il est possible de travailler avec un groupe de neurones. Le résultat de leur fonction d'entrée, NET, est traité comme un vecteur et celui-ci est normalisé. Cela permet de résoudre des problèmes plus complexes et d'obtenir des résultats plus précis.

VI.4.5.4. Apprentissage de la couche de Grossberg

L'ajustement d'un poids ne se fait que s'il connecte un neurone de Grossberg à un neurone de Kohonen dont l'output est différent de zéro. Cet ajustement est proportionnel à la différence entre le poids et l'output désiré du neurone de Grossberg auquel il est connecté.

$$V_{ij\ new} = W_{ij\ old} + \beta\ (Y_i - W_{ij\ old})\ K_i;$$

avec :

- K_i = output du neurone i de Kohonen;

- Y_i = composant j du vecteur d'output désiré;

- β = coefficient fixé initialement 0.1 et réduit lors de l'apprentissage.

VI.4.5.5. Le réseau Counterpropagation complet

Dans le réseau complet, des vecteurs d'input X et Y sont appliqués simultanément et le réseau produit les vecteurs d'output X' et Y' approximant X et Y. Durant l'apprentissage, X et Y sont à la fois vecteurs d'input et vecteurs d'output désiré. Une particularité intéressante réside dans le fait que, lorsqu'on applique uniquement le vecteur d'input X (le vecteur d'input Y = 0), le réseau produit à la fois X' et Y'. Le réseau Counterpropagation permet donc de générer une fonction ainsi que son inverse si elle existe. [DAVAL,89], [WASSE,89].

VI.4.6. Self-Organizing Map ou Kohonen

Le réseau de Kohonen est un réseau à couche unique (de sortie) à inputs binaires et est du type non-supervisé. Les neurones y sont fortement interconnectés . Chacun de ces neurones reçoit l'ensemble des entrées du réseau et donc également des entrées en provenance des autres neurones. Chaque entrée est connectée à chaque noeud de sortie via des connexions à poids variables (figure VI.10.).

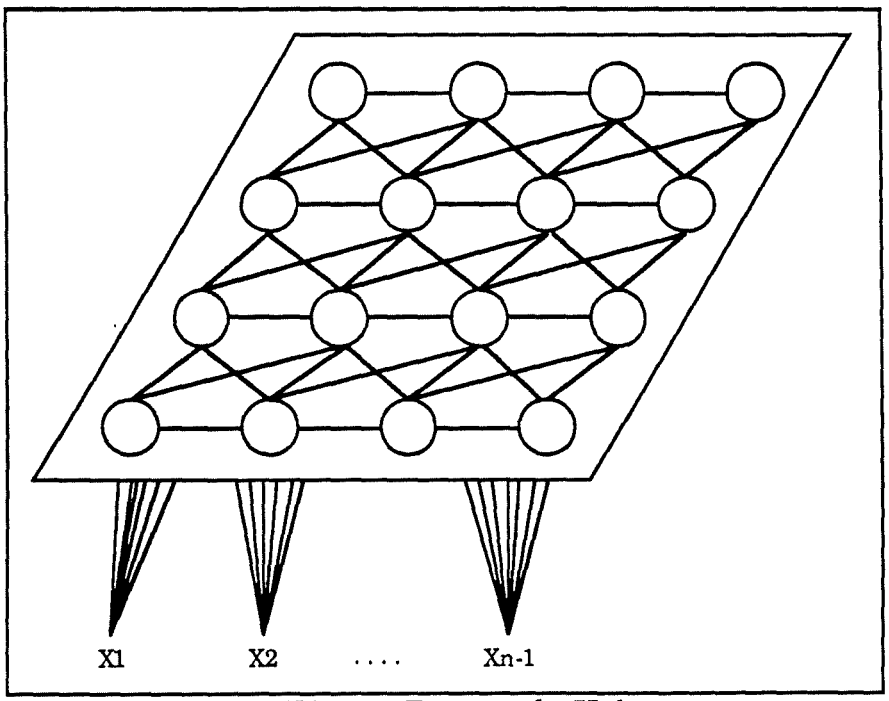


Figure VI.10. : Réseau de Kohonen.

Deux points doivent être vérifiés avant toute utilisation. Tout d'abord, il faut s'assurer que les poids des vecteurs des neurones dans la

couche sont correctement initialisés. Ensuite, il faut utiliser des vecteurs de poids et des vecteurs d'entrée qui ont été normalisés à une constante, généralement l'unité.

Les noeuds présents dans les réseaux de Kohonen ont la particularité d'être en compétition pour l'apprentissage. Chaque neurone calcule la différence entre son vecteur poids et son vecteur d'entrée. Le neurone qui obtient la plus petite différence est déclaré vainqueur de cette compétition et sera le seul à pouvoir émettre une sortie. La philosophie est "the winner takes all". En réalité, seul le vainqueur et ses voisins peuvent ajuster leurs poids.

Chaque neurone vainqueur est autorisé à réaliser son activation, laquelle agit comme une inhibition sur les sorties des autres neurones. Dans le voisinage immédiat du vainqueur, par contre, l'effet est excitateur. Cela entraîne donc que chaque neurone va essayer de gagner cette compétition. S'il voit qu'il n'y arrivera pas, il essayera d'aider son voisin à la gagner.

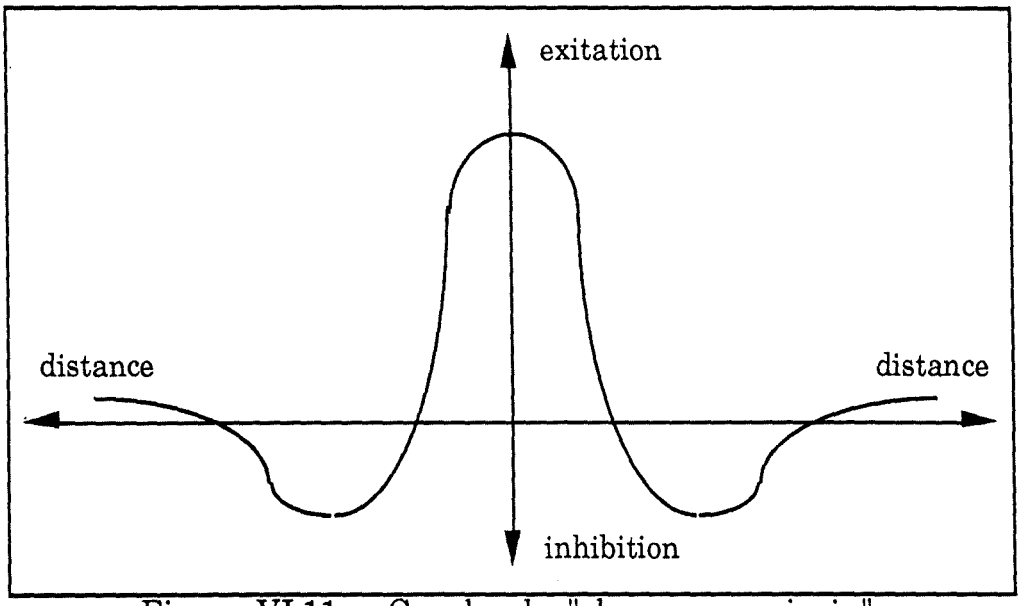


Figure VI.11. : Graphe du "chapeau mexicain".

Cette dépendance entre les neurones est représentée dans le graphe ci-dessus. Cette fonction est souvent appelée "chapeau mexicain" (figure VI.11.). L'on remarque aisément que dans la zone proche du neurone vainqueur, l'action des neurones est excitatrice. Par contre plus loin, l'action est inhibitrice, et enfin, pour les neurones les plus éloignés, l'action est négligeable.

En ce qui concerne l'initialisation des vecteurs d'input et des vecteurs de poids, ainsi que l'apprentissage, on peut se référer au réseau Counterpropagation (décrit ci-dessus), et plus spécifiquement à la couche de neurone "Kohonen".

Avant l'apprentissage, le voisinage d'un neurone peut englober l'ensemble des neurones présents dans le réseau. Pendant l'apprentissage le voisinage se réduit jusqu'à ne plus englober que le neurone lui-même. [DAVAL,89], [LIPPM,88], [CAUDI,88-2], [HECHT,88].

VI.4.7. Hamming

Le réseau de Hamming est également un réseau à valeurs d'input binaires. L'algorithme implémenté par le réseau est l'"optimum error classifier". Ce réseau est donc un "classeur". L'algorithme calcule la distance de Hamming, pour chacune des classes, entre l'exemplaire de classe et le modèle d'entrée. Cette distance est un réel. La "distance de Hamming", entre un input X à N composants et un exemplaire de classe Ej en comprenant autant, représente le nombre de composants différents entre l'input et l'exemplaire de classe.

Le graphe ci-dessous (figure VI.12.) représente un réseau de Hamming. Il est composé de deux sous-réseaux. Le réseau inférieur calcule les scores de correspondance (cfr. p. 58).

Les poids et les seuils sont d'abord fixés dans le réseau inférieur de façon à ce que les scores de correspondance générés par les sorties des neurones intermédiaires soient égaux à (N - la distance de Hamming) séparant le modèle d'entrée de l'exemplaire de classe.

Ces scores de correspondance seront donc compris entre zéro et N (nombre d'entrées), et seront plus élevés pour les neurones correspondant aux classes dont l'exemplaire se rapproche le plus du modèle d'entrée.

Les seuils et les poids sont ensuite fixés dans le réseau supérieur. Tous les seuils sont égaux à zéro et les poids reliant un neurones à lui-même sont égaux à un. Les poids entre neurones sont inhibiteurs et sont égaux à $-\varepsilon$ avec $\varepsilon = 1/M$.

Le réseau supérieur va sélectionner le neurone du réseau inférieur dont le score de correspondance est le plus élevé. Ensuite, il itérera jusqu'à ce que la sortie d'un seul neurone soit positive.

$$OUT_j(t+1) = g\left(OUT_j(t) - \varepsilon \sum_{k \neq j} OUT_k(t) \right);$$

avec :

- t = unité de temps;

- $OUT_j(t)$ = output du neurone j du sous-réseau supérieur au temps t;

- $OUT_j(t+1)$ = output du neurone j du sous-réseau supérieur au temps t+1;

- ε = poids reliant deux neurones différents du sous-réseau supérieur;

- g = fonction de transfert $g(\alpha) = \alpha$ pour $\alpha > 0$
$g(\alpha) = 0$ pour $\alpha < 0$;

- $0 \leq j, k \leq M-1$.

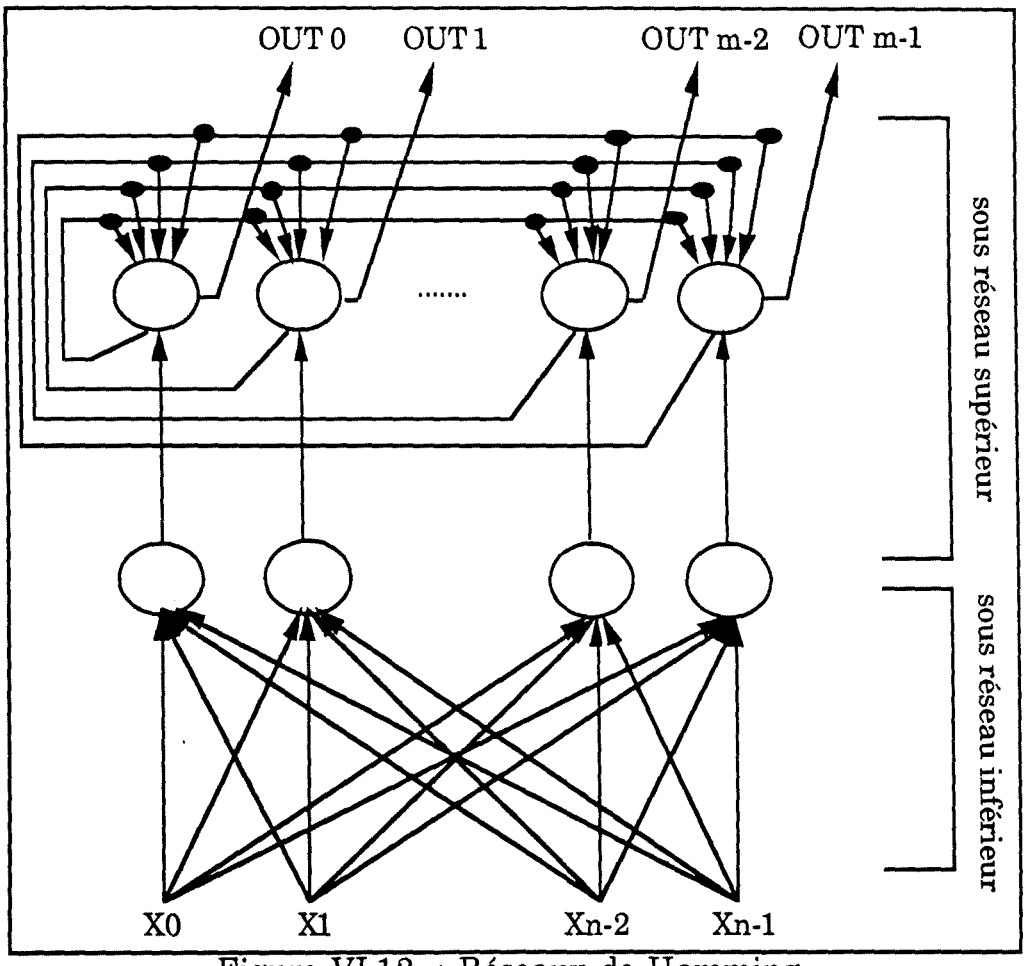


Figure VI.12. : Réseaux de Hamming.

La classe correspondant au neurone avec une sortie positive sera la classe recherchée.

Le réseau de Hamming possède un certain nombre d'avantages par rapport au réseau de Hopfield : il est plus performant pour des problèmes tels que la reconnaissance de caractères, la reconnaissance d'entrées

aléatoires ou encore le recouvrement bibliographique (retrouver un livre à partir d'une cote incomplète). De plus le réseau de Hamming demande beaucoup moins de connexions que le réseau de Hopfield. [LIPPM,88].

VI.4.8. Carpenter/Grossberg Classifier ou Adaptive Resonance Theory

Le réseau de Carpenter/Grossberg est un réseau non supervisé, à entrées binaires, implémentant un algorithme de "clustering". L'algorithme sélectionne la première entrée comme étant l'exemplaire du premier "cluster" (grappe). L'entrée suivante ("it follows the leader") est comparée à l'exemplaire de la première grappe et est rajoutée à celle-ci si la distance (de Hamming) la séparant de l'exemplaire est inférieure à un seuil donné. Si la distance est supérieure à ce seuil, une nouvelle grappe sera créée et l'entrée en deviendra l'exemplaire. Ces différentes grappes dépendront donc de l'ordre de présentation des entrées.

La structure du réseau est semblable à celle de Hamming. Le réseau de Carpenter/Grossberg diffère principalement de part le fait que des connexions "feed-back" (rétroaction) sont fournies entre les neurones de sortie et les neurones d'entrée.

Les différentes opérations effectuées par l'algorithme sont :

- initialisation des poids à zéro;
- établissement d'un seuil de correspondance appelé "vigilance" et compris entre zéro et un. Plus la valeur est proche de un, plus la correspondance doit être importante;
- présentation d'une nouvelle entrée au réseau. Cette entrée est comparée, en parallèle, à tous les exemplaires des grappes, et les scores de correspondance sont calculés;
- l'exemplaire avec le plus grand score de correspondance est choisi et ensuite comparé à l'entrée. Pour cela, il faut calculer le rapport entre le nombre de bits de valeur 1, commun à l'entrée et à cet exemplaire, et le nombre de bits de l'entrée;
- si ce rapport est supérieur au seuil de vigilance, l'entrée est considérée comme similaire à l'exemplaire et celui-ci est mis à jour en effectuant une opération logique ET entre les bits de l'entrée et ceux de l'exemplaire. Si le rapport est inférieur au seuil, une nouvelle grappe sera créée et l'entrée en deviendra l'exemplaire.

La figure ci-dessous (figure VI.13.) donne un exemple du comportement de cet algorithme. On remarquera que l'ordre de présentation des différents exemples influence la création des différents exemplaires de classe.

Comme pour le réseau de Kohonen, on peut se référer au réseau Counterpropagation, et plus spécialement à la couche de neurones de "Grossberg".

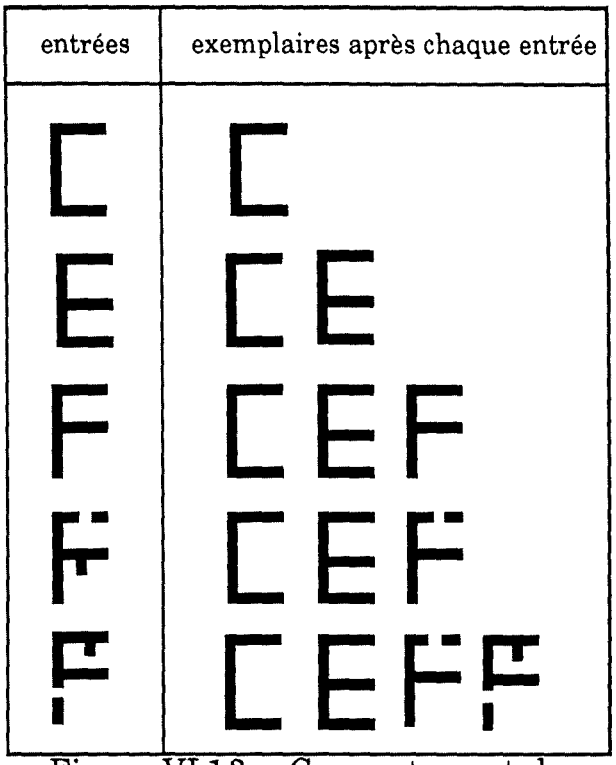


Figure VI.13. : Comportement du réseau de Carpenter/Grossberg.

Une limitation du modèle est la nécessité de disposer d'entrées correctes. Le moindre bruit dans la représentation de l'entrée peut engendrer des problèmes. [LIPPM,88], [HECHT,88], [WASSE,89].

VI.4.9. Adaline-Madaline

VI.4.9.1. Adaline

Adaline : ADaptive LInear NEuron.

Le réseau Adaline est un réseau à valeurs d'entrée binaires. C'est un réseau à couche unique (figure VI.14.).

Un poids seuil est connecté à une entrée fixée à +1 fonction d'entrée affine). Les entrées peuvent prendre les valeurs +1 ou -1. Les poids peuvent être positifs ou négatifs. La somme pondérée des entrées par les poids (y compris le poids seuil) (fonction d'entrée NET) est appliquée à un quantificateur, qui la convertit de façon à obtenir une sortie (OUT) égale à +1 ou -1.

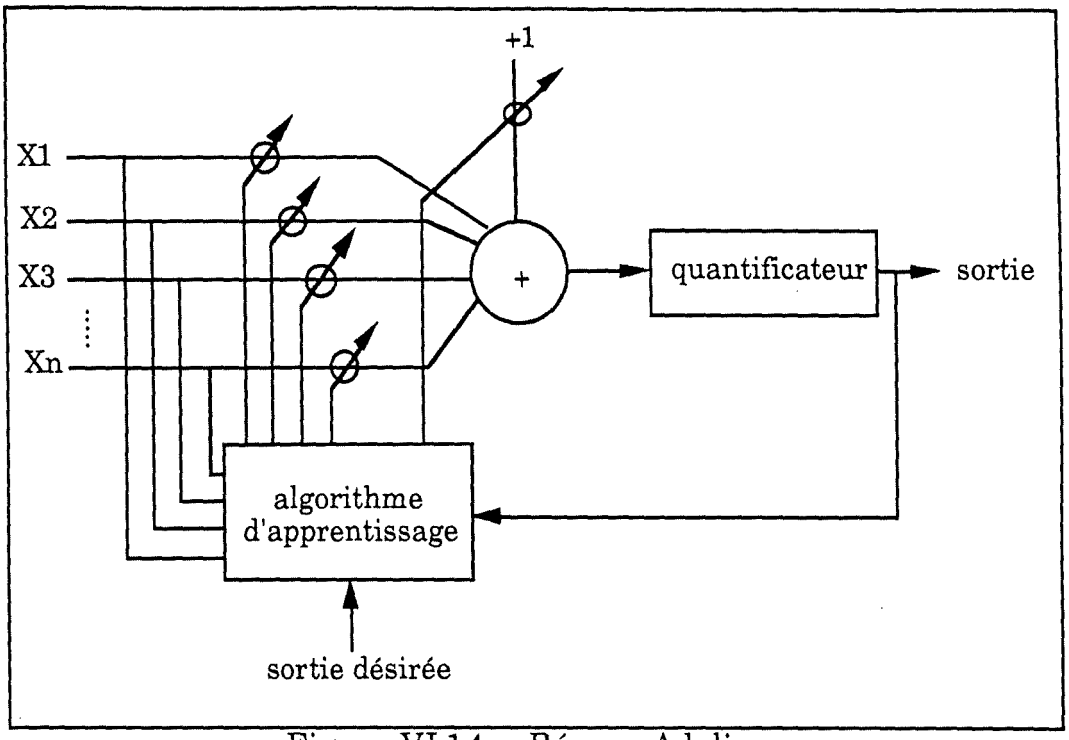


Figure VI.14. : Réseau Adaline.

Il y a plusieurs variantes de l'algorithme d'apprentissage. L'un d'eux est le "pattern recognizing control systems" de B. Widrow :

- initialiser les poids à des valeurs aléatoires (si les poids étaient initialisés à la même valeur, on ajouterait ou soustrairait toujours la même valeur, ce qui risquerait de bloquer le système),
- présenter un élément d'entrée de l'ensemble d'apprentissage, et la sortie désirée,
- calculer l'erreur, c'est-à-dire la différence entre la sortie désirée et la sortie calculée par le quantificateur,
- ajuster les poids de façon à réduire cette erreur,
- répéter le processus avec l'entrée suivante de l'ensemble d'apprentissage.

VI.4.9.2. Madaline

Madaline : Multiple Adaptive Linear Neuron.

Un réseau Adaline en deux couches ou plus est un réseau Madaline.

Cette architecture utilise plusieurs Adaline en parallèle avec une sortie unique. Si la moitié des entrées du neurone de sortie sont égales à +1 (Net = +1), alors, la sortie sera +1 (OUT = +1), sinon, elle sera égale à -1. D'autres fonctions de transfert, telles que le OU et le ET, sont possibles.

Un réseau Madaline se présente de la façon suivante (figure VI.15.) :

- une couche de neurones d'entrée agissant comme buffer,
- une couche de neurones intermédiaires composée d'éléments Adaline et entièrement connectée avec la couche des neurones d'entrée,
- un neurone de sortie du type Madaline.

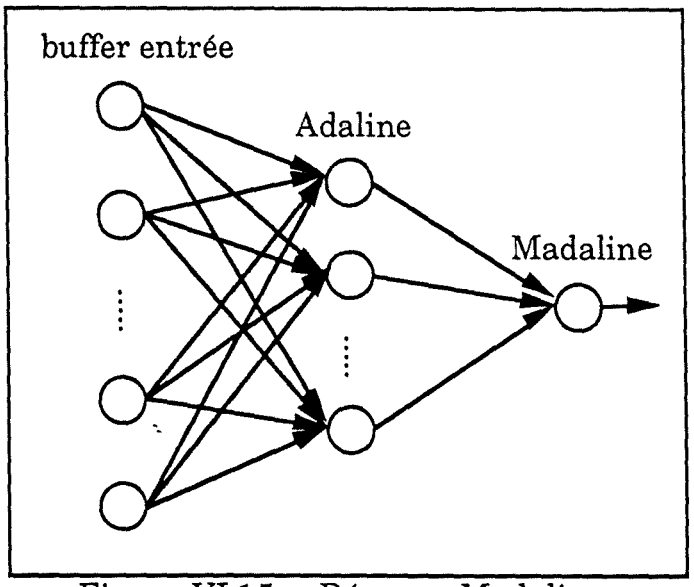


Figure VI.15. : Réseaux Madaline.

Les principales applications des réseaux Madaline sont les prévisions Météorologiques et la reconnaissance de la parole. [KLIMA,87-2], [HECHT,88].

VI.4.10. Brain-State-in-a-Box (BSB)

VI.4.10.1. Le modèle

Le modèle BSB ("Brain-State-in-a-Box") est un modèle particulier de réseau neuronal non linéaire. Ce réseau est un système auto-associatif à feed-back.

Comme pour tout réseau neuronal, la programmation du modèle BSB se déroule en deux phases distinctes : une phase d'apprentissage suivie d'une phase de vérification. L'apprentissage est basé sur la règle de Widrow-Hoff et la vérification procède d'une fonction linéaire ou bien, le plus souvent, d'une fonction sigmoïde non linéaire.

Pour décrire le système à un moment donné, il faut représenter l'activité de toutes les cellules à ce moment, soit le vecteur d'état. Initialement, le vecteur d'état contient les caractéristiques de l'objet que l'on veut faire reconnaître au réseau.

L'apprentissage associatif du système se réalise d'après le cycle décrit ci-dessous :

- le vecteur d'entrée est présenté au système qui produit un vecteur de sortie;
- la matrice des poids des connexions est modifiée d'après l'écart qu'il y a entre le vecteur d'entrée et le vecteur de sortie; la valeur du cosinus entre ces deux vecteurs constitue la mesure de l'écart.

La rétroaction vers le système est positive ce qui a pour conséquence de produire une sortie qui augmente sans limite. Dès lors, on fixe au départ les valeurs minimales et maximales que peuvent prendre les seuils des différents neurones. Ceci a pour effet de limiter le vecteur de sortie, d'où le nom de "Brain-State-in-a-Box".

VI.4.10.2. Un exemple d'utilisation du modèle

L'exemple décrit ici est la différenciation de mots identiques d'après le contexte. En d'autres termes, le réseau peut déterminer la signification de certains mots ambigus d'après le contexte qui lui est donné.

Tout d'abord, on enseigne au système le nom des objets. Ces noms sont ambigus car ils peuvent référer à plus d'une sorte d'objet. Par exemple, le nom oiseau est ambigu parce qu'il peut se référer à moineau, merle, corbeau, ... Chaque stimulus d'apprentissage consiste en un nom et un ensemble complet d'associations auxquelles le nom pourrait se référer. Par exemple, un stimulus d'apprentissage qui contient le mot "fruit" est associé à des caractéristiques telles que "pomme", "rond", "jaune".

Exemple :
OISEAU-rouge_gorge-été-vers-rouge-vole-aile
OISEAU-canari-chante-cage-jaune-vole-aile
OISEAU-corbeau-dérobe-ferme-noir-vole-aile
CHIEN-fourrure-maison-noir-mord-mammifère
.
.
.

Une fois tous les stimuli appris au système, on peut passer à la phase de répétition ("retrieval phase"). On teste la capacité du système à reconnaître la signification des différents mots en les lui présentant avec et sans contexte. Le contexte est constitué d'informations à propos de l'objet auquel le mot se réfère.

Le système, tel que décrit plus haut, retrouve toujours la signification d'un mot, avec ou sans contexte. Quand il n'y a qu'une

signification possible, le système retrouve toujours la bonne. Cependant, le réseau réalise plus d'itérations lorsqu'il y a plusieurs significations possibles.

Finalement, un nom connu est présenté au système avec des caractéristiques inconnues. Le système essaye alors de retrouver le ou les autres objets qui contiennent ces mêmes caractéristiques.Une fois qu'il a trouvé ces objets, il prend le complément d'information y associé et essaye d'inférer quel contexte est associé au nom de départ, c'est-à-dire quel contexte possible contient l'une ou l'autre des nouvelles caractéristiques. [ANDER, 86], [HECHT,88].

VI.4.11. Bidirectional Associative Memory (BAM)

Le réseau BAM est un réseau hétéro-associatif. Cela signifie qu'à un vecteur d'input donné, il peut associer un vecteur d'output différent.

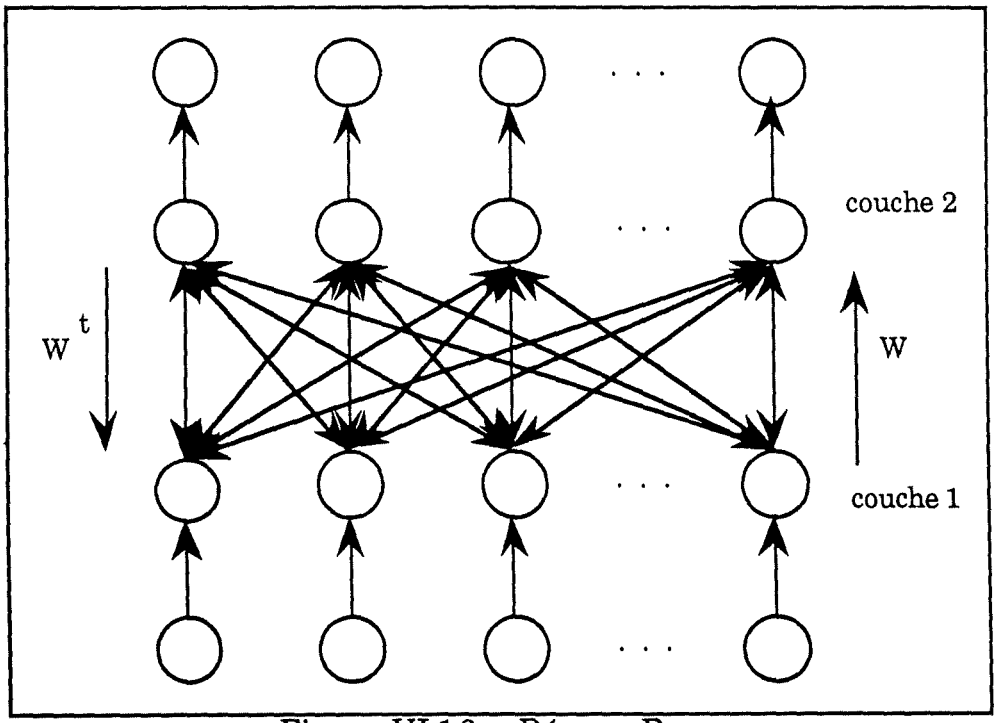


Figure VI.16. : Réseau Bam.

Le réseau BAM possède deux couches centrales de neurones, et des couches d'entrée et de sortie agissant comme "buffer" (figure VI.16.).

Un vecteur d'input binaire A, est appliqué à la matrice de poids W, ce qui produit le vecteur d'output B. Ce vecteur est ensuite appliqué à la transposée de la matrice des poids, soit W^t, ce qui produit un nouveau vecteur d'input. Il est important de remarquer que les vecteurs A et B ne

doivent pas nécessairement posséder le même nombre de composants. Ce processus est répété jusqu'à convergence. Pour cela le feed-back est utilisé. La sortie est ensuite transférée depuis la seconde couche intermédiaire vers la couche de sortie.

Chaque neurone du réseau calcule la somme pondérée des inputs par les poids (NET), et y applique ensuite une fonction d'activation, afin de produire l'output du neurone (OUT). Cette fonction est généralement une fonction sigmoïde.

Nous avons donc :

$$b_i = G\left(\sum_j a_j W_{ij}\right);$$

ce qui donne en notation matricielle : B = F(AW);

avec :

- B = vecteur d'output de la couche 2;
- A = vecteur d'output de la couche 1;
- W = matrice des poids entre les couches 1 et 2;
- G = fonction d'activation.

Nous obtiendrons donc : $A = G\left(BW^t\right)$ avec W^t matrice transposée de la matrice de poids W.

Etant donné que les vecteurs sont binaires, il faut que les outputs des neurones soient binaires. Pour cela, OUT_i sera égal à 1 si NET_i est strictement positif, et égal à 0 si NET_i est strictement négatif. L'output restera inchangé si NET_i est égal à zéro.

L'ensemble d'apprentissage est composée de paires de vecteurs A et B, et l'apprentissage se fait par calcul. La matrice des poids W est déterminée en faisant la somme du produit vectoriel de la transposée du vecteur A par le vecteur B, sur l'ensemble d'apprentissage.

$$W = \sum_i A_i^t \, B_i.$$

On parle de "**resonance**" car lorsqu'on applique un vecteur d'input A à W, cela produit un vecteur d'output B, qui appliqué à W^t reproduit le vecteur A.

[BLUM,90], [KOLON,87], [KLIMA,88-3], [HECHT,88], [WASSE,89].

VI.4.12. Cognitron et Neocognitron

VI.4.12.1. Cognitron

Structure

Le réseau Cognitron est un réseau à couches relativement complexe. Ces couches contiennent deux types de neurones différents. D'une part, il y a les neurones excitateurs, et d'autre part les neurones inhibiteurs. Les neurones excitateurs tenteront de renforcer les connexions (augmenter la valeur des poids) les reliant aux neurones d'une couche supérieure alors que les neurones inhibiteurs, quant à eux, joueront un rôle inverse. L'excitation d'un neurone dépend de la somme pondérée des ses inputs excitateurs et inhibiteurs par les poids.

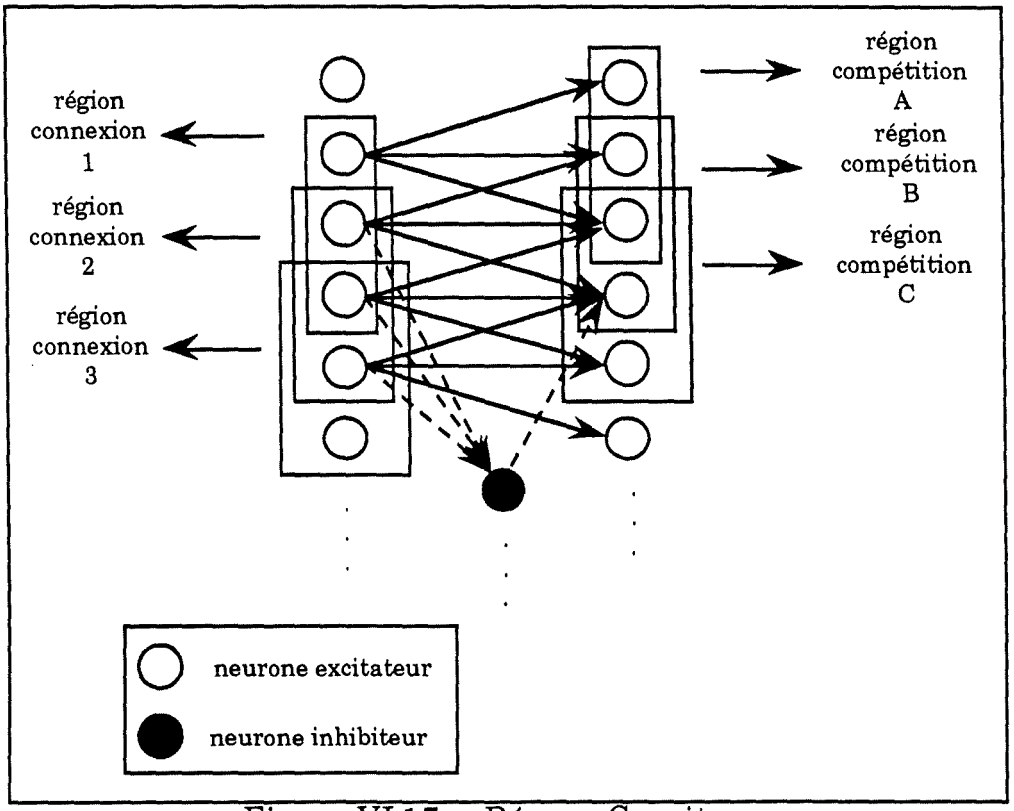


Figure VI.17. : Réseau Cognitron.

Dans les différentes couches on peut également distinguer deux types de régions : les régions de connexion et les régions de compétition.

- Une région de connexion est un ensemble de neurones d'une couche, qui sont tous connectés à un même neurone de la couche

supérieure (le réseau Cognitron n'est donc pas un réseau entièrement connecté).

- La région de compétition est un ensemble de neurone d'une même couche et qui sont tous connectés à un même neurone de la couche inférieure. La caractéristique de cette région provient du fait que, lors de l'apprentissage, seul le neurone ayant l'output le plus élevé pourra modifier les poids de ses connexions.

Les graphes (figures VI.17. et VI.18.) représentent un réseau Cognitron. On remarquera que tout neurone de la couche supérieure est associé à un neurone inhibiteur de la couche inférieure. Ce neurone inhibiteur a les mêmes inputs (excitateurs) que le neurone de la couche supérieure auquel il est associé. En d'autre mot, ces deux neurones possèdent la même région de connexion. Les poids, reliant les neurones de la région de connexion au neurone inhibiteur, sont fixes lors de l'apprentissage et leur somme vaut un.

Apprentissage

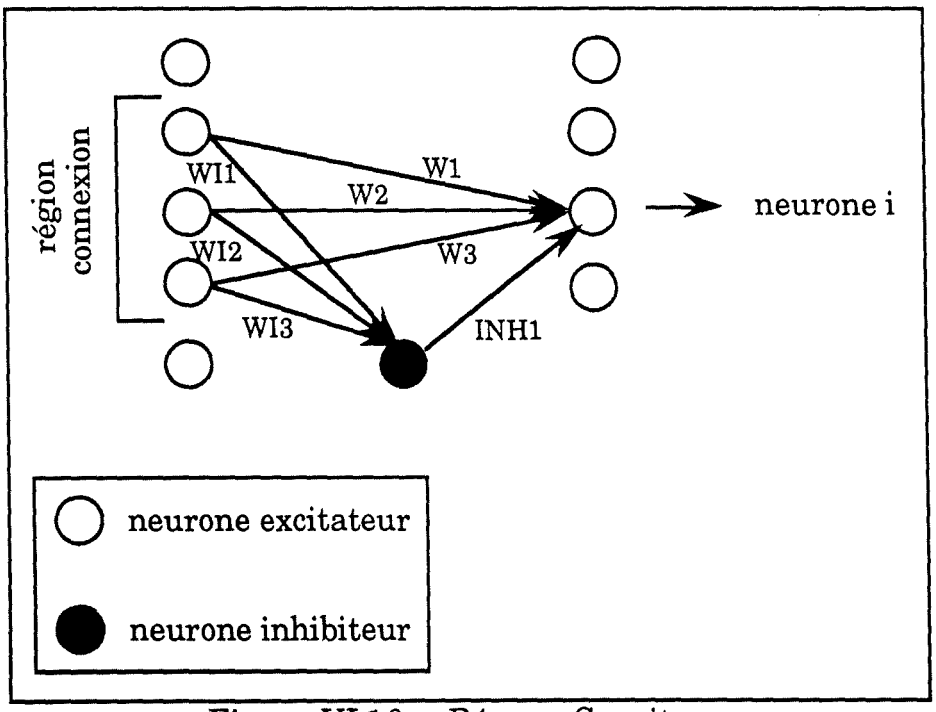


Figure VI.18. : Réseau Cognitron.

L'apprentissage est non supervisé. Il faut distinguer les neurones excitateurs des neurones inhibiteurs.

a) Neurone excitateurs.

Pour calculer l'input d'un neurone excitateur, il faut d'abord calculer la somme pondérée des inputs excitateurs et inhibiteurs par les poids.

$$- E = \sum_i W_i \, OUT_i;$$

$$- I = \sum_i INH_i \, OUTINH_i;$$

avec :

- E = input excitateur total;
- I = input inhibiteur total;
- W_i = poids reliant le neurone au $i^{ème}$ neurone excitateur de la couche inférieure;
- OUT_i = output du $i^{ème}$ neurone excitateur de la couche inférieure;
- INH_i = poids reliant le neurone au $i^{ème}$ neurone inhibiteur de la couche inférieure;
- $OUTINH_i$ = output du $i^{ème}$ neurone inhibiteur de la couche inférieure.

Le calcul de l'output est alors le suivant :

$$NET = \left(\frac{(1 + E)}{(1 + I)}\right) - 1;$$

$$OUT = OUT \text{ si } NET \geq 0;$$

$$OUT = 0 \text{ si } NET < 0;$$

avec :

- Net = fonction d'entrée du neurone.

Ce qui donne $\frac{(E - I)}{(1 + I)}$ avec NET positif.

Afin de limiter la croissance sans limite de l'output, c'est-à-dire lorsque lorsque E et I sont supérieurs à zéro, il est possible de calculer l'output en prenant le ratio de E sur I plutôt que leur différence.

$$OUT = \frac{E}{I} - 1.$$

b) Neurone inhibiteur.

L'output d'un neurone inhibiteur est égal à la somme pondérée de ses inputs par les poids le reliant à la région de connexion. Ces inputs correspondent, en fait, aux outputs de la région de connexion associée au neurone.

c) Changement des poids des connexions.

1) Le changement des poids reliant les neurones excitateurs au neurone i de la couche supérieure se fait de la façon suivante (cfr graphe ci-dessus) :

$$- \delta W_i = q \, WI_j \, OUT_j ;$$

avec :

- W_i = poids reliant le neurone i de la couche supérieure au neurone j de la couche inférieure;
- q = taux d'apprentissage;
- WI_j = poids reliant le neurone j de la couche inférieure, au neurone inhibiteur associé au neurone i de la couche supérieure;
- OUT_j = output du neurone j de la couche inférieure.

2) Pour la connexion entre le neurone inhibiteur et le neurone i de la couche supérieure, le changement de poids est proportionnel au ratio, de la somme pondérée des inputs excitateurs par les poids, et de deux fois l'input inhibiteur.

$$\delta INH_i = q \left(\frac{\sum_j W_j \, OUT_j}{2 \, OUTINH_i} \right) ;$$

avec :

- INH_i = poids reliant le neurone inhibiteur associé au neurone i de la couche supérieure;
- q = taux d'apprentissage;
- W_j = poids reliant le neurone j de la couche inférieure au neurone i de la couche supérieure;

- OUT_j = output du neurone j de la couche inférieure; ou input du neurone inhibiteur associé au neurone i de la couche supérieure;

- $OUTINH_i$ = output du neurone inhibiteur associé au neurone i de la couche supérieure.

3) Lorsque, dans une même région de compétition, aucun neurone n'est sélectionné pour modifier ses valeurs de poids, les formules suivantes sont appliquées :

- $\delta W_i = q' WI_j OUT_j$;

- $\delta INH_i = q' OUTINH_i$.

C'est notamment le cas au début de l'apprentissage, car tous les poids excitateurs sont initialisés à zéro.

Inhibition latérale

Un processus intéressant peut être d'associer, à chaque neurone "vainqueur" d'une région de compétition, un neurone inhibiteur latéral. Celui-ci somme les inputs des autres neurone de la région, et envoie un signal inhibiteur au neurone vainqueur. Cette méthode est efficace, mais elle ralentit considérablement le processus d'apprentissage.

Structure complète

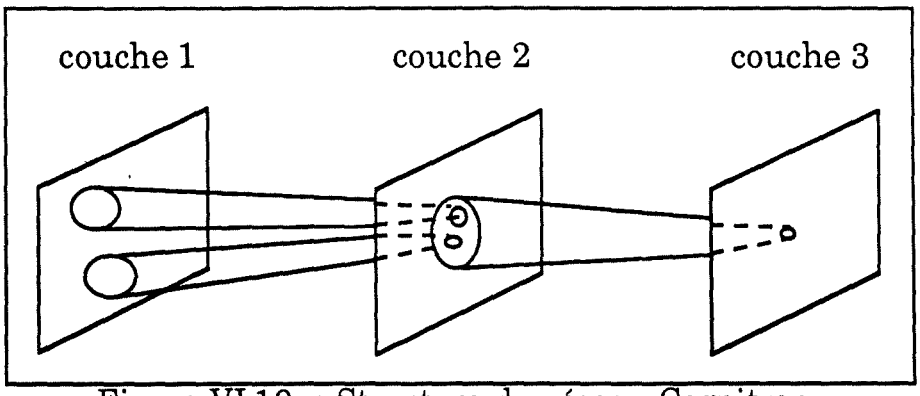


Figure VI.19. : Structure du réseau Cognitron.

En réalité, le réseau Cognitron comprend des couches à deux dimensions. Un neurone d'une couche supérieure correspond à une petite région de la couche qui lui est juste inférieure (figure VI.19.). On peut ainsi arriver à une structure où chaque neurone de la dernière couche correspond à toute la région d'input.

VI.4.12.2. Le Neocognitron

Le réseau Neocognitron a été créé (84-87) afin d'augmenter les capacités du réseau Cognitron. Cependant, il y a des différences importantes.

Contrairement au réseau Cognitron, qui ne dispose que d'un apprentissage non-supervisé, le réseau Neocognitron dispose également de l'apprentissage supervisé.

Une autre différence importante réside dans la structure des couches de neurones. Chaque couche est composée de deux types de neurones (simples et complexes) travaillant par paires. De plus, chaque neurone correspond à un tableau à deux dimensions.

Pour le reste, les notions de région de connexion, région de compétition, neurones inhibiteurs et excitateurs sont conservées.

VII Applications

VII.1. INTRODUCTION

Notre objectif dans cette partie est de présenter onze modèles qui adressent une large gamme de problèmes dans des domaines aussi variés que la vision, la robotique ou l'économie. Ces applications ont été choisies parce que considérées par DARPA ("Defense Advanced Research Project Agency") du département de la défense américaine ("Department Of Defense" -DOD) comme l'état de l'art en matière de résolution de problèmes par réseau neuronal.

Cependant, chacune de ces disciplines fait appel à des notions propres, parfois plus ou moins approfondies. Lorsque ces connaissances nous semblaient trop techniques nous les avons volontairement occultées. Le lecteur intéressé est renvoyé à [DARPA,88,403-498] où il trouva une description plus technique.

En outre, vu leur statut, les modèles présentés étaient considérés en 1988 (année de plublication de l'ouvrage) comme relativement confidentiels. Dès lors, les auteurs font preuve d'une grande circonspection dans leurs explications, se gardant bien de révéler les détails d'implémentation. Nous nous excusons donc de la sobriété des explications qui entourent parfois l'un ou l'autre modèle présenté. Les modèles présentés sont :

1- le pendule inversé;
2- le contrôle de processus;
3- l'identificateur de mots;
4- le classifieur de signaux sonars;
5- l'analyse de risques;
6- le robot élévateurs;
7- la reconnaissance de cibles;
8- le classifieur d'images;
9- le modèle de la rétine;
10- le pistage de cible;
11- la fusion d'images.

Pour terminer, nous voudrions signaler que les initiateurs de ces applications sont essentiellement américains. Cela est du essentiellement a une plus grande disponibilité de documents américains en même temps qu'un meilleur état d'avancement des travaux aux Etats-Unis.

VII.2. PRESENTATION DES APPLICATIONS

VII.2.1. Le pendule inversé

(Viral V. Tolat, Dr Bernard Widrow, Stanford University)

Il y a plusieurs raisons qui ont poussé le "System Application Panel" de DARPA à choisir cette application. Premièrement, le pendule inversé est un problème classique de contrôle qui a été étudié en profondeur et qui est bien compris. De plus, ce problème est représentatif de toute une classe de problèmes de contrôle et comprendre comment le résoudre via un réseau neuronal permettra de résoudre d'autres problèmes du même genre. Finalement, la complexité du pendule inversé est suffisamment importante pour que le problème soit intéressant tout en étant simple assez que pour être soluble.

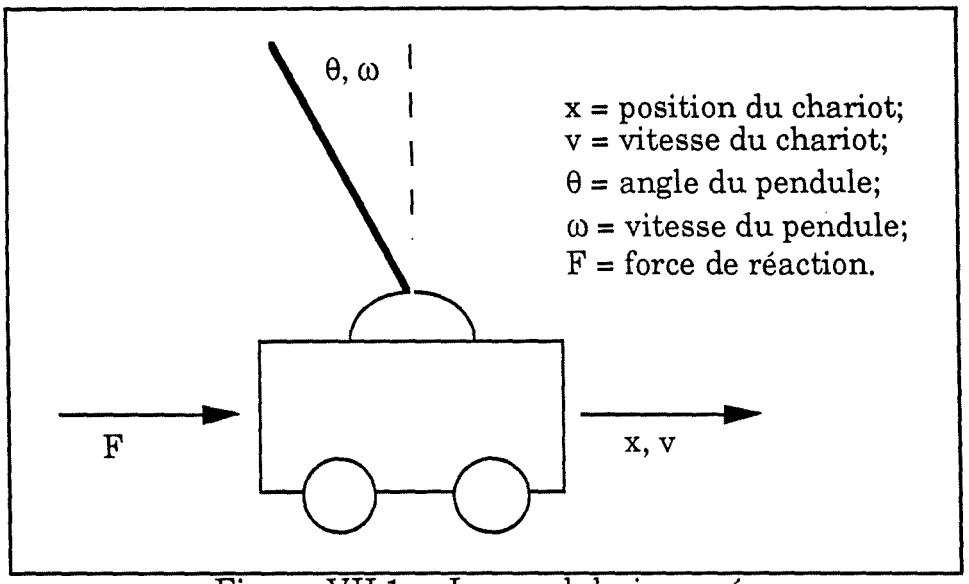


Figure VII.1. : Le pendule inversé.

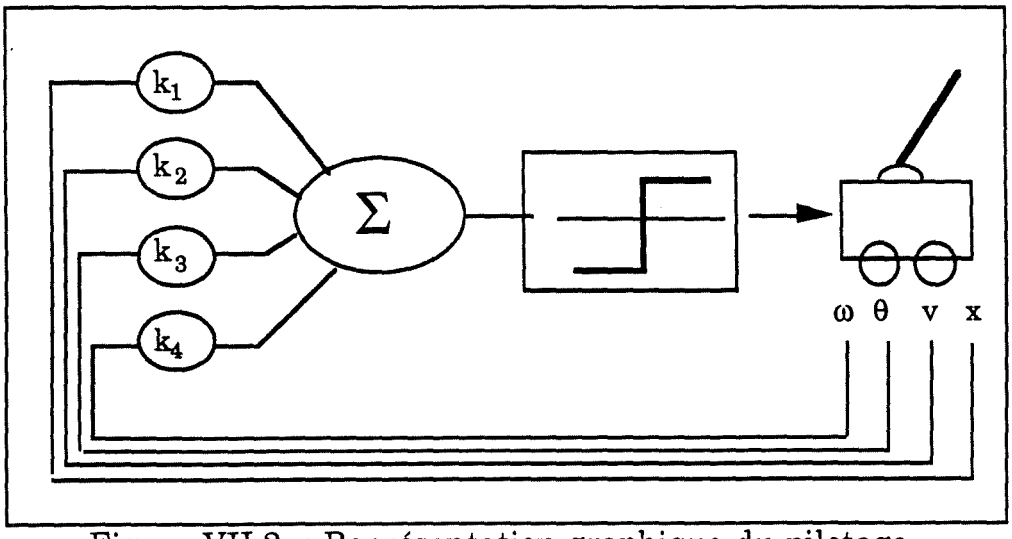


Figure VII.2. : Représentation graphique du pilotage.

Le pendule inversé est illustré à la figure VII.1.. Quatre variables décrivent l'état du système : la position du chariot (x), sa vitesse (v),

l'angle du pendule (θ) et sa vitesse angulaire (ω). La force requise pour stabiliser le système est :

$$F = U \, sgn [k_1 x + k_2 v + k_3 \theta + k_4 \omega]$$

où U est une constante représentant la grandeur de la force qui doit être appliquée au système et les coefficients k_1, k_2, k_3, k_4 sont dérivés de caractéristiques physiques du pendule et de la théorie du contrôle optimal.

Un réseau neuronal qui réalise ce pendule est présenté à la figure VII.2. Les détecteurs sur le chariot envoient l'état des variables dans le circuit neuronal qui conduit mécaniquement le chariot, fermant ainsi la boucle de contrôle.

VII.2.2. Le contrôle de processus

(R.S. Sutton, GTE Laboratories)

Cette application utilise un réseau neuronal qui prédit, à partir de données provenant de 100 à 200 détecteurs, la production d'une usine d'ampoules électriques et réalise des mesures de performances de la production. En identifiant mieux les variables qui influencent la performance de l'usine, des améliorations de coût de qualité peuvent être apportées.

Le réseau est composé d'un réseau **Adaline** mono-couche qui calcule les mêmes quantités qu'une régression linéaire c'est-à-dire les nombres qui lient les variables indépendantes (mesures des détecteurs) aux variables dépendantes (production de l'usine et autres mesures de performances). En plus, on utilise un réseau **Rétro-propagation** multi-couches pour saisir les non linéarités et un algorithme d'apprentissage tenant compte du temps écoulé.

Les objectifs à moyen terme de cette application sont de passer d'un simple réglage de processus à un contrôle de processus plus complet.

VII.2.3. L'identificateur de mots

(M.E. Hoff)

Un système de reconnaissance complète du langage naturel semble hors de portée de la technologie actuelle et ce à cause de l'énorme complexité du langage humain. Les systèmes de reconnaissance de la parole tiennent donc compte d'un certain nombre d'hypothèses pour

pouvoir être effectifs. Le système proposé ici tient compte des contraintes suivantes :

- une et une seule personne qui parle. Le système est entrainé avec des mots provenant du langage de cette personne et prononcés par elle;
- on ne considère que des phrases ou mots isolés plutôt qu'un flot de parole continu;
- le vocabulaire est limité. Le système fonctionne avec seulement quelques centaines de mots plutôt qu'avec les dizaines de milliers que connaissent la pupart des individus. Ceci permet d'économiser le stockage.

Avec ces restrictions, le langage peut être échantillonné pour former des patterns qui seront sockés durant la phase d'apprentissage. En phase de reconnaissance, le pattern qui correspond le mieux à l'input sera produit comme résultat. Il faut tout de même que le résultat de la comparaison soit supérieur à un certain seuil pour être valable; autrement, le réseau fournit un résultat indiquant qu'il ne reconnaît pas l'input. Le réseau, appelé "**Matched Filter Word Recognizer**" (figure VII.3.), est composé de neurones correspondants chacuns à un pattern mémorisé (représentation locale ou modèle ponctuel). Ces neurones possèdent un seuil qui varie dynamiquement de telle sorte qu'un seul neurone à la fois puisse tirer pour un input donné.

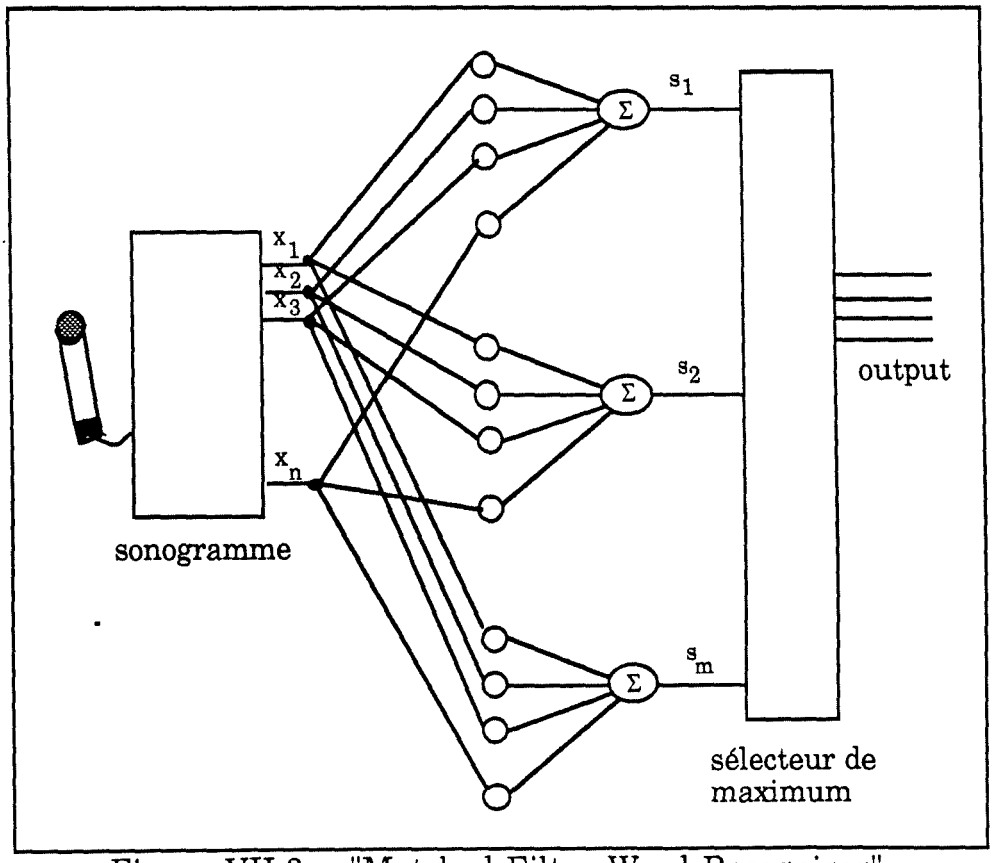


Figure VII.3. : "Matched Filter Word Recogniser".

La parole récoltée par le microphone est convertie en un pattern représenté par l'ensemble des valeurs d'input x_i, i=1,...,n. Chacun des patterns est stocké comme un ensemble de poids associé à un neurone qui génère un output S_j, j=1,...,m. Le sélectionneur de maximum choisit le plus grand output j et, s'il est supérieur à un certain seuil, fournit le code j en output.

VII.2.4. Le classificateur de signaux sonars

(R. Paul Gorman, Allied-Signal Aerospace Technology Center, Columbia & Terrence J. Sejnowski, John Hopkins University)

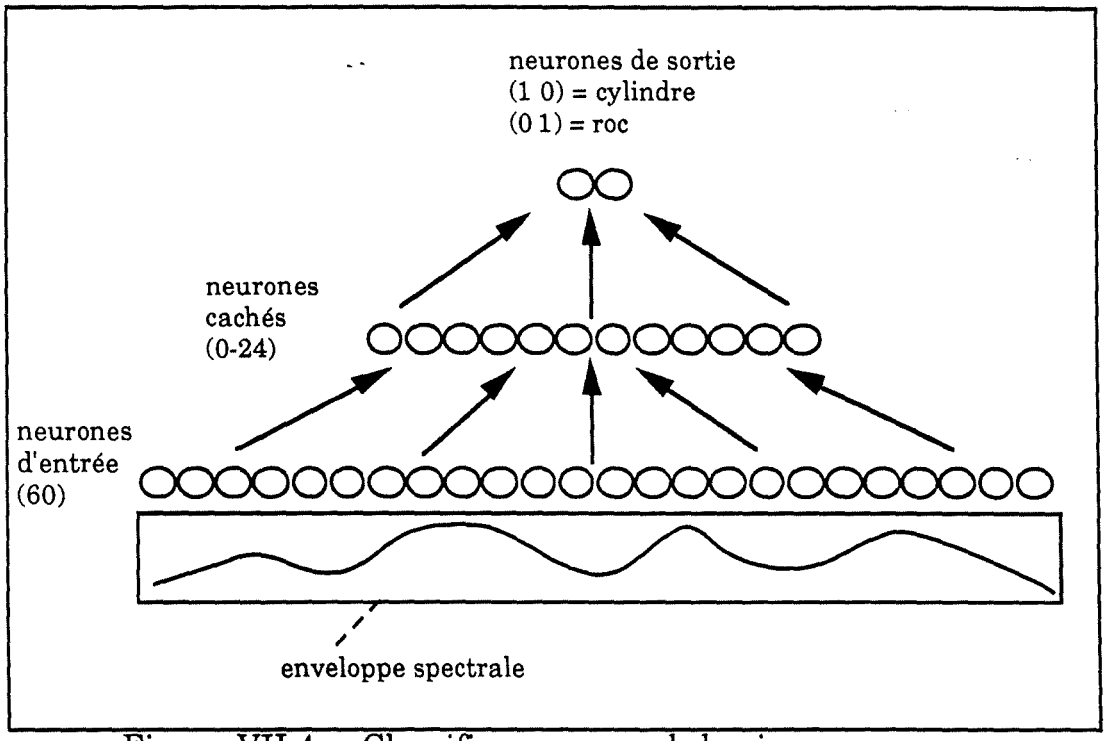


Figure VII.4. : Classifieur neuronal de signaux sonars.

A l'heure actuelle, il faut compter sur la compétence des opérateurs sonar pour distinguer l'écho provenant d'une mine de celui provenant d'un roc sous-marin. Ces nécessités d'identification se présentent surtout dans les eaux peu profondes des ports. Le réseau neuronal mis au point est un réseau **rétro-propagation** qui permet une discrimination automatique de ce genre de signal. Il utilise des données sonar provenant d'un roc et d'une mine de mêmes dimensions. Le réseau est composé d'une couche d'entrée de 60 neurones, d'une couche cachée de 24 neurones et d'une couche de sortie de 2 neurones (figure VII.4.). Le réseau est complètement interconnecté et chaque neurone utilise une

fonction d'activation non linéaire qui produit un output compris entre 0.0 et 1.0. Les données sont présentées sous forme d'enveloppes spectrales du signal sonar. Chaque enveloppe consiste en 60 échantillons compris entre 0.0 et 1.0.

Le travail de classification consiste donc pour le réseau à transformer un signal d'entrée en un pattern de sortie indiquant à quelle classe le signal appartient : (0 1) pour le roc ou (1 0) pour la mine. L'algorithme d'apprentissage utilisé est celui de rétro-propagation comme indiqué par le nom du réseau.

L'ensemble d'entraînement est constitué de 50 signaux retournés par le roc, de 50 signaux retournés par la mine et est présenté 300 fois au réseau. L'expérience a été renouvelée 10 fois avec des poids synaptiques différents pour avoir des performances relativement indépendantes de la distribution initiale des poids. Les résultats de l'apprentissage sont présentés à la figure 7.5. pour 0, 2, 3, 6, 12 et 24 neurones cachés respectivement. 0 neurone signifie qu'il n'y a qu'une couche d'entrée et une couche de sortie.

Nombre d'unités cachées	**Performances moyennes sur l'ensemble d'entraîn. (%)**	**Déviaion standard sur l'ensemble d'entraîn. (%)**	**Performance moyenne sur l'ensemble de test (%)**	**Déviation standard sur l'ensemble de test (%)**
0	79.3	3.4	73.1	4.8
2	96.2	2.2	85.7	6.3
3	98.1	1.5	87.6	3.0
6	99.4	0.9	89.3	2.4
12	99.8	0.6	90.4	1.8
24	100.0	0.0	89.2	1.4

Figure VII.5. : Résultats de classifications du sonar.

La déviation standard exprime en pourcentage le nombre de fois où le réseau se trompe sur l'ensemble d'entraînement. On constate qu'avec 12 neurones cachés, le réseau apprend 99.8% des inputs et n'en classifie mal que 0.6% lors de la phase de rappel. Avec 24 neurones, il apprend et classe tout correctement.

Le tableau reprend également les résultats pour les series de tests qui comportent des signaux sonars différents de ceux de l'entraînement. Avec 24 neurones, le réseau classifie correctement près de 90% des signaux inconnus et ne se trompe que 1.4 fois sur 100.

VII.2.5. Analyse de risques

(Edward Collins, Sushmito Ghosh, Christopher Scofield de la société Nestor Inc.).

Beaucoup de problèmes du monde réel requièrent des décisions basées sur un très grand nombre de données dont l'origine est par exemple des formulaires d'assurance ou l'output de détecteurs.

La compagnie Nestor a élaboré un système, le "Nestor Multiple Neural Network Decision System" pour répondre à ce type de problème. Il est utilisé ici pour résoudre le problème du prêt hypothécaire, une application spécifique d'analyse de risque faisant intervenir des données du monde réel.

Le système de Nestor adresse 3 niveaux de problèmes :

1- Validation de la demande d'emprunt.

Le système Nestor filtre la population des propriétaires potentiels et évalue leur demande sur base de certains ratios prédéfinis (emprunts/valeur du bien, dépenses/revenus,...). Il tient également compte de questions clés telles le statut d'emploi du demandeur, sa situation de divorce éventuelle ou son passé financier. Cette première étape permet d'écarter un certain nombre de demandeurs et de faire subir un examen plus approfondi à ceux qui restent;

2- Demande d'assurance hypothèque.

Les souscripteurs d'une assurance hypothèque sont par nature des personnes à plus haut risque que la population générale des souscripteurs d'hypothèque. Ils ont déjà été enregistrés à l'étape (1) et jugés comme peu sûrs. D'où cette souscription de deuxième ordre qu'est l'assurance.

3- Le processeur de risque de non paiement.

Certains risques assurés par la compagnie se réaliseront dans le futur. Comme ces paiements d'assurance constituent les principales dépenses de la compagnie, on comprend son souhait de vouloir les réduire. L'évaluation du risque d'occurrence est un filtre de troisième ordre de la population générale en vue donc de tester la performance du prêt.

Nous allons essentiellement nous attacher à décrire la validation de la demande d'emprunt (1). Celle-ci a été réalisée via le "Multiple Neural Network Learning System" (MNNLS), ou "Nestor Learning System" (NLS) en abrégé, développé par Nestor. Retenons quand même que la demande d'une assurance hypothèque (2) est traitée par un assureur automatique d'assurance hypothèque ("Mortgage Insurance Underwriter") et que le risque de non paiement (3) est traité par un proceseur de risque ("Delinquency Risk Processor") qui évalue le risque qu'un emprunteur soit défaillant pour un prêt donné. Ce sont là les trois

composantes du système d'aide à la décision mis au point par la compagnie Nestor.

Le réseau NLS est composé de 3 couches de 3 sous-réseaux, chaque couche traitant une partie non exclusive des caractéristiques du candidat (figure VII.6.).

Les réseaux du sommet traitent des discriminations évidentes alors que les réseaux du bas, déchargés de la majorité des décisions, se concentrent sur une discrimination plus fine. Le NLS est en outre constitué d'un contrôleur qui à la fois synthétise une réponse à partir des outputs des réseaux et dirige leur entraînement. Le module contrôleur est constitué d'un ensemble de règles pour déterminer l'output du système et son entraînement.

Chaque sous-réseau est un RCE ("**Restricted Coulomb Energy**") qui permet de catégoriser une caractéristique du client. Si la catégorisation est non ambiguë, un seul neurone de la couche de sortie de ce réseau est actif. Si la catégorisation est ambiguë, deux ou plusieurs neurones de sortie sont actifs. Mais même dans ce cas, le NLS peut trouver la bonne solution.

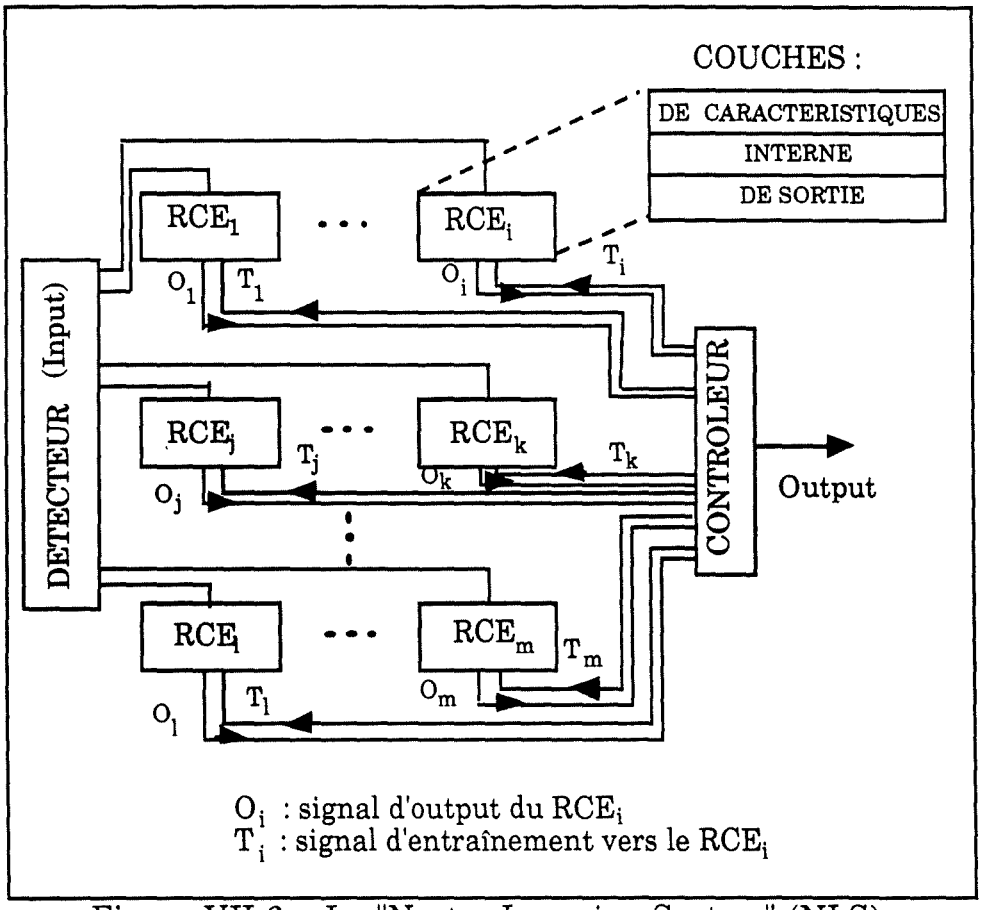


Figure VII.6. : Le "Nestor Learning System" (NLS).

Supposons par exemple que le NLS ne soit constitué que de 2 réseaux RCE et qu'il doive classer des inputs α, β et γ (figure VII.7.). Dans cette situation, le premier réseau peut distinguer α de β mais pas β de γ. Le deuxième réseau quant à lui peut séparer γ de α et β mais pas α de β. Si l'on présente l'input β à chacun de ces réseaux, ils fourniront les réponses ambiguës (β,γ) et (α,β) respectivement. Le contrôleur peut synthétiser ces réponses pour arriver à la classification β. Donc, en l'absence de réseau qualifié pour identifier b, les réponses confuses des 2 réseaux peuvent être utilisées pour produire une classification non ambiguë.

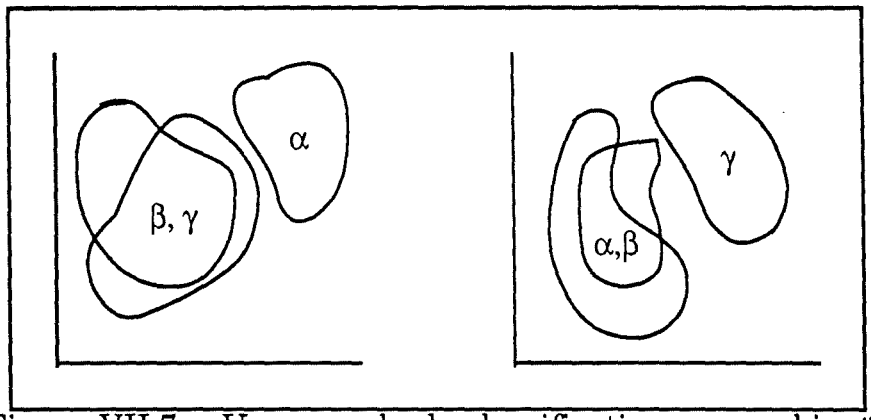


Figure VII.7. : Un exemple de classification non ambiguë.

Le NLS est entièrement basé sur un logiciel et est opérationnel sur IBM PC/AT, Sun Microsystems ou environnements Apollo.

VII.2.6. Robot élévateur

(Von Ayre Jennings, Martin Marietta)

L'un des objectifs des réseaux neuronaux est d'améliorer de manière spectaculaire la capacité des robots à fonctionner dans un monde réel. Comme première étape dans cette voie, nous proposons ici un réseau qui apprend à guider les mouvements d'un robot par l'observation du comportement humain.

Le réseau utilisé dans cette expérience est le "**Cerebellar Model Articulation Controller**" (CMAC) développé par Dr. James Albus. Il peut implémenter n'importe quelle fonction continue en utlisant la procédure d'apprentissage de Widrow-Hoff et possède l'avantage de pouvoir s'exécuter en temps réel sur un ordinateur digital conventionnel.

Le robot utilisé pour l'expérience est un robot industriel avec bras élévateur permettant de soulever des palettes. Plusieurs détecteurs de

proximité équipent le bras, ce qui permet au réseau neuronal de guider son insertion dans la pallette (figure VII.8). Ceci va obliger le réseau à apprendre à générer une trajectoire cartésienne (x, y, z, degré de pente, roulement, déviation) basée sur les valeurs d'inputs des détecteurs. Cette tâche est très difficile pour des systèmes de contrôle conventionnels parce que la relation entre un pattern particulier des détecteurs et le mouvement approprié est très compliquée et non linéaire.

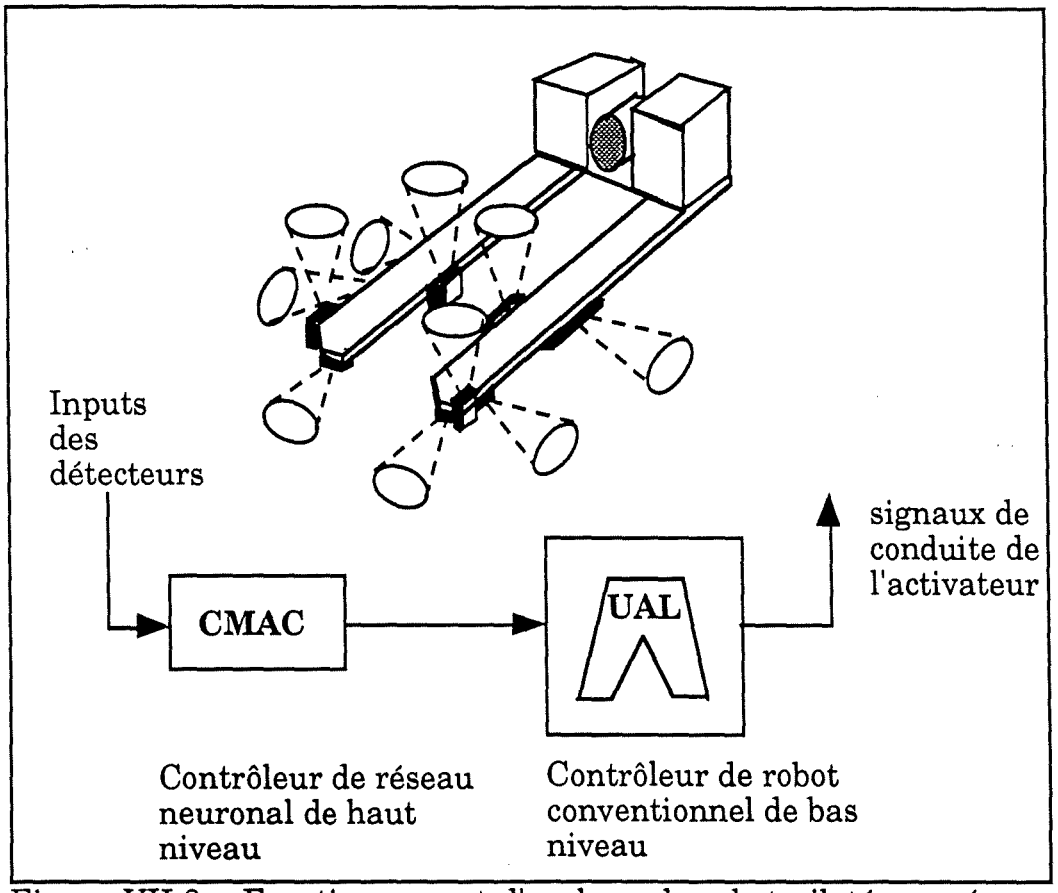


Figure VII.8. : Fonctionnement d'un bras de robot piloté par réseau neuronal.

La méthode utilisée pour entraîner le réseau neuronal est classique en ce sens où elle se base sur des exemples -ici, de manipulations- fournis par un opérateur humain. Ce dernier dirige le bras de robot via une manette du genre "joystick". La direction des forces exercées sur la manette est utilisée pour spécifier la commande de trajectoire idéale au système de contrôle du robot et à la procédure d'apprentissage (figure VII.9.) L'objectif est donc d'adapter les poids du réseau de telle sorte que pour des inputs de détecteurs donnés, le réseau fournisse une trajectoire qui est identique ou très similaire à celle générée par l'opérateur humain.

Après l'apprentissage, le réseau est directement connecté au contrôleur de bas niveau pour guider le bras de robot de manière autonome. Remarquons qu'une des principales préoccupations du

professeur lors de l'apprentissage est de n'utiliser que des informations qui seront accessibles par le robot par la suite. Par exemple, des mouvements de bras qui sont au delà de la portée des détecteurs ne peuvent évidemment être utilisés comme exemples d'apprentissage.

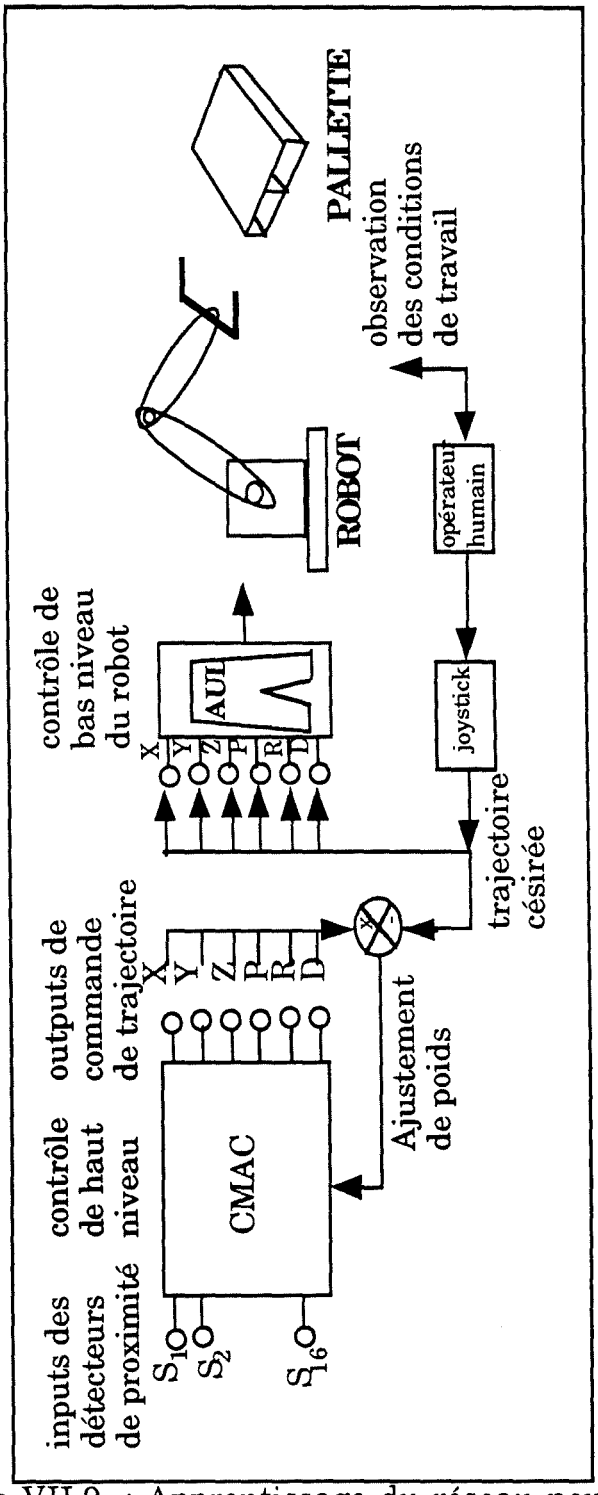


Figure VII.9. : Apprentissage du réseau neuronal.

Les résultats obtenus pour ce genre de guidage sont assez encourageants : le réseau a appris à saisir la pallette depuis n'importe quel endroit de départ et ce, avec seulement 7 exemples d'entraînement.

VII.2.7. Reconnaissance de cibles

(M. Oyster, Hughes Aircraft, Electro-Optical & Data Systems Group)

Le problème abordé ici est la reconnaissance automatique de cible(s) de jour ou de nuit. Le réseau neuronal utilisé est présenté à la figure VII.10. Il combine des attributs intrinsèques d'images pour fournir des agrégats de formes plus complexes. Une technique dite de relaxation intègre les agrégats pour produire une interprétation de forme unique.

Une caractéristique marquante de ce type d'application est que l'apprentissage n'est pas un problème clé. Le réseau ne doit pas apprendre de lui-même toutes les représentations intermédiaires requises pour produire une solution au problème de vision parce que cela prendrait trop de temps. La structure en couche du réseau et le pattern d'interconnexion permettent de réaliser automatiquement les représentations intermédiaires.

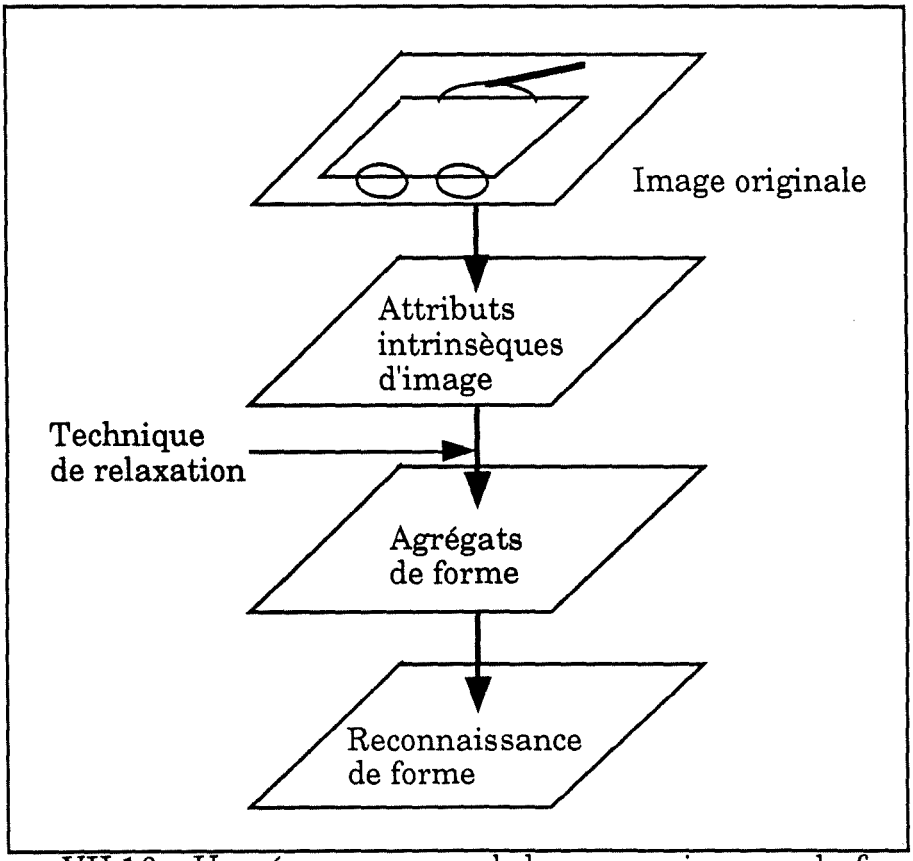


Figure VII.10 : Un réseau neuronal de reconnaissance de formes.

VII.2.8. Classifieur d'images

(Murali Menon MIT/Lincoln Laboratory)

Le **Neocognitron** de Fukushima (1980) est un réseau neuronal multi-couches feed-forward utilisé en reconnaissance de formes et classification de patterns à 2 dimensions (cfr IV.8. Tâches exécutée par un réseau neuronal et VI.4.12.2. le Neocognitron). Le modèle est invariant à des déplacements de l'objet dans le plan et tolère des images bruitées et même un peu déformées. Son architecture modélise de façon qualitative l'anatomie de la rétine humaine.

Un exemple de la capacité du Neocognitron à classer des patterns a été démontré en entraînant le système sur trois patterns d'inputs différents (une voiture, un camion et un char). Le réseau était composé de 4 couches de six plans et avait appris à classer les images après 20 présentations de chaque exemplaire. Chaque plan d'un niveau était sensible à des caractéristiques particulières (barres horizontales ou verticales,...) qu'il assemblait sans l'aide d'un professeur.

Le Neocognitron de l'expérience était implémenté comme simulation software sur un mini-ordinateur.

VII.2.9. Modèle de la rétine

(Michael H. Brill, Doreen W. Bergeron et William W. Stoner de SAIC)

Les spécialistes ont établi que l'intensité de la lumière varie d'un facteur 10 milliards en une journée de temps. Notre système visuel, par contre, n'enregistre lui des variations que de l'ordre d'un facteur 1000, ce qui reste suffisant puisque la plupart des surfaces ne diffèrent dans leur réflexion que d'un facteur 100. Comme la lumière varie graduellement dans l'espace ou dans le temps, tout dispositif photosensible ne doit être sensible qu'à de petits changements de luminosité par rapport à une intensité moyenne de lumière. Cette acquisiton de l'adaptation de la lumière est une expérience commune dans les problèmes de vision et de vision par ordinateur.

Un autre objectif d'un système visuel est de s'adapter pour continuer à fonctionner lorsque la lumière devient faible. Une technique de l'oeil humain consiste à élargir sa pupille, ce qui compromet cependant sa résolution spatiale.

Le réseau neuronal réalisé, Iris, est un modèle de la rétine et s'attaque à ces problèmes que sont l'adaptation au contraste et l'adaptation à l'intensité de la lumière. Nous n'exposons pas le fonctionnement détaillé de ce modèle parce qu'il nous semble en dehors

du cadre du présent travail. Nous renvoyons toutefois le lecteur intéressé à [DARPA,88,469-475].

Le modèle a été simulé sur une station Sun Microsystems en Fortran. Aucune indication n'est donnée quant au type de réseau utilisé ou quant au type d'apprentissage employé, s'il existe.

VII.2.10. Pistage de cibles

(R. Kuczewski, TRW ANS Center San Diego)

Le pistage de plusieurs cibles en même temps est un problème important dans les systèmes aéronautiques ou de contrôle du traffic aérien. Le problème consistant à trouver une solution optimale croît avec n! ou n est le nombre de pistes suivies.

Dans les cas les plus simples de pistage multi-cibles, les informations viennent sous forme d'instantanés d'objets à un moment donné, l'objectif étant d'identifier le chemin suivi par chacun des objets. La figure VII.11. montre un exemple de séquences d'instantanés. Le problème est de corréler les instantanés tout en tenant compte de contraintes sur la dynamique des objets.

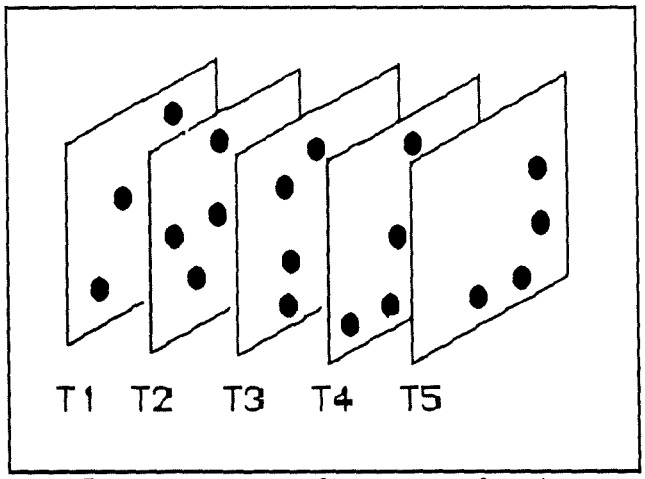


Figure VII.11. : Instantanés utilisés pour le pistage multi-cibles.

Le réseau neuronal réalisé fonctionne pour le pistage de cibles dans un espace à deux dimensions mais peut être étendu à un espace à n dimensions. La technique inventée pour le système est appelée "Interpolative Probability Fields" (IPF). Elle est une combinaison du "**Boundary Contour System**" (BCS) de Grossberg et Mingolla et de la dynamique probabiliste du pistage de cibles.

Les inputs présentés au système sont de la forme de la figure VII.12.. Le réseau IPS trouve la direction la plus probable pour chaque objet à chaque instant, ce qui permet de connecter les "points" et

d'assigner un trajet aux objets. La direction la plus probable est choisie d'après une distribution gaussienne pour ce qui est de la distance, de l'angle de rotation et de la déviation de direction.

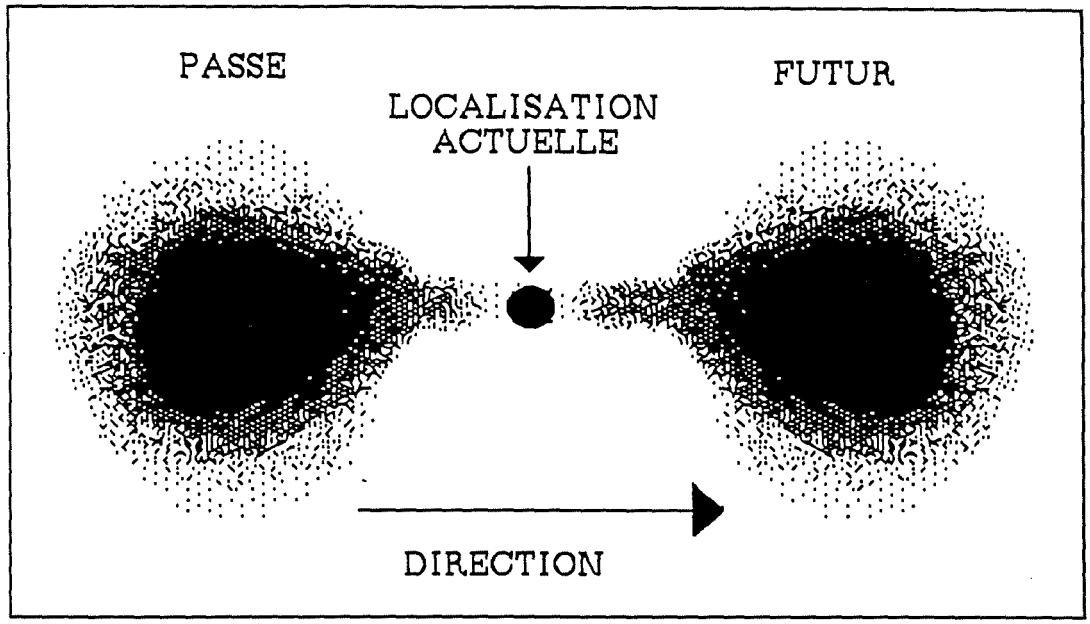


Figure VII.12. : Distribution probable du mouvement de l'objet.

Les zones plus foncées indiquent une plus grande probabilité de localisation et les plus claires une plus faible probabilité.

Le système IPF est constitué de nombreux éléments de traitement largement interconnectés. Chaque élément est assigné à chaque point de l'espace des caractéristiques. L'espace des caractéristiques a ici 4 dimensions : localisation (x,y), temps (t), direction (θ). L'output d'un élément représente la probabilité que le point associé soit inclu dans la solution du problème. Les éléments de traitement sont interconnectés de manière à trouver la meilleure solution globale en trouvant la meilleure solution locale. Par exemple, il existe des connexions positives (excitatoires) de chaque neurone vers des points qui dans la tranche de temps suivante correspondent à la forme de la distribution de la figure VII.12.; il existe aussi des connexions négatives (inhibitrices) entre des éléments qui représentent une même localisation au même moment mais qui ont des directions différentes. La solution émerge après un certain nombre d'itérations entre les différents éléments.

Le réseau IPF a d'abord été testé sur le suivi simultané de 2 cibles (figure VII.13.). Il résoud le problème en choisissant une des huit possibilités de direction (hypothèse de départ) à chaque instant.

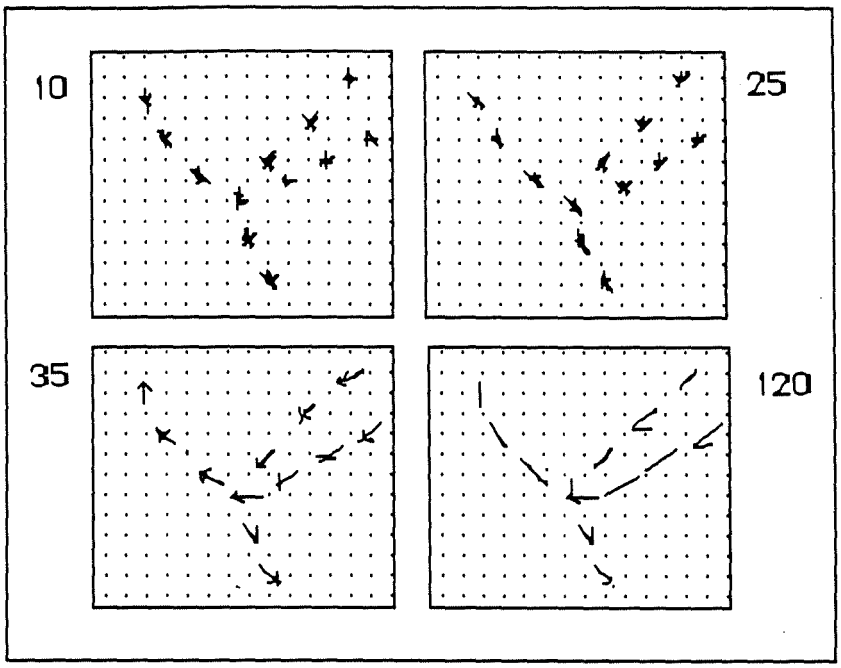


Figure VII.13. : Pistage de 2 cibles via l'IPS. 8 possibilités de direction.

La direction est représentée par un vecteur dont la longueur correspond à la probabilité accordée par le réseau que l'objet se dirige dans cette direction. Intitialement, chaque direction a la même probabilité. Comme le réseau itère (± 120 fois), la meilleure direction émerge pour chaque point. L'IPF a ensuite été utilisé sur 15 cibles avec un temps de résolution du problème identique. Cette propriété a déjà été évoquée au point IV.6. lorsque nous avons dit que le temps de traitement était indépendant du nombre d'états mémorisés.

VII.2.11. Fusion d'image

(S. Grossberg, Center Adaptive vision, Boston)

Beaucoup d'algorithmes d'intelligence artificielle permettent de traiter des problèmes de vision. Cependant, chacun d'eux ne traite qu'un aspect de la vision et se combine très difficilement aux autres, ce qui explique les maigres performances des algorithmes conventionnels de vision.

Le cerveau humain, par contre, combine aisément différentes sources visuelles ambiguës pour générer rapidement une représentation globale et cohérente des couleurs et des formes en profondeur.

Il fallait donc de nouveaux principes et techniques pour réaliser ces fameux problèmes de vision. C'est ce qu'à fait le "Center for Adaptive Systems" de Boston en développant une architecture de vision générale et

automatique. Cette architecture permet de prendre en considération à la fois les frontières des objets, leurs texture, ombre, profondeur, échelle et mouvement pour former une représentation cohérente des couleurs et des formes en profondeur.

Les processus utilisés pour traiter automatiquement les données visuelles humaines sont appelés *segmentation émergente* et *remplissage de caractéristiques.* On les retrouve dans l'architecture automatique via des interactions nonlinéaires entre les deux réseaux neuronaux "**Boundary Countour System**" (BCS) et "**Feature Countour System**" (FCS) (figure VII.14.).

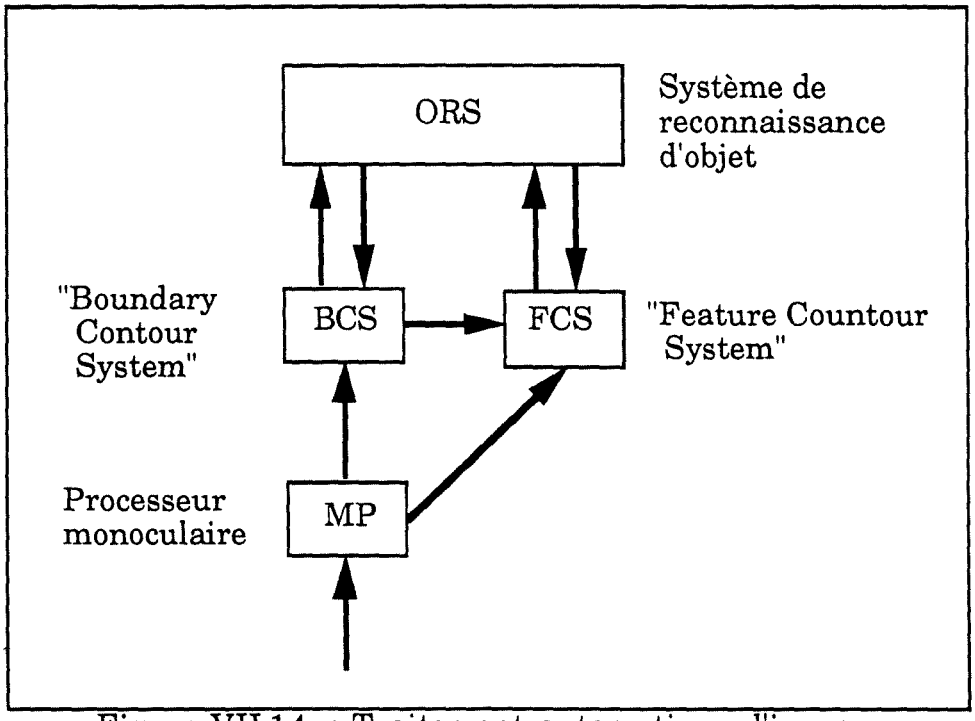


Figure VII.14. : Traitement automatique d'images multi-dimensionnelles bruitées.

Des signaux prétraités par le "Monocular Preprocessor" (MP) sont envoyés à la fois au BCS et au FCS. Le BCS génère une segmentation d'une scène en 3 dimensions. Cette segmentation est entre autre capable de détecter et compléter des frontières et de grouper des textures. Cette segmentation est envoyée à la fois au FCS et au ORS ("Object Recognition System"). L'ORS, à son tour, renvoie les signaux appris au BCS. Ces signaux vont permettre de modifier la structure issue de la segmentation. Le BCS passe les modification opérées sur la segmentation au FCB, lequel va découper la structure en régions perceptuelles pouvant être remplies de couleurs et lumiosités visibles (la segmentation réalisée par le BCS est invisible dans le BCS). Le remplissage est activé par des signaux du MP et, une fois terminée, la représentation peut interagir avec l'ORS.

VII.3. CATEGORISATION DES APPLICATIONS

Les 11 applications présentées ci-dessus ont été choisies pour montrer les différents domaines dans lesquels les réseaux neuronaux constituent une alternative aux technologies conventionnelles, lorsqu'il y en a. Elles ont également été choisies pour leur potentiel d'applicabilité à court-terme.

TOTAL	55	11	54	12	77		22	10		11
SYSTEMES DE DECISIONS	6	2	5		7			1		1
SYS. DYN./BIOLOGIE	7		5	2	7		6	1		1
ROBOTIQUE	6		5	3	8		5	2		2
TRAITEMENT DU SIGNAL	3		2	1	3		1			
RADAR	6	3	7		10		3	1		1
SONAR	2	4			4		1			1
PAROLE	3	1	3	2	6		1	1		1
VISION	19	5	24	4	33		5	4		4
	Nbr de chercheurs	A l'état de concept	Simulation	Implémentation hardware	Nbr d'applications		Utilité < 3 ans	Utilité présente		Les 11 applications présentées

Tableau VII.1. : Résumé des champs d'application, des chercheurs et du statut des réseaux de neurones.

Elles sont issues d'un panel plus général de 77 applications provenant de 55 chercheurs ayant une expérience pratique dans le domaine des réseaux neuronaux (Tableau VII.1.). Ces applications peuvent être répertoriées dans 8 catégories : vision, reconnaissance de la parole, sonar, radar, traitement du signal, robotique, systèmes dynamiques et fonctions cognitives [Darpa,88].

La catégorie *traitement du signal* regroupe les applications qui ne tombent pas explicitement dans les domaines de vision, parole, sonar ou radar. Les *systèmes dynamiques* regroupent des applications qui font intervenir explicitement l'intégration temporelle et qui couvrent aussi des aspects biologiques. Les *systèmes de décision* incluent des applications qui utilisent les réseaux neuronaux pour exécuter des actions de plus haut niveau, telle la gestion de commande & contrôle. Ces problèmes sont en général traités par des techniques d'intelligence artificielle.

Il est en outre indiqué pour chacun de ces domaines, le nombre d'applicatons à l'état de concept, de simulation et d'implémentation hardware. L'utilité actuelle[5] et à court-terme (3 ans) est également renseignée pour chacun des domaines.

Nous constatons que c'est le domaine de la vision qui regroupe le plus d'applications et qui présente la plus grande utilité actuelle. Quatre des applications que nous avons présentées tombent d'ailleurs dans cette catégorie.

Le Tableau VII.2. reprend chacune des 11 applications par ordre croissant de complexité et décrit pour chacune d'elles, la fonction réalisée, le degré de connectivité, les types d'apprentissage, de structure de connexion et le statut. Comme on peut le voir, seules 3 applications fonctionnent de manière efficace sur des problèmes de taille réelle en 1988 : le contrôle de processus, l'identificateur de mots et l'analyseur de risque.

Les fonctions réalisées vont de la correspondance linéaire et non linéaire à des fonctions spécialisées telles que celles du modèle de la rétine, du pistage de cibles ou de la segmentation d'image. Les connectivités évoluent quant à elles de l'interconnexion complète vers l'interconnexion faible et locale. En d'autres termes, la liste des applications définit un ordre de complexité et de spécificité.

Une autre caractéristique intéressante est qu'en même temps qu'on se déplace d'un apprentissage supervisé à un non supervisé puis à une absence d'apprentissage (poids fixes), on passe presque simultanément de réseaux feed-forward à des réseaux feed-back. Si l'on considère

[5] Projection réalisée à partir des chiffres de 1988.

également les applications ayant fait leurs preuves, on peut en conclure que les applications matures tendent à être des systèmes *feed-forwards supervisés*.

Application	Fonction	Connectivité	Appr.	Type	Statut
Pendule inversé	Correspond. linéaire	Complète	Sup	Feedf	Sim
Contrôle de processus	Régression linéaire	Sur mesure	Sup	Feedf	Appl
Identificateur de mots	Classification	Complète	Sup	Feedf	Appl
Sonar	Corresp. non linéaire	Complète	Sup	Feedf	Démo
Analyseur de risques	Corresp. non linéaire	Faible	Sup	Feedf	Appl
Bras élévateur de robot	Corresp. non linéaire	Faible	Sup	Feedf	Démo
Reconnais. de cibles	Optimisation	Sur mesure	Sup	Feedb	Simul
Classifieur d'images	Corresp. non linéaire	En couches	Non Sup	Feedf	Simul
Modèle de la rétine	Vision "front end"	Locale	Poids fixes	Feedf	Démo
Pistage de cibles	Interpolation	Locale	Poids fixes	Feedb	Simul
Fusion d'image	Groupement global	Locale	Poids fixes	Feedb	Simul

Tableau VII.2. : Caractéristiques des 11 applications.

Il peut être également intéressant de classer les applications d'après le *modèle de réseau utilisé*. Sur les 77 applications répertoriées, les approches les plus populaires sont le Perceptron multi-couches utilisant la rétro-propagation et le réseau de Hopfield. Le réseau rétro-propagation est le seul qui s'applique à toutes les catégories énumérées, excepté la biologie. Le tableau VII.3. reprend cette classification pour les 11 applications passée en revue. Alors que le réseau Back-propagation est

toujours à l'honneur, le réseau de Hopfield semble moins utilisé, laissant la place aux réseaux RCE et Paramètre.

Pour la majorité des applications, une solution peut être trouvée par plus de 4 modèle de réseau neuronal. Dès lors, quel réseau est-il le plus adapté pour une application donnée ? Cette question reste à l'heure actuelle en suspend vu l'absence d'analyse comparative d'efficacité[6].

	PI	CP	IM	S	R	BR	RC	CI	R	PC	FI
Perceptron/Adaline	X	X	X		x						
Back-propag. multi-c.	x	x	x	X	x	x	x	x			
Avalanche				x							
Réseau RCE		x	x	x	X	x	x	x			
CMAC	x					X	x				
Réseau Paramètre		x		x	x	x	X	x			
Neocognitron							x	X			
BSB							x			x	
CAM							x	x		x	x
Carte quantitative				x		x	x	x		x	x
Automate cellulaire			x	x				x	X	x	x
"Masking field"			x	x							
BCS/FCS								x		X	X
ART I/II							x	x			
Système dynamique									x	x	x

PI : pendule inversé;
CP : contrôle de processus;
IM : identificateur de mots;
S : Sonar;
R : analyseur de risque;
BR : bras de robot;
RC : reconnaissance de cibles;
CI : classifieur d'images;
R : rétine;
PC : pistage de cibles;
FI : fusion d'images.

Tableau VII.3. : Implantations réelles et potentielles des 11 applications.

VII.4. RESEAUX NEURONAUX ET SYSTEMES EXPERTS

VII.4.1. Du point de vue de l'acquisition des connaissances

Les processus d'acquisition des connaissances des systèmes experts sont fondamentalement différents de ceux utilisés dans les réseaux neuronaux. Dans le premier cas, on observe tout d'abord le processus de résolution de problème des experts humains pour capturer leurs

[6] Excepté pour la classification des classifieurs effectuée par Lippmann (Point VI. Taxonomies des réseaux neuronaux).

connaissances. Ensuite, ces connaissances spécifiques à un domaine sont encodées sous la forme de règles, par exemple, et implantés dans un programme d'ordinateur qui peut les utiliser pour résoudre des problèmes spécifiques.

Les systèmes biologiques procèdent autrement : l'acquisition des connaissances se réalise à partir de l'*expérience directe* dans un environnement naturel. *Les réseaux neuronaux utilisent aussi ce principe de l'expérience directe* -via un apprentissage par l'exemple- *et sont donc très similaires aux systèmes biologiques du point vue de l'acquisition des connaissances.*

VII.4.2. Du point de vue de l'utilisation

La plupart des décisions que tout un chacun prend chaque jour consiste en des problèmes de classifications : vais-je accorder un crédit à un tel ? Devons-nous promouvoir ce produit ou non ?

Les réponses possibles à ces questions constituent des catégories. Lorsque ces classifications que nous devons faire conduisent à des actions, on parle alors de "decision making by pattern classification". C'est là typiquement le rôle qui est assigné aux réseaux neuronaux : faire de la classification après avoir appris à partir d'exemples.

Les systèmes experts, quant à eux, examinent une caractéristique particulière et essayent d'approcher une solution possible pas à pas.

La question qui se pose ici est de savoir quand on doit utiliser un réseau neuronal à la place d'un système expert. La réponse est que si un problème peut être résolu par un système expert alors il faut utiliser un système expert. Il existe cependant des situations dans lesquelles il est plus rentable d'utiliser un réseau neuronal. Pour cela, certaines conditions doivent être remplies :

1) toutes les données nécessaires pour prendre une décision doivent être réunies, contrairement aux systèmes experts;
2) les inputs doivent être très corrélés. Par exemple, dans le problème de l'octroi d'un crédit, le montant de la demande d'emprunt est fortement corrélé au salaire du demandeur;
3) les catégories possibles se chevauchent ou ne sont qu'au nombre de deux;
4) le problème possède des catégories dont les limites changent avec le temps.

VII.5. AVANTAGES DES RESEAUX NEURONAUX

D'après une étude réalisée par DARPA [DARPA,88,265-270] auprès de 55 chercheurs, les principaux avantages des réseaux neuronaux seraient les suivants (par ordre décroissant d'importance) :

1- Vitesse;
2- Parallélisme;
3- Facilité d'implémentation;
4- Apprentissage supervisé.

Parmi les autres avantages mis en évidence par ces chercheurs, on retrouve entre autre l'auto-organisation, l'apprentissage non supervisé, la généralisation, la tolérance aux fautes et l'immunité contre le bruit.

	PI	CP	IM	S	R	BR	RC	CI	R	PC	FI	AC
Vitesse	○		○	○	○	○	●	○	○	○	○	✷
Parallélisme		○	●	○	○	○	○	○	○	○	○	✷
Facilité d'implém.	●	○	○	○	○	○						
Apprentis. supervisé	○	○	○	●	○	○	○					✷
Contrôle hiérarchique					●	○						✷
Auto-organisation				○	○			○		●	○	
Fusion de données		○			○	○					●	✷
Apprentis. non superv.									●			
Généralisat.	○	○		○	○	●		○		○	○	✷
Tolérance aux fautes		○		○	○	○	○	○	○	○	○	✷
Spatio-temporalité		○							●	○	○	✷
Beaucoup de données		●			○	○			○			✷
Immunité au bruit		○		○	○	○	○	○	○	○		✷

PI : pendule inversé;
CP : contrôle de processus;
IM : identificateur de mots;
S : Sonar;
R : analyseur de risque;
BR : bras de robot;
RC : reconnaissance de cibles;
CI : classifieur d'images;
R : rétine;
PC : pistage de cibles;
FI : fusion d'images;
AC : approche conventionnelle.

Tableau VII.4. : Avantages des réseaux neuronaux et technologies conventionnelles.

Cependant, ces avantages ne sont pas l'apanage des seuls réseaux neuronaux. En effet, considérons le tableau VII.5. qui reprend ces avantages pour les 11 applications présentées au point VII.2. ainsi que pour les techniques conventionnelles. La qualité la plus marquante de l'application est indiquée par un (●) et les autres avantages perçus par un (○). On constate que la plupart des avantages énumérés sont aussi partagés par les techniques conventionnelles. Seuls l'apprentissage non supervisé, l'auto-organisation et la facilité d'implémentation sont caractéristiques des réseaux neuronaux.

S'il semble que la *facilité d'implémentation* soit une propriété marquante de la modélisation neuronale, cela provient du fait que la programmation de ce type de réseaux se réalise *par exemples*. Cependant, à moins que le problème à résoudre n'ait une taille triviale, une implémentation sérieuse utilisant un réseau neuronal requerra de l'attention, du jugement et de l'intuition.

En ce qui concerne *l'auto-organisation*, il semble qu'il faille distinguer deux catégories : l'*auto-organisation à petite échelle* et l'*auto-organisation globale*. On retrouve la première, par exemple, dans les réseaux Back-propagation, dans le classifieur sonar ou dans le "Nestor Learning System" qui peuvent organiser leurs neurones comme des champs réceptifs. La deuxième est caractéristique des systèmes vivants, plus complexes donc et que l'on retrouve dans certains systèmes auto-supervisés tel le Darwin III. Ce dernier est une simulation de bras de robot auto-supervisé qui ne se base que sur un feed-back visuel et de toucher de l'environnement.

VIII Implémentations software et hardware

VIII.1. INTRODUCTION

De nombreux produits, software ou hardware, sont actuellement disponibles sur le marché. Ce chapitre a pour but de donner, pour chacun des principaux produits, une brève description ainsi que ses caractéristiques principales.

Ce chapitre sera composé de quatre parties : dans la première nous ferons une brève mise au point sur les termes utiliséspour décrire les performances des réseaux neuronaux; dans la seconde nous décrirons dix produits software actuellement disponibles; dans une troisième partie nous développerons un système mixte (software et harware) de réseau neuronal; enfin nous passerons en revue quelques produits hardware.

VIII.2. TERMINOLOGIE

Lorsqu'on parle de réseaux neuronaux et de performances, il n'est pas possible de parler d'instructions par seconde ou de capacité en mot mémoire. De part la structure des réseaux, et du fait qu'une information ne peut être localisée de manière précise, on parlera du nombre d'interconnexions pour la capacité et du nombre d'interconnexions par seconde pour la vitesse (tableau VIII.1.). Mais il faut être vigilant et s'assurer de l'équivalence des termes avant toute comparaison. Il est en effet fréquent que certains calculs ou sommations ne soient pas pris en compte, lors du calcul de la vitesse ou de la capacité d'un réseau.

	capacité mémoire	vitesse (opérations par sec.)
ordinateurs digitaux	mots	IPS, FLOPS
réseaux neuronaux	Interconnexions	Interconnexions/sec.
IPS = instructions par seconde; FLOPS = opération en virgule flottante par seconde.		

Tableau VIII.1. : Performances.

VIII.3. SOFTWARES

VIII.3.1. SYSPRO de Martingale Research.

Le SYSPRO offre un système de démonstration écrit en Fortran. C'est essentiellement un outil de simulation par lequel une implémentation BEP ("Back-Error-Propagation") a été écrite. Le SYSPRO complet comprend le code source, et nécessite au minimum une machine du type XT et 512 K de mémoire centrale. Ce système permet de fixer les "momentum" et taux d'apprentissage. [SCHWA,88-1].

VIII.3.2. NEURALSHELL de Ward Systems Group.

Ce software est d'un emploi facile et permet de résoudre de petits problèmes en utilisant le BEP ("Back-Error Propagation"). Un PC avec 256 K de mémoire centrale est nécessaire. Le nombre de neurones d'entrée et de sortie ainsi que le type des valeurs d'entrée et de sortie (binaires ou continues) peuvent être définis par l'utilisateur. Le NEURALSHELL créera un réseau à trois couches de neurones. Le nombre de neurones de la couche intermédiaire sera le même que celui de la couche d'entrée. Le système choisit alors, par défaut, toutes les valeurs nécessaires au calcul (momentum, ...). Le fait de fixer automatiquement le nombre de neurones de la couche intermédiaire ainsi que le momentum est une limitation importante de ce système. Pour le problème du OU-Exclusif par exemple, il n'y a pas de convergence car le momentum est trop élevé. [SCHWA,88-1].

VIII.3.3. AWARENESS de Neural Systems.

Awareness est avant tout un ensemble didactique car toutes les architectures et tous les paramètres des réseaux sont prédéfinis. Il y a quatre modèles de base :

- BEP (Back-Error Propagation),
- Hopfield,
- problème du voyageur de commerce de Tank,
- Kohonen.

Le modèle BEP consiste en un réseau à trois couches de neurones. La première couche est composée de deux éléments, la deuxième en contient huit et la dernière un seul.

AWARENESS nécessite un PC avec 256 K de mémoire centrale minimum. [SCHWA,88-1].

VIII.3.4. MAC BRAIN de Neuronics.

MAC BRAIN offre cinq règles d'apprentissage et la possibilité d'en définir d'autres. MAC BRAIN permet également de changer la fonction de transfert. Un défaut important de cette implémentation software est l'impossibilité de détruire des neurones qui auraient été créés accidentellement. MAC BRAIN est utilisable sur Mac II et Mac SE et demande 1 MB de mémoire centrale. [SCHWA,88-1].

VIII.3.5. COGNITRON de Cognitive software.

Le COGNITRON est utilisable, tout comme MAC BRAIN, sur Mac II et Mac SE, mais ne nécessite que 400 K de mémoire centrale.

L'utilisation du COGNITRON est relativement aisée. Sept exemples de réseaux sont implémentés. De plus le COGNITRON permet de construire ses propres réseaux. Les fonctions de transfert et d'apprentissage sont implémentées en Lisp, mais d'autres fonctions (écrites en C, Pascal ou Fortran) peuvent être importées. [SCHWA,88-1].

VIII.3.6. NEURALWORKS de NeuralWare.

Le NEURALWORKS version 1 fonctionne sur PC et nécessite 512 K de mémoire centrale. Il offre un très grand contrôle sur chacun des éléments des réseaux. Huit types de réseaux sont disponibles : Perceptron, Adaline, Madaline, Hopfield, Brain State in a Box (BSB), Bidirectional Associative Memory (BAM), Back-Error-Propagation (BEP) et Counterpropagation. L'interface est du type Macintosh. Une version II existe également. Celle-ci est également disponible sur station de travail Sun. [SCHWA,88-1]. Ce logociel fait l'objet d'une description plus complète à l'annexe D.

VIII.3.7. ANSim de Saic.

ANSim est disponible en deux versions. Chacune contient 13 exemples de réseaux neuronaux. La version de base nécessite un PC-AT; une carte EGA, Windows et une souris. Dans la deuxième version, une copie de Windows ainsi qu'une souris sont fournis. Ce logiciel présente un certain nombre d'avantages. Il possède un système d'aide et un graphe des différentes erreurs dans le réseau peut être obtenu, ce qui permet de suivre à volonté l'apprentissage du réseau. De plus, l'état de celui-ci peut être sauvé à tout moment lors de la phase d'apprentissage. Le logiciel n'autorise pas l'introduction des valeurs d'entrée depuis le clavier. Elles ne peuvent être spécifiées que par fichiers. ANSim peut travailler avec les formats ASCII, Lotus et dBase. Saic a également créé son propre accélérateur hardware pour ce software (Delta I. et II. 22 MFLOPS). [SCHWA,88-1]. Ce logiciel fait également l'objet d'une description plus détaillée à l'annexe E.

VIII.3.8. AI Net de AI Ware.

AI Net est un logiciel tournant sur Windows. Il possède les mêmes particularités que Neuralshell mais est plus flexible que ce dernier. Il est plus facile de développer un réseau BEP (Back-Error Propagation) à l'aide d'AI Net. AI Ware fournit également un accélérateur hardware permettant de réduire le temps de calcul d'un facteur 20. AI Net tourne sur PC-AT. [SCHWA,88-1].

VIII.3.9. ANZA de Hecht-Nielsen Corporation (HNC).

HNC a principalement développé un accélérateur hardware et un software en relation avec cet accélérateur. ANZA contient une implémentation complète du réseau "Counterpropagation" développé par R. Hecht-Nielsen. [SCHWA,88-1].

VIII.3.10. Nestor Development System (NDS) de Nestor Corporation.

Nestor Corp. a construit son propre réseau neuronal, supervisé, le RCE (Restricted Coulomb Energy). Ce réseau est développé dans le NDS. Celui-ci nécessite un PC-AT, une carte EGA, une souris et 640 K de mémoire centrale. Si un problème peut être résolu par de multiples hyperplans dans une simple hypersphère, alors souvent un simple élément RCE est nécessaire. Le NDS demande un compilateur C. Nestor Corp. espère fournir dans le futur une interface pour fichiers Lotus, dBase et ASCII. Un aspect important du RCE est sa vitesse d'apprentissage par rapport à d'autres logiciel implémentant des réseaux BEP. [SCHWA,88-1].

VIII.4. UN SYSTEME INTEGRE POUR LA SIMULATION DE RESEAUX NEURONAUX

Cette partie a pour objet d'exposer comment un hardware particulier a été créé pour simuler un réseau neuronal. Ce hardware est en fait un ordinateur parallèle qui fournit une puissance équivalente à un mainframe dans le cas particulier de la simulation de réseau neuronal, le tout au prix d'un mini-ordinateur.

L'ensemble est complété par un système d'exploitation qui assiste l'utilisateur dans le partitionnement des problèmes et qui permet l'exécution en parallèle -sans modification de code- des programmes développés sur d'autres machines.

VIII.4.1. Le modèle

Le réseau est constitué d'unités (neurones) reliées entre elles. Chaque sortie d'unité est communiquée comme entrée aux autres unités auxquelles elle est reliée. L'excitation de chaque unité est déterminée en faisant le produit de ses entrées par ses poids. Le résultat de cette somme de produits est ensuite passé comme argument à la fonction de transfert, non linéaire, du neurone.

Vu le nombre important de connexions dans le système, les neurones ont été regroupés en structures régulières (matrice à deux

dimensions par exemple). Le système ainsi construit ne nécessite plus que des communications entre unités de groupement. Même de cette façon, le système va être très vite saturé au niveau communications et ce, malgré l'utilisation d'un processeur de communication. La solution retenue est alors de régulariser la structure de chaque unité en interconnectant complètement tous les neurones d'une unité et en limitant le nombre de connexions avec l'extérieur. Si une unité contient M entrées et N sorties, M * N poids seront nécessaires mais seules M + N connexions inter-unités seront utilisées.

Les sorties et les entrées des différentes unités peuvent être reliées à des places arbitraires, permettant ainsi de modifier la structure du réseau à souhait. Les interconnexions entre unités sont réalisées au niveau software.

Deux fonctions peuvent être identifiées comme limitant la vitesse de simulation des réseaux neuronaux :

- le calcul de la somme des produits des sorties et des poids,
- la modification des poids pendant la phase d'apprentissage.

La première est toujours la même tandis que la seconde peut être légèrement différente suivant l'algorithme d'apprentissage. Ces opérations doivent être effectuées pour chacun des poids et prennent beaucoup de temps calcul.

VIII.4.2. Le simulateur Netsim

La base de la machine est un ensemble distribué de simulateurs de réseaux neuronaux autonomes appelés "Netsim" ("Network simulator"). Chaque Netsim est constitué d'un microprocesseur, d'un circuit intégré de solution (un coprocesseur spécialisé implémentant à haute vitesse des fonctions des réseaux neuronaux), et d'un circuit intégré de communication autorisant l'interconnexion de nombreux Netsim. L'ensemble des cartes Netsim connectées entre elles constitue le processeur parallèle "Griffin".

Le Netsim est un réseau neuronal complet avec mémoire, un microprocesseur local et un moyen de communication. Il peut être utilisé comme simulateur d'un réseau neuronal unique, ou il peut faire partie d'une machine de simulation plus importante.

VIII.4.3. Le Griffin

En vue d'améliorer l'usage que l'on peut faire d'un tel système, il est nécessaire de le baser sur une architecture existante et pour laquelle nombre de logiciel ont été développés (figure VIII.1.). La carte est composée d'un Intel 80186, d'une mémoire Ram et d'une mémoire Eprom

contenant le Bios et les instructions qui servent à programmer le proceseur. De cette façon, un certain nombre de logiciels utilisant des langages de haut niveau peuvent tourner sur les cartes Netsim (Turbo C de Borland par exemple). Un corollaire de ce choix est que l'on peut utiliser un PC standard pour développer des applications destinées à tourner sur des Netsims. Cela ce fait en créant une version de la carte Netsim dans laquelle le microprocesseur local et la mémoire sont enlevés et remplacés par des accès aux bus du PC. Enfin, un PC peut remplacer une carte Netsim.

Les communications entre les noeuds et l'ordinateur hôte s'effectuent via deux canaux. Le premier, la ligne série à grande vitesse, permet aux noeuds de communiquer entre eux et avec l'hôte. Le second, le canal des messages, permet à l'ordinateur de communiquer avec les noeuds en utilisant des adresses absolues.

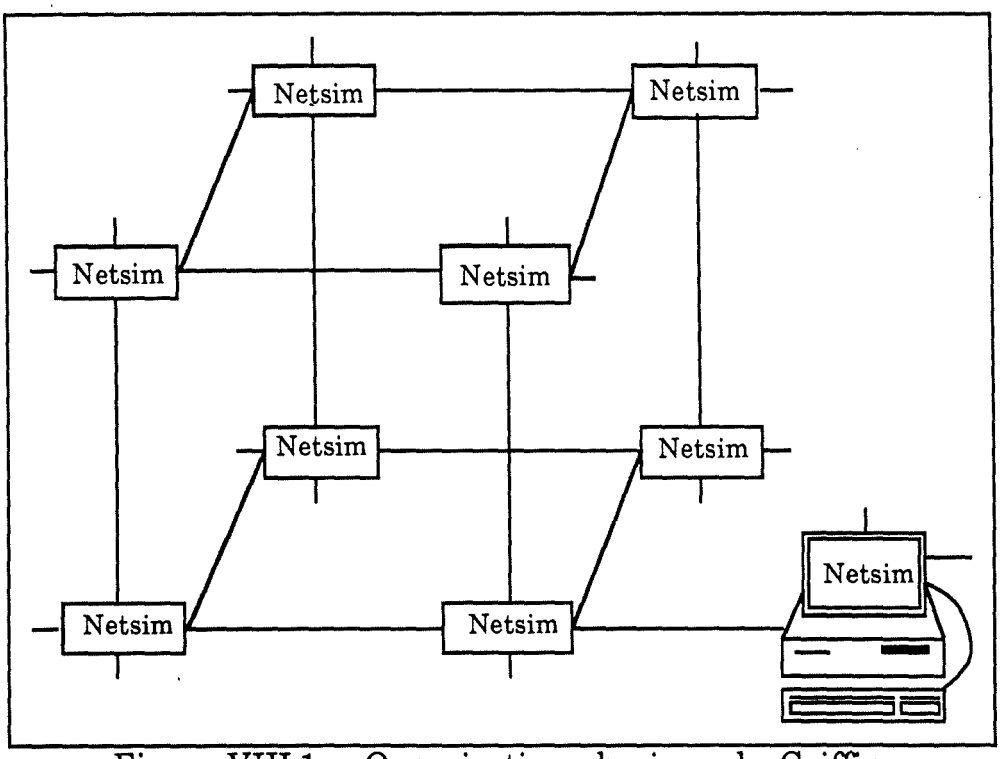


Figure VIII.1. : Organisation physique du Griffin.

VIII.4.4. L'environnement software

L'environnement logiciel est constitué de deux parties : l'interface de l'hôte et le système d'exploitation local. Le plus compliqué est le système d'exploitation local parce qu'il doit offrir à l'utilisateur la possibilité d'intégrer ses propres routines avec un minimum d'effort (routines de diagnostic, de set-up). Ces routines ne doivent pas être critiques du point de vue temps CPU pour ne pas ralentir la simulation.

La plus grande partie de la mémoire d'un noeud local est divisée en une série de sous-routines, chacune d'elles ayant ses propres zones de données et de paramètres. La machine fonctionne en recevant des instructions depuis l'hôte et qui sont ensuite traduites en appels aux sous-routines.

Les applications Netsim sont développées à partir de l'hôte et sont ensuite chargées dans les noeuds. La partie interface-utilisateur du programme doit comprendre des fonctions telles que le choix de la forme du réseau et de la fonction de transfert. [GARTH,86].

VIII.5. HARDWARE

VIII.5.1. Les différentes architectures

Les réseaux neuronaux peuvent être très puissants pour résoudre certains problèmes. Mais, ces solutions doivent être exploitées de façon à profiter au maximum de l'avantage "physique", résultant du parallélisme massif, procuré par ces réseaux. Il faut donc que ces réseaux ne se limitent pas seulement à un problème "software".

Quelques **neuro-ordinateurs** ou "**machines spécialisées capables d'implémenter, efficacement et à un coût raisonnable, des réseaux neuronaux**" existent (en annexe B on trouvera trois tableaux reprenant les principaux simulateurs hardware ainsi que leurs caractéristiques). Un **neuro-ordinateur** est principalement un ensemble d'unités de traitement inteconnectées, et fonctionnant en parallèle.

Trois paramètres principaux permettent de définir une classification des différentes architectures (figure VIII.2.) :

- la complexité d'une unité élémentaire;
- le nombre d'unités indépendantes;
- la possibilité de programmer le graphe des connexions et le mode de communication entre les unités.

L'ensemble des architectures permettant la mise en oeuvre de réseaux neuronaux peut être divisée en quatre grande classe :

- les ordinateurs séquentiels ordinaires;
- les calculateurs parallèles ou vectoriels conventionnels;
- les neuro-ordinateurs à usage général; ceux-ci se rapprochent des calculateurs parallèles, mais ils sont en général intégrés dans une architecture spécialisée pour simuler des réseaux neuronaux (figure VIII.3.);
- les neuro-ordinateurs spécialisés; ils sont composés d'un ensemble d'unités simples et non programmables reliées par des

connexions fixes ou non; cette architecture est généralement destinée à des problèmes particuliers.

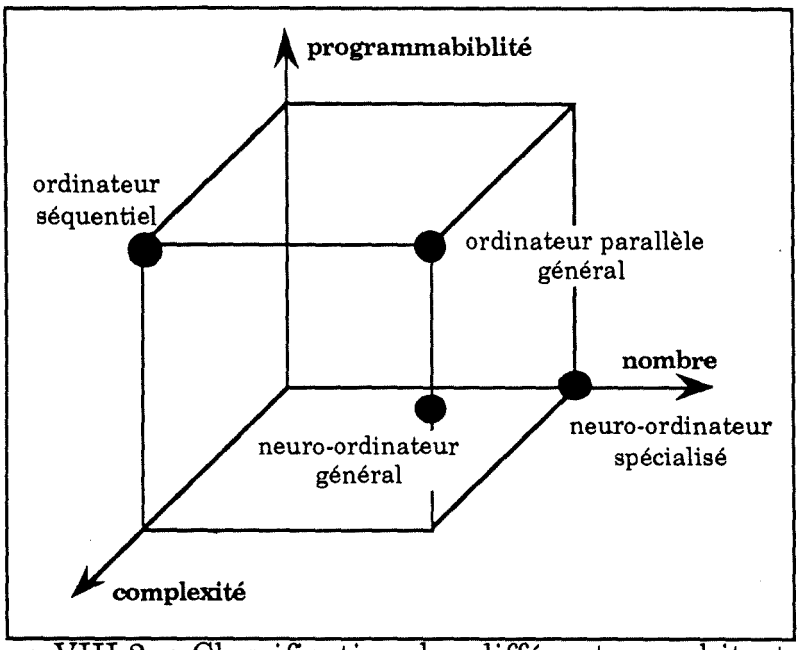


Figure VIII.2. : Classification des différentes architectures.

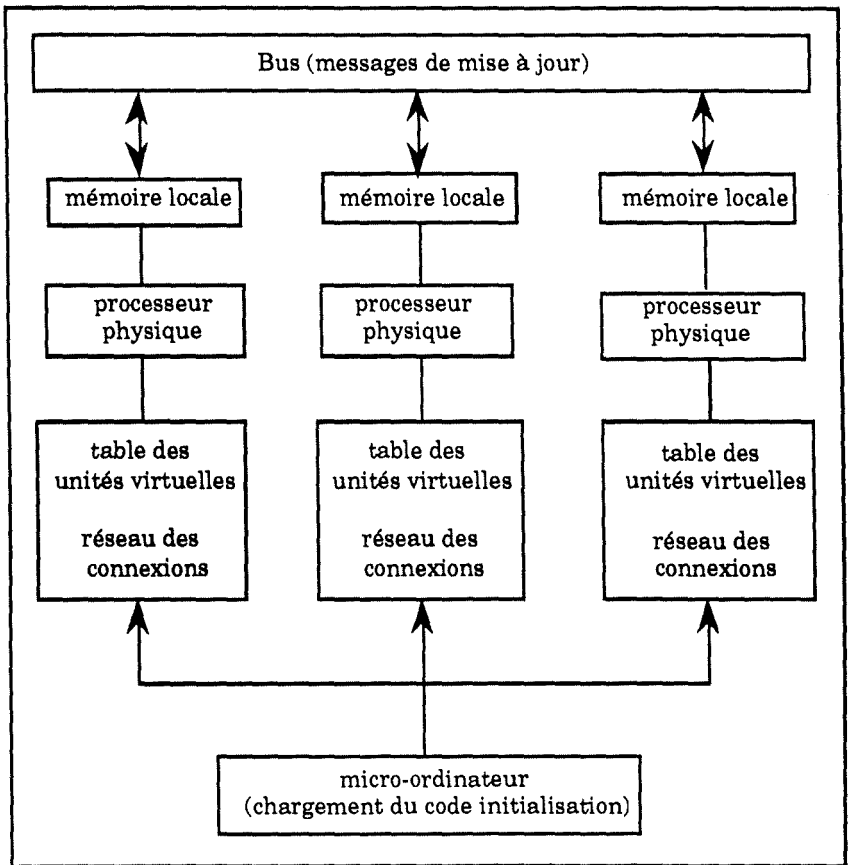


Figure VIII.3. : Schéma d'un neuro-ordinateur général.

Le tableau VIII.2. ci-dessous reprend quelques simulateurs hardware de réseaux neuronaux et donne pour chacun d'eux le nombre maximal d'interconnexions ainsi que leur vitesse en nombre d'interconnexions par seconde.

	interconnexion	interconnexion par seconde
PC/AT	100 K	25 k
SUN3	250 K	250 K
VAX	32 M	100 K
SYMBOLICS	10 M	35 K
ANZA	500 K	45 K
DELTA 1 (*)	1 M	10 M
TRANSPUTER (*)	2 M	3 M
MARK III, IV	1 M	500 K
ODYSSEY	256 K	20 M
MX-1/16 (*)	50 M	120 M
CM-2 (64 K)	64 M	13 M
WARP (10)	320 K	10 M
BUTTERFLY (64)	60 M	8 M
CRAY XMP 1-2	2 M	50 M
(*) Performances espérées		

Tableau 8.2. : Performances de réseaux hardware. [DARPA,88].

Le MX-1/16 est actuellement une architecture expérimentale mais les chercheurs espèrent obtenir des performances de l'ordre de 120 M d'interconnexions par seconde et de 50 M d'interconnexions. Les différences entre les différents systèmes sont relativement importante, mais, plus le système est puissant, plus il est difficile de le programmer.

Différentes techniques hardware permettront, à l'avenir, de repousser les limites de capacité (interconnexions) et de vitesse (interconnexions par seconde) :

- le "Gallium ArSenide" et le "Charge-Coupled Devices" permettront l'augmentation de la vitesse;
- le développement des mémoire "RAM" (Random-Access Memory) et des "puces à trois dimensions" influenceront la capacité des réseaux;
- le "Multiprocessing" devrait modifier et la vitesse et la capacité.

La figure VIII.4. ci-dessous montre l'état des performances actuelles ainsi que les perspectives pour les prochaines années.

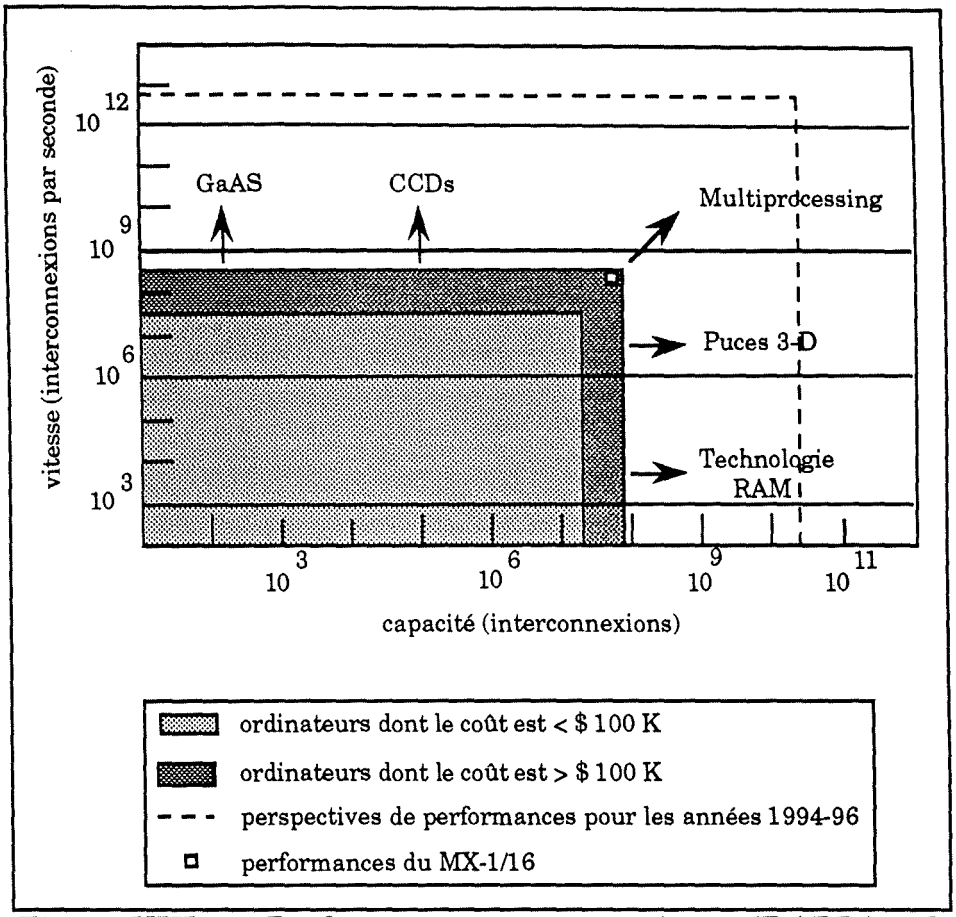


Figure VIII.4. : Performances et perspectives. [DARPA,88].

Les réseaux neuronaux sont simulés en mode batch ou en mode interactif. Le mode batch utilise principalement deux types d'architecture alors que le mode interactif en utilise généralement trois. La figure ci-dessous (figure VIII.5.) reprend ces différents modes et leurs types d'architectures.

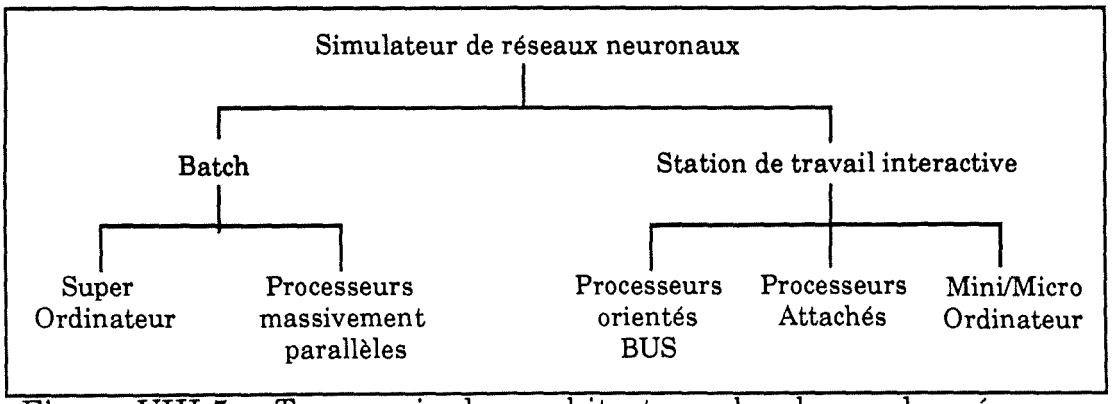


Figure VIII.5. : Taxonomie des architectures hardware des réseaux.

Les super ordinateurs

Les super ordinateurs sont des ordinateurs à très hautes vitesses. Leur architecture est généralement basée sur des semi-conducteurs à hautes vitesses (ECL ou GaAS). Le nombre de processeurs parallèles est relativement faible (<20).

Les processeurs massivement parallèles

Cette catégorie de processeurs donne de très hautes vitesses d'exécution grâce à l'utilisation d'un grand nombre de processeurs (jusqu'à plusieurs milliers). Cependant, de grandes différences existent quant aux communications entre ces différents processeurs. Celles-ci constituent un point important de ce type d'ordinateur, et elles peuvent constituer un goulet d'étranglement et donc faire chuter considérablement les performances.

Les processeurs orientés BUS

Dans cette architecture, des processeurs (< 20) sont connectés sur un BUS. Ce type de machine rencontre beaucoup de succès, mais la vitesse actuelle des BUS constitue une limitation importante.

Les processeurs attachés

Ce type d'architecture consiste en de petits réseaux neuronaux couplés à un mini ou micro-ordinateur. Cette architecture offre un bon rapport prix/performances.

Les mini/micro processeurs

Les mini et micro-processeurs composent l'architecture traditionnelle. Celle-ci offre comme avantage de nombreux supports graphiques et software. Malheureusement les performances sont assez médiocres.

Les autres types d'architectures

Certaines machines n'appartiennent pas à l'une ou l'autre catégorie, mais essayent de combiner les différentes techniques et d'optimiser les performances. C'est notamment le cas pour le "Parallon 3" et le "RP3".

Chaque architecture de réseaux neuronaux peut prendre différentes formes physiques :

- électronique ("**direct VLSI**") (tout est composé d'appareils et de circuits électroniques); cette technologie est limitée au réseau à faible

densité de connexions; elle demande beaucoup de mémoire car tous les poids doivent être mémorisés;

- "**Analog VLSI**"; cette technique est plus intéressante car elle permet de mémoriser les poids des connexions en les implémentant dans des résistances;
- électro-optique (des signaux optiques relient des éléments de traitement électroniques);
- entièrement optique (des signaux lumineux relient des éléments de traitement optiques); cette technique de part sa nature en trois dimension permet de travailler avec des réseaux à très grande densité de connexions.

Il faut noter que des combinaisons de ces différentes technologies sont toujours possibles.

Toute ces machines sont configurées comme coprocesseur d'un ordinateur standard de série, qui agit comme hôte. Le "neurocomputer" coprocesseur est relié à l'hôte via un bus de données ou une liaison périphérique standard (Ethernet par exemple).

En tant que coprocesseur, le "neurocomputer" se comporte comme un autre type de périphérique. L'hôte donne et appelle des données au "neurocomputer" via des routines software. [HECHT,88].

VIII.5.2. Quelques simulateurs hardware

VIII.5.2.1. Mark III et Mark IV

R. Hecht-Nielsen a développé les accélérateurs Mark III et Mark IV. Dans le Mark III des processeurs virtuels de neurones multiplexent les processeurs physiques. Le Mark III contient 8100 éléments de traitements (transistors) et 417.000 connexions (résistances) et tourne comme un périphérique sur Vax. L'implémentation Mark III-Vax d'un réseau neuronal permet d'augmenter la vitesse d'exécution d'un facteur 29 par rapport à la même implémentation en Pascal sur MicroVax.

L'accélérateur Mark IV utilise un système ADAPT ("Adaptive Distributed Analog Processor Technology") qui permet d'implémenter une plus grande variété de réseaux neuronaux.

VIII.5.2.2. NEP (Network Emulation Processor)

Le NEP fournit un support hardware efficace pour l'émulation de réseaux neuronaux importants. Il est utilisé comme émulateur pour les stations de travail PAN ("Parallel Associative Network") sur IBM PC. Un NEP peut modifier un réseau neuronal, de taille maximale, jusqu'à 30 fois par seconde. Grâce à ses interfaces très rapides, plusieurs NEP peuvent être placés en cascade de façon à augmenter la taille des

réseaux. Dans ce cas chaque NEP correspond à une partie précise du réseau.

VIII.5.2.3. Butterfly

L'ordinateur "Butterfly" a été construit par "Bolt Beranek and Newman Inc.". La configuration de base est du type MIMD ("Multiple-Instruction/Multiple-Datastream") et comprend entre quatre et 256 processeurs. Chaque processeur, élément de base de l'ordinateur "Butterfly", appelé "Processor Node" (PN), utilise un microprocesseur Motorola 68020 capable d'exécuter un million d'instruction par seconde. Chaque PN possède une mémoire pouvant aller de un à quatre Mbytes. De plus chaque processeur peut accéder à toutes les autres mémoires.

Les communications inter-processeurs se font grâce à des techniques semblables à la communication par paquets. Le développement software d'applications est fait sur DEC VAX ou sur station de travail SUN tournant sous UNIX 4.2. Ce système supporte le C et le Fortran 77. Une version LISP est prévue.

VIII.5.2.4. CAPP (Content Adressable Parallel Processor)

Le CAPP est un ordinateur du type SMID (Single-Instruction/ Multiple-Datastream) construit à l'université du Massachussets. Le processeur parallèle contient 262.144 cellules arrangées en un tableau de 512 sur 512, chaque cellule consistant en 32 bits de mémoire statique. Une cellule est connectée à ses quatre voisins les plus proches.

VII.5.2.5. Connection machine

La "Connection Machine" de Thinking Machines Corp. contient 65536 processeurs et fourni une puissance de 1000 MIPS. On peut subdiviser le système en quatre parties égales utilisant leur propre flux d'instructions exécutées sur un quart des processeurs. Cela devient donc un ordinateur du type MSMID (Multiple Single-Instruction/ Multiple-Datastream). Chaque processeur possède 4 Kbytes de mémoire centrale.

Deux types de communications sont possibles entre les différents processeurs. Le premier type consiste en un réseau où les processeurs sont connectés avec les voisins se situant au Nord, Sud, Est et Ouest. Dans le second type de connexions, le CMR (Connection Machine Router) connecte les 65536 processeurs à 16 autres processeurs physique dont les adresses diffèrent seulement d'un bit. Le "Router" permet la communication entre tous les processeurs.

La "Connection Machine" supporte le langage assembleur REL-2, ainsi que les langage C et LISP.

VIII.5.2.6. GAPP (Geometric Array Parallel Processor)

Le GAPP, développé par "Martin Marietta Corp." et commercialisé par "NCR corp's Microelectronics Division", est un processeur du type SIMD (Single-Instruction/Multiple-Datastream) et est composé d'un "tableau" de 6 fois 12 PE (Processing Element). Sa puissance permet de faire 921 millions d'additions par secondes.

Chaque PE est connecté avec ses voisins se situant aux quatre points cardinaux. De plus un bus de communication permet d'entrer des données au sud et de les sortir au nord sans interférer sur la puissance de calcul de chacun des éléments.

VIII.5.2.7. iPSC

Le système multi-processeurs iPSC d'INtel Scientific Computer peut opérer simultanément avec 128 unités connectées en réseau hypercube. Chaque unité consiste en un microcomputer équipé de l'Intel 80286 ou 80287. Chaque noeud (unité) possède sa propre mémoire ainsi que ses propres canaux de communication. Un de ses canaux est utilisé comme canal Ethernet et fourni un accès direct au gestionnaire du Cube.

Le gestionnaire tourne sous les système d'exploitation XENIX ou UNIX et supporte les langages LISP, FORTRAN, C et l'Assembleur.

Le système iPSC-VX est un système de vecteur concurrent construit sur la base du iPSC. Ce système fournit de très hautes performances (1280 MFLOPS pour la version iPSC-VX/64).

VIII.5.2.8. MPP (Massively Parallel Processor)

Le MPP a été construit pour le "NASA Goddard Space Flight Center" par Goodyear Aerospace Corp. Ce processeur est du type SIMD (Single-Instruction/Multiple-Datastream) et possède 16.384 éléments de traitement (PE).

Chaque PE est un processeur et possède sa propre mémoire centrale, pouvant aller jusqu'à 65.536 bits. Chacun des éléments communique avec ses quatre voisins les plus proches.

Le MPP a été implémenté sur un DEC PDP-11 travaillant sous le système d'exploitation DEC's RSX-11M

VIII.5.2.9. RP3 (Research Parallel Processor Project)

Le RP3 a été conçu comme un ordinateur MIMD (Multiple-Instruction/Multiple-Datastream) par IBM. Il possède 512 "processor/memory elements" avec un réseau d'interconnexion. Un système complet devrait fournir 1.3 GIPS, 800 MFLOPS, 1-2 Gbytes de

mémoire principale et un taux d'entrées/sortie de 192 Mbytes par seconde. Chaque PME contient notamment un processeur 32-bits et 4 Mbytes de mémoire centrale.

VIII.5.2.10. WARP

Le WARP est un ordinateur du type MIMD (Multiple-Instruction /Multiple-Datastream) et a été dessiné et construit à l'université de Carnegie Mellon. Il offre de haute performance de l'ordre de 100 MFLOPS. Il consiste en 10 cellules identiques, chacune contenant deux processeurs à virgule flottante pouvant donner chacun 5 MFLOPS. Chaque cellule peut être programmée individuellement pour exécuter différents calculs. Le WARP peut être placé en interface avec un VAX 11/780 via un ordinateur ayant 1 Mbytes de mémoire centrale et une largeur de bande de 24 Mbytes par seconde.

[DARPA,88], [DAVAL,89], [HECHT,88], [KOLON,87].

IX Quid des réseaux neuronaux

IX.1. INTRODUCTION

Le but de ce chapitre est de faire une mise au point sur ce qui se dit sur les réseaux neuronaux ainsi que de voir quelles sont les limites et les perspectives de cette technologie pour les 25 prochaines années. Pour ce dernier point nous ferons appel à une enquête menée par la DARPA (Defense Advanced Research Project Agency) américaine, auprès d'une cinquantaine de chercheurs.

IX.2. MISE AU POINT ET LIMITES

Avant toute chose il faut distinguer les simulateurs software de réseaux neuronaux des réseaux hardware. Les simulateurs de réseaux sont, en fait, des programmes software tournant sur des machines traditionnelles. Les réseaux hardware sont, quant à eux, soit des accélérateurs hardware, soit des processeurs ayant une structure de réseaux (cfr. point VIII.).

1) Le terme "*Neural Network*" a été utilisé pour la première fois dans les années 50, et le concept de réseau neuronal est plus ancien encore. Depuis quelques années il y a une résurgence d'intérêt pour les réseaux neuronaux, car ceux-ci offrent de nouvelles possibilités. Il y a donc un développement récent mais les concepts de base sont anciens. En cela, on ne peut dire que la technologie des réseaux neuronaux est ancienne ou récente.

2) Certaines critiques affirment que les réseaux neuronaux, systèmes concurrents, ne peuvent être simulés sur des ordinateurs du type Von Neumann. Le "*contrôle*" du facteur temps lors de la simulation permet d'éviter de telles remarques. En effet, il sera alors possible à un temps t de faire plusieurs opérations séquentielles.

3) Certains pourraient penser que puisque les réseaux neuronaux sont capables d'apprendre et de s'adapter, les informaticiens ne sont plus utiles. Si le travail de ceux-ci a changé, il n'est pas supprimé pour autant. Il y aura toujours une phase de recherche et d'implémentation des algorithmes. De même il faudra toujours s'occuper de la phase d'apprentissage. Le travail change, mais il n'est pas supprimé.

4) Pour ce qui est des réseaux hardware, ceux-ci sont plus efficients de part leur structure massivement parallèle. Mais pour pouvoir faire des comparaisons valables, il faut des systèmes de complexité comparable. Or un réseau hardware est beaucoup plus complexe qu'un PC ou un Micro-ordinateur. De plus, à l'heure actuelle, même si les développements hardware sont en bonne voie, on assiste principalement à des simulations sur machines séquentielles. Les rares réseaux

neuronaux hardware réalisés sont encore au stade expérimental. [DAVAL,88] [DARPA,88].

De plus, actuellement, les résultats obtenus montrent à suffisance que les simulateurs et les réseaux hardware offrent de meilleurs résultats que des programmes ou machines digitales, et ce pour un grand nombre d'applications (reconnaissance de la parole, de la vision, ...).

5) Une autre remarque souvent énoncée est le manque de précision des réseaux neuronaux. Il ne faut cependant pas perdre de vue le champ d'application de cette technologie qui se situe moins dans le domaine des mathématiques que dans le domaine du traitement de données sensorielles (classification de patterns, contrôle de bras de robots, vision primaire, traitement du signal, reconnaissance de la parole). Très rarement, les réseaux neuronaux ont été utilisés pour résoudre des problèmes nécessitant du raisonnement et de l'inférence ou des calculs mathématiques complexes, tel que tester la primalité de très grand nombres [DARPA,88].

6) On l'a vu, les réseaux neuronaux s'inspirent de considérations biologiques. On entend par là qu'ils se réfèrent à l'organisation du cerveau pour ce qui est de la configuration des réseaux et des algorithmes d'apprentissage.

Cependant, la connaissance du cerveau est à ce point limitée qu'elle aide peu ceux qui désire l'émuler. Dès lors, les concepteurs de réseaux doivent aller au-delà des connnaissances biologiques. Ceci les conduit à créer de nouvelles structures qui exécutent les fonctions du cerveau mais qui sont infaisables d'un point de vue organique ou qui exigent des hypothèses hautement improbables sur l'anatomie du cerveau et sur son fonctionnement.

L'analogie entre réseau neuronal existe donc bien d'un point de vue fonctionnel mais pas structurel dans la majeure partie des cas. De plus, le cerveau humain est tellement complexe qu'il semble certain que jamais de tels réseaux n'atteindront ce niveau de complexité. Néanmoins, cette simplicité relative permet d'obtenir des bons résultats (cfr. VII. Applications).

7) Le temps d'apprentissage est en général très long et augmente plus rapidement que la taille du problème à résoudre.

8) Dans l'état actuel des connaissances, on ne peut pas dire combien de neurones il faut mettre dans les couches cachées des réseaux multi-couches. Tout ce qu'on peut dire, c'est que ce n'est pas parce qu'on en met plus que le problème est automatiquement plus vite résolu (cfr. notre étude dans le cas particulier du ou-exclusif -annexe F.). Même si

c'est le cas, il faut prendre en compte un temps de calcul plus important suite à l'accroissement des connexions.

IX.3. PERSPECTIVES

La figure ci-dessous (figure IX.1.) montre les performances actuelles obtenues avec les meilleurs réseaux hardware. On remarque que ces performances sont très médiocres face à celles obtenues par des systèmes vivants.

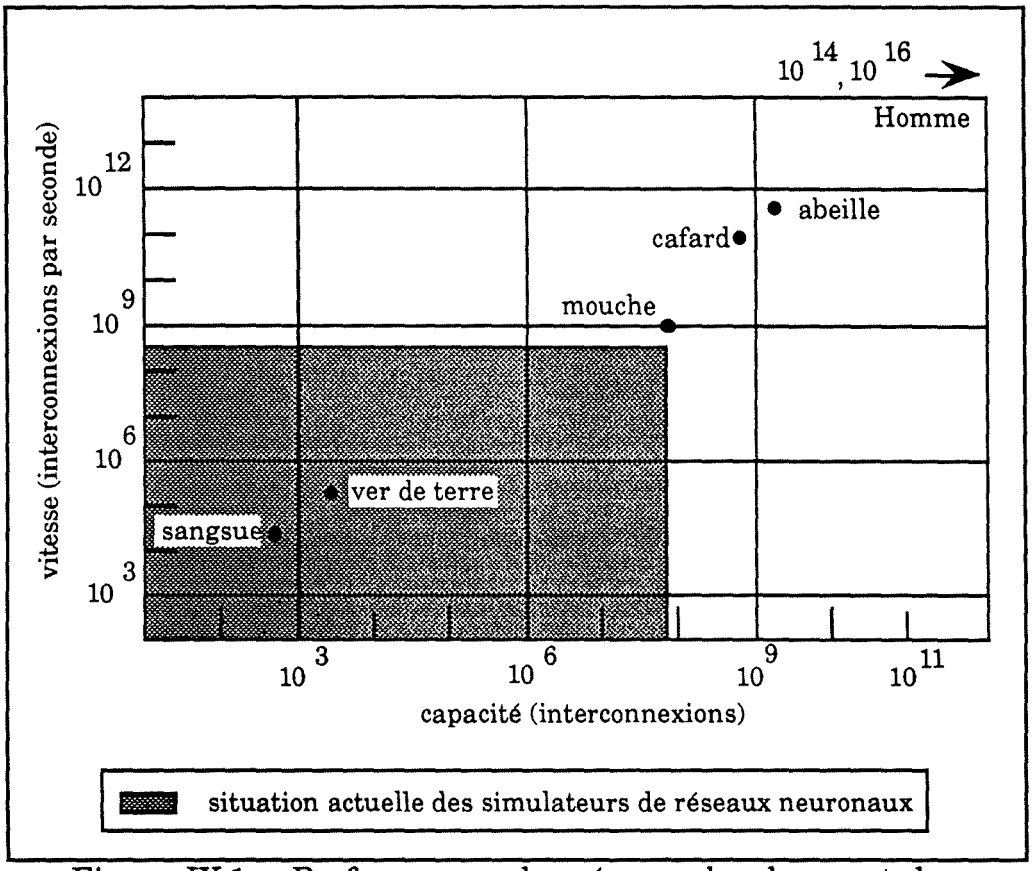


Figure IX.1. : Performances des réseaux hardware et des systèmes vivants. [DARPA,88].

Malgré les limitations actuelles, les réseaux neuronaux ont montré qu'ils étaient capables de s'adapter et d'apprendre, ce qui en fait leur force principale. Ces réseaux offrent de nouvelles perspectives de résolutions de problèmes.

Actuellement, le hardware limite les possibilités de développement, mais le domaine étant en pleine évolution, les chercheurs espèrent obtenir, dans les prochaines années, des performances proches, voire supérieures à celles d'une abeille.

L'étude menée par la DARPA auprès d'experts américains, et dont nous donnons les résultats ci-dessous, avait pour but de répondre aux questions : "**Que seront les réseaux neuronaux dans les 25 prochaine années ?**" et "**Qu'est-il possible de faire avec de tels réseaux ?**". Dans l'annexe C on trouvera des graphiques reprenant plus en détails les résultats de cette enquête.

La technologie des réseaux neuronaux n'est pas une technologie à la mode et qui risque de disparaître d'ici peu. En effet, les experts estiment que cette technologie va devenir majeure dans les cinq à dix ans à venir.

De plus, 64% de ces experts estiment que les réseaux neuronaux permettront de générer de nouvelles applications et ce principalement en robotique (43%) et pour des problèmes d'interface homme-machine (33%). Les problèmes de classement et de reconnaissance seraient les plus touchés par cette technologie (49%). Ces applications permettront d'accoître principalement la capacité de traitement de l'information (33%) et le temps de réponse (27%).

Les experts estiment encore, avec quelques réserves, que les réseaux neuronaux pourront remplacer la technologie existante (34%).

Les applications de réseaux neuronaux seraient particulièrement développées dans le secteur militaire. Le département américain de la défense a d'ailleurs décider de lancer un vaste programme d'étude des réseaux neuronaux et de leurs possibilités.

Les facteurs les plus limitatifs au développement de cette technique sont, la mauvaise connaissance de la théorie mathématique des réseaux (49%) et la difficulté de création de règles pour des réseaux complexes.

[DAVAL,88] [DARPA,88].

X Conclusion

Après avoir examiné les fondements théoriques, l'état de l'art dans le domaine des applications ainsi que les outils de simulation et d'implémentation, nous pouvons conclure que :

• De part leur adaptivité, leur capacité à apprendre ainsi que leur parallélisme massif, les réseaux neuronaux offrent une *nouvelle alternative au traitement de l'information*;

• *La recherche dans la modélisation neuronale a acquis beaucoup de maturité depuis les années 50.* Celle-ci est due à trois facteurs :

- l'avancement des théories mathématiques;
- le développement de nouveaux outils sur ordinateur;
- une compréhension accrue de la neurobiologie;

• D'un point de vue théorique, il est nécessaire :

- *d'encourager le développement de nouveaux modèles de réseaux neuronaux.* Les chercheurs doivent explorer des systèmes plus complexes, comportant plusieurs milliers de neurones organisés en sous-réseaux. Ces systèmes ne devraient pas se limiter à des éléments de traitement utilisant simplement des sommations suivies de non linéarités. Des modèles biologiquement plus plausibles devraient être étudiés;
- *d'encourager le développement de meilleurs algorithmes d'apprentissage* qui tiennent mieux compte des caractéristiques des données;
- *de définir des critères de performance* qui indiquent comment et où les réseaux neuronaux sont les plus efficients;
- *d'explorer les différents types de représentation des données pour différentes tâches.*

• Du point de vue des implémentations :

- *les ordinateurs actuels ont un pouvoir de calcul très éloigné des systèmes biologiques les plus modestes* (mouche,...). Leur inadéquation au niveau du stockage, de la vitesse et de la flexibilité ralentit les forces en présence;
- presque toutes les applications de réseaux neuronaux sont à l'heure actuelle implémentées sous forme d'algorithmes tournant sur un ordinateur digital, ce qui ne fournit pas la vitesse, la robustesse et la forme compacte des *implémentations hardware*. En fait, il y a très peu de réseaux neuronaux implémentés sous forme hardware et la majeure partie est *expérimentale*.

• Du point de vue des applications :

- malgré les limitations engendrées par le stockage, les vitesses de simulations, il y a eu d'*importantes démonstrations des capacités des*

réseaux neuronaux, principalement dans les domaines de la vision, de la parole, du traitement du signal et de la robotique;

- les *applications commerciales* sont attendues; la plupart d'entre elles sont toujours au stade du prototypage ou de la simulation;
- les réseaux neuronaux offrent d'importants avantages pour les *systèmes de défense* futurs grâce à leur utilisation dans le traitement avancé de la vision, des images et de leurs capacités en robotique.

Pour terminer, nous pensons qu'il est nécessaire que ces axes de recherches soient encouragés par des fonds privés ou publics. En effet, les capacités hardware actuelles limitent le développement des applications. Dès lors, si les capacités de simulation et d'implémentation ne sont pas améliorées, le champ des réseaux neuronaux retombera de nouveau dans l'ombre (cfr. III. : Historique). Cette dernière considération nous permet de laisser le mot de la fin à Carver Mead, inventeur des processeurs "Retina" et "Cochlea" :

"In a field like neural networks, one is usually too optimistic in the short run but one is never optimistic enough over the long run".

PARTIE II :

APPLICATIONS

XI Introduction

Cette partie a pour objectifs :

1) de montrer sur un exemple simple en robotique comment il est possible d'apprendre à un réseau neuronal de rétro-propagation à déduire une courbe de commutation de commande optimale dans le cas d'une inertie pure (cfr. XII.). En résumé, des coordonnées cartésiennes (x,y), identifiant des éléments de deux ensembles, sont présentées à un réseau neuronal, lequel doit pouvoir apprendre la frontière qui sépare les deux ensembles (la courbe de commutation). Le réseau a été implémenté en pascal et ensuite avec un logiciel spécifique de simulation, ANSim;

2) de présenter une application de reconnaissance automatique de caractères imprimés via un réseau neuronal de rétro-propagation (cfr. XIII.). On présente au réseau des caractères sous forme de matrices de valeurs et on attend du réseau qu'il les reconnaisse. Nous montrerons que tous les réseaux présentent des performances différentes selon le paramétrage utilisé; les plus performants ne sont pas nécessairement les plus puissants. Par performance on entend le temps d'apprentissage comme fonction de la taille du réseau et de la valeur des principaux paramètres (taux d'apprentissage, momentum,...).

Le lecteur pourra en outre trouver en annexe la présentation de deux logiciels de simulation de réseau neuronaux que nous avons utilisés :

1) NeuralWorks Professional I de NeuralWare Inc. (Annexe D);
2) Ansim de Saic (Annexe E).

Une petite comparaison de notre implémentation pascal et du logiciel Ansim sur le Ou-Exclusif est présentée à l'annexe F.

Finalement, nous avons inclu les spécifications et le code pascal de notre première application en robotique à l'annexe G et le rapport de stage au Brésil à l'annexe H.

XII Parabole de commutation en robotique

XII.1. LE PROBLEME

Dans cette application notre but est de développer un contrôleur de processus, sous forme de réseau neuronal, permettant de piloter un bras de robot. Voici brièvement les principes sous-jacents à cette application.

Processus à contrôler

Il nous faut *contrôler un processus*, ce qui ne peut se faire que via un *contrôleur* et un *moteur* si le processus est mécanique (un bras de robot), ou via des muscles et le cerveau, si le processus est biologique (un oeil par exemple).

Prenons le cas de l'oeil dont on tente de contrôler le processus de mouvement par rapport à l'orbite et imaginons que cet oeil fixe un point. Notre but est de parvenir à ce que l'oeil fixe directement une consigne de position, une lampe qui s'allume par exemple (figure XII.1.).

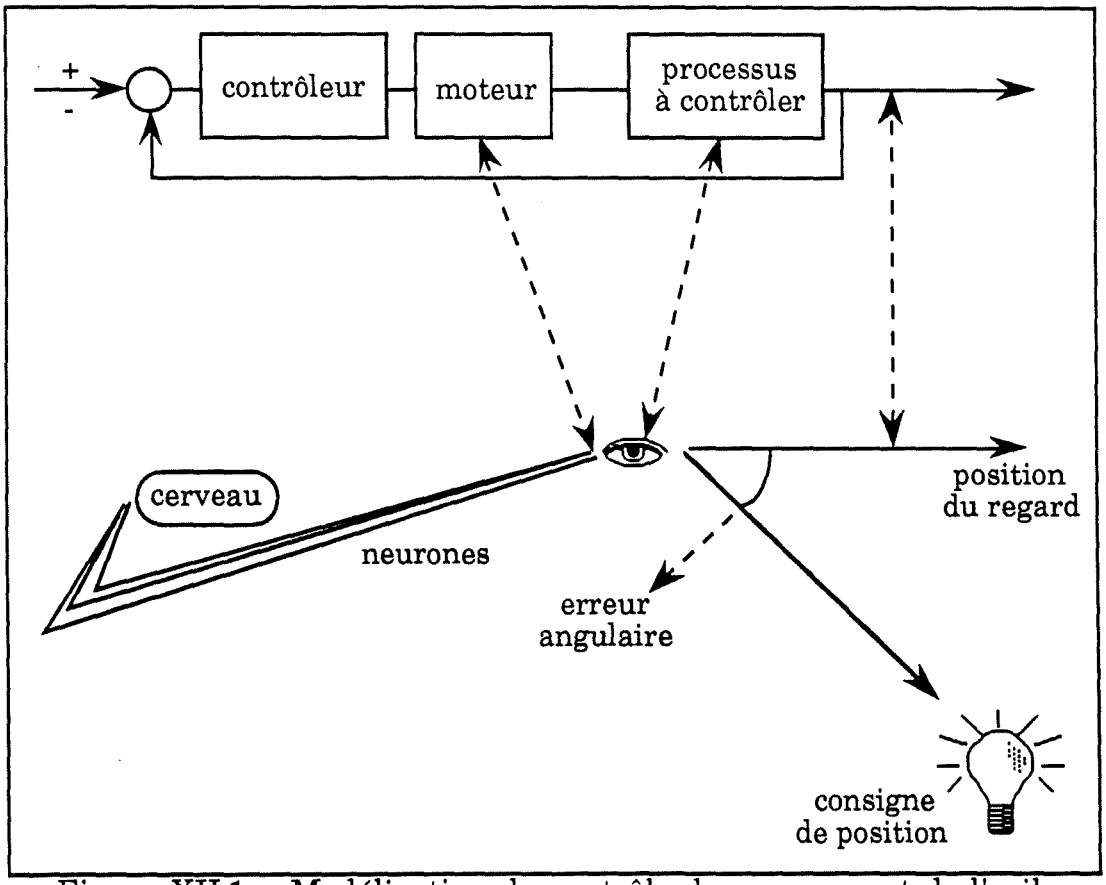


Figure XII.1. : Modélisation du contrôle du mouvement de l'oeil.

Cette consigne de position constitue la valeur d'entrée du système. Elle est ensuite comparée avec la position réelle de l'oeil, ce qui permet de calculer l'erreur de positionnement. L'erreur calculée est ensuite

introduite dans le contrôleur, lequel envoie une commande au moteur qui actionne le processus en conséquence. La nouvelle position du processus est calculée et comparée à la consigne pour être réinjectée dans le système selon un mécanisme de feed-back, et ainsi de suite.

L'oeil possède une région sensible, la fovea, sur laquelle tout ce que l'on voit à tendance à ce concentrer. Reliés à celle-ci, des neurones fournissent les informations utiles au culliculus, la partie du cerveau réservée à la vision et qui transforme l'information en commande vers les muscles. C'est à ce moment que se produit un changement dans le processus, ce que nous appelerons *la commutation.*

Notre contrôle du processus doit se faire de façon à minimiser l'erreur de position (d'où l'utilisation du feed-back) et ce le plus rapidement possible.

Métaphore

"Le mouvement se produit en un temps minimal". Nous supposons ici que la nature a développé le mouvement de façon à ce qu'il soit exécuté en un temps minimal. Cette supposition n'est pas vérifiée, mais elle ne nuit en rien au raisonnement que nous allons tenir.

Il nous faut donc développer un réseaux neuronal qui va prendre la place du contrôleur. Ce réseau neuronal est nécessaire pour réaliser la commutation du processus (oeil) en un temps minimal.

Résultats théoriques connus

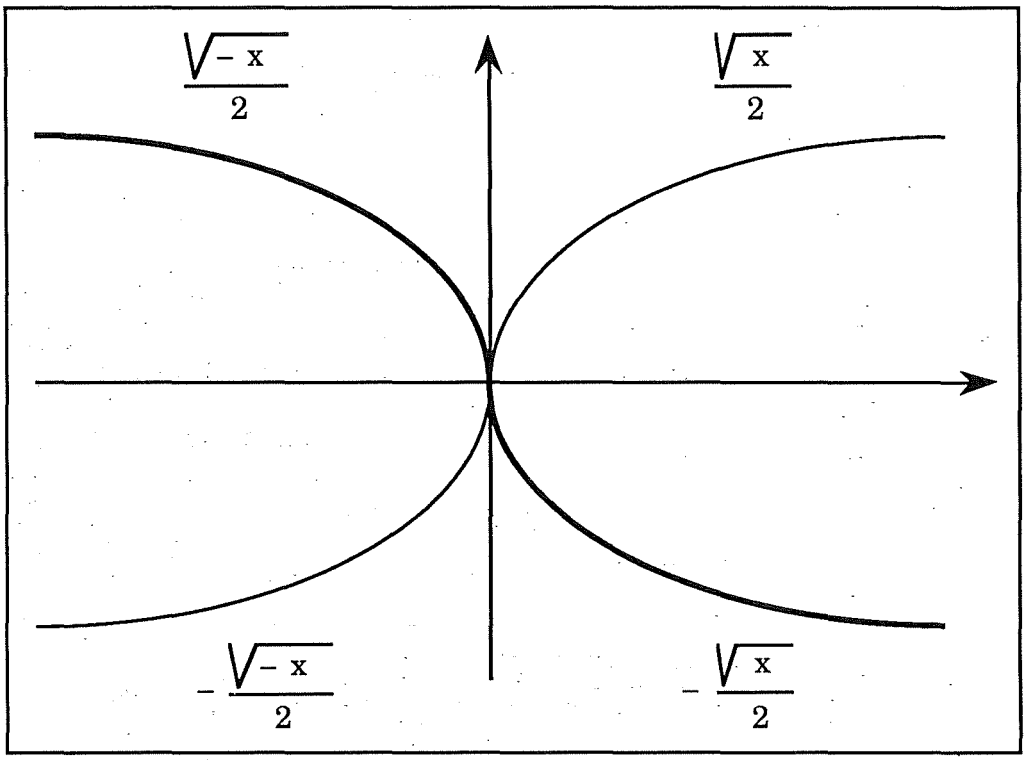
- Si la dynamique du processus est définie par des équations différentielles linéaires à coefficients constants (ce qui est vrai pour la plupart des cas étudiés en littérature comme étude préalable), la commande est de type "*bang-bang*". Cela signifie que l'on va accélérer le mouvement au maximum et ensuite le freiner au maximum, Ce mouvement sera donc le plus rapide possible.

- Si nous considérons une inertie pure, c'est-à-dire sans frottement, on prouve que le contrôleur est une parabole de commutation (figure XII.2.). Cette inertie pure est une équation du second degré proportionnelle à la vitesse.

ERRATA

1 - p. 21-22 : remplacer "X" par "NET"; et $OUT = a * \frac{\left(e^{kNET} - 1\right)}{\left(e^{kNET} + 1\right)}$;

2 - p. 154 : la figure XII.2. est la suivante :



3 - I.1 : le titre n'est pas **CONCLUSION** mais **BIBLIOGRAPHIE**.

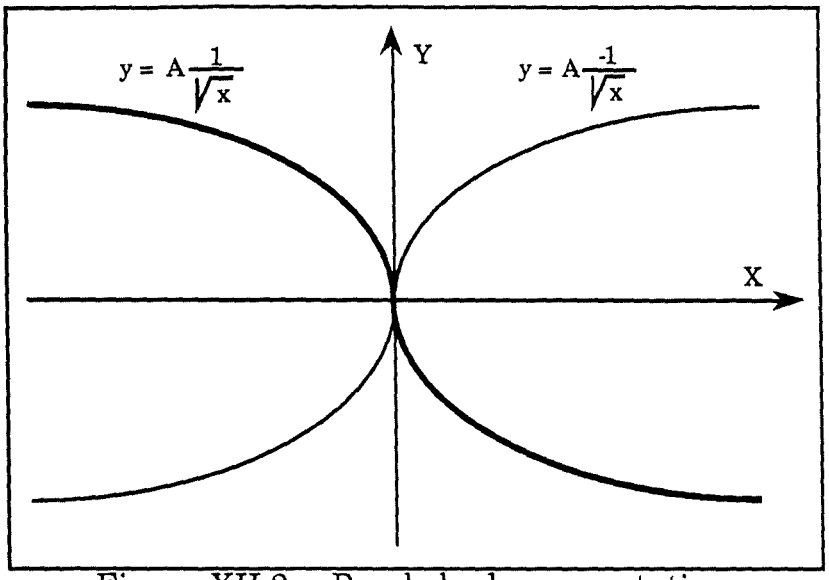


Figure XII.2. : Parabole de commutation.

Dans le cas de l'oeil, le frottement étant très faible, nous obtenons une courbe semblable à celle de la parabole de commutation. Cela n'est plus vrai pour le bras de robot. Le frottement est nettement plus important et la parabole de commutation est dégénérée. Il est même fort probable qu'une solution analytique ne soit plus possible.

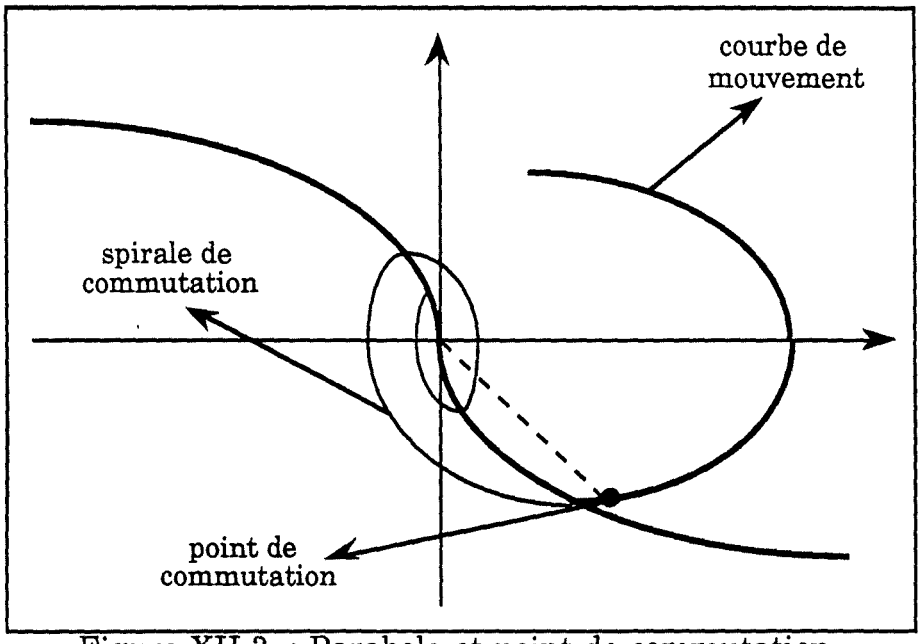


Figure XII.3. : Parabole et point de commutation.

Une manière de résoudre ce problème est de partir de la parabole de commutation dont les poids ont été calculés par notre réseaux neuronal et de les changer via un nouvel apprentissage, de manière à obtenir une

courbe de commutation inconnue. Cette expérience peut être obtenue par observation du comportement de l'oeil.

Lorsqu'il y a frottement, il est nécessaire de commuter plus tôt afin d'éviter d'avoir un comportement oscillatoire (figure XII.3.).

XII.2. RESEAU RETRO-PROPAGATION IMPLEMENTE EN TURBO PASCAL V. 5.5.

Dans notre application, nous avons résolu le problème de la parabole de commutation dans le cas d'une inertie pure. Pour ce faire, nous avons implémenté un algorithme de rétro-propagation en Turbo Pascal V. 5.5. La structure de notre réseau est relativement simple :

1- il s'agit d'un réseau rétro-propagation à trois couches : la couche d'entrée comprend deux neurones, la couche cachée onze et la couche de sortie un seul. Les deux neurones de la couche d'entrées servent à introduire les valeurs continues x et y alors que le neurone de la couche de sortie sert à indiquer si ces coordonnées sont au-dessus (1.0) ou en-dessous de la courbe de commutation;

2- pour des raisons de capacité mémoire, nous avons réduit l'ensemble de quelque 300 coordonnées choisies à 73 dans l'intervalle [- 0.5, 0.5], partie de la courbe la plus difficile à apprendre (changement de concavité);

3- le taux d'apprentissage a été fixé à 0.75[1] et le momentum à 0.9[2]. Les poids ont été fixés de manière aléatoire dans l'intervalle [0,5]. La première valeur peut paraître élevée mais elle est dictée par la nécessité d'accélérer le processus de convergence qui, pour des valeurs moindre, est très lent;

4- la fonction de transfert utilisée est une sigmoïde qui prend ses valeurs dans l'ensemble des réels et fournit ses résultats dans l'intervalle [0,1];

5- l'erreur tolérée est fixée à 0.1.

Avec le réseaux ainsi construit, nous sommes parvenus, après plus de 24 heures, à une convergence des poids synaptiques pour 80% des inputs. En d'autres termes, un/cinquième des inputs considérés n'étaient pas "engrammés" par le réseau.

Notre programme ne comportant pas de procédure de rappel des informations, nous n'avons pu tester la capacité de généralisation du réseau à des inputs inconnus. Cette capacité a été testée dans l'implémentation ANSim (ci-dessous).

---

[1] Valeur optimale du réseau rétro-propagation de Rumelhart [RUMEL,88].

[2] Valeur généralement conseillée dans la littérature.

XII.3. RESEAU RETRO-PROPAGATION ANSIM

Nous avons également résolu ce problème à l'aide du logiciel ANSim. La structure du réseau utilisé est identique à celle décrite plus haut, excepté que la couche cachée contient 15 neurones. Le taux d'aprentissage a été fixé à 0.1, le coefficient du momentum à 0.6 et les poids ont été initialisés dans l'intervalle [-0.3,0.3]. Dans ce cas, la convergence a été obtenue pour l'ensemble des inputs en 36412 itérations et plus de trois jours.

Etant donné la très faible vitesse de convergence, nous avons légèrement modifié le réseau. Le bon sens aurait voulu que nous le rendions plus puissant en rajoutant des neurones dans la couche cachée, ce que nous n'avons pas fait eu égard à notre mince expérience dans le domaine. Au contraire, nous avons ramené ce nombre à 8 neurones, les autres paramètres restant inchangés. Quelle ne fut pas notre suprise de voir que la convergence était atteinte en 1235 itérations, soit un peu plus de deux heures! Au vu de ces bons résultats, nous avons décidé de créer un dernier réseaux dont la couche cachée ne contenait plus que quatre neurones (paramètres inchangés). Après quelques centaines d'itérations cette fois, rebelotte : avec la configuration choisie, la convergence a été atteinte en exactement 986 itérations , ce qui est plus que satisfaisant.

Comme nous avons pu le constater avec le OU-Exclusif le nombre de neurone dans la couche cachée peut influencer fortement la convergence. Placer plus de neurones n'est pas nécessairement gage d'une meilleure convergence. Trouver le nombre optimal n'est pas chose aisée et ne peut se faire que par essais et erreurs. Il en va de même pour l'établissement des différents paramètres. Un taux d'apprentissage élevé permet jusqu'à un certain point d'obtenir une meilleure convergence. Au delà, le risque d'oscillation et donc de non convergence augmentent également.

Une application future en continuité avec ce travail serait de résoudre le problème pour une courbe de commutation tenant compte d'un frottement.

XIII Reconnaissance automatique de caractères imprimés

XIII.1. LE PROBLEME

Le problème consiste à apprendre à un réseau rétro-propagation à reconnaître la police de caractères définie à la figure XIII.1.

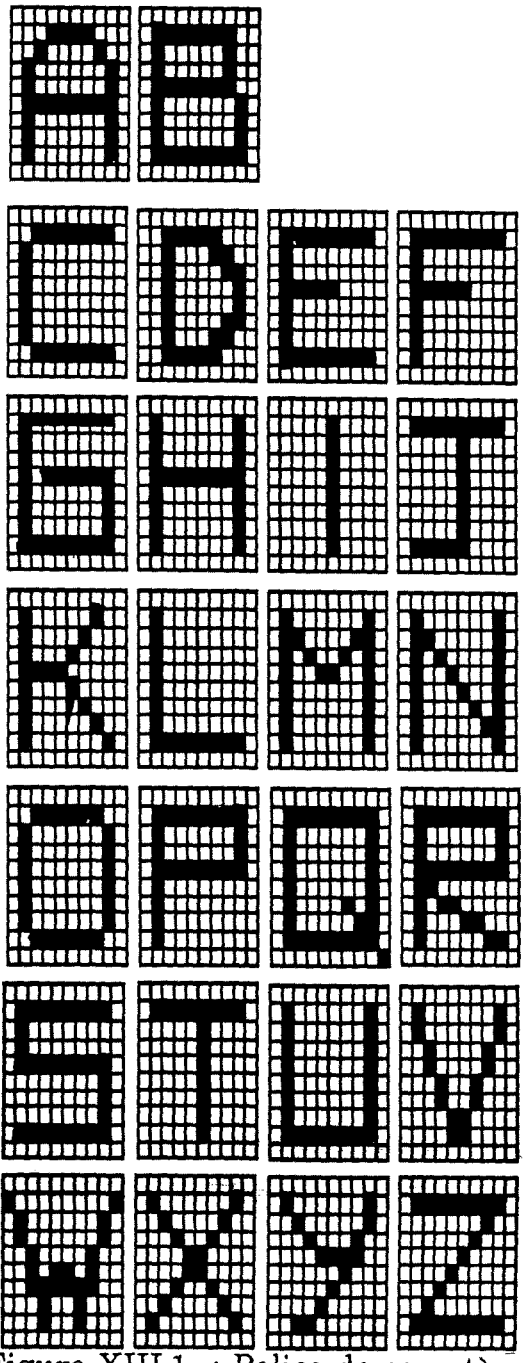


Figure XIII.1. : Police de caractères utlisée pendant l'apprentissage.

XIII.2. RESEAU RETRO-PROPAGATION ANSIM

Chaque lettre de l'alphabet de la figure XIII.1. est représentée par une matrice 10x10 et chaque entrée dans la matrice est une valeur -0.5 ou +0.5 selon qu'il s'agisse d'une case blanche ou noire dans la représentation graphique.

Le réseau utilisé comporte une couche d'entrée et de sortie de 10x10 neurones et une couche cachée de 5x5 neurones (figure XIII.2.). L'ensemble des poids est initialisé à de petites valeurs aléatoires comprises entre -0.3 et +0.3 (valeurs par défaut du logiciel).

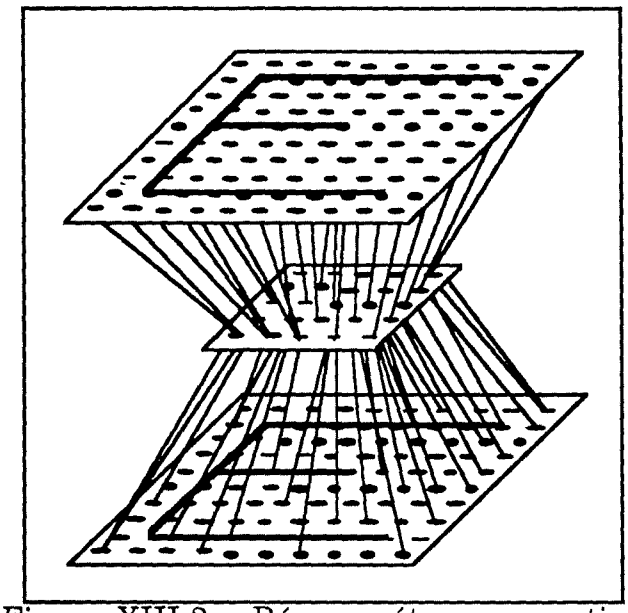


Figure XIII.2. : Réseau rétro-propagation de reconnaissance de caractères.

Nous avons fixé le taux d'apprentissage (η) à 0.75 et le momentum (α) à 0.9. Rappelons que le taux d'apprentissage représente la rapidité avec laquelle le réseau apprend et le momentum la force avec laquelle le réseau tient compte de ses anciens poids de connexions lors du calcul des nouveaux.

Le choix de $\eta = 0.75$ et $\alpha = 0.9$ provient du résultat des expériences de Rumelhart sur le réseau de rétro-propagation ([RUMEL,88]). En ce qui le concerne, la valeur $\eta = 0.75$ est optimale en ce sens qu'au dessus de cette valeur on tombe plus facilement dans un minimum local et qu'en dessous on y perd au niveau temps d'apprentissage.

XIII.3. PERFORMANCES

La chance a voulu que nous réussions l'apprentissage de notre réseau du premier coup. Après quelque 110 cycles sur l'alphabet, le réseau avait correctement appris les inputs et, qui plus est, était capable de reconnaître des versions fortement bruitées, méconnaissables à l'oeil nu.

Nous avons alors fait varier le nombre de neurones de la couche cachée. De 5x5 neurones, nous sommes passés à 6x6, 7x7, 8x8, 9x9 puis à 2x2, 3x3, 4x4 neurones mais sans succès. Force était donc de constater que le premier choix était le bon. Pour nous en assurer, nous avons reconstruit un réseau identique au premier, avec 5x5 neurones au milieu, et l'avons entraîné. Contrairement au premier essai, celui-ci constituait un échec. Les formes apprises, comme dans la majeure partie des autres essais, se ressemblaient toutes et reprenaient les caractéristiques communes à presque toutes les lettres sans toutefois s'apparenter à l'une d'entre elles. De plus, ces résultats étaient atteints après quelques dizaines de cycles, voir quelques cycles. Ce qui nous fit suspecter que le taux d'apprentissage était à l'origine de ces échecs.

Les réseaux ont alors été reconfigurés et réinitialisés avec un taux d'apprentissage égal à 0.1, le momentum restant le même. Ce qu'on observe alors est beaucoup plus encourageant : après quelques cycles, au lieu d'avoir un résultat pratiquement définitif (toutes les valeurs à -0.5 ou 0.5), les neurones représentant des caractéristiques communes à un grand nombre de lettres produisent le bon output et les autres neurones ont des valeurs moins certaines ([-0.2,0.2]) mais plus dépendantes de la lettre en cours d'apprentissage. Si l'on considère la distribution des caractéristiques communes aux différentes lettres au tableau XIII.1., le réseau a à priori tendance à apprendre un 8 entouré de blanc (les bords blancs sont présents à chaque lettre de l'alphabet, soit 26 fois). Au fur et à mesure de l'apprentissage il affine cette représentation pour l'adapter aux particularités des différentes lettres.

26	26	26	26	26	26	26	26	26	26
26	21	14	15	15	15	15	13	18	26
26	20	18	18	0	3	3	6	11	26
26	19	1	4	2	3	3	4	11	26
26	18	9	8	12	14	6	9	12	26
26	18	4	3	4	8	4	3	11	26
26	17	1	4	2	4	4	2	11	26
26	17	2	1	2	4	3	6	10	26
26	18	13	14	13	15	12	11	15	26
26	26	26	26	26	26	26	26	26	1

Tableau XIII.1. : distribution des caractéristiques de lettres.

L'apprentissage est beaucoup plus lent (un bon après-midi soit 200 cycles) qu'avec $\eta = 0.75$ mais le réseau converge plus certainement vers les bonnes valeurs. On voit les lettres se dessiner lentement mais sûrement.

	$\eta = 0.1$	$\eta = 0.4$
3x3	pas OK	pas OK
5x5	OK	pas OK
7x7	OK	pas OK
30	OK	-

Tableau XIII.2. : Synthèse des apprentissages

Les apprentissages ont été refaits avec des taux d'apprentissages différents ($\eta = 0.1$ et $\eta = 0.4$) pour des réseaux comportant 3x3, 5x5, 6x6, 7x7, 9x9 et 30 éléments dans la couche cachée.

Le tableau XIII.2. reprend une synthèse de ces tests. Lorsque l'apprentissage réussit, l'erreur maximale d'un neurone est inférieure à 0.1 (on exige qu'il fournisse soit -0.5 soit 0.5) et on indique "OK" dans la case correspondante. Lorsque l'apprentissage échoue, le "pas OK" peut signifier plusieurs choses. Tout d'abord le réseau refuse purement et simplement d'apprendre les inputs et le seul pattern appris est une sorte de mixture des différentes lettres. Dans d'autres cas, il peut également signifier que l'apprentissage a presque réussi, les lettres étant pratiquement reconnues à l'un ou l'autre détail près. Nous avons estimé qu'il était préférable, pour la facilité de l'analyse, de regrouper ces résultats dans la même catégorie, "pas OK".

Il ressort du tableau XIII.2. que le réseau 5x5 apprend de nouveau l'ensemble des inputs, tout comme les réseaux 7x7 et 30 d'ailleurs, mais ce n'est qu'en contrepartie d'un apprentissage très long (700 cycles).

Le réseau avec 3x3 neurones dans la couche du milieu semble insuffisant pour résoudre le problème, quelque soit le taux d'apprentissage et le nombre de cycles effectués (± 500 cycles).

Les autres réseaux testés, 6x6 et 9x9 n'ont pas donnés de résultats significatifs. On peut noter qu'au delà de 9x9 la mémoire de l'Olivetti M380 XP/3 est déjà saturée. Dans de telles conditions, on comprend mieux les limites de la simulation, surtout sur des problèmes de taille réelle.

Après les apprentissages, nous sommes passés à la phase de rappel pour les réseaux ayant le mieux appris, soit les réseaux 5x5 (η = 0.1 et η = 0.4) et 30. Le test de rappel était constitué de deux parties :

1- reconnaissance d'un input bruité à 50%;
2- reconnaissance d'un input déformé : un "A" majuscule sans sa partie supérieure.

Les résultats graphiques sont présentés aux figures XIII.3., XIII.4., XIII.5, XIII.6. et XIII.7. Il ressort que :

1- les inputs bruités sont mieux reconnus par les réseaux ayant un taux d'apprentissage maximal (η = 0.75) (cfr. figures XIII.4 et XIII.6);
2- plus la convergence est rapide et moins le réseau est résistant à une déformation dans les inputs, du moins dans le cas de couches cachées à 2 dimensions (cfr. figures XIII.3 et XIII.5);
3- Le réseau à 30 neurones dans la couche cachée est meilleur que les autres réseaux à couche cachée bidimensionnelle (malgré un apprentissage de 256 cycles seulement) (cfr. figure XIII.7).

Les résultats d'apprentissages avec bruit ne sont pas inclus ici parce qu'incomplets. Notons seulement que le taux d'échec lors de tels apprentissages est encore plus élevé que lorsqu'il n'y a pas de bruit. Cela se comprend aisement par le fait que certaines lettres comme le O et le Q, le T et le J le M et le N ou le E et le F ne diffèrent déjà que par quelques "pixels" en temps normal, sans bruitage...

Pour finir, nous signalerons simplement que, vu l'absence de critère permettant d'estimer la performance d'un réseau pour un type d'application donné, vu la durée en général très longue des apprentissages (une après-midi minimum), il est difficile de concevoir un grand nombre de tests. Néanmoins, c'est sur des comparaisons simples comme celles présentées ci-dessus que l'on peut se rendre compte au mieux des disparités qui affectent un type de réseau, même lorsque les paramétres sont fort semblables.

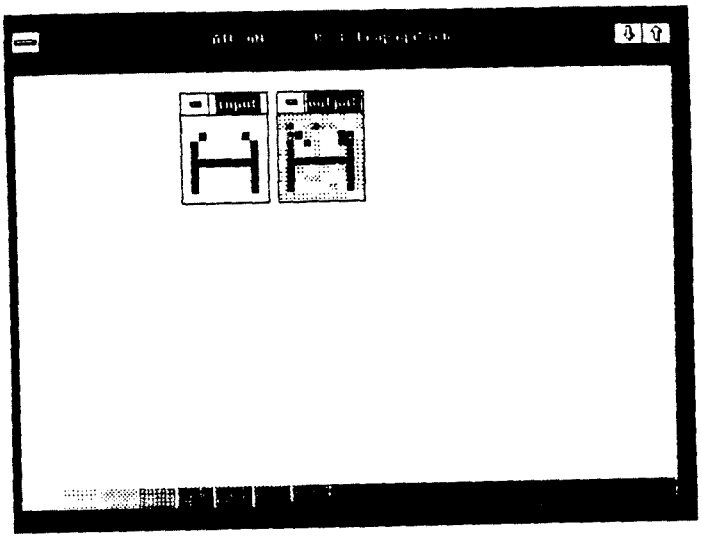


Figure XIII.3. : Ecran 1.

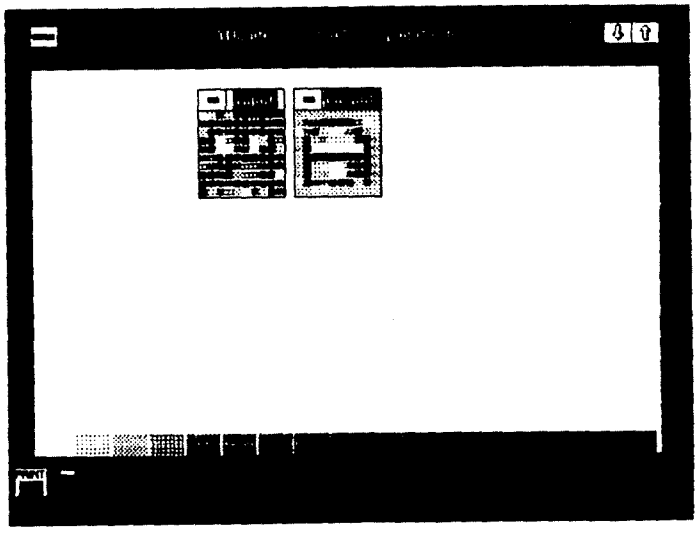


Figure XIII.4. : Ecran 2.

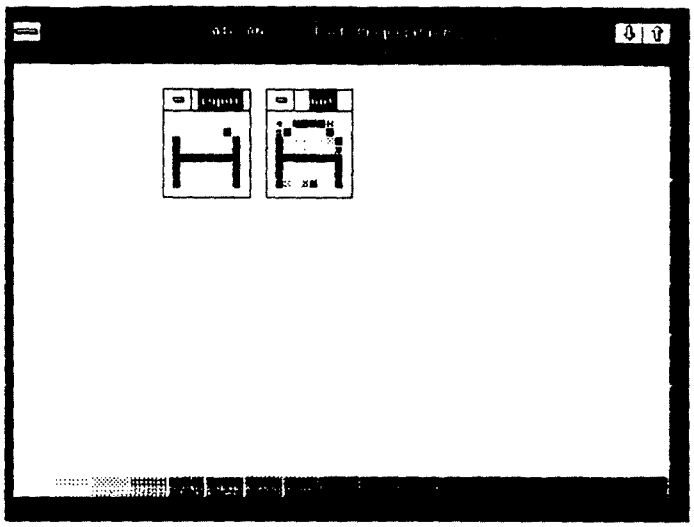


Figure XIII.5. : Ecran 3.

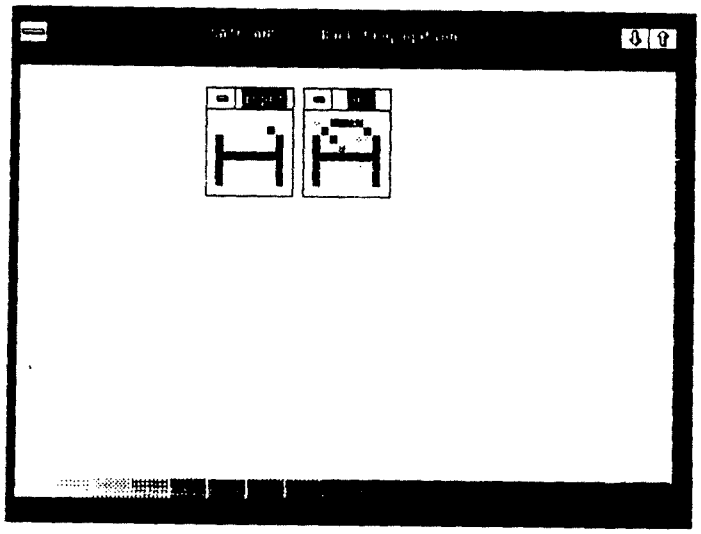


Figure XIII.6. : Ecran 4.

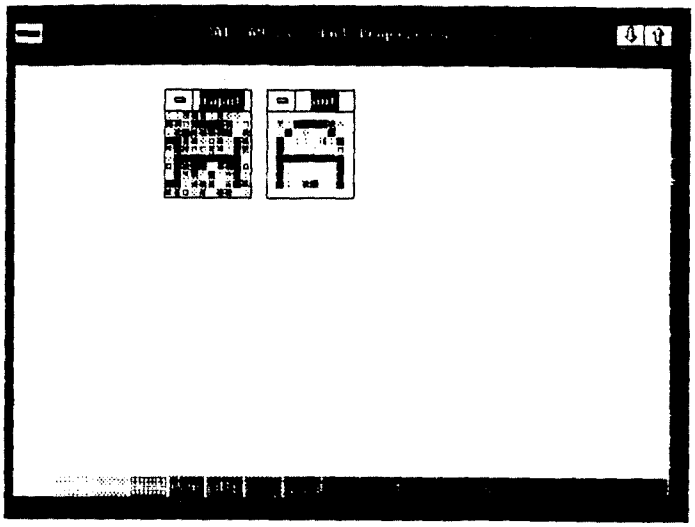


Figure XIII.7. : Ecran 5.

XIV Conclusion

Dans les applications du Ou-Exclusif, du bras de robot et de la reconnaissance automatique de caractères, nous nous sommes exclusivement limités au paradigme de la rétro-propagation. Les applications que nous avons développées avec celui-ci sont relativement simples mais déjà riches d'enseignements pour les non-initiés que nous étions.

Si le réseau de rétro-propagation est un, sinon le plus, populaire dans le domaine de la reconnaissance des formes, il n'en reste pas moins que beaucoup de mystères l'entourent encore. Que peut-on dire du taux d'apprentissage $\eta = 0.75$? Celui-ci est-il optimal en général ou spécifique à un type d'application ? Vu les résultats obtenus dans l'application de reconnaissance de caractères, nous pencherions plutôt en faveur de la seconde possibilité.

Le momentum a été maintenu à 0.9 en toute circonstance, d'une part parce que les résaux convergeaient nettement mieux avec un momentum que sans et que, d'autre part, la littérature conseille lors de son emploi de le maintenir un peu inférieur à 1.

Les poids initiaux ont toujours été choisis de manière aléatoire entre 0 et 1 dans notre implémentation pascal, et entre -0.3 et 0.3 lors de l'utilisation d'ANSim (valeurs par défaut). Il apparaît comme certain que ces valeurs sont capitales dans l'apprentissage d'un réseau. Ceci nous a été confirmé par notre expérience (cfr. l'étude du Ou-Exclusif à l'annexe F) en même temps que par des conférienciers spécialisés dans le domaine.

La grande inconnue reste cependant le choix du nombre de neurones dans la couche cachée. Ce choix serait plus facile à effectuer si l'on pouvait interpréter de manière univoque ce que ces neurones représentent durant la phase d'apprentissage, ce que personne n'a encore pu établir jusqu'à présent.

Nous ne pourrions terminer de parler des réseau neuronaux sans parler des durées d'apprentissage, actuellement astronomiques (de l'ordre de la journée pour les problèmes les plus simples) en ce qui concerne la simulation sur ordinateur de type PC/AT.

Finalement, vu la très grande difficulté d'interpréter la combinaison de paramètres à effets peu ou pas connus, il est difficile pour quelqu'un de pouvoir affirmer qu'il a résolu le problème de manière optimale dans le champ même de la modélisation neuronale. Il est très possible qu'avec quelques neurones en plus, ou en moins, des paramètres un peu mieux ajustés, on arrive encore à une meilleure solution.

Nous voudrions conclure ce mémoire sur une dernière considération : la technique des réseaux neuronaux dans le champ

d'action des informaticiens. Le comportement des réseaux neuronaux est soumis à l'interprétation mathématique, aptitude que possède en général peu l'informaticien. Dès lors, nous pensons que celui-ci est, et sera de plus, confiné dans le rôle de la simulation, le rôle de recherche et du développement étant assuré par la collaboration de physiciens, électroniciens, mathématiciens, psychologues et neurobiologistes. En conséquence, en l'absence de formation spécifique, l'outil de travail actuel que nous recommandons aux autres étudiants en informatique et dont nous avons fait grand usage, est le "pifomètre".

A Annexe : Glossaire

Adaptabilité :

Capacité d'un système de calcul neuronal à s'auto-ajuster. L'apprentissage, l'auto-organisation et la généralisation en sont les manifestations.

Algorithme d'apprentissage :

Ensemble d'équations qui permettent de modifier les poids des connexions des neurones en réponse à un signal d'output et de valeurs fournies par la fonction de sortie. Ces équations permettent au réseau de s'auto-ajuster pour produire un résultat spécifique.

Apprentissage auto-supervisé :

Un moyen d'entraîner un réseau neuronal qui requiert un feed-back interne de l'erreur pour pouvoir accomplir certaines tâches spécifiques. Par exemple, dans le cas d'un automate qui apprend à suivre un point lumineux en contrôlant les muscles d'un oeil, l'erreur est fournie par la distance qui existe entre le reflet du point lumineux sur l'oeil et le centre de la rétine artificielle.

Apprentissage compétitif :

Un algorithme d'apprentissage non supervisé qui permet à des groupes de neurones d'un réseau de concourir pour répondre à un stimulus d'entrée. Le vainqueur de chaque groupe est celui dont les poids de connexions lui permettent de répondre plus fortement au stimulus. Il peut alors ajuster ses poids vers le pattern qui lui a permis de gagner.

Apprentissage supervisé :

Entraînement d'un réseau qui requiert des données d'entraînement étiquetées et un professeur externe. Le professeur connaît la réponse correcte et fournit un signal d'erreur lorsqu'une erreur est faite par le réseau. Ceci est parfois appelé "apprentissage par renforcement" ou "apprentissage avec un critique" lorsque le professeur indique si la réponse était correcte ou incorrecte mais ne fournit pas d'information détaillée sur l'erreur. L'apprentissage supervisé s'oppose aux apprentissages non supervisés et auto-supervisés.

Apprentissage non supervisé :

Entraînement d'un réseau qui requiert des données non étiquetées et pas de professeur externe. Les données sont présentées au réseau et des catégories internes sont formées qui compriment la quantité d'inputs qui doit être traitée à des niveaux supérieurs. Cette tâche de "clustering" est parfois appelée "vector-quantization".

Auto-Organisation :

Il faut distinguer l'auto-organisation à petite échelle de l'auto-organisation à grande échelle. L'auto-organisation à petite échelle permet à un réseau d'organiser ses neurones afin de réaliser certaines performances. L'auto-organisation à grande échelle est la propriété des systèmes vivants qui apprennent de manière auto-supervisée.

Axone :

Partie du neurone par laquelle un neurone classique envoie un potentiel d'action ou impulsion aux autres neurones.

Classifieur de patterns :

Réseau qui permet de découper l'espace des caractéristiques en régions qui correspondent à des classes d'appartenance. Le classifieur est capable de déterminer pour chaque input la classe à laquelle il appartient.

Connectivité :

Modèle d'interconnexion fonction du type de réseau utilisé. Les éléments de traitement ou neurones peuvent être complètement interconnectés entre eux, localement interconnectés à des noeuds voisins ou faiblement interconnectés. De plus, les réseaux peuvent être formés de couches et les neurones dans ces couches reliés via des connexions feed-back ou feed-forward aux neurones des couches adjacentes.

Connectivité complète :

Tous les éléments du réseau sont connectés entre eux.

Connectivité locale :

Les éléments de traitement d'une couche d'un réseau multi-couches sont uniquement connectés aux éléments des couches adjacentes. Cette connectivité est dite locale.

Connexionisme :

Ce terme vient d'une hypothèse partagée par la plupart des architectures massivement parallèles : seul un petit nombre de bits d'information peuvent être envoyé d'un processeur vers un autre. Il n'est pas possible de passer de structure symbolique complexe comme dans les formalismes conventionnels. Dès lors, toute la charge du travail est supportée par la structure des connexions du réseau. Les systèmes connexionistes sont devenus synonymes des réseaux neuronaux. [DARPA,88].

Couches :

Les éléments d'un réseau neuronal sont arrangés en couches. La fonction de transfert est didentique pour tous les éléments d'une même couche.

Dendrites :

Dans le système biologique, les dendrites représentent les nerfs attachés au soma et qui recoivent des inputs d'autres neurones via des contacts spécialisés appelés synapses.

Dégradation gracieuse :

Dans un réseau neuronal, la notion qu'aucun élément de traitement n'est essentiel au fonctionnement du réseau; la performance du réseau décroît graduellement au fur et à mesure que des élements de traitement sont détruits mais il n'existe pas de point critique au delà duquel la performance chute brutalement.

"Dryware" :

Idiômes pour les systèmes réalisés par les hommes et qui effectuent des traitements de l'information, du contrôle ou émulent l'intelligence humaine. Dryware est utilisé par opposition à "wetware" (voir ce terme).

Elément de traitement :

L'élément de traitement, encore appelé unité de traitement, cellule, noeud, neurone artificiel ou tout simplement neurone, est une modélisation du neurone biologique et est la brique de base de tout réseau neuronal. Il est constitué d'une petite mémoire et d'une faible puissance de traitement. Un neurone artificiel calcule la somme de ses inputs pondérés par des poids synaptiques et transmet son output à un certain nombre d'autres éléments.

Feed-back :

Caractéristique d'interconnexion par laquelle un neurone envoie son output vers les couches antérieures d'un réseau multi-couches ou aux autres neurones dans le cas d'un réseau complètement interconnecté. Un exemple de réseau feed-back est le réseau de Hopfield. Un réseau à feed-back doit itérer de nombreuses fois avant de produire un output, contrairement à un réseau feed-forward.

Feed-forward :

Caractéristique d'interconnexion d'un réseau multi-couches par laquelle les neurones d'une couche propagent exclusivement leurs

outputs vers la couche supérieure. Le réseau opère donc jusqu'à ce que ses inputs aient atteint la couche de sortie.

Fonction d'entrée :

Partie d'un élément de traitement qui additionne les signaux d'entrée.

Fonction de sortie :

Partie d'un élément de traitement qui calcule l'output de chaque neurone en fonction de la valeur fournie par la fonction de transfert. En général, la fonction de sortie est la fonction identité. Elle permet, par exemple, de n'autoriser qu'un seul neurone d'une couche à tirer, celui ayant le plus grand output.

Fonction de transfert :

Equation mathématique qui détermine l'opération effectuée par chaque neurone. Ces équations décrivent comment le signal d'output évolue dans le temps en fonction des signaux d'input fournis par la fonction d'entrée. La fonction de transfert inclut la loi d'apprentissage de l'élément de traitement.

Généralisation :

Abilité des réseaux neuronaux à fournir des outputs corrects à partir d'inputs différents de ceux de l'apprentissage. La généralisation est une propriété intrinsèque des réseaux neuronaux.

Interconnexion :

Connexion pondérée entre deux éléments de traitement.

Mémoire associative ou **mémoire adressable par le contenu :**

Un réseau neronal qui, une fois stimulé par un pattern incomplet ou bruité, répond avec le pattern complet. Le réseau peut donc être adressé en utilisant des éléments de la mémoire plutôt que juste leur location.

Mémoire hétéro-associative :

Type de mémoire qui permet aux réseaux neuronaux de déterminer la classe à laquelle appartient un objet. L'exemplaire de classe est différent de l'objet fournit en input, contrairement à une mémoire auto-associative.

Neurone :

"Ensemble constituant la cellule nerveuse et comprenant : une masse protoplasmique qui entoure le noyau (péricaryone), de nombreuses arborisations protoplasmiques ou dendrites (dont l'ensemble constitue le dendrone) et un long prolongement cylindrique, le cylindraxe ou axone" [GARNI,89].

Neurone formel :

Représentation formelle simplifiée du neurone biologique telle que proposée par McCulloch & Pitts (cfr. [McCul,43]). Le comportement du neurone est explicité sous forme de propositions logiques.

Poids :

Coefficient d'adaptation associé à une connexion et qui en détermine l'intensité. Les valeurs des poids sont fonction de l'architecture du réseau et des informations qu'ils ont apprises. Les poids représentent la mémoire du réseau, ce que le réseau connaît.

Procédure de convergence :

Dans certains réseaux neuronaux, après un certain nombre de présentations de couples (stimulus-réponse), les valeurs des poids du réseau approchent l'ensemble des valeurs qui représentent le calcul ou la classification qui lient les stimuli aux réponses.

Règle de Hebb :

Règle formulée au départ pour décrire le système nerveux humain. Elle a été généralisée pour devenir un algorithme d'apprentissage pour réseau neuronal. Cet algorithme stipule qu'une connexion entre deux neurones est renforcée chaque fois qu'elle est utilisée.

Représentation distribuée :

Représentation où chaque concept est représenté par plusieurs neurones et où chaque neurone participe à la représentation de plusieurs concepts. Une représentation entièrement distribuée sur l'ensemble des connexions est encore appelée "représentation holographique". La représentation distribuée est opposée à la représentation locale (voir ci-dessous).

Représentation locale :

Représentation qui consiste à associer à chaque neurone d'un réseau un et un seul concept. La représentation locale est opposée à la représentation distribuée (voir ci-dessus).

Réseau neuronal artificiel :

Des neurones artificiels opérant en parallèle et largement interconnectés forment un réseau neuronal artificiel. La fonction du réseau est déterminée par la structure de ses interconnexions (connectivité locale, complète, faible, feed-back ou feed-forward), les poids synaptiques et le traitement effectué dans les cellules (fonctions d'addition, de transfert et de sortie). Le réseau est entièrement spécifié lorsqu'on a déterminé en plus sa règle d'apprentissage, c'est-à-dire la manière dont le réseau va modifier ses poids.

Rétro-propagation :

Un algorithme d'apprentissage pour adapter les poids dans un réseau neuronal multi-couches feed-forward. Cet algorithme minimise l'erreur quadratique entre l'output fournit et l'output désiré.

Synapse :

Dans les sytèmes biologiques, les tissus qui connectent les neurones. Les synapses sont les contacts spécialisés entre les dendrites du neurone et les axones d'autres neurones. Les synapses sont capables de changer la force de l'implulsion qui les traversent.

Transformation :

Correspondances ou associations d'objets ou de représentations dans un ensemble (tes que des mots écrits) avec des objets ou des représentations dans un autre ensemble (tels que des sons prononcés par quelqu'un) selon une certaine règle qui n'est pas connue du programmeur humain.

Unités cachées :

Les éléments d'un réseau multi-couches qui sont situés entre la couche d'entrée et la couche de sortie. Ces éléments permettent au réseau d'entreprendre des calculs plus complexes que les réseaux sans unités cachées. Les couches comportant de tels éléments sont appelées couches cachées.

"**Wetware**" :

Idiôme pour les systèmes biologiques qui présentent une capacité à réaliser des fonctions de traitement de l'information, des fonctions de contrôle ou l'intelligence. Utilisé par opposition à "dryware" (voir ce terme).

B Annexe : Hardware spécifique

	Processeur	Intercon. par sec.	coût
Super ordinateur	Cray X-MP	50 M/sec	> $ 1 M
Ordinateurs Massivement Parallèles	Connection Machine	13 M/sec	> $ 1 M
	Hypercube		> $ 1 M
	Butterfly	8 M/sec	> $ 1 M
	Warp (10)	10 M/sec	$ 300 K
	Parallon 16/2X	20 M/sec	$ 40 K
Ordinateurs Orientés BUS	TRW MK III	500 K/sec	$ 75 K
	TRW MK V (16)	10 M/sec	$ 100 K
	MX-1/16	120 M/sec	$ 300 K
Processeurs Attachés	SAIC SIGMA-1	5-8 M/sec	$ 15 K
	TI Odyssey	5 M/sec	$ 15 K
	AI Net	45 K/sec	$ 3 K
Mini-Micro Ordinateurs, Station de travail	IBM PC/AT	160 K/sec	$ 8 K
	SUN 3	250 K/sec	$ 20 K
	Apple MacIntosh	5 K/sec	$ 8 K
	Symbolics	35 K/sec	$ 100 K
	DEC VAX	2 M/sec	$ 400 K

Tableau B.1. : Architecture et performances de réseaux hardware. [DARPA,88].

	Hardware	Capacité (K interc.)	Vitesse (K. int./sec)	Coût ($)	rapport Vit./coût
Mini/micro Ordinateurs	PC/AT	100	25	5 K	5,0
	SUN 3	250	250	20 K	10,0
	VAX	100	100	300 K	0,3
	SYMBOLICS	32.000	35	100 K	0,4
Processeurs attachés	ANZA	500	45	10 K	5,0
	Delta -1	1.000	10.000	15 K	300,0
	Transputer	2.000	3.000	4 K	750,0
Ordinateurs orientés BUS	Mark III, IV	1.000	500	75 K	7,0
	ODYSSEY	256	20.000	15 K	1000,0
	MX-1/16	50.000	120.000	300 K	400,0
Ordinateurs Massivement parallèles	CM-2 (64K)	64.000	13.000	2.000 K	6,0
	WARP (10)	320	10.000	300 K	30,0
	Butterfly (64)	60.000	8.000	500 K	16,0
Super ordinateurs	Cray XMP (1)	2.000	50.000	4.000	15,0

Tableau B.2. : Architecture et performances de réseaux hardware. [DARPA,88].

Neurocomputer	Dates	Auteurs, développeurs	Technologie	Capacité			Vitesse	Status
				Nbr. d'élém. de trait.	Nbr. de connex.	Nbr. de réseaux †	Connexions par seconde ‡	
Perceptron	1957	F. Rosenblatt C. Wightman, Cornell Aeronautical Laboratory	Electromécanique et électronique	8	512	1	103	Exp.
Adaline, Madaline	1960/62	B. Widrow, Stanford U.	Electrochimique	1 8	16 128	1	104	Com.
Electro-optic Crossbar	1984	D. Psaltis, California Inst. of Technology	Electro-optique	32	103	1	105	Exp.
Neural Emulation Processor	1985	C. Cluz, IBM	Electronique	4x10E3	1.6x10E4	1	4.9x10E5	Exp.
Optical Resonator	1985	B. Soffer, Y. Owechko, G. Dunning, Hughes Malibu Research Labs	Optique	6.4x10E3	1.6x10E7	1	1.6x10E5	Exp.
Mark III Mark IV	1985 1986	R. Hecht-Nielsen, T. Gutschow, M. Myers, R. Kuczewski, TRW	Electronique	8x10E3 2.5x10E5	4x10E5 5x10E6	1 1	3x10E5 5x10E6	Com. Exp.
Odessey	1986	A. Penz, R. Wiggins, Texas Instr. Research Labs	Electronique	8x10E3	2.5x10E5	1	2x10E6	Com.
Crossbar Chip	1986	L. Jackel, J. Denker, AT&T Bell Labs	Electronique	256	6.4x10E4	1	6x10E9	Exp.
Optical Novelty filter	1986	D. Anderson, U. of Colorado	Optique	1.6x10E4	2x10E6	1	2x10E7	Exp.
Anza Anza Plus	1987 1988	R. Hecht-Nielsen, T. Gutschow, Hecht-Nielsen Neurocomputer Corp.	Electronique	3x10E4 10E6	5x10E5 1.5x10E6	pas de limite	2.5x10E4 (1.4x10E5) 1.5x10E6 (6x10E6)	Com.
Parallon 2 Parallon 2X	1987 1987	S. Bogoch, O. Clark, I. Bason, Human Device	Electronique	10E4 9.1x10E4	5.2x10E4 3x10E5	pas de limite	1.5x10E4 (3x10E4)	Com.
Delta Floating-point Processor	1987	G.A. Works, W.L. Hicks, S. deiss, R. Kasbo, Science Applications Int'l Corp.	Electronique	10E6	10E8	pas de limite	2x10E5 (10E7)	Com.

† Nombre de réseaux qui peuvent être résidents sur la carte.

‡ Vitesse avec apprentissage entre parenthèses; vitesse sans apprentissage hors parenthèses.

Tableau B.3. : Architecture et performances de réseaux hardware. [HECHT,88].

C Annexe : Questionnaire

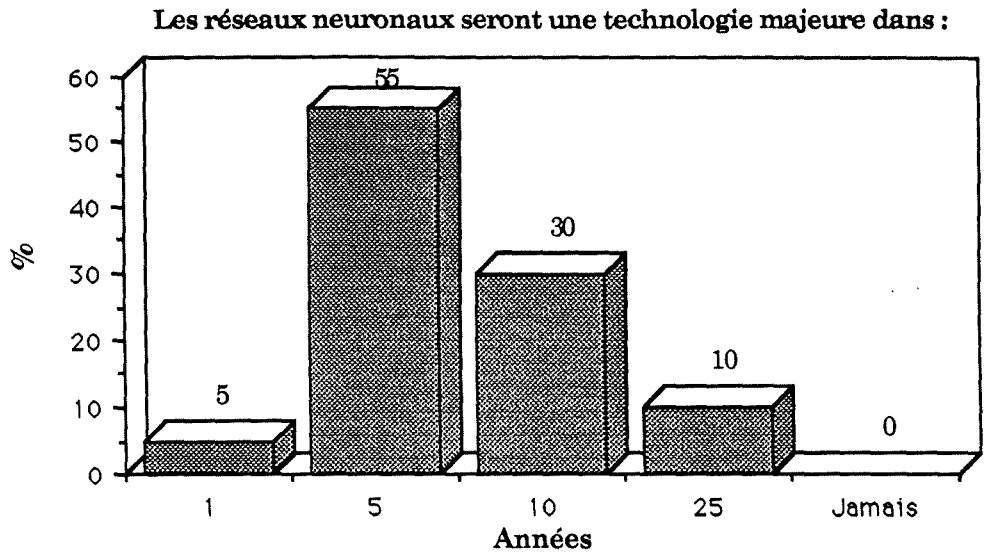


Figure C.1. : Question 1.

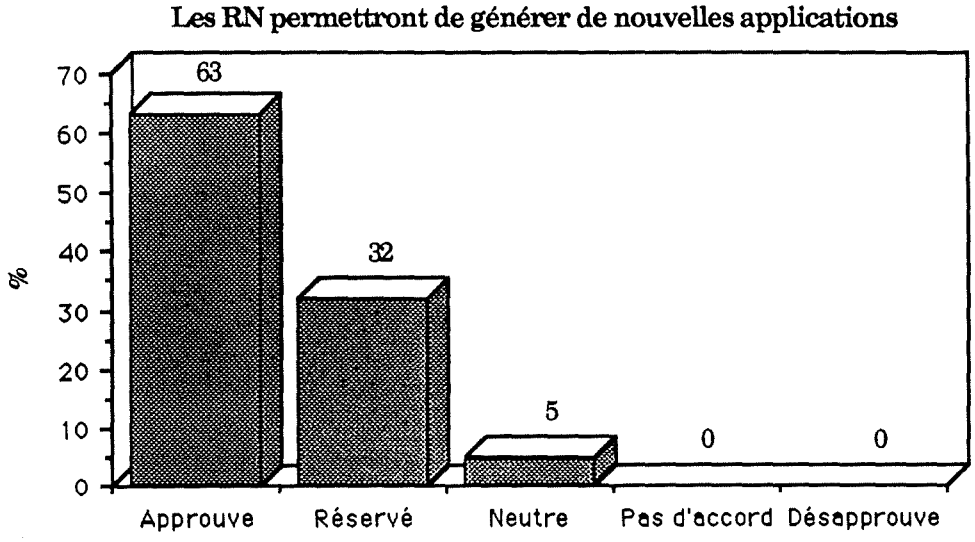


Figure C.2. : Question 2.

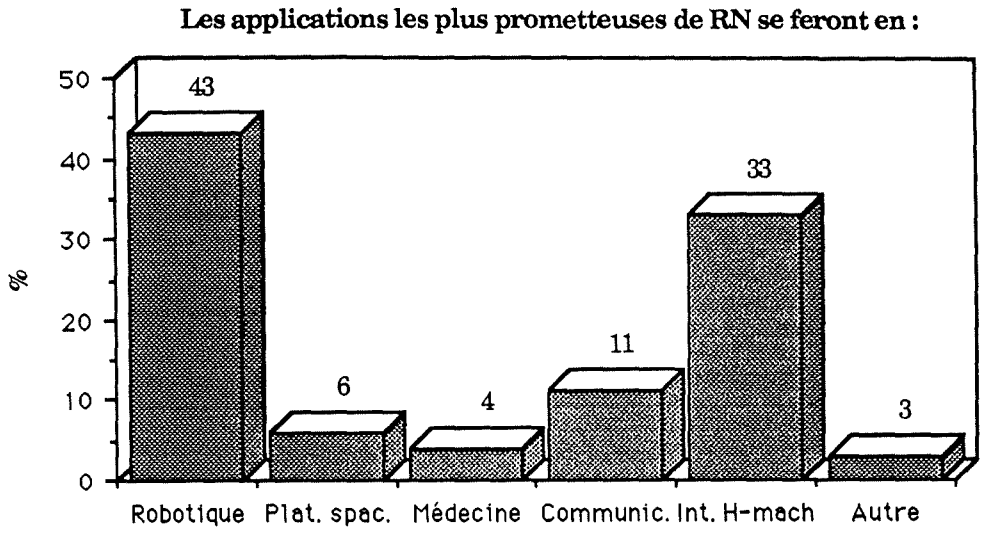


Figure C.3. : Question 3.

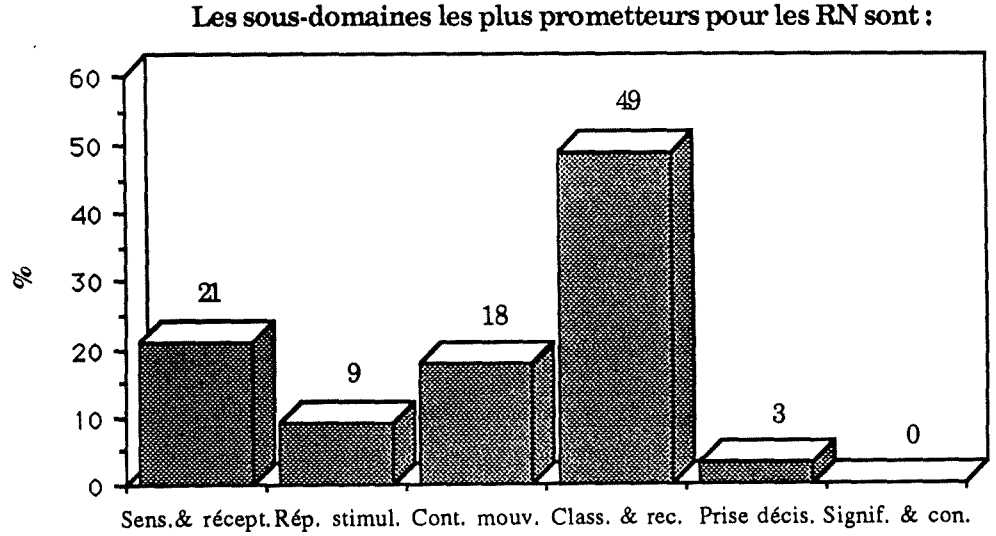


Figure C.4. : Question 4.

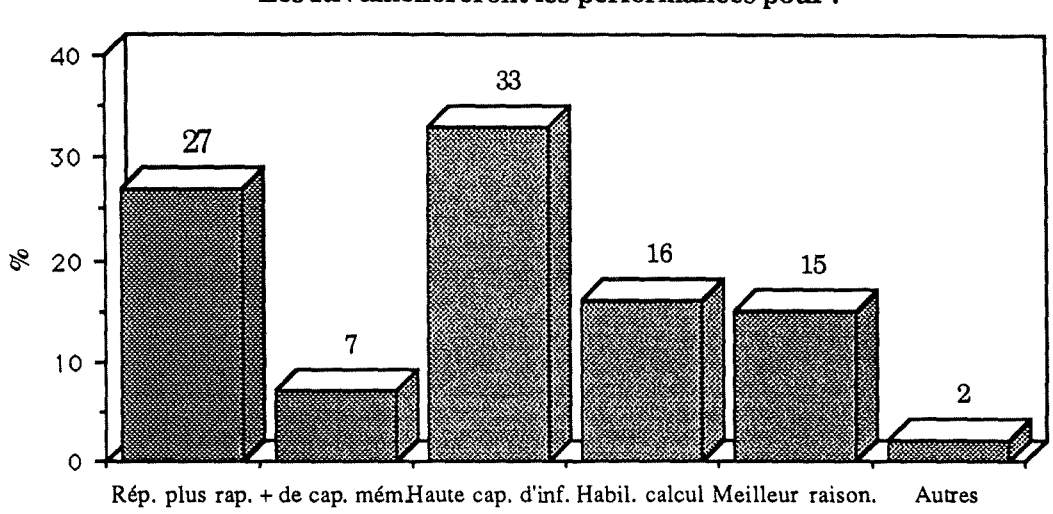


Figure C.5. : Question 5.

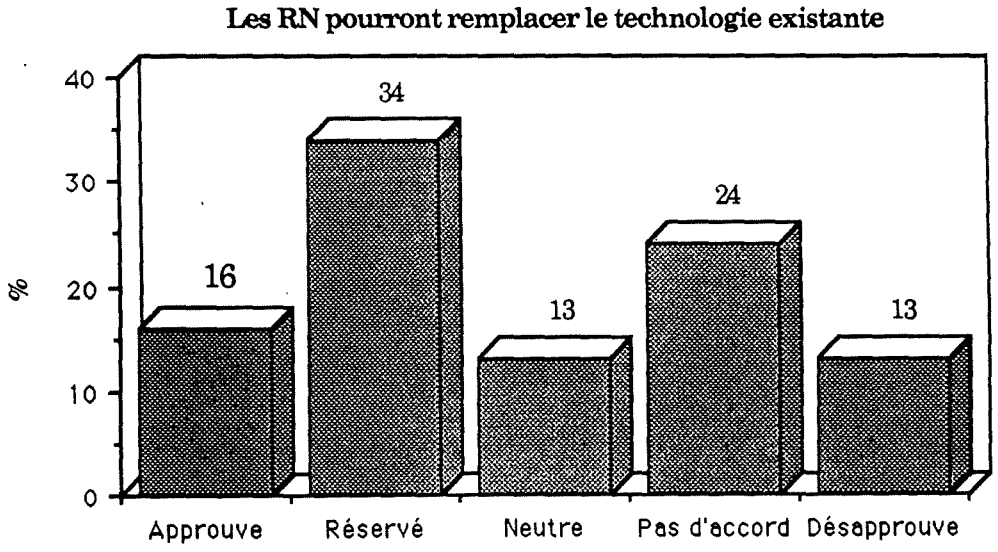


Figure C.6. : Question 6.

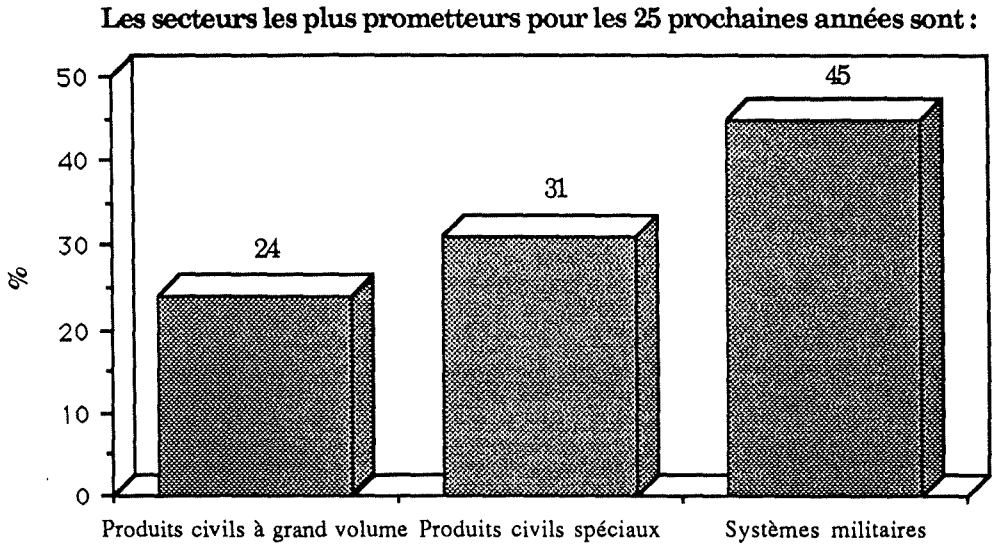


Figure C.7. : Question 7.

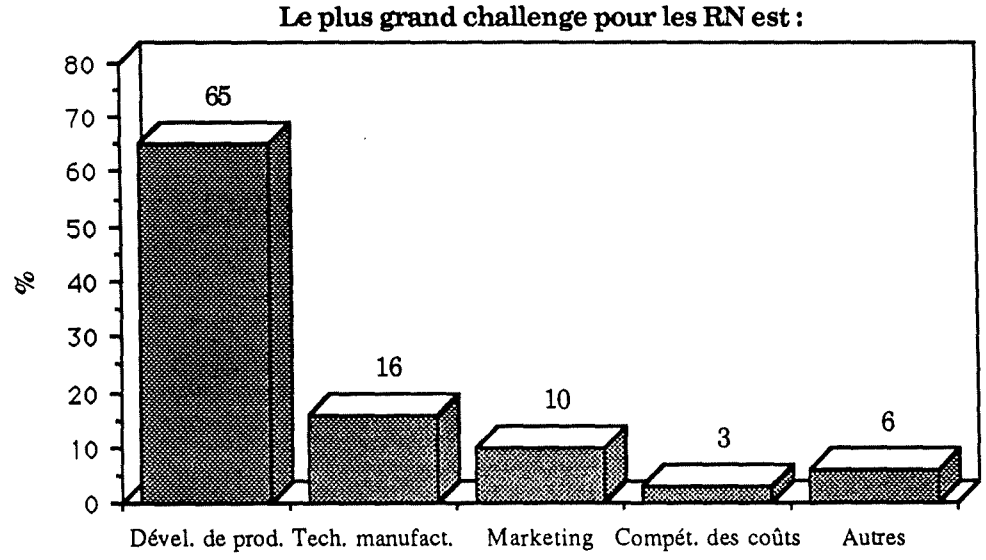


Figure C.8. : Question 8.

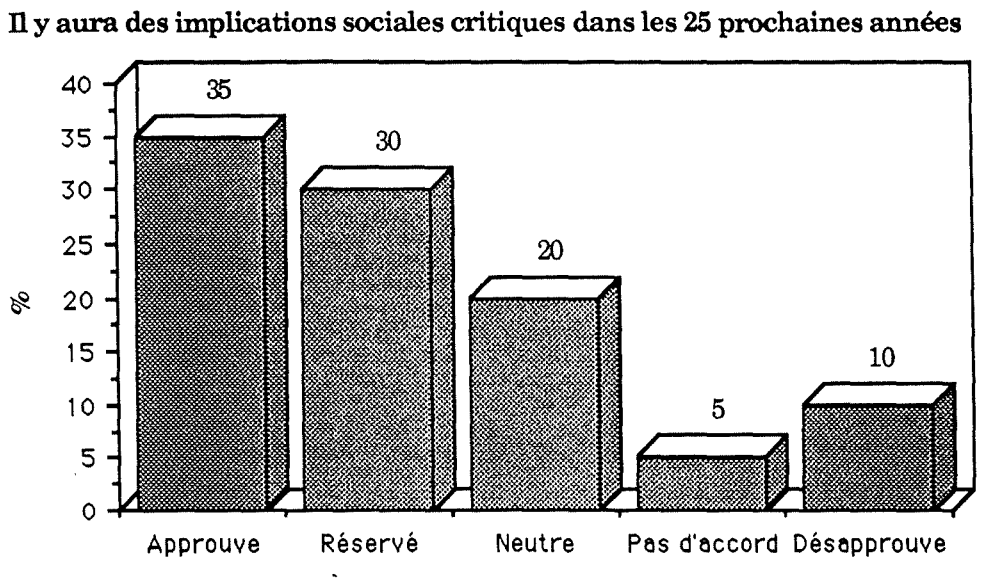


Figure C.9. : Question 9.

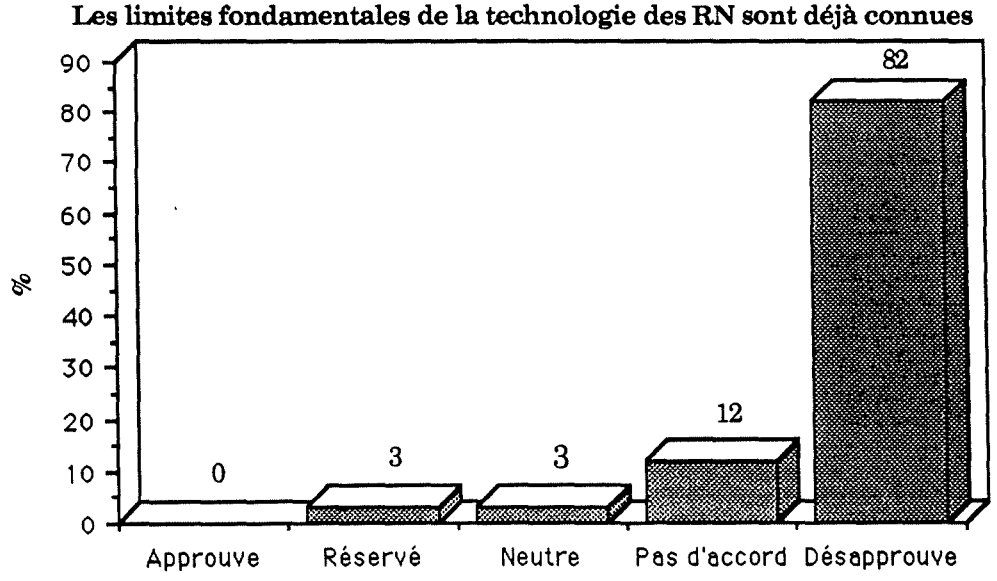


Figure C.10. : Question 10.

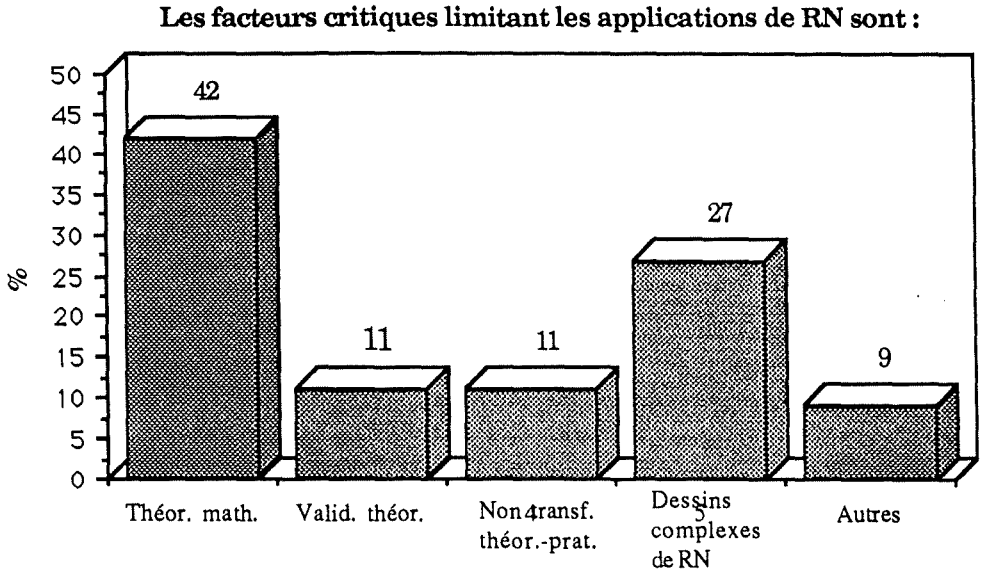


Figure C.11.: Question 11.

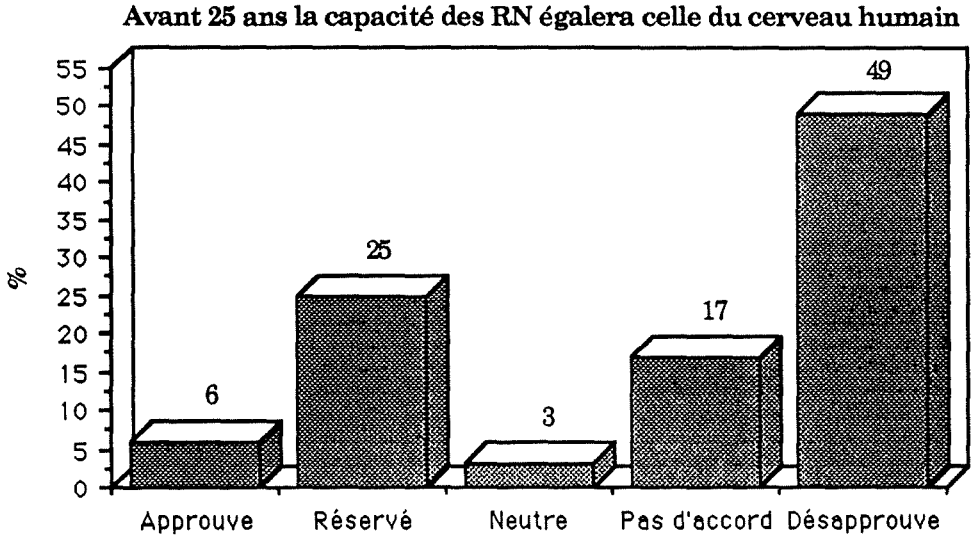


Figure C.12. : Question 12.

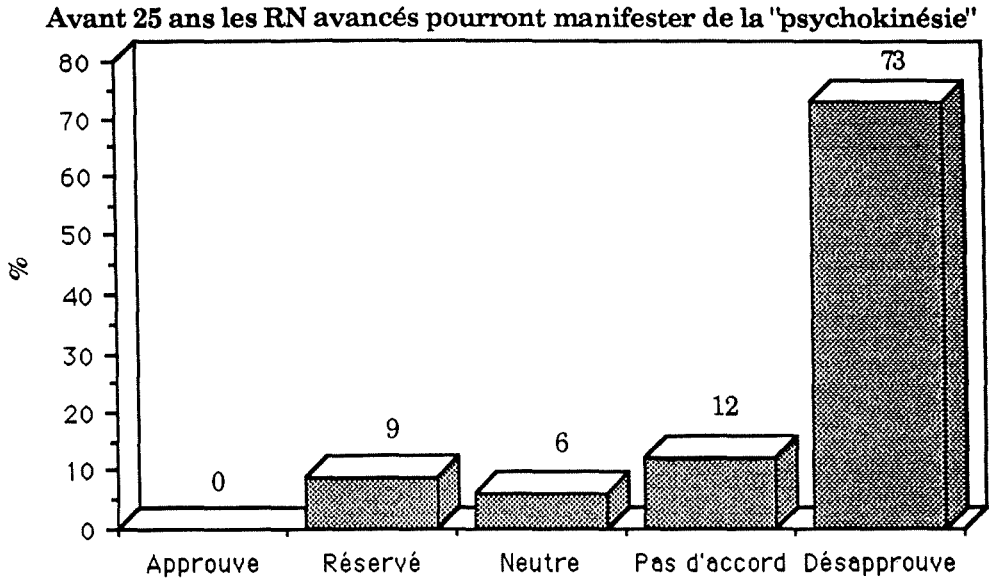


Figure C.13. : Question 13.

NeuralWorks Professional I
de NeuralWare Inc.

Le logiciel NeuralWorks de simulation de réseaux neuronaux est l'outil que nous avons utilisé au Brésil pour tenter de développer nos modèles. Nous disons bien tenter parce que le logiciel ayant subi certaines modifications officieuses, il fut quasiment impossible de travailler de manière satisfaisante. Néanmoins, nous pensons connaître suffisamment son fonctionnement pour le décrire de manière précise.

Tout d'abord, le logiciel fonctionne sur PC/XT ou AT et possède un environnement graphique couleur et des outils qui permettent à l'utilisateur de visualiser le réseau en cours de construction (figure D.1.). Les neurones sont représentés par des rectangles dont la taille et la couleur sont fonction de la valeur du neurone. Les connexions entre les éléments sont représentées par des lignes en pointillés qui prennent différentes couleurs d'après la valeur des poids synaptiques. Les neurones de la couche d'entrée sont également représentés[3].

Le menu permettant de définir les principaux paramètres du réseau est représenté à la figure D.1. L'utilisateur peut choisir entre un réseau auto ou hétéro-associatif[4], déterminer le type du réseau (Perceptron, Hopfield,...), la façon de calculer l'erreur ("L/R Schedule") ou encore spécifier si ces données proviennent d'un fichier ou d'un programme ("Alpha vs User").

Une fois le réseau construit, onpeut déterminer les paramètres de la (des) couche(s) (figure D.2.). Ces paramètres sont entre autres :

- la fonction d'entrée f[5] ("Summation");
- la fonction de transfert ou d'activation g ("Transfer");
- la fonction de sortie h ("Output");
- la règle d'apprentissage ("Learning rule").

Ces paramètres sont propres à une couche et peuvent donc être différents à d'autres endroits du réseau. La figure D.3. Représente les différentes manières d'aborder l'apprentissage et la phase de rappel. L'utilisateur a le choix entre traiter 1, N ou bien tous les inputs à la fois. La boîte de dialogue en bas de l'écran est utilisée pour les messages d'erreur.

Le programme possède en outre une option de vidage de la mémoire, ce qui permet d'obtenir dans un fichier toutes les valeurs des neurones et paramètres du réseau à un instant donné.

---

[3] Voir à ce sujet la discussion du point IV.3.

[4] Rappelons qu'un réseau auto-associatif associe par exemple un "2" à un "2" alors qu'un réseau hétéro-associatif associerait plutôt un "2" à "deux".

[5] Cfr pt IV.2.

Pour terminer, nous signalons la bonne qualité du mode d'emploi qui constitue une excellente introduction aux réseaux neuronaux (présentation des différents paradigmes, glossaire sommaire, facilité de compréhension,...).

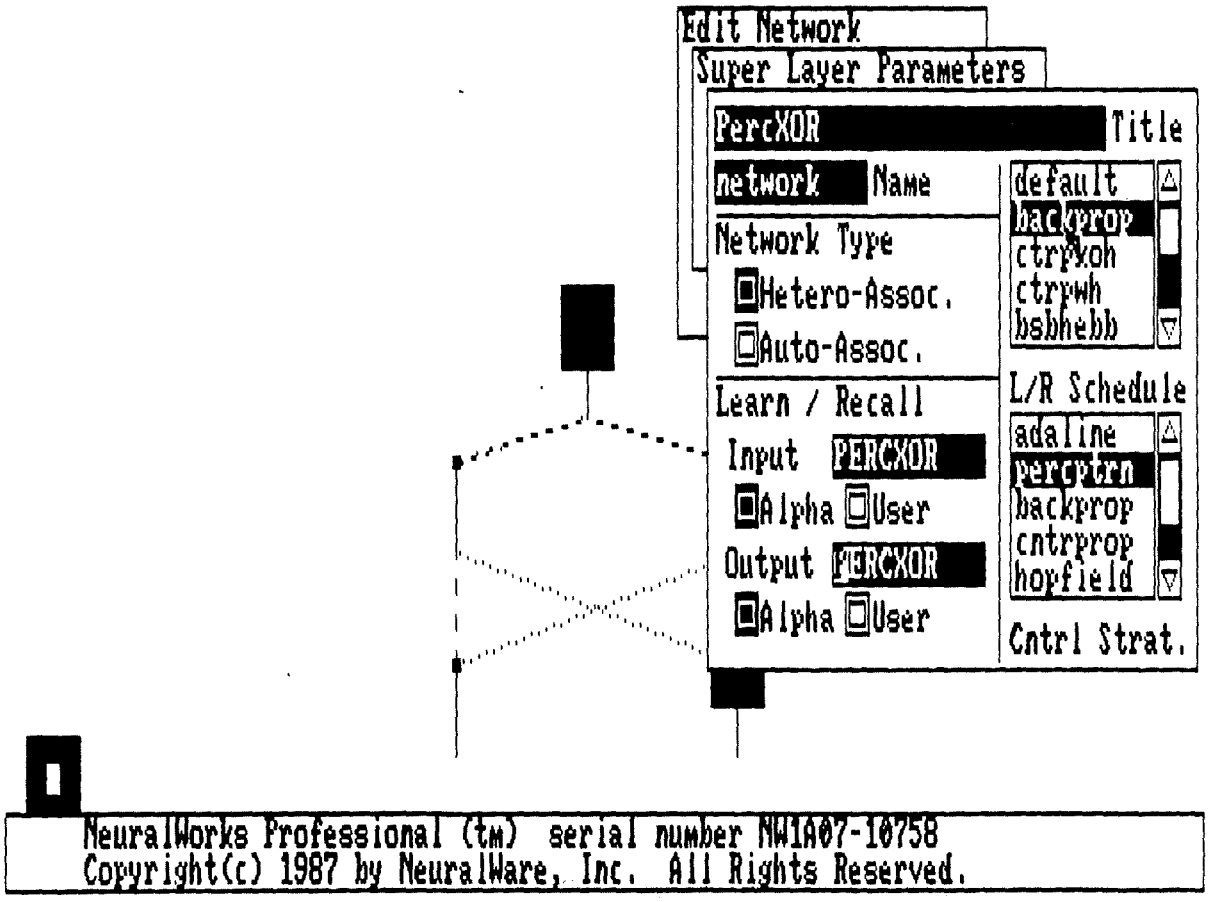


Figure D.1. : Ecran 1.

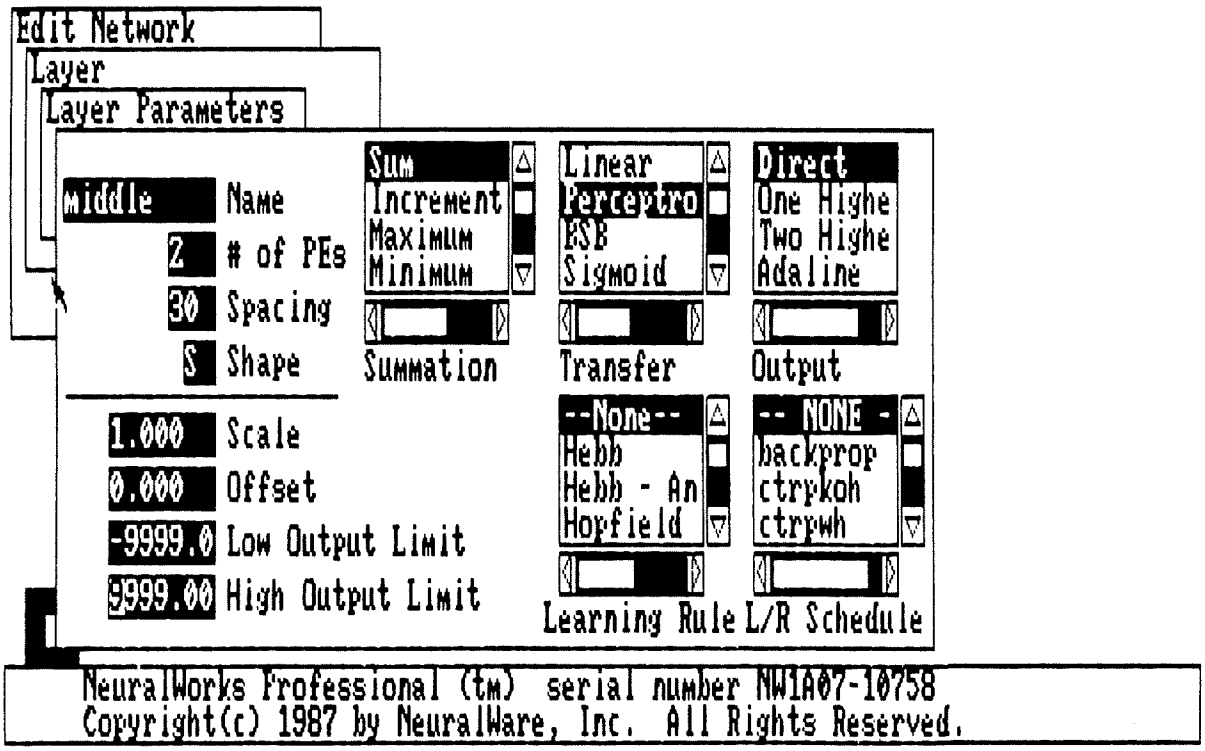


Figure D.2. : Ecran 2.

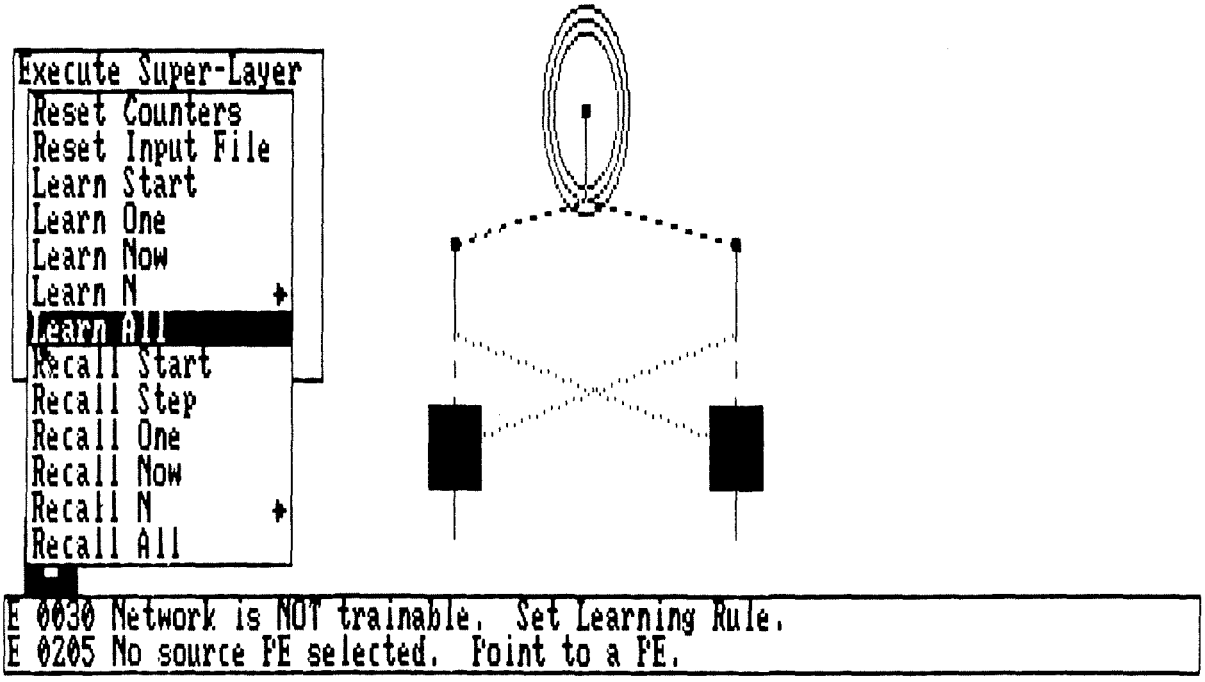


Figure D.3. : Ecran 3.

Ansim de Saic

Le logiciel ANSim V. 2.30 de la Société SAIC est un programme de simulation de réseaux neuronaux tournant sur PC/AT sous les environnements Microsoft Windows 286 et 386, versions 2.03 et 2.10. De ce fait, il bénéficie des facilités apportées par cet environnement : présentations des réseaux sous formes de fenêtres, "multitasking",... Le logiciel supporte deux coprocesseurs mathématiques spécifiques, Delta I et Delta II (22 MFlop), qui peuvent être utilisés pour une grande partie des réseaux disponibles.

L'utilisateur a le choix entre 13 paradigmes de réseau neuronal (figure E.1.). Chaque paradigme a ses propres menus et ses propres options de présentation contrairement à NeuralWorks Professional I. En effet, dans ce dernier, pour chaque réseau construit, l'utilisateur a accès au menu de la figure D.2 qui permet à tout moment de changer la règle d'apprentissage et les fonctions f, g et h. Ce qui n'est pas cohérent avec le menu de la figure D.1., lui aussi accessible à tout moment, et qui permet de déterminer le paradigme du réseau. Dès lors, on est souvent confronté à des incompatibilités d'options.

Dans ANSim, de plus, pour un paradigme donné, l'utilisateur n'a pas le choix des fonctions f, g et h; tout est prédéfini et ne peut être changé, contrairement a NeuralWorks Professional I.

Au niveau présentation graphique, le nombre de choses pouvant être représentées à l'écran est assez grand. L'utilisateur peut visualiser les différentes activations des neurones du réseau (une fenêtre par couche du réseau dans laquelle chaque neurone est représenté par un carré dont la couleur varie selon la valeur du neurone -figure E.2.). Il peut également afficher les valeurs des connexions, différents types d'erreurs calculées durant l'apprentissage du réseau et un statut qui indique entre autre le nom du fichier contenant le réseau, le nombre de cycles opérés sur le fichier de données, l'erreur moyenne (RMS) et l'erreur maximale d'un neurone de sortie. Dans NeuralWorks, par contre, l'utilisateur n'a pas d'autre possibilité d'affichage durant l'apprentissage que celle des neurones et connexions du réseau.

On constate donc que, d'un point de vue graphique, le logiciel ANSim V. 2.30 offre des possibilités nettement supérieures à celles de NeuralWorks Professional I. D'un point de vue performances, nous ne pouvons pas les comparer pour les raisons mentionnées plus haut (impossibilité de faire tourner correctement NeuralWorks Professional I).

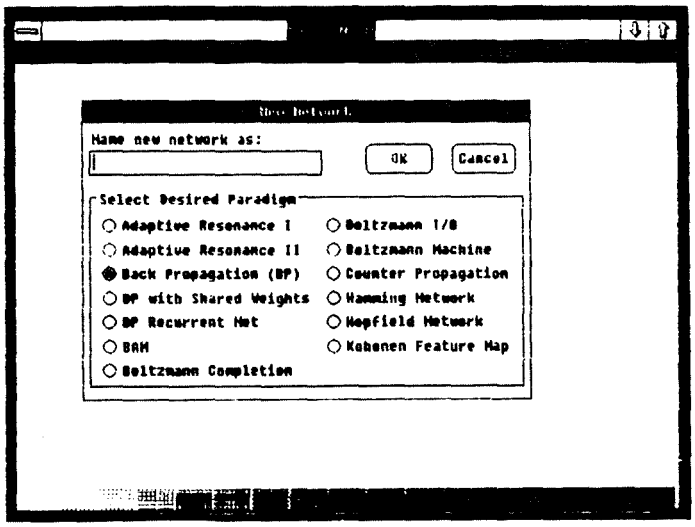


Figure E.1. : Ecran 1.

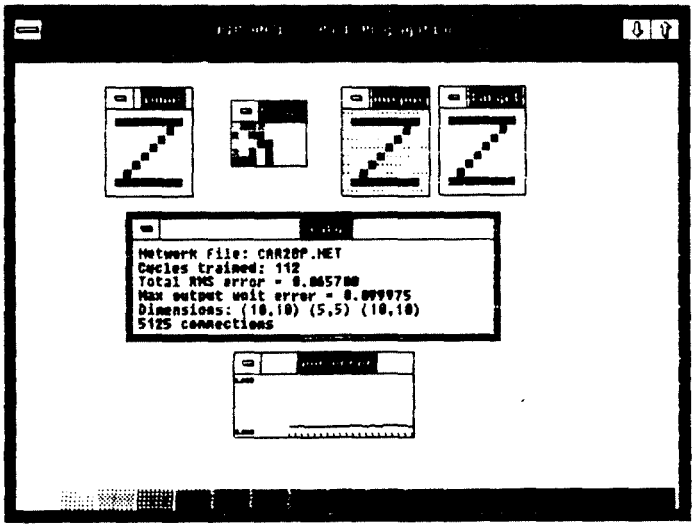


Figure E.2. : Ecran 2.

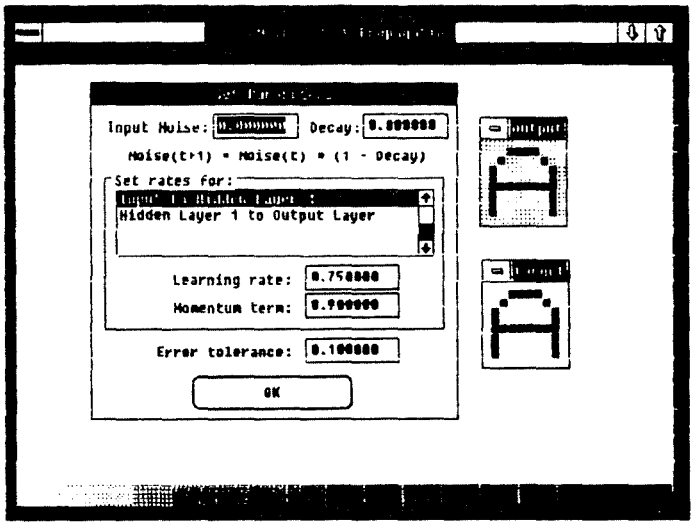


Figure E.3. : Ecran 3.

F Etude comparative du Ou-Exclusif

Nous comparons ici, sur le ou-exclusif, les résultats obtenus avec notre implémentation pascal d'un réseau rétro-propagation et les résultats obtenus avec ANSim.

1. PARAMETRAGE

Le paramétrage des réseaux est le suivant :

1.1. Notre réseau :

- réseau rétro-propagation en Turbo Pascal V. 5.5.;
- taux d'apprentissage : 0.75;
- momentum : 0.9;
- sigmoïde [0, 1];
- données :

Input 1	Input 2	Output
0	1	1
1	0	1
0	0	0
1	1	0

- erreur acceptée : 0.1;
- poids initiaux : dans l'intervalle [0,1].

1.2. Réseau ANSim :

- réseau rétro-propagation;
- taux d'apprentissage : 0.75;
- momentum : 0.9;
- sigmoïde [-0.5, 0.5];
- données :

Input 1	Input 2	Output
-0.5	0.5	0.5
0.5	-0.5	0.5
-0.5	-0.5	-0.5
0.5	0.5	0.5

- erreur acceptée : 0.1;
- poids initiaux : [-0.3, 0.3] (valeurs par défaut du logiciel).

2. RESULTATS OBTENUS SUR 5 ESSAIS

2.1. Notre réseau :

le tableau ci-dessous reprend le nombre d'itérations sur l'ensemble des inputs en fonction du nombre de neurones dans la couche cachée du réseau (ordonnées). Le test a été répété 5 fois (abscisse).

ESSAIS/ Nbr de NEUR.	1	2	3	4	5
2	420	1435	906	431	467
3	423	844	459	2951	322
4	512	351	432	700	1897
5	316	292	487	420	306
6	311	284	459	317	351
7	400	479	354	428	264
8	435	360	383	391	333
9	379	411	313	467	360
10	251	359	375	367	275
15	237	269	383	273	328
20	300	**212**	392	408	263
30	239	249	220	299	289

2.2. Réseau Ansim :

ESSAIS/ Nbr de NEUR.	1	2	3	4	5
2	287	306	273	181	452
3	322	257	354	272	427
4	204	342	287	301	351
5	326	227	328	579	199
7	247	278	323	203	284
8	395	181	342	245	337
9	252	248	305	249	258
10	262	257	187	270	219
15	274	204	233	179	244
20	311	184	230	212	206
30	**152**	184	195	159	220

3. ANALYSE DES RESULTATS

Au vu des résultats annoncés, il semble que le logiciel ANSim soit plus performant que notre programme pascal. En effet, pour chaque type de réseau ayant un certain nombre de neurones dans la couche cachée, la meilleure performance appartient au logiciel ANSim. La meilleure performance globale de ce dernier, sur les essais réalisés, est 152 itérations (30 neurones dans la couche cachée), contre 212 pour notre implémentation (20 neurones dans la couche cachée).

Nous pensons que cela ne peut provenir que de l'initialisation des poids du réseau, les autres paramètres étant identiques. De fait, si l'on observe notre implémentation avec un réseau (2,2,1) (2 neurones en entrée, 2 neurones dans la couche du milieu et 1 en sortie), la disparité des résultats est énorme (de 420 à 1435 itérations) compte tenu du fait que seuls les poids initiaux varient d'un essai à l'autre.

Si les deux séries de tests s'accordent pour placer les réseaux (2,15,1), (2,20,1) et (2,30,1) en tête du classement, il n'en va pas de même pour les autres réseaux, notamment (2,2,1), (2,7,1), (2,9,1). Peut-être qu'avec une série de tests plus long arriverait-on à une meilleure homogénéisation des résultats.

Le problème à résoudre étant relativement simple et nos séries de tests assez réduites, nous n'avons pas observé de minimum local durant les tests. Cependant, l'expérience nous a appris que si les réseaux plus puissants convergent plus vite, ils tombent aussi plus facilement dans un minimum local. Il ne sont donc pas les meilleurs, loin sans faut (cfr. la reconnaissance automatique de caractères supra).

G Spécifications et code pascal

G.1. ARCHITECTURE LOGIQUE

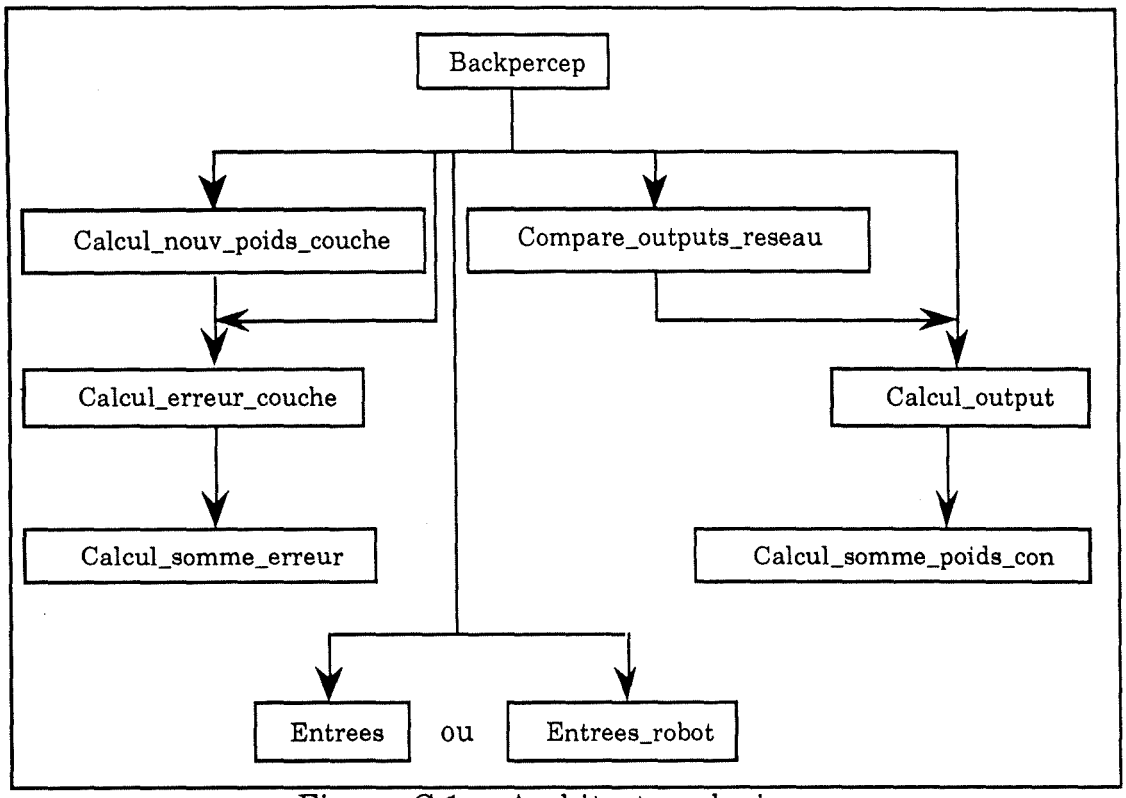


Figure G.1. : Architecture logique.

G.2. SPECIFICATIONS EXTERNES

G.2.1. Déclaration des constantes, types et variables globales.

Const. :

Maxcouche = 4; nbr maximum de couches d'un réseau;
Maxelem = 30; nbr maximum d'éléments dans une couche;
Maxinput = 295; nbr maximum autorisé d'inputs;
Alpha = 0.9; Momentum dans l'algorithme d'apprentissage;
Comp = 0.04; Erreur d'approximation;
Gain = 0.75; Taux d'apprentissage dans l'algo. d'apprentissage;

Types :
Valcouche = array[1..Maxelem] of real;
(* valcouche : la valeur des outputs des neurones d'une couche *)
Valreseau = array[1..Maxcouche] of valcouche;
(* valreseau : l'ensemble des outputs des neurones du réseau *)
Uninput = array[1..Maxelem] of real;
(* un input présenté au réseau *)

```
Valinput = array[1..Maxinput] of uninput;
(* l'ensemble des inputs que le réseau doit apprendre *)
Valoutputdes = array[1..Maxinput] of uninput;
(* l'ensemble des outputs désirés. Il y en a autant (maxinput) que
d'inputs introduits *)
Longueur = array[1..Maxcouche] of integer;
(* longueur : le nombre d'éléments dans chacune des couches *)
Poidselem = array[1..Maxelem] of real;
(* poidselem : les poids des connexions reliant un noeud à la couche
inf *)
Poidscouche = array[1..Maxelem] of poidselem;
(* poidscouche : tous les poids des connexions d'une couche vers la
couche inférieure *)
Poidsreseau = array[2..Maxcouche] of poidscouche;
(* poidsreseau : les poids de toutes les connexions du réseau *)
(* 2 car il n'y a pas de connexions en dessous de la couche d'input *)
Biascouche = array[1..Maxelem] of real;
(* poids des connexions des élém. d'une couche vers la référence *)
Biasreseau = array[1..Maxcouche] of biascouche;
(* poids de toutes les connexions du réseau vers la référence *)
Erreurscouche = array[1..Maxelem] of real;
(* erreurs des neurones d'une couche *)
```

Variables globales :

```
Arret : boolean;                (* indique si le réseau a fini
                                   l'apprentissage*)
Ensvaleurs: valreseau;          (* tableau des outputs de tous les
                                   neurones *)
Ensinputs: valinput;            (* tableau des inputs *)
Ensoutputdes: valoutputdes;     (* tableau des outputs désirés du
                                   réseau *)
Enspoids: poidsreseau;          (* tableau des poids des couches *)
Ancpoids: poidsreseau;          (* tableau des poids au temps t-1 *)
Ensbias: biasreseau;            (* tableau des bias du réseau *)
Ancbias: biasreseau;            (* tableau des bias au temps t-1 *)
Nbrcouches: integer;            (* le nombre de couches du réseau *)
Nbrinputs: integer;             (* nbr d'inputs *)
Longcouches: longueur;          (* tableau des longueurs des
                                   différentes couches *)
Erreursreseau: array[1..Maxcouche] of eErreurscouche;
                                (* tableaux servant au calcul des
                                   erreurs retro-propagées *)
Erreurs: valinput;              (* tableau contenant les différences
                                   entre les outputs désirés et les outputs
                                   calculés *)
Tour: integer;                  (* nbr d'inputs appris *)
```

Par la suite, les abréviations "**Arg.**", "**Pré.**", "**Rés.**" et "**Post.**" signifient respectivement "arguments", "préconditions", "résultats" et "postconditions".

G.2.2. Spécifications externes des modules.

A. Module CALCUL_SOMME_ERREURS.

Arg. :

- Numcouche : entier représentant le numéro de la couche dont on calcule l'erreur de chacun de ses éléments;
- Numelem : entier représentant le numéro de l'élément dont on calcul l'erreur;
- Longcouches;
- Enspoids;
- Erreursreseau;

Pré. :

- $2 \leq$ Numcouche $\leq$ Nbrcouches;
- $1 \leq$ Numelem $\leq$ Longcouches[Numcouche];
- Longcouches contient les valeurs de longueur des couches;
- Enspoids contient l'ensemble des valeurs de poids du réseau;
- Erreursreseau contient l'ensemble des valeurs d'erreur du réseau.

Rés. :

- Somme : valeur réelle contenant l'erreur du neurone;

Post. :

- Somme contient la somme du produit des erreurs des éléments de la couche supérieure par le poids reliant ces éléments à l'élément NUMELEM;
- les autres variables sont inchangées.

B. Module CALCUL_ERREUR_COUCHE.

Arg. :

- Numcouche : entier représentant le numéro de la couche dont on calcule l'erreur de chacun de ses éléments;
- Numinp : entier représentant le numéro de l'input dont on calcul l'erreur;
- Ensoutputdes;
- Longcouches;
- Nbrcouches;

- Ensvaleurs;
- Erreursreseau;
- Enspoids.

Pré. :

- 2 <= Numcouche <= Nbrcouches;
- Numinp ≤ Nbrinputs;
- Ensoutputdes est initialisé;
- Longcouches[Numcouche] ≤ Maxelem;
- Nbrcouches ≤ Maxcouche;
- Ensvaleurs contient l'ensemble des valeurs des neurones du réseau;
- Erreursreseau contient l'ensemble des valeurs des erreurs du réseau;
- Enspoids est initialisé.

Rés. :

- Erreursreseau;

Post. :

- Erreursreseau contient les erreurs des neurones de la couche actuelle Numcouche;
- les autres variables sont inchangées.

C. Module CALCUL_SOMME_POIDS_CON.

Arg. :

- Couche : entier contenant le numéro de la couche;
- Neur : entier contenant le numéro du neurone dans la couche;
- Longcouches;
- Enspoids;
- Ensvaleurs;
- Ensbias;

Pré. :

- 2 ≤ Couche ≤ Nbrcouches;
- 1 ≤ Neur ≤ Longcouches[Couche];
- Ensbias est initialisé;
- les autres variables respectent les préconditions énoncées dans les modules ci-dessus.

Rés. :

- Sommepoids : valeur entière.

Post. :

- Sommepoids représente la somme des produits des différents poids des conexions reliant le neurone Neur à la couche Couche-1 par les outputs des neurones extrémités de ces connexions;
- les autres variables sont inchangées.

D. Module CALCUL_NOUVEAU_POIDS_COUCHE.

Arg. :

- Gain : valeur de type réel intervenant dans le calcul d'un poids;
- Numcouche : valeur de type Entier indiquant le numéro de la couche dont on calcule les poids des connexions la reliant à la couche inférieure;
- Longcouches;
- Erreursreseau;
- Alpha;
- Enspoids;
- Ancpoids;
- Ensbias;
- Ancbias;
- les autres variables respectent les préconditions énoncées dans les modules ci-dessus.

Pré. :

- $0 < \text{Gain} < 1$;
- $2 \leq \text{Numcouche} \leq \text{Nbrcouches}$;
- Alpha;
- Ancpoids contient les poids du réseau précédent l'exécution de la procédure Calcul_Nouveau_Poids_Couche; lors de la première exécution, Ancpoids est indéterminé;
- Ancbias contient l'ensemble des poids des connexions des neurones vers la référence avant l'exécution de la procédure Calcul_Nouveau_Poids_Couche; lors de la première exécution, Ancbias est indéterminé;

Rés. :
- Enspoids;
- Ancpoids;
- Ensbias;
- Ancbias.

Post. :

- Enspoids contient les nouveaux poids de connexions du réseau;
- Ancpoids contient les anciens poids de connexions du réseau;
- Ensbias contient les nouvelles valeurs de bias du réseau;

- Ancbias contient les anciennes valeurs de bias du réseau.

E. Module CALCUL_OUTPUT.

Arg. :

- Nbrcouches;
- Longcouches.

Pré. :

- les variables respectent les préconditions énoncées dans les modules ci-dessus.

Rés. :

- Ensvaleurs.

Post. :

- Ensvaleurs contient les outputs des neurones du réseau.

F. Module COMPARE_OUTPUT_RESEAU.

Arg. :

-Arret : variable indiquant si le réseau a suffisamment appris et s'il faut dès lors arrêter l'exécution;
- Numinp : numéro d'un input;
- Nbrinputs;
- Longcouches;
- Ensinputs;
- Ensvaleurs;

Pré. :

$1 \leq$ Numinp $\leq$ Maxinput;
- Numinp contient le numéro de l'input courant;
- Les autres variables respectent les préconditions énoncées dans les modules ci-dessus.

Rés. :

- Arret.

Post. :

- Arret contient la valeur indiquant s'il faut poursuivre ou non l'exécution du programme.

G.1. Module ENTREE

Arg. :

- Gain;
- Maxelem;
- Maxcouche;
- Maxinputs;
- Nbrinputs;
- Nbrcouches;
- Longcouches;

Pré. :

Rés. :

- Ensoutputdes;
- Enspoids;
- Ensbias;
- Ensinputs;

Post. :

- Ensinputs contient les inputs de l'utilisateur;
- Ensoutputdes contient les outputs désirés de l'utilisateur;
- Les inputs et les outputs désirés sont rangés par ordre croissant dans Ensinputs et Ensoutputdes respectivement;
- Enspoids contient les valeurs des poids du réseau;
- Ensbias contient les valeurs des poids des neurones vers la référence;

G.2. Module ENTREER.

Spécifications identiques au module Entrees. Le module Entreer permet à l'utilisateur d'introduire le nom d'un fichier de données plutôt que d'introduire les données manuellement.

H. Module SORTIES

Arg. :

- Numinp : valeur entière;
- Compt : valeur entière;
- Tour : valeur entière;
- Longcouches;
- Ensoutputdes;
- Ensvaleurs;

Pré. :

1≤ Numinp ≤ Maxinputs;
- Numinp contient le numéro du dernier input traité;
- Compt contient le nombre de cycles effectué sur les inputs;
1≤ Tour ≤ Nbrinputs;
- Tour contient le nombre d'inputs tels que la différence entre la valeur de leur output désiré correspondant et l'output effectif est inférieure à Comp;
- Les autres variables respectent les préconditions énoncées dans les modules ci-dessus.

I. PROGRAMME PRINCIPAL

Le but de ce programme est de réaliser un apprentissage de rétro-propagation sur réseau Perceptron multi-couches. L'utilisateur indique le nombre de couches qu'il veut employer (≤ 4), le nombre d'éléments par couches, les inputs et outputs désirés qu'il souhaite utiliser et le taux d'apprentissage.

Arg. :

- Longcouches;
- Ensvaleurs;
- Ensinputs;
- Arret;
- Nbrcouches;

D.3. TEXTE PASCAL

```
(----------------------------------------------------------------------------------------)

unit dec;

interface

(*****************************************************************)
(*                                                               *)
(* Déclarations des constantes, types et variables globales      *)
(*                                                               *)
(*****************************************************************)

  const

    Maxelem = 30;   (* nbr maximum d'éléments d'une couche *)
```

```
    Maxcouche = 3;  (* nbr maximum de couche du réseau *)
    maxinput = 50;  (* nbr maximum d'inputs que le réseau doit
                       apprendre *)
    comp = 0.04;    (* erreur d'approximation *)
    alpha = 0.9;    (* constante du momentum *)

  type

    valcouche = array[1..Maxelem] of real;
(* valcouche : la valeur des outputs des neurones d'une couche *)

    valreseau = array[1..Maxcouche] of valcouche;
(* valreseau : l'ensemble des outputs des neurones du réseau *)

    uninput = array[1..Maxelem] of real;
(* un input présent_ au réseau *)

    valinput = array[1..maxinput] of uninput;
(* l'ensemble des inputs que le réseau doit apprendre *)

    valoutputdes = array[1..maxinput] of uninput;
(* l'ensemble des outputs désirés.  Il y en a autant (maxinput) que
d'inputs introduits *)

    longueur = array[1..Maxcouche] of integer;
(* longueur : le nombre d'éléments dans chacune des couches *)

    poidselem = array[1..Maxelem] of real;
(* poidselem : les poids des connexions reliant un noeud à la couche inf *)

    poidscouche = array[1..Maxelem] of poidselem;
(* poidscouche : tous les poids des connexions d'une couche vers la couche
inférieure *)

    poidsreseau = array[2..Maxcouche] of poidscouche;
(* poidsreseau : les poids de toutes les connexions du réseau *)
(* 2 car il n'y a pas de connexions en dessous de la couche d'input *)

    biascouche = array[1..Maxelem] of real;
(* poids des connexions des éléments d'une couche vers la référence *)

    biasreseau = array[1..Maxcouche] of biascouche;
(* poids de toutes les connexions du réseau vers la référence *)

    erreurscouche = array[1..Maxelem] of real;
(* erreurs des neurones d'une couche *)
```

```
var
    ensvaleurs: valreseau;          (* tableau des outputs de tous les
                                       neurones *)
    ensinputs: valinput;            (* tableau des inputs *)
    ensoutputdes: valoutputdes;     (* tableau des outputs désirés du
                                       réseau *)
    enspoids: poidsreseau;          (* tableau des poids des couches *)
    ancpoids: poidsreseau;          (* tableau des poids au temps t-1 *)
    ensbias: biasreseau;            (* tableau des bias du réseau *)
    ancbias: biasreseau;            (* tableau des bias au temps t-1 *)
    nbrcouches: integer;            (* le nombre de couches du réseau *)
    nbrinputs: integer;             (* nbr d'inputs *)
    longcouches: longueur;          (* tableau des longueurs des
                                       différentes couches *)
    erreursreseau: array[1..Maxcouche] of erreurscouche;
                                    (* tableaux servant au calcul des
                                       erreurs retro-propagées *)
    erreurs: valinput;              (* tableau contenant les différences
                                       entre les outputs désirés et les outputs
                                       calculés *)
    tour: integer;                  (* nbr d'inputs appris *)

implementation

(*****************************************************************)
(*                                                               *)
(* fin des déclarations.                                         *)
(*                                                               *)
(*****************************************************************)

end.

(-------------------------------------------------------------------------------)

unit cse;

interface
  uses
    dec;

  procedure calcul_somme_erreurs (numcouche: integer; numelem:
integer; var somme: real);

(*****************************************************************)
(*                                                               *)
(* Cette procédure calcule, pour un élément NUMELEM de la        *)
(* couche NUMCOUCHE,                                             *)
(* la somme (SOMME) du produit des erreurs des éléments de la    *)
```

```
(* couche supérieure par le poids reliant ces éléments à l'élément     *)
(* NUMELEM                                                             *)
(*                                                                     *)
(***********************************************************************)

implementation

   procedure calcul_somme_erreurs (numcouche: integer; numelem:
integer; var somme: real);

      var
         valeur: real;
         i: integer;

   begin
      somme := 0;
      for i := 1 to longcouches[numcouche + 1] do
         begin
            valeur := enspoids[numcouche + 1, i, numelem] *
                      erreursreseau[numcouche + 1, i];
            somme := somme + valeur
         end
   end;

(***********************************************************************)
(*                                                                     *)
(* fin de la procedure CALCUL_SOMME_ERREURS                            *)
(*                                                                     *)
(***********************************************************************)

end.

(-----------------------------------------------------------------------------------------)
unit CEC;

interface
   uses
      CSE, DEC;

   procedure CALCUL_ERREUR_COUCHE (numcouche: integer;
numinp: integer);

(***********************************************************************)
(*                                                                     *)
(* Cette procédure calcule les erreurs des neurones de la couche       *)
(* NUMCOUCHE.  S'il s'agit de la couche d'output, l'output désiré      *)
(* OUTPUTDES est utilisé dans le calcul.                               *)
(*                                                                     *)
(***********************************************************************)
```

```
implementation

   procedure CALCUL_ERREUR_COUCHE (numcouche: integer;
numinp: integer);

      var

         lc,              (* longueur d'une couche *)
         nbc,             (* nbr de couches *)
         nrc,             (* num de la couche *)
         i: integer;      (*compteur *)
         somme: real; (* somme des erreurs par rapport à la couche sup *)
         Oneur: real;  (* output d'un neurone   *)


   begin
      lc := longcouches[numcouche];
      nbc := nbrcouches;
      nrc := numcouche;
      if nrc = nbc then
         begin
            i := 1;
            while i <= lc do
               begin
                  Oneur := ensvaleurs[nrc, i];
                  erreursreseau[nrc, i] := Oneur * (1 - Oneur) *
                  (ensoutputdes [numinp, i] - Oneur);
                  i := i + 1
               end
         end
      else
         begin
            i := 1;
            while i <= lc do
               begin
                  CALCUL_SOMME_ERREURS(nrc, i, somme);
                  Oneur := ensvaleurs[nrc, i];
                  erreursreseau[nrc, i] := Oneur * (1 - Oneur) * somme;
                  i := i + 1
               end
         end;
   end;

(*****************************************************************)
(*                                                               *)
(* fin de la procédure CALCUL_ERREUR_COUCHE                      *)
(*                                                               *)
(*****************************************************************)
```

```
end.

(-------------------------------------------------------------------------------)

unit CSPC;

interface
  uses
    DEC, CRT;


  procedure CALCUL_SOMME_POIDS_CON (couche: integer; neur:
integer; var sommepoids: real);

(*****************************************************************)
(*                                                               *)
(* Cette proc_dure calcule l'input d'un neurone. Le neurone NEUR *)
(* de la couche COUCHE à un input égal à SOMMEPOIDS.             *)
(*                                                               *)
(*****************************************************************)

implementation

  procedure CALCUL_SOMME_POIDS_CON (couche: integer; neur:
integer; var sommepoids: real);

    var
      ch: char;
      coucheinf,          (* num_ro de la couche inf_rieure *)
      k, lcinf: integer;  (* longueur couche inf_rieure *)

  begin
    coucheinf := couche - 1;
    sommepoids := 0;
    lcinf := longcouches[coucheinf];
    for k := 1 to lcinf do
      sommepoids := sommepoids + enspoids[couche, neur, k]
                    * ensvaleurs[coucheinf, k];
    sommepoids := sommepoids + ensbias[couche, neur]
                    * 1;(*référence*)
  end;

(*****************************************************************)
(*                                                               *)
(* fin de la procédure CALCUL_SOMME_POIDS_CON                    *)
(*                                                               *)
(*****************************************************************)
```

```
end.

(-------------------------------------------------------------------------------------)

UNIT cnpc;

interface
   uses dec;

   procedure calcul_nouv_poids_couche (gain: real; numcouche:
integer);

(*****************************************************************)
(*                                                               *)
(* Cette procédure calcule les poids reliant les éléments d'une  *)
(* couche (NUMCOUCHE) au différents éléments de la couche        *)
(* inférieure                                                    *)
(*                                                               *)
(*****************************************************************)

implementation

   procedure calcul_nouv_poids_couche (gain: real; numcouche:
integer);

      var
         i: integer;
         j: integer;
         valeur: real;
         produit: real;

   begin
      for i := 1 to longcouches[numcouche] do
         begin
            for j := 1 to longcouches[numcouche - 1] do
               begin
                  produit := gain * erreursreseau[numcouche, i] *
                             ensvaleurs[numcouche - 1, j];
                  valeur := enspoids[numcouche, i, j] + produit +
                            alpha * (enspoids[numcouche, i, j] -
                            ancpoids[numcouche, i, j]);
                  ancpoids[numcouche, i, j] := enspoids[numcouche, i, j];
                  enspoids[numcouche, i, j] := valeur
               end;
            produit := gain * erreursreseau[numcouche, i] * 1;  (* ref=tjs 1 *)
            valeur := ensbias[numcouche, i] + produit +
                      alpha * (ensbias[numcouche, i] -
                      ancbias[numcouche, i]);
            ancbias[numcouche, i] := ensbias[numcouche, i];
```

```
            ensbias[numcouche, i] := valeur
        end
    end;

(*****************************************************************)
(*                                                               *)
(* fin procedure calcul_nouv_poids_couche                        *)
(*                                                               *)
(*****************************************************************)

end.

(-------------------------------------------------------------------------------------)

UNIT co;

interface
    uses CSPC,DEC;

procedure CALCUL_OUTPUT;

(*****************************************************************)
(*                                                               *)
(* Cette procédure calcule l'output de chacun des neurones du    *)
(* réseau et range le résultat dans la variable globale ENSPOIDS.*)
(*                                                               *)
(*****************************************************************)

implementation

procedure CALCUL_OUTPUT;

var

  nbc,                  (* nbr de couches *)
  i,j,
  lc : integer;         (* longueur d'une couche (= nbr d'éléments) *)
  poidscon : real;      (* somme des poids des connexions reliant un
                           élément aux autres éléments de la couche
                           inférieure *)
  temp : real;          (* var temporaire *)
  fich : file of real;


begin
     nbc:=nbrcouches;
    for i:=2 to nbc do
    begin
        lc:=longcouches[i];
```

```
      for j:=1 to lc do
      begin
         CALCUL_SOMME_POIDS_CON(i,j,poidscon);
         if poidscon<-10  (*-5*)
           (* avec -4.6--> 1/(1.exp(-poidscon)) < 0.01 *)
         then temp:=0
         else if poidscon>10 (*5*)
            then temp:=1 (* avec 4.96 --> 1/(1+exp(-poidscon)) > 0.99*)
            else temp:=1/(1+exp(-poidscon));
          ensvaleurs[i,j]:=temp
      end
   end
end;

(*****************************************************************)
(*                                                               *)
(* fin de la procédure CALCUL_OUTPUT                             *)
(*                                                               *)
(*****************************************************************)

end.

(---------------------------------------------------------------------------------------)

unit cor;

interface

   uses
      dec, printer, co;

(*****************************************************************)
(*                                                               *)
(* Cette procédure compare les outputs calculés avec leur output *)
(* désiré respectif.  Il y a correspondance lorsque les deux outputs *)
(* calculés et désirés diffèrent d'une valeur au moins égale à COMP. *)
(* Si plus de 5% des output calculés diffèrent des outputs désirés, la *)
(* variable ARRET prendra la valeur FALSE (TRUE dans le cas      *)
(* contraire; tous les outputs convergent.                       *)
(*                                                               *)
(*****************************************************************)

   procedure COMPARE_OUTPUTS_RESEAU (var arret: boolean;
numinp: integer);

implementation

   procedure COMPARE_OUTPUTS_RESEAU (var arret: boolean;
numinp: integer);
```

```
    var
      i, j, total: integer;
      egal: boolean;

  begin
    total := 0;
    arret := true;
    i := 1;
    while (i <= nbrinputs) and arret do
      begin
        for j := 1 to longcouches[1] do
          ensvaleurs[1, j] := ensinputs[i, j];
        CALCUL_OUTPUT;
        j := 1;
        egal := true;
        while (j <= longcouches[nbrcouches]) and (egal = true) do
          begin
            if abs(ensoutputdes[i, j] - ensvaleurs[nbrcouches, j])
            > comp then
              egal := false;
            j := j + 1
          end;
        if not (egal) then
          begin
            total := total + 1;
            if total >= trunc(nbrinputs * 0.05) - 1 then
              arret := false
          end;
        i := i + 1
      end;
    tour := i - 1;
    for j := 1 to longcouches[1] do
      ensvaleurs[1, j] := ensinputs[numinp, j];
    CALCUL_OUTPUT
  end;

(*****************************************************************)
(*                                                               *)
(* fin de la procédure COMPARE_OUTPUTS_RESEAU                    *)
(*                                                               *)
(*****************************************************************)

end.

(-------------------------------------------------------------------------------)

unit entree;
```

```
interface
  uses
    DEC, CRT, PRINTER;

  procedure entrees (var gain: real);

(*****************************************************************)
(*                                                               *)
(* Cette procédure a pour but de lire les différentes entrées;   *)
(* le nombre de couche (< Maxcouche), le nombre d'éléments dans  *)
(* chacune des couches (< Maxelem), l'input, l'output désiré (= 1*)
(* ou = 0) et le gain (0 < gain <). La matrice des poids est initialisée. *)
(*                                                               *)
(*****************************************************************)

implementation

  procedure entrees (var gain: real);

    var
      y: integer;
      i: integer;
      j: integer;
      k: integer;
      signe: integer;
      valeur: real;
      nimp, altern: integer;
      continue: char;

  begin
    clrscr;
    y := 1;
    gotoxy(31, y);
    write('ENTREE DES VALEURS');
    y := y + 2;
    gotoxy(1, y);
    write('nombre maximum de couche autorisées : ', Maxcouche);
    y := y + 1;
    gotoxy(1, y);
    write('nombre de couche dans le r_seau (y compris l''input) : ');
    read(nbrcouches);
    while (nbrcouches > Maxcouche) or (nbrcouches < 2) do
      begin
        gotoxy(55, y);
        write('          ');
        gotoxy(55, y);
        read(nbrcouches)
      end;
    y := y + 2;
```

```
gotoxy(1, y);
write(' Gain ( > 0 ) : ');
read(gain);
while (gain <= 0) do
   begin
      gotoxy(21, y);
      write('          ');
      gotoxy(21, y);
      read(gain)
   end;
y := y + 2;
gotoxy(1, y);
write('nombre maximum d''éléments par couche : ', Maxelem);
y := y + 1;
gotoxy(1, y);
write('nombre d''éléments de la couche d''input : ');
read(longcouches[1]);
while (longcouches[1] > Maxelem) or (longcouches[1] < 1) do
   begin
      gotoxy(42, y);
      write('          ');
      gotoxy(42, y);
      read(longcouches[i])
   end;
y := y + 1;
for i := 2 to nbrcouches do
   begin
      gotoxy(1, y);
      write('nombre d''éléments dans la ', i, ' ème couche : ');
      gotoxy(43, y);
      read(longcouches[i]);
      while (longcouches[i] > Maxelem) or (longcouches[i] < 1) do
         begin
            gotoxy(43, y);
            write('       ');
            gotoxy(43, y);
            read(longcouches[i]);
         end;
      y := y + 1
   end;
nimp := 1;
continue := 'O';
while (continue = 'O') or (continue = 'o') do
   begin
      writeln;
      writeln;
      writeln('Introduction de l''input n° ', nimp);
      writeln;
      for i := 1 to longcouches[1] do
```

```
        begin
          write('valeur de l''élément ', i, ' de l''input : ');
          readln(ensinputs[nimp, i]);
        end;
      writeln;
      writeln('le nombre d''éléments de l''output désiré doit', 'être
              égal à : ', longcouches[nbrcouches]);
      writeln('les éléments doivent être égaux à 0 ou 1');
      writeln;
      for i := 1 to longcouches[nbrcouches] do
        begin
          write('valeur de l''élément n° ', i, 'de l''output désiré : ');
          readln(ensoutputdes[nimp, i]);
          while not (ensoutputdes[nimp, i] = 1) and not
                (ensoutputdes[nimp, i] = 0) do
            begin
              gotoxy(51, wherey);
              write('      ');
              gotoxy(51, wherey);
              read(ensoutputdes[nimp, i])
            end;
          writeln;
        end;
      writeln;
      write('Encore des inputs à introduire ? (O/N) ');
      continue := readkey;
      nimp := nimp + 1;
    end;
  nbrinputs := nimp - 1;
  altern := 0;
  randomize;
  for i := 2 to nbrcouches do
    for j := 1 to longcouches[i] do
      begin
        for k := 1 to longcouches[i - 1] do
          begin
            valeur := random;
            if altern = 1 then
              begin
                valeur := valeur * - 1;
                altern := 0
              end
            else
              altern := 1;
            enspoids[i, j, k] := valeur * 5
          end;
      end;
  for i := 2 to nbrcouches do
    for j := 1 to longcouches[i] do
```

```
            begin
               valeur := random;
               valeur := valeur * - 1;
               ensbias[i, j] := valeur * 5
            end;
      for i := 1 to nbrinputs do
         for j := 1 to Maxelem do
            erreurs[i, j] := 1000;
   end;

(*****************************************************************)
(*                                                               *)
(* fin de la procedure ENTREES                                   *)
(*                                                               *)
(*****************************************************************)

end.

(-------------------------------------------------------------------------------------------)

unit entreer;

interface
   uses
      DEC, CRT, PRINTER;

   procedure ENTREES_ROBOT (var gain: real; var fin: boolean);

(*****************************************************************)
(*                                                               *)
(*  Cette procédure demande à l'utilisateur le nombre de couche  *)
(* qu'il veut utiliser, le gain et le nombre de neurones de la (des) *)
(*  couche(s) cachée(s).                                         *)
(*                                                               *)
(*  Les entrées et sorties sont lues à partir d'un fichier.      *)
(*                                                               *)
(*****************************************************************)

implementation

   procedure ENTREES_ROBOT (var gain: real; var fin: boolean);

      type
         record1 = record
               input: array[1..Maxelem] of real;
               output: array[1..Maxelem] of real;
            end;
         record2 = record
               nomf: string[12];
```

```
        inn: integer;
        outt: integer
      end;

  var
    i, j, k, y, signe, altern: integer;
    valeur: real;
    erreur, trouve, mistake: boolean;
    nomfich: string[12];
    fich1: file of record1;
    fich2: file of record2;
    elem1: record1;
    elem2: record2;

begin
  clrscr;
  fin := false;
  y := 1;
  gotoxy(31, y);
  write('ENTREE DES VALEURS');
  y := y + 2;
  gotoxy(1, y);
  write('nombre maximum de couches autorisées : ', Maxcouche);
  y := y + 1;

 (* lecture du nombre de couches du réseau (<=Maxcouche et >=2)  *)

  gotoxy(1, y);
  write('nombre de couches dans le réseau', ' (y compris la couche
        d''input) : ');
  readln(nbrcouches);
  while (nbrcouches > Maxcouche) or (nbrcouches < 2) do
    begin
      gotoxy(66, y);
      write('          ');
      gotoxy(66, y);
      readln(nbrcouches)
    end;

 (* lecture du nombre de neurones dans les différentes couches *)

  y := y + 1;
  gotoxy(1, y);
  for i := 1 to nbrcouches do
    begin
      gotoxy(1, y);
      write('nombre d''éléments dans la ', i, ' ème couche : ');
      gotoxy(43, y);
      readln(longcouches[i]);
```

```
        while (longcouches[i] > Maxelem) or (longcouches[i] < 1) do
          begin
            gotoxy(43, y);
            write('        ');
            gotoxy(43, y);
            readln(longcouches[i]);
          end;
        y := y + 1
      end;

(* lecture du nom de fichier de données *)

  erreur := true; (* le fichier n'existe pas*)
  trouve := false; (* le fichier est dans la liste *)
  mistake := false; (* le nbr de neur In et Out est correct *)
  while (erreur = true) do
    begin
      y := y + 1;
      gotoxy(1, y);
      write('Quel est votre fichier de données (8 char) ?', '(extension
            .NNI) : ');
      gotoxy(65, y);
      readln(nomfich);
      nomfich := nomfich + '.NNI';
      assign(fich1, nomfich);
      (*$I-*)
      reset(fich1);
      (*$I+*)
      if IOresult = 0 then
        erreur := false
      else
        begin
          y := y + 1;
          gotoxy(1, y);
          write('Fichier inexistant...Try again');
          y := y + 1;
        end;
      if not (erreur) then
        begin
          assign(fich2, 'repert.rob');
          (*$I-*)
          reset(fich2);
          (*$I+*)
          if IOresult <> 0 then
            begin
              y := y + 1;
              gotoxy(1, y);
              write('Liste non trouvée');
            end
```

```
            else
               begin
                  trouve := false;
                  while not (trouve) and not (eof) do
                    begin
                       read(fich2, elem2);
                       if elem2.nomf = nomfich then
                          begin
                             trouve := true;
                             mistake := false;
                             if (elem2.inn <> longcouches[1]) or
                             (elem2.outt <> longcouches[nbrcouches])
                             then
                                mistake := true
                          end
                    end;
                  close(fich2)
               end
         end
   end;
if not (trouve) then
   begin
      y := y + 1;
      gotoxy(1, y);
      write('Fichier pas présent dans la liste...Relancer');
      y := y + 1
   end;
if mistake then
   begin
      y := y + 1;
      gotoxy(1, y);
      write('Nbr de neurones d''In ou d''Out incorrect...Relancer');
      y := y + 1;
   end;
if not (mistake) and not (erreur) and trouve then
   begin

 (* lecture du gain *)

      y := y + 2;
      gotoxy(1, y);
      write('  Gain ( > 0 ) : ');
      readln(gain);
      while (gain <= 0) do
         begin
            gotoxy(21, y);
            write('          ');
            gotoxy(21, y);
            readln(gain)
```

```
      end;

(* garnissage du tableau ENSINPUTS des inputs *)

    i := 1;
    while not eof(fich1) and (i <= maxinput) do
      begin
        read(fich1, elem1);
        for j := 1 to longcouches[1] do
          ensinputs[i, j] := elem1.input[j];
        for j := 1 to longcouches[nbrcouches] do
          ensoutputdes[i, j] := elem1.output[j];
        i := i + 1
      end;
    nbrinputs := i - 1;
    close(fich1);


(* fixation des poids des connexions du réseau *)

    altern := 0;
    randomize;
    for i := 2 to nbrcouches do
      for j := 1 to longcouches[i] do
        begin
          for k := 1 to longcouches[i - 1] do
            begin
              valeur := random;
              if altern = 1 then
              begin
              valeur := valeur * - 1;
              altern := 0
              end
              else
              altern := 1;
              enspoids[i, j, k] := valeur * 5;
              ancpoids[i, j, k] := enspoids[i, j, k]
            end;
        end;

(* fixation des poids des connexions à la référence *)

    for i := 2 to nbrcouches do
      for j := 1 to longcouches[i] do
        begin
          valeur := random;
          valeur := valeur * - 1;
          ensbias[i, j] := valeur * 5;
          ancbias[i, j] := ensbias[i, j]
```

```
                end;

        (* garnissage du tableau des erreurs à une grde valeur arbitraire *)

            for i := 1 to nbrinputs do
              for j := 1 to longcouches[nbrcouches] do
                erreurs[i, j] := 1000
          end
        else
          fin := true
      end;

(*****************************************************************)
(*                                                               *)
(*  fin de la procédure ENTREES_ROBOT                            *)
(*                                                               *)
(*****************************************************************)

end.

(-------------------------------------------------------------------------------)

unit SORTIES;

interface
  uses
    DEC, crt;

  procedure SORTIE (numinp: integer; compt: longint);

(*****************************************************************)
(*                                                               *)
(* Cette procédure affiche à l'écran les différents poids des    *)
(* connexions du réseau, l'output du réseau et l'output désiré.  *)
(*                                                               *)
(*****************************************************************)

implementation

  procedure SORTIE (numinp: integer; compt: longint);

    var
      lc,                          (* longueur d'une couche *)
      i, j, k, coucheinf: integer; (* numéro de la couche inférieure *)
      posy: integer;               (* position de la ligne du curseur *)

  begin
    clrscr;
    window(40, 1, 80, 25);
```

```
    lc := longcouches[nbrcouches];
    for j := 1 to lc do
      writeln('ensoutputdes [', numinp, j, ']');
    gotoxy(14, 1);
    for j := 1 to lc do
      writeln(ensoutputdes[numinp, j] : 4 : 3, ' ');
    writeln;
    posy := wherey;
    for i := 1 to lc do
      writeln('ensvaleurs [', nbrcouches, i, ']');
    gotoxy(16, posy);
    for i := 1 to lc do
      writeln(ensvaleurs[nbrcouches, i] : 5 : 4, ' ');
    writeln;
    write('numinp : ', numinp);
    writeln;
    write('Compteur : ', compt);
    writeln;
    write('Tours : ', tour)
  end;

(*************************************************************************)
(*                                                                       *)
(* fin de la procédure SORTIE.                                           *)
(*                                                                       *)
(*************************************************************************)

end.

(-------------------------------------------------------------------------------------------)

program backpercep;

  uses
    cnpc, cec, cse, co, cor, cspc, entreer, sorties, dec, crt, printer;

(*************************************************************************)
(*                                                                       *)
(* Programme principal                                                   *)
(*                                                                       *)
(*************************************************************************)

  var
    stop, arret: boolean;
    ch: char;
    gain: real;
    i: integer;
    compteur: longint;
    numinp: integer;
```

```
begin
   stop := false;
   compteur := 0;
   ENTREES_ROBOT(gain, stop);
   if stop then exit;
   numinp := 1;
   for i := 1 to longcouches[1] do ensvaleurs[1, i] := ensinputs[numinp, i];
   CALCUL_OUTPUT;
   COMPARE_OUTPUTS_RESEAU(arret, numinp);
   while not (arret) do
     begin
       SORTIE(numinp, compteur);
       compteur := compteur + 1;
       i := 1;
       for i := nbrcouches downto 2 do
         CALCUL_ERREUR_COUCHE(i, numinp);
       for i := nbrcouches downto 2 do
         CALCUL_NOUV_POIDS_COUCHE(gain, i);
       numinp := numinp + 1;
       if numinp > nbrinputs then
         numinp := 1;
       for i := 1 to longcouches[1] do
         ensvaleurs[1, i] := ensinputs[numinp, i];
       CALCUL_OUTPUT;
       COMPARE_OUTPUTS_RESEAU(arret, numinp);
     end;
   compteur := compteur + 1;
   SORTIE(numinp, compteur);

(*****************************************************************)
(*                                                               *)
(* fin programme principal                                       *)
(*                                                               *)
(*****************************************************************)

end.

(-----------------------------------------------------------------------------)
(-----------------------------------------------------------------------------)

program repert;

   uses
     DEC, CRT;

(*****************************************************************)
(*                                                               *)
(* Ce programme a pour but de créer et de mettre à jour un       *)
(* répertoire des différents fichiers utilisés. Ce répertoire contient *)
```

```
(* noms des fichiers et le nombre d'éléments d'input et d'output.      *)
(*                                                                     *)
(***********************************************************************)

  type
    record2 = record
          nom: string[12];
          inn: integer;
          outt: integer
        end;

  var
    elem2, elemt: record2;
    fich2: file of record2;
    nbr, i: integer;
    ch: char;
    stop, trouve: boolean;


begin
  clrscr;
  assign(fich2, 'repert.rob');
  reset(fich2);
  write('Nom du nouv fich de data à ajouter (sans l''ext .NNI) : ');
  readln(elem2.nom);
  writeln;
  while length(elem2.nom) > 8 do
    begin
      writeln('Nom trop long...Try again');
      write('Nom du nouv fich de data à ajouter (sans l''ext .NNI) : ');
      readln(elem2.nom);
      writeln
    end;
  elem2.nom := elem2.nom + '.NNI';
  write('Nombre de neurones d''entrée : ');
  readln(nbr);
  while (nbr > Maxelem) or (nbr < 0) do
    begin
      writeln('Nombre de neurones incorrect...Try again');
      write('Nombre de neurones d''entrée : ');
      readln(nbr);
    end;
  elem2.inn := nbr;
  write('Nombre de neurones de sortie : ');
  readln(nbr);
  while (nbr > Maxelem) or (nbr < 0) do
    begin
      writeln('Nombre de neurones incorrect...Try again');
      write('Nombre de neurones de sortie : ');
```

```
      readln(nbr);
    end;
  i := 0;
  elem2.outt := nbr;
  trouve := false;
  while not (eof(fich2)) and not (filesize(fich2) <= 0) and not (trouve) do
    begin          (* si pas filesize alors même si fich2=vide il y a lecture *)
      read(fich2, elemt); (* car fich2 contient des blancs.
                             Or blanc<>elemt *)
      i := i + 1;
      if elemt.nom = elem2.nom then
        begin
          trouve := true;
          ch := ' ';
          stop := false;
          while not (stop) do
            begin
              writeln;
              writeln('Voulez-vous supprimer l''ancien fichier ?
                       (O/N)');
              readln(ch);
              if (ch = 'o') or (ch = 'O') or (ch = 'n') or (ch = 'N') then
                stop := true;
            end;
          if (ch = 'o') or (ch = 'O') then
            begin
              seek(fich2, i - 1);
              write(fich2, elem2)
            end
        end
    end;
  if not (trouve) or (filesize(fich2) = 0) then
    begin
      seek(fich2, filesize(fich2));
      write(fich2, elem2)
    end;
  close(fich2)

end.

(-----------------------------------------------------------------------------------------)
(-----------------------------------------------------------------------------------------)

program lecinp;

  uses
    crt, dec;
```

```
(*****************************************************************)
(*                                                               *)
(* Ce programme a pour effet de créer un fichier contenant les   *)
(* inputs et outputs désirés.                                    *)
(*                                                               *)
(*****************************************************************)

  type

    record1 = record
          ch1: array[1..Maxelem] of real;
          ch2: array[1..Maxelem] of real;
        end;


  var

    fich1: file of record1;
    elem1: record1;
    i, j: integer;

begin
  clrscr;
  assign(fich1, 'temp');
  rewrite(fich1);
  for i := 1 to 16 do
    begin
      for j := 1 to 2 do
        begin
          write('input n° ', i, ',', ' partie ', j, ' : ');
          readln(elem1.ch1[j]);
        end;
      write('ouput d_sir_ pour l''input ', i, ' : ');
      readln(elem1.ch2[1]);
      write(fich1, elem1);
    end;
  close(fich1);
  write('Le fichier s''appelle "temp."');
end.

(---------------------------------------------------------------------------------------)
```

H Rapport de stage au Brésil

Philippe Nivelles,
Vincent Thiry,

Facultés Universitaires
Notre Dame de la Paix
5000 Namur,
Belgique.

Prof. Walter C. de Lima
UFSC, EEL, GPEB,
88049 Florianopolis S.C.

Brésil.

Cher professeur Lima,

nous vous communiquons notre rapport de stage effectué dans votre laboratoire d'ingéniérie électrique. Nous avons cru bon de diviser ce compte rendu en trois parties. La première est relative aux activités de travail que nous avons réalisées; la seconde concerne l'expérience humaine que le stage nous a apporté; et enfin, nous faisons quelques propositions d'améliorations sur le plan professionnel.

1. L'aspect professionnel.

Nos objectifs de travail en quittant la Belgique se situaient dans le domaine des systèmes experts et des langages y afférents. Après nous être documenté sur le sujet et avoir emporté avec nous une documentation appropriée, il s'est avéré que notre stage se déroulerait dans le domaine des réseaux neuronaux. Ce sujet n'était pas pour nous déplaire étant donné que nous avions déjà réalisé un travail -à orientation philosophique- dans ce domaine.

Le stage a commencé par la lecture d'articles traitant des réseaux de neurones et provenant de revues internationnales (IEEE, AIII, Byte,...). A partir de ces lectures nous avons réalisé un premier travail de synthèse qui a donné lieu au premier rapport. Nos connaissances des réseaux neuronaux étaient, à ce moment, suffisantes pour commencer l'apprentissage d'un logiciel de simulation de réseaux, NeuralWorks Professional I. Malheureusement, l'apprentissage n'a pu débuter immédiatement, par manque de la documentation appropriée. En attendant de la recevoir, nous nous sommes attachés à implémenter, en Pascal (Turbo Pascal V. 5.0), un algorithme d'apprentissage de Back-propagation sur réseau Perceptron.

Pour des raisons encore inconnues à ce jour[1], relevant probablement de subtilités d'implémentations, le réseau ainsi créé n'a pas donné les résultats escomptés (absence de convergence des poids synaptiques). Notre attitude première a été de nous pencher sur notre programme aux fins d'y déceler d'éventuelles erreurs de programmation. Cette approche nous a conduit dans un premier temps à réécrire le programme d'une autre façon et ensuite, vu le nouvel échec dans les résultats obtenus, à l'instancier au cas particulier du XOR (ou-exclusif). Les résultats obtenus par cette dernière programmation, que l'on pourrait qualifier de minimale et de plus facilement contrôlable, n'ont pas été meilleurs que ceux obtenus avec les deux versions génériques : divergence entre résultats escomptés et résultats fournis.

Nous avons tout de même continué à travailler sur notre programme, et ce pendant près de trois semaines. Parallèlement au travail de programmation, nous avons continué à lire des articles traitant des réseaux de neurones, ce qui a donné lieu à un second travail de synthèse et donc à un second rapport.

Par après, nous avons reçu la documentation relative au logiciel NeuralWorks Professional I. Nous avons alors travaillé sur ce logiciel en trois étapes. Tout d'abord par la lecture du mode d'emploi qui contenait de nouveaux concepts relatifs aux différents types de réseaux mis en oeuvre par le logiciel (Adaline, Madaline, Counter-propation,...). Ces concepts ont du être assimilés pour pouvoir, dans un second temps, étudier le comportement des réseaux déjà implémentés. L'étude de ces réseaux à révélé certaines étrangetés et notamment l'impossibilité de réobtenir les résultats fournis par le logiciel avant toute modification. Si on combine ceci au fait que le logiciel, d'origine américaine, présente des parties de menus en portugais, ça nous incite à penser que celui-ci a subi des modifications pouvant influencer son fonctionnement normal. Il se peut également que, n'ayant pas reçu d'autres informations d'ordre pratique que celles se trouvant dans le mode d'emploi, nous n'ayons pas réalisé le paramétrage correct de nos réseaux.

La troisième phase consistait en l'implémentation du "ou-exclusif" à partir du logiciel fourni. Cela a pu être réalisé sans problème, dès lors que les poids des connexions du réseau étaient fixées à la main. Cette manière de procéder permet en effet de créer un réseau qui ne nécessite pas d'apprentissage puisque ses poids de connexions sont fixés de telle sorte que le réseau produise, dès la première utilisation, le comportement désiré. Par la suite, nous avons remplacé les poids fixés à la main par des poids ayant de petites valeurs aléatoires. Le réseau ainsi modifié

[1] Nous savons maintenant qu'il s'agissait d'une erreur méthodologique dans le calcul de la rétro-propagation de l'erreur. Celle-ci a été détectée en examinant un programme Fortran lors de notre retour en Belgique. Notre programme pascal était sémantiquement et syntaxiquement correct eu égard aux spécifications reçues pendant le stage.

nécessitait donc un apprentissage pour amener ses sorties à tendre vers celles désirées. C'est justement sur ce point que notre travail s'est considérablement ralenti : les différentes combinaisons de règles de transfert, de paramètres d'apprentissage et de valeurs de poids ne permettaient jamais d'obtenir l'égalité entre résultats désirés et résultats obtenus.

Cette situation de blocage de notre travail, combinée aux différentes opportunités qui nous étaient offertes en Belgique de poursuivre notre travail, nous ont incité à rentrer plus tôt que prévu. Notre stage ne s'inscrivant pas dans un programme de recherche financé par un quelconque organisme belge ou étranger, le retour avancé n'a pas posé de problème d'un point de vue contractuel.

2. L'aspect relationnel.

Le second point que nous désirons évoquer dans ce rapport concerne l'expérience humaine apportée par le stage. Avant d'aborder les choses d'un point de vue que l'on pourrait qualifier de macroscopique, nous voudrions d'abord mettre l'accent sur les rapports que nous avons eus avec les étudiants de l'université et plus particulièrement ceux du laboratoire. Sur ce dernier point, nous sommes enchantés de l'accueil chaleureux que nous avons reçu de la part de tous et du climat de camaraderie et d'entraide sans lequel nous n'aurions pu réaliser nos activités. Nous n'oublions pas non plus les professeurs qui ont favorisé notre intégration et les remercions pour tout ce qu'ils ont fait dans cette direction. Il faut également mentionner que notre situation de quatre belges dans un même laboratoire à favorisé une communication interne au groupe, peut être au détriment de celle en direction des autres personnes du département. Néanmoins, notre avis est que l'intégration a été réussie donnant ainsi un bilan largement positif.

D'un point de vue plus général, notre vie de tous les jours a été très enrichissante et pleine d'enseignements. Nous avons appris, de jour en jour, a mieux percevoir la mentalité des gens ainsi que leur façon de vivre et leurs valeurs. En outre, l'économie n'est pas la même qu'ici en Europe et il est intéressant d'observer la manière dont cela se répercute sur les secteurs de la distribution, des banques ou plus simplement sur le comportement des gens vis à vis de la gestion de leur budget. Sur le plan humain en général, nous pouvons donc affirmer que le stage fut très gratifiant et que notre expérience nous sera certainement très utile.

3. Améliorations éventuelles.

En vue d'une plus grande efficacité dans le travail, voici selon nous quelques points qui devraient être améliorés.

I Conclusion

[ANDER,87] : James A. Anderson, Edward J. Wisniewski et Susan R. Viscuso, "Software for Neural Networks". Departement of Cognitive and Linguistic Sciences and Department of Psychology, Brown University Providence, R1 02912;

[ANDER,86] : James A. Anderson, Edward J. Wisniewski & Susan R. Viscuso, "Software for Neural Networks", Sigarch (ACM special interset group on computer architecture) (14) 1, 1986, pp 26-36.

[BLUM,90] : Adam Blum, "Bidirectional Associative Memory Systems in C++", Dr. Dobb's Journal, april 1990.

[CAUDI,88-1] : Maureen Caudill, "Neural Networks Primer Part II", AI Expert, February 1988, pp 55-61.

[CAUDI,88-2] : Maureen Caudill, "Neural Networks Primer Part IV", AI Expert, August 1988, pp 61-67.

[DARPA,88] : Defense Advanced Research Project Agency, "DARPA Neural Network Study (U.S.)", AFCEA International Press, 1988.

[DAVAL,89] : Eric Davalo & Patrick Naïm, "Des RESEAUX de NEURONES", Eyrolles 1989.

[FELDM,86] : Jerome A. Feldman, "Neural representation of conceptual knowledge", Technical Report 189, Department of Computer Science, The University of Rochester, New York 14627, juin 1986;

[GARNI,89] : Marcel Garnier, valery Delamare, Jean Delamare, Thérèse Delamare-Riche, "Dictionnaire des termes médicaux", Maloine, 1989, ISBN 2-224-01910-6;

[GARTH,86] : Simon Garth & Danny Pike, "An integrated system for neural network simulations", Sigarch (ACM special interest group on computer architecture) (14) 1, 1986, pp 37-44.

[HAMME,86] : Dan Hammerstrom, David Maier & Shreekant Thakkar, "The cognitive architecture project", Sigarch (ACM special interest group on computer architecture) (14) 1, 1986, pp 9-21.

[HECHT,88] : Robert Hecht-Nielsen, "Neurocomputing : Picking the human brain", IEEE Spectrum, March 1988, pp 36-41.

[HINTO,87] : Geoffrey E. Hinton, "Connectionist learning procedures", Technical Report CMU-CS-87-115 (version 2), Computer Science Department, Carnegie-Mellon University Pittsburgh PA 15213;

[JONE,87] : William P. Jones & Josiah Hoskins, "Back-Propagation. A generalized delta learning rule.", Byte, October 1987, pp 155-162.

[JOSIN,88] : Gary Josin, "Integrating Neural Network with Robots", AI Expert, August 1988, pp 50-58.

[JUDGE,89] : Peter Judge, "The age of Reason" dans "Systems International", mai 1989;

[KLIMA,87-1] : Casimir C. Klimasauskas, "Teaching your computer to learn : applications of Neaural Computing", Neural Ware Inc., pp 5-32.

[KLIMA,87-2] : Casimir C. Klimasauskas, dans "NeuralWorks ProfessionalTM, Networks I" de NeuralWare Incorporated, septembre 1987, pp 3-74.

[KLIMA,87-3] : Casimir C. Klimasauskas, dans "NeuralWorks ProfessionalTM, Networks II" de NeuralWare Incorporated, septembre 1987, pp 3-15.

[KLIMA,87-4] : Casimir C. Klimasauskas, "An introduction to Neural Computing" dans "NeuralWorks ProfessionalTM" de NeuralWare Incorporated, septembre 1987;

[KOHON,84] : Teuvo Kohonen, "Self-organization and Associative Memory", Springer-Verlag Berlin Heidelberg New York Tokyo, 1984, ISBN 3-540-12165;

[KOLON,87] : Michele A. Kolonay & Casimir C. Klimasauskas. Neural Ware Professional I, "An introduction to neural computing", Neural Ware Inc., pp 3-67.

[KOSKO,87] : Bart Kosko, "Fuzzy Associative Memories", "Fuzzy Expert Systems", Addison-Wesley, après 87;

[LIPPM,88] : Richard P. Lippmann, "An introduction to computing with Neural Nets", Sigarch (ACM special interest group on computer architecture) (16) 1, 1988, pp 7-25.

[McCUL,43] : Warren S. McCulloch and Walter Pitts, "A logical calculus of the ideas immanent in nervous activity", "The Bulletin of Mathematical Biophysics", december 1943;

[MICHA,83] : Jaime G. Carbonell,Ryszard S. Michalski, et Tom M. Mitchell, "An overview of Machine Learning", "Machine learning : An artificial intelligence approach (Volume 1)", p. 3-16, Springer-Verlag Berlin Heidelberg New York Tokyo, 1983, ISBN 0-935382-05-4;

[MICHA,86] : Ryszard S. Michalski, "Understanding the nature of learning : issues and research directions", "Machine Learning : An artificial intellignece approach (Volume 2)", p. 3-26, Morgan Kaufmann Publishers, Inc., 1986, ISBN 0-934613-00-1;

[MINSK,69] : Marvin L. Minsky, Seymour S. Papert, "Perceptrons", presse du MIT, 1969;

[PAO,89] : Yoh-Han Pao, "Adaptive Pattern Recognition and Neural Networks", Addison Wesley Publishing Company, Inc., 1989.

[PERSO,88] : Léon Personnaz, Gérard Dreyfus & Isabelle Guyon, "Les machines neuronales", La Recherche, vol 19, numéro 204, Novembre 1988, pp 1362-1371.

[REMY,87] : Claire Rémy. "Les mémoires associatives : Quand l'ordinateur s'inspire du cerveau", Micro Systèmes, Mars 1987, pp 85-96.

[RUMEL,88] : David E. Rumelhart, James L. McClelland. "Parallel Distributed Processing. Exploration in the microstucture of cognition.", Volume 1 : Foundations, MIT press.

[SCHWA,88-1] : Tom J. Schwartz, "12-Product-Wrap-up : Neural Networks", AI Expert, August 1988, pp 73-85.

[SCHWA,88-2] : Tom J. Schwartz & Roger Schelm, "Brain Waves", AI Expert, August 1988, pp 85-87.

[SIMON,83] : Herbert A. Simon, "Why should Machines Learn ?" dans "Machine Learning : an artificial intelligence approach (Volume 2)", cfr. [MICHA,83];

[TANK,88] : David W. Tank et John J. Hopfield, "Collective computation in neuronlike circuits" dans "Scientific American", décembre 1987;

[TRELE,88] : Robert B. Trelease, "Connectionism, Cybernetics, and the Cerebellum", AI Expert, August 1988, pp 30-35.

[YOON,89] : Dr. Barbara Yoon, "Artificial Neural Network Technology", Sigsmall/Pc Notes, Vol 15, Number 3, August 1989, pp 3-16.

[WASSE,89] : Philip D. Wasserman, "Neural Computing. Theory and Practice.", Van Nostrand Reinhold 1989.

Réalisation, impression et reliures : SPRL Centre de Reprographie - Tél. : 081/22.87.72