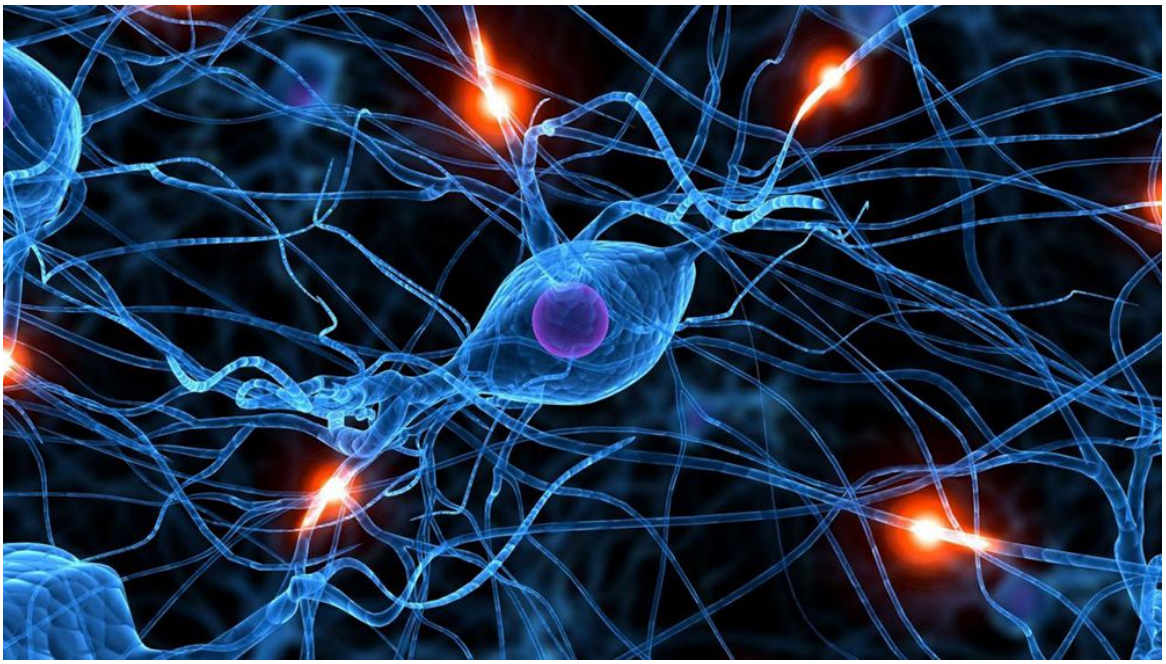
Les Réseaux de Neurones Artificiels



I - Introduction

Le cerveau humain a une fantastique puissance de traitement de l'information si l'on considère ses capacités à prendre en charge certaines taches nécessaires pour obtenir un comportement intelligent. La nature de l'intelligence a longtemps été un sujet difficile et controverse. D'une certaine manière, chacun a une idée assez vague de ce qu'est l'intelligence : la capacité à observer, à comprendre, à se souvenir, à résoudre des problèmes, à apprendre, à créer,…etc. Deux disciplines sont concernées par la définition et la modélisation du comportement intelligent : ce sont la psychologie cognitive et l'intelligence artificielle (IA en jargon informatique).

La psychologie cognitive est l'étude scientifique des structures de mémorisation et des processus de raisonnement humain. Le but de l'IA est le développement des machines capables d'avoir un comportement intelligent. Le critère pour estimer la qualité d'un modèle en « IA » diffère de ceux de la neurophysiologie ou des sciences cognitives. En « IA », on met l'accent sur le développement des systèmes à hautes performances plutôt que sur la reproduction fidèle ou l'explication du comportement humain et a fortiori des données neuro-physiologiques. Au fur et à mesure que progresse l'informatique dans les années cinquante et soixante, il devient clair que les domaines privilégiés à court terme de l'ordinateur sont le calcul scientifique et la gestion. Les performances de celui-ci dans des tâches plus ″intelligentes″ (traduction automatique, vision et reconnaissance des formes sont plus discutables).

Les techniques de l'intelligence artificielle telles que les systèmes experts, la logique floue, les algorithmes génétiques et les réseaux de neurones artificiels (RNA) ont été largement utilisées dans le domaine de l'électronique de puissance et de la commande des machines électriques. L'objectif recherché dans l'utilisation des techniques de l'intelligence artificielle est d'arriver à l'émulation du raisonnement humain sur un DSP (Digital Signal Processor) (processeur de signal numérique) de telle sorte que le système complet commande - machine puisse penser et réagir intelligemment comme un être humain. Un système commande - machine équipé d'un algorithme développant un calcul par intelligence est appelé système intelligent. En effet, un système intelligent possède la caractéristique d'apprentissage, d'auto-organisation et d'auto-adaptation. Les techniques de l'intelligence artificielle ont été discutées pendant longtemps et le seront également à l'avenir.

Actuellement, les techniques de l'intelligence artificielle sont largement utilisées dans de nombreux domaines tels que la régulation de processus industriels, le traitement d'image, le diagnostic, la médecine, la technologie spatiale et les systèmes de gestion de données informatiques. Parmi toutes les techniques intelligentes, les réseaux de neurones artificiels (RNA) semble avoir le maximum d'impact dans le domaine de l'électronique de puissance et dans la commande de machines électriques, ce qui est évident par le nombre important de publications réalisées dans la littérature.

Les RNA constituent une technique de traitement de données bien comprise et bien maitrisée. Ces techniques s'intègrent parfaitement dans les stratégies de commande. En effet, elles réalisent des fonctionnalités d'identification, de contrôle ou de filtrage, et prolonge les techniques classiques de l'automatique non linéaire pour aboutir à des solutions plus efficaces et robustes.

Dans ce chapitre, nous présentons dans un premier temps un bref historique sur RNA et des généralités sur leurs concepts de base. Nous abordons ensuite le processus d'apprentissage. L'apprentissage d'un RNA se réalise par le choix du type d'apprentissage et de l'algorithme de mise à jour. Comme type, seul l'apprentissage supervisé sera détaillé vu son intérêt dans la commande et l'identification des systèmes dynamiques. Nous présentons trois algorithmes d'apprentissage : l'apprentissage par correction d'erreur, l'apprentissage à rétropropagation du gradient d'erreur et l'apprentissage par renforcement. Seuls les deux premiers algorithmes seront

II - Historique

Les recherches sur les méthodes neuronales de traitement de l'information en vue de modéliser le comportement du cerveau humain ne sont pas récentes, en :

- 1890 : W. James, célèbre psychologue américain introduit le concept de mémoire associative, et propose ce qui deviendra une loi de fonctionnement pour l'apprentissage sur les réseaux de neurones connue plus tard sous le nom de loi de Hebb.
- 1943 : J. Mc Culloch et W. Pitts, laissent leurs noms à une modélisation du neurone biologique (un neurone au comportement binaire). Ceux sont les premiers à montrer que des réseaux de neurones formels simples peuvent réaliser des fonctions logiques, arithmétiques et symboliques complexes (tout au moins au niveau théorique).
- 1949 : D. Hebb, physiologiste américain explique le conditionnement chez l'animal par les propriétés des neurones eux-mêmes. Ainsi, un conditionnement de type pavlovien tel que, nourrir tous les jours à la même heure un chien, entraine chez cet animal la sécrétion de salive à cette heure précise même en l'absence de nourriture. La loi de modification des propriétés des connexions entre neurones qu'il propose explique en partie ce type de résultats expérimentaux.
- 1957 : F. Rosenblatt développe le modèle du Perceptron. Il construit le premier neuro-ordinateur basé sur ce modèle et l'applique au domaine de la reconnaissance de formes. Notons qu'à cet époque les moyens à sa disposition sont limités et c'est une prouesse technologique que de réussir à faire fonctionner correctement cette machine plus de quelques minutes.
- 1960 : B. Widrow, un automaticien, développe le modèle ADALINE (ADAptive LInear NEuron). Dans sa structure, le modèle ressemble au Perceptron, cependant la loi d'apprentissage est différente. Celle-ci est à l'origine de l'algorithme de retro propagation de gradient très utilisé aujourd'hui avec les Perceptrons multicouches. Les réseaux de type ADALINE restent utilisés de nos jours pour certaines applications particulières.
- 1969 : M.L. Minsky et S. Papert publient ensuite un ouvrage qui met en évidence les limitations théoriques du Perceptron. Ces limitations concernent l'impossibilité de traiter des problèmes non linéaires en utilisant ce modèle.
- 1972 : T. Kohonen présente ses travaux sur les mémoires associatives et propose des applications à la reconnaissance de formes.
- 1982 : J.J. Hopfield est un physicien reconnu à qui l'on doit le renouveau d'intérêt pour les réseaux de neurones artificiels. Il présente une théorie du fonctionnement et des possibilités des réseaux de neurones.
- 1983 : La machine de Boltzmann est le premier modèle connu apte à traiter de manière satisfaisante les limitations recensées dans le cas du Perceptron. Mais l'utilisation pratique

s'avère difficile, la convergence de l'algorithme étant extrêmement longue (les temps de calcul sont considérables).

- 1985 : La rétropropagation de gradient apparaît. C'est un algorithme d'apprentissage adapté aux réseaux de neurones multicouches (aussi appelés Perceptrons multicouches). Sa découverte réalisée par trois groupes de chercheurs indépendants indique que « la chose était dans l'air ». Dés cette découverte, nous avons la possibilité de réaliser une fonction non linéaire d'entrée/sortie sur un réseau en décomposant cette fonction en une suite d'étapes linéairement séparables. De nos jours, les réseaux multicouches et la rétropropagation de gradient reste le modèle le plus productif au niveau des applications.

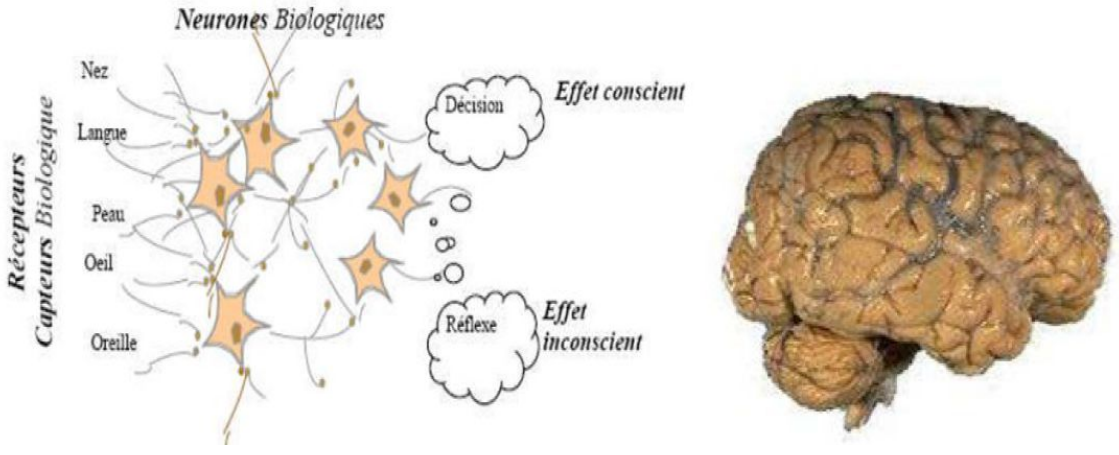
Depuis ce temps, le domaine des réseaux de neurones fourni constamment de nouvelles théories, de nouvelles structures et de nouveaux algorithmes. Dans ce chapitre, nous allons tenter d'exposer les plus importants.

III - Domaines d'application des réseaux de neurones artificiels

Aujourd'hui, les réseaux de neurones artificiels ont de nombreuses applications dans des secteurs très variés :

- Traitement d'images : reconnaissance de caractères et de signatures, compression d'images, reconnaissance de forme, cryptage, classification, etc.
- Traitement du signal : filtrage, classification, identification de source, traitement de la parole…etc.
- Contrôle : commande de processus, diagnostic, contrôle qualité, asservissement des robots, systèmes de guidage automatique des automobiles et des avions…etc.
- Défense : guidage des missiles, suivi de cible, reconnaissance du visage, radar, sonar, lidar, compression de données, suppression du bruit…etc.
- Optimisation : planification, allocation de ressource, gestion et finances, etc.
- Simulation : simulation du vol, simulation de boîte noire, prévision météorologique, recopie de modèle…etc.

IV - Définition des réseaux de neurones



(a) (b)

Fig.1 - Structure d'un réseau de neurone biologique et le cerveau humain.

L'origine des réseaux de neurones vient de l'essai de modélisation mathématique du cerveau humain les premiers travaux datent de 1943 et sont l'œuvre de W.M. Culloch et W. Pitts. Ils supposent que l'impulsion nerveuse est le résultat d'un calcul simple effectué par chaque neurone et que la pensée née grâce à l'effet collectif d'un réseau de neurone interconnecté (figure 1).

Un réseau de neurones est un assemblage de constituants élémentaires interconnectés (appelés «neurones» en hommage à leur modèle biologique), qui réalisent chacun un traitement simple mais dont l'ensemble en interaction fait émerger des propriétés globales complexes. Chaque neurone fonctionne indépendamment des autres de telle sorte que l'ensemble forme un système massivement parallèle. L'information est stockée de manière distribuée dans le réseau sous forme de coefficients synaptiques ou de fonctions d'activation, il n'y a donc pas de zone de mémoire et de zone de calcul, l'une et l'autre sont intimement liés.

Un réseau de neurone ne se programme pas, il est entraîné grâce à un mécanisme d'apprentissage. Les tâches particulièrement adaptées au traitement par réseau de neurones sont: l'association, la classification, la discrimination, la prévision ou l'estimation, et la commande de processus complexes.

Les réseaux de neurones artificiels consistent en des modèles plus ou moins inspirés du fonctionnement cérébral de l'être humain en se basant principalement sur le concept de neurone.

IV.1 - Introduction aux réseaux de neurones biologiques

Le cerveau humain contient environ 100 milliards de neurones. Ces neurones vous permettent, entre autre, de lire ce texte tout en maintenant une respiration régulière permettant d'oxygéner votre sang, en actionnant votre cœur qui assure une circulation efficace de ce sang pour nourrir vos cellules, etc.

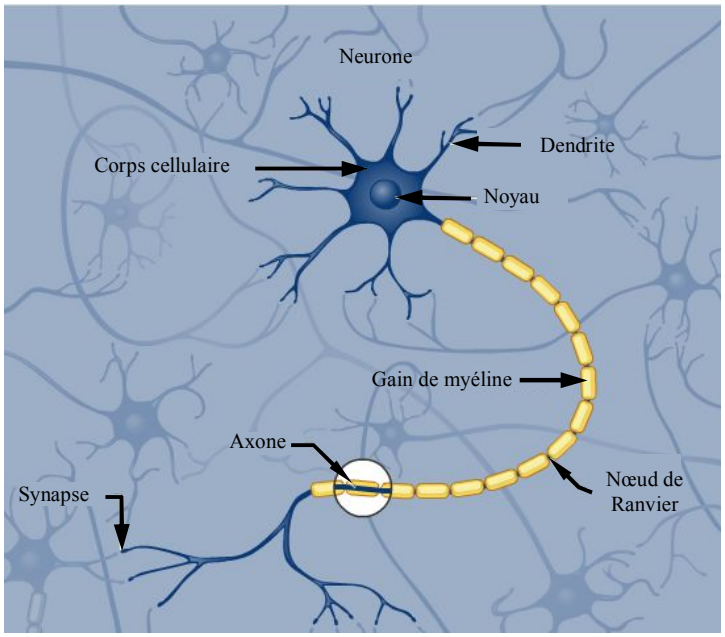


Fig.2 - Modèle d'un neurone biologique.

Chacun de ces neurones est par ailleurs fort complexe. Essentiellement, il s'agit de tissu vivant et de chimie. Les spécialistes des neurones biologiques (neurophysiologistes) commencent à peine à comprendre quelques uns de leurs mécanismes internes. On croit en général que leurs

différentes fonctions neuronales, y compris celle de la mémoire, sont stockées au niveau des connexions (synapses) entre les neurones. C'est ce genre de théorie qui a inspiré la plupart des architectures de réseaux de neurones artificiels que nous aborderons dans ce chapitre. L'apprentissage consiste alors soit à établir de nouvelles connexions, soit à en modifier des existantes.

Un neurone est une cellule particulière comme la montre la figure 2. Elle possède des extensions par lesquelles elle peut distribuer des signaux (axones) ou en recevoir (dendrites).

Dans le cerveau, les neurones sont reliés entre eux par l'intermédiaire des axones et des dendrites. En première approche. On peut considérer que ces sortes de filaments sont conductrices d'électricité et peuvent ainsi véhiculer des messages depuis un neurone vers un autre. Les dendrites représentent les entrées du neurone et son axone sa sortie.

Un neurone émet un signal en fonction des signaux qui lui proviennent des autres neurones. On observe en fait au niveau d'un neurone, une intégration des signaux reçus au cours du temps, c'est à dire une sorte de sommations des signaux. En général, quand la somme dépasse un certain seuil, le neurone émet à son tour un signal électrique.

La notion de synapse explique la transmission des signaux entre un axone et une dendrite. Au niveau de la jonction (c'est à dire de la synapse), il existe un espace vide à travers lequel le signal électrique ne peut pas se propager. La transmission se fait alors par l'intermédiaire de substances chimiques, les neuromédiateurs. Quand un signal arrive au niveau de la synapse, il provoque l'émission de neuromédiateurs qui vont se fixer sur des récepteurs de l'autre côté de l'espace inter-synaptique (figure 3).

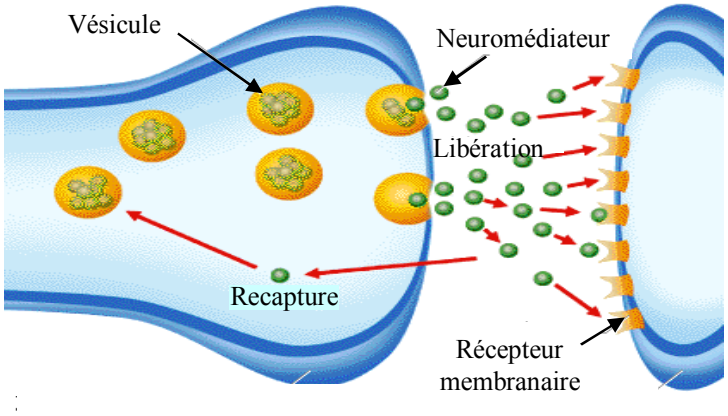


Fig.3 - Transmission de l'information entre les neurones biologiques.

Quand suffisamment de molécules se sont fixées, un signal électrique est émis de l'autre côté et on a donc une transmission. En fait, suivant le type de la synapse, l'activité d'un neurone peut renforcer ou diminuer l'activité de ces voisins. On parle ainsi de synapse excitatrice ou inhibitrice.

IV.2 - Les neurones formels

Un "neurone formel" (ou simplement "neurone") est une fonction algébrique non linéaire et bornée, dont la valeur dépend des paramètres appelés coefficients ou poids. Les variables de cette fonction sont habituellement appelées "entrées" du neurone, et la valeur de la fonction est appelée sa "sortie".

Un neurone est donc avant tout un opérateur mathématique, dont on peut calculer la valeur numérique par quelques lignes de logiciel. On a pris l'habitude de représenter graphiquement un neurone comme indiqué sur la figure 4.

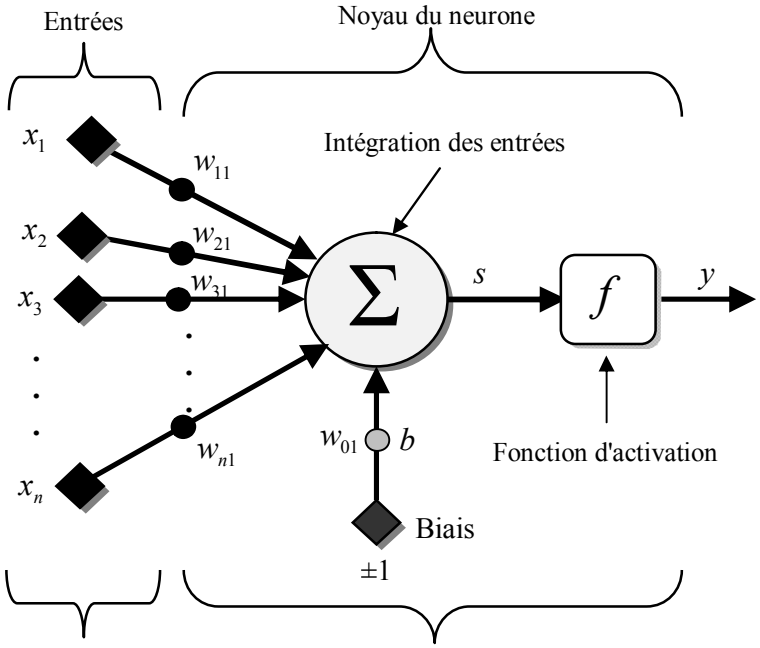


Fig.4 - Modèle d'un neurone artificiel.

Des observations de neurone biologique, découle le modèle du neurone formel proposé par W. M. Culloch et W. Pitts en 1943 :

- Les x_i représentent les vecteurs d'entrées, elles proviennent soit des sorties d'autres neurones, soit de stimuli sensoriels (capteur visuel, sonore…) ;
- Les w_{ij} sont les poids synaptiques du neurone j. Ils correspondent à l'efficacité synaptique dans les neurones biologiques ($w_{ij} > 0$: synapse excitatrice; $w_{ij} < 0$: synapse inhibitrice). Ces poids pondèrent les entrées et peuvent être modifiés par apprentissage ;
- Biais : entrée prend souvent les valeurs -1 ou +1 qui permet d'ajouter de la flexibilité au réseau en permettant de varier le seuil de déclenchement du neurone par l'ajustement des poids et du biais lors de l'apprentissage;
- Noyau : intègre toutes les entrées et le biais et calcul la sortie du neurone selon une fonction d'activation qui est souvent non linéaire pour donner une plus grande flexibilité d'apprentissage.

IV.2.1 - Modélisation d'un neurone formel

La modélisation consiste à mettre en œuvre un système de réseau de neurones sous un aspect non pas biologique mais artificiel, cela suppose que d'après le principe biologique on aura une correspondance pour chaque élément composant le neurone biologique, donc une modélisation pour chacun d'entre eux. On pourra résumer cette modélisation par le tableau 1, qui nous permettra de voir clairement la transition entre le neurone biologique et le neurone formel.

Neurone biologique	Neurone formel
Synapses	Poids des connexions
Axones	Signal de sortie
Dendrites	Signal d'entrée
Noyau ou Somma	Fonction d'activation

Tab.1 - Analogie entre le neurone biologique et le neurone formel.

Le modèle mathématique d'un neurone artificiel est illustré à la figure 4. Un neurone est essentiellement constitué d'un intégrateur qui effectue la somme pondérée de ses entrées. Le résultat s de cette somme est ensuite transformé par une fonction de transfert f qui produit la sortie y du neurone. En suivant les notations présentées à la section précédente, les n entrées du neurone correspondent au vecteur $x = [x_1, x_2, x_3 ... x_n]^T$, alors que $w = [w_{11}, w_{21}, w_{31}, ... w_{n1}]^T$ représente le vecteur des poids du neurone. La sortie s de l'intégrateur est donnée par l'équation suivante :

$$\begin{aligned} s &= \sum_{i=1}^{n} w_{i1} x_i \pm b \\ &= w_{11} x_1 + w_{21} x_2 + w_{31} x_3 + ... + w_{n1} x_n \pm b \end{aligned} \tag{1}$$

Que l'on peut aussi écrire sous forme matricielle :

$$s = w^T x \pm b \tag{2}$$

Cette sortie correspond à une somme pondérée des poids et des entrées plus ce qu'on nomme le biais b du neurone. Le résultat s de la somme pondérée s'appelle le niveau d'activation du neurone. Le biais b s'appelle aussi le seuil d'activation du neurone. Lorsque le niveau d'activation atteint ou dépasse le seuil b, alors l'argument de f devient positif (ou nul). Sinon, il est négatif.

On peut faire un parallèle entre ce modèle mathématique et certaines informations que l'on connaît (ou que l'on croit connaître) à propos du neurone biologique. Ce dernier possède trois principales composantes : les dendrites, le corps cellulaire et l'axone (voir figure 2). Les dendrites forment un maillage de récepteurs nerveux qui permettent d'acheminer vers le corps du neurone des signaux électriques en provenance d'autres neurones. Celui-ci agit comme une espèce d'intégrateur en accumulant des charges électriques. Lorsque le neurone devient suffisamment excité (lorsque la charge accumulée dépasse un certain seuil), par un processus électrochimique, il engendre un potentiel électrique qui se propage à travers son axone pour éventuellement venir exciter d'autres neurones. Le point de contact entre l'axone d'un neurone et la dendrite d'un autre neurone s'appelle la synapse. Il semble que c'est l'arrangement spatial des neurones et de leur axone, ainsi que la qualité des connexions synaptiques individuelles qui détermine la fonction précise d'un réseau de neurones biologique. C'est en se basant sur ces connaissances que le modèle mathématique décrit ci-dessus a été défini.

Un poids d'un neurone artificiel représente donc l'efficacité d'une connexion synaptique. Un poids négatif vient inhiber une entrée, alors qu'un poids positif vient l'accentuer. Il importe de

retenir que ceci est une grossière approximation d'une véritable synapse qui résulte en fait d'un processus chimique très complexe et dépendant de nombreux facteurs extérieurs encore mal connus. Il faut bien comprendre que notre neurone artificiel est un modèle pragmatique qui, comme nous le verrons plus loin, nous permettra d'accomplir des tâches intéressantes. La vraisemblance biologique de ce modèle ne nous importe peu. Ce qui compte est le résultat que ce modèle nous permettrons d'atteindre.

Un autre facteur limitatif dans le modèle que nous nous sommes donnés concerne son caractère discret. En effet, pour pouvoir simuler un réseau de neurones, nous allons rendre le temps discret dans nos équations. Autrement dit, nous allons supposer que tous les neurones sont synchrones, c'est à dire qu'à chaque temps t, ils vont simultanément calculer leur somme pondérée et produire une sortie:

$$y(t) = f\big(s(t)\big) \tag{3}$$

Dans les réseaux neurones biologiques, tous les neurones sont en fait asynchrones.

Revenons donc à notre modèle artificiel tel que formulé par l'équation (2) et ajoutons la fonction d'activation f pour obtenir la sortie du neurone :

$$y = f(s) = f(w^T x \pm b) \tag{4}$$

En remplaçant w^T par une matrice $W = w^T$ d'une seule ligne, on obtient une forme générale que nous adopterons tout au long de ce chapitre :

$$y = f(Wx \pm b) \tag{5}$$

L'équation (5) nous amène à introduire un schéma de notre modèle plus compact que celui de la figure 4. La figure 5 illustre celui-ci. On y représente les n entrées comme un rectangle noir. De ce rectangle sort le vecteur x dont la dimension matricielle est $n \times 1$. Ce vecteur est multiplié par une matrice W qui contient les poids (synaptiques) du neurone. Dans le cas d'un neurone simple, cette matrice possède la dimension $1 \times n$. Le résultat de la multiplication correspond au niveau d'activation qui est ensuite comparé au seuil b (un scalaire) par soustraction. Finalement, la sortie du neurone est calculée par la fonction d'activation f. La sortie d'un neurone est toujours un scalaire.

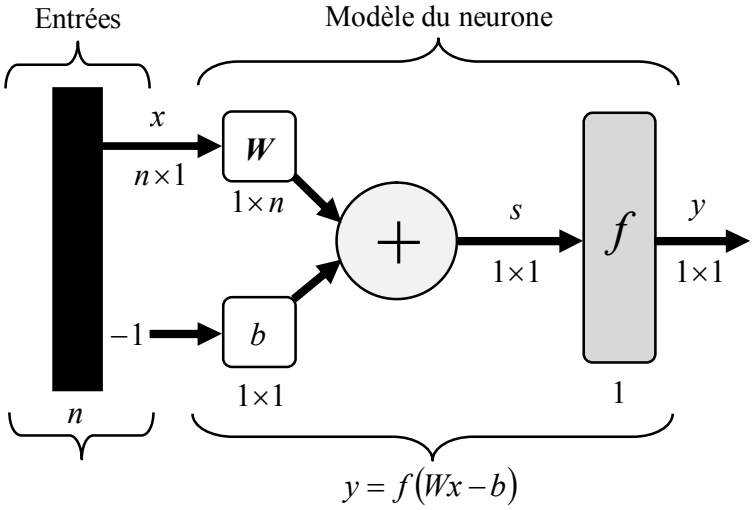


Fig.5 - Représentation matricielle du modèle d'un neurone artificiel.

IV.2.2 - Fonctions d'activations

Jusqu'à présent, nous n'avons pas spécifié la nature de la fonction d'activation de notre modèle. Il se trouve que plusieurs possibilités existent. Différentes fonctions de transfert pouvant être utilisées comme fonction d'activation du neurone sont énumérées au tableau 2. Les fonctions d'activations les plus utilisées sont les fonctions «seuil» (en anglais «hard limit»), «linéaire» et «sigmoïde». Comme son nom l'indique, la fonction seuil applique un seuil sur son entrée. Plus précisément, une entrée négative ne passe pas le seuil, la fonction retourne alors la valeur 0 (on peut interpréter ce 0 comme signifiant *faux*), alors qu'une entrée positive ou nulle dépasse le seuil, et la fonction retourne à 1 (*vrai*). Utilisée dans le contexte d'un neurone, cette fonction est illustrée à la figure 6.a. On remarque alors que le biais b dans l'expression de $y = hard\,lim(w^T x - b)$ (équation 4) détermine l'emplacement du seuil sur l'axe $w^T x$, où la fonction passe de 0 à 1. Nous verrons plus loin que cette fonction permet de prendre des décisions binaires.

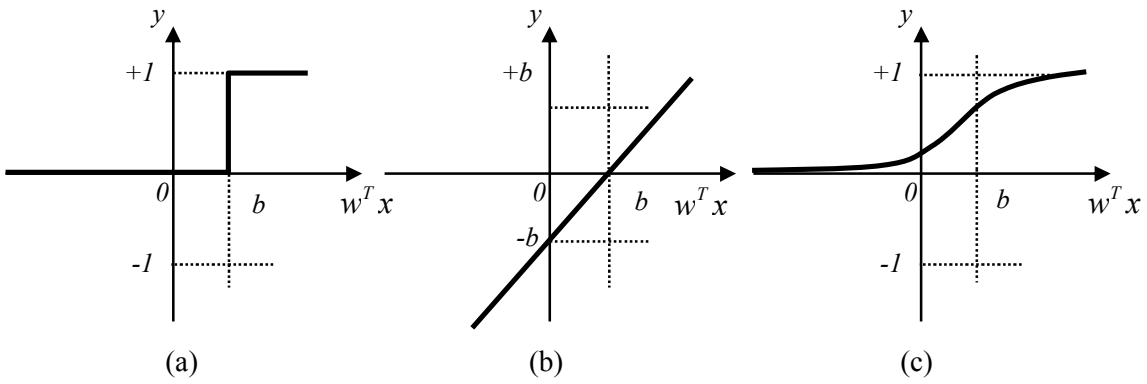


Fig.6 - Fonctions d'activations: (a) du neurone «seuil» ; (b) du neurone «linéaire», et (c) du neurone «sigmoïde».

La fonction linéaire est très simple, elle affecte directement son entrée à sa sortie :

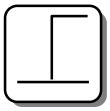
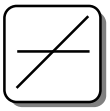
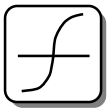
$$y = s \tag{6}$$

Appliquée dans le contexte d'un neurone, cette fonction est illustrée à la figure 6.b. Dans ce cas, la sortie du neurone correspond à son niveau d'activation dont le passage à zéro se produit lorsque $w^T x = b$.

La fonction de transfert sigmoïde est quant à elle illustrée à la figure 6.c. Son équation est donnée par :

$$y = \frac{1}{1 + exp^{-s}} \tag{7}$$

Elle ressemble soit à la fonction seuil, soit à la fonction linéaire, selon que l'on est loin ou près de b, respectivement. La fonction seuil est très non linéaire car il y a une discontinuité lorsque $w^T x = b$. De son côté, la fonction linéaire est tout à fait linéaire. Elle ne comporte aucun changement de pente. La sigmoïde est un compromis intéressant entre les deux précédentes. Notons finalement, que la fonction «tangente hyperbolique (*tanh*)» est une version symétrique de la sigmoïde.

Nom de la fonction	Relation entrée/sortie	Icône	Nom MATALB
Seuil	$y = 0 \quad si \quad s < 0$ $y = 1 \quad si \quad s \geq 0$		hardlim
Seuil symétrique	$y = -1 \quad si \quad s < 0$ $y = 1 \quad si \quad s \geq 0$		hardlims
Linéaire	$y = s$		purelin
Linéaire saturée	$y = 0 \quad si \quad s \leq 0$ $y = s \quad si \quad 0 \leq s \leq 1$ $y = 1 \quad si \quad s \geq 1$		satlin
Linéaire saturée symétrique	$y = -1 \quad si \quad s < -1$ $y = s \quad si \quad -1 \leq s \leq 1$ $y = 1 \quad si \quad s > 1$		satlins
Linéaire positive	$y = 0 \quad si \quad s \leq 0$ $y = s \quad si \quad s \geq 0$		poslin
Sigmoïde	$y = \frac{1}{1 + exp^{-s}}$		logsig
Tangente hyperbolique	$y = \frac{e^{s} - e^{-s}}{e^{s} + e^{-s}}$		tansig
Compétitive	$y = 1 \quad si \quad s \quad maximum$ $y = 0 \quad autrement$		compet

Tab.2 - Différentes fonctions d'activations utilisées dans les RNA.

V - Architecture des réseaux de neurones

L'architecture d'un réseau de neurones est l'organisation des neurones entre eux au sein d'un même réseau. Autrement dit, il s'agit de la façon dont ils ordonnés et connectés. La majorité des réseaux de neurones utilise le même type de neurones. Quelques architectures plus rares se basent sur des neurones dédiés. L'architecture d'un réseau de neurones dépend de la tâche à apprendre. Un réseau de neurone est en général composé de plusieurs couches de neurones, des entrées jusqu'aux sorties. On distingue deux grands types d'architectures de réseaux de neurones : les réseaux de neurones *non bouclés* et les réseaux de neurones *bouclés*.

V.1 - Les réseaux de neurones non bouclés

Un réseau de neurones non bouclé réalise une (ou plusieurs) fonctions algébriques de ses entrées, par composition des fonctions réalisées par chacun de ses neurones. Un réseau de neurones non bouclé est représenté graphiquement par un ensemble de neurones "connectés" entre eux, l'information circulant des entrées vers les sorties sans "retour en arrière"; si l'on représente le réseau comme un graphe dont les nœuds sont les neurones et les arêtes les "connexions" entre ceux-ci, le graphe d'un réseau non bouclé est acyclique. Le terme de "connexions" est une métaphore : dans la très grande majorité des applications, les réseaux de neurones sont des formules algébriques dont les valeurs numériques sont calculées par des programmes d'ordinateurs, non des objets physiques (circuits électroniques spécialisés) ; néanmoins, le terme de connexion, issu des origines biologiques des réseaux de neurones, est passé dans l'usage, car il est commode quoique trompeur. Il a même donné naissance au terme de connexionnisme.

V.1.1 - Réseaux de neurones monocouches

La structure d'un réseau monocouche est telle que des neurones organisés en entrée soient entièrement connectés à d'autres neurones organisés en sortie par une couche modifiable de poids (figure 7).

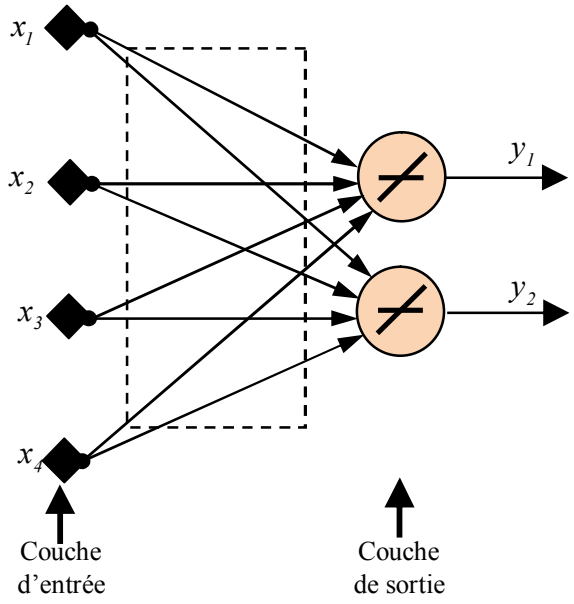


Fig.7 - Schéma d'un réseau de neurones monocouche.

V.1.2 - Réseaux de neurones multicouches

Les neurones sont arrangés par couche. Il n'y a pas de connexion entre neurones d'une même couche, et les connexions ne se font qu'avec les neurones de couches avales. Habituellement, chaque neurone d'une couche est connecté à tous les neurones de la couche suivante et celle-ci seulement. Ceci nous permet d'introduire la notion de sens de parcours de l'information (de l'activation) au sein d'un réseau et donc définir les concepts de neurone d'entrée, neurone de sortie. Par extension, on appelle couche d'entrée l'ensemble des neurones d'entrée, couche de sortie l'ensemble des neurones de sortie. Les couches intermédiaires n'ayant aucun contact avec l'extérieur sont appelées couches cachées.

La figure 8 représente un réseau de neurones non bouclé qui a une structure particulière, très fréquemment utilisée : il comprend des entrées, deux couches de neurones cachés et des neurones de sortie. Les neurones de la couche cachée ne sont pas connectés entre eux. Cette structure est appelée *Perceptron multicouches*.

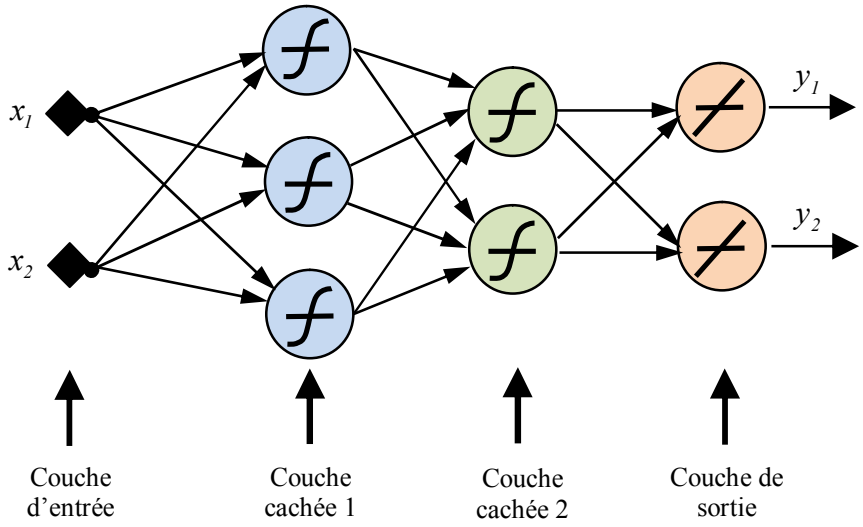


Fig.8 - Schéma d'un réseau de neurones non bouclé (Perceptron multicouches).

On note aussi que Les réseaux multicouches sont beaucoup plus puissants que les réseaux simples à une seule couche. En utilisant deux couches (une couche cachée et une couche de sortie), à condition d'employer une fonction d'activation sigmoïde sur la couche cachée, on peut entraîner un réseau à produire une approximation de la plupart des fonctions, avec une précision arbitraire (cela peut cependant requérir un grand nombre de neurones sur la couche cachée). Sauf dans des rares cas, les réseaux de neurones artificiels exploitent deux ou trois couches.

V.1.3 - Réseaux de neurones à connexions locales

Il s'agit d'une structure multicouche, mais qui à l'image de la rétine conserve une certaine topologie. Chaque neurone entretient des relations avec un nombre réduit et localisé de neurones de la couche avale. Les connexions sont donc moins nombreuses que dans le cas d'un réseau multicouche classique (figure 9).

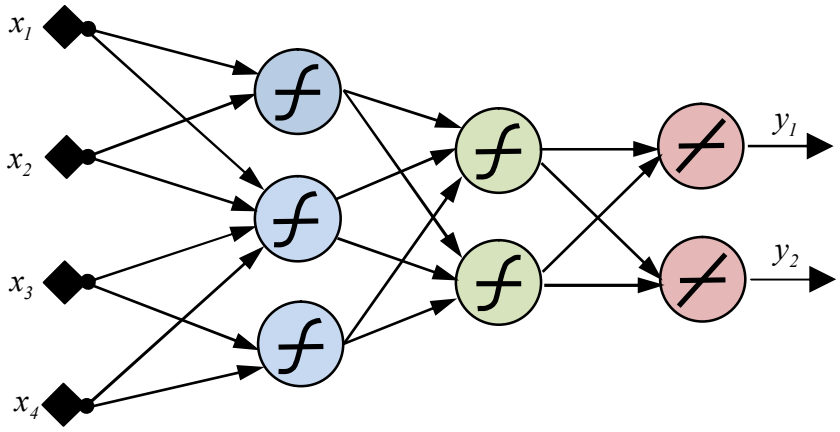


Fig.9 - Schéma d'un réseau de neurones à connexions locales.

Les réseaux de neurones non bouclés sont des objets *statiques* : si les entrées sont indépendantes du temps, les sorties le sont également. Ils sont utilisés principalement pour effectuer des tâches d'approximation de fonction non linéaire, de classification ou de modélisation de processus statiques non linéaire.

V.2 - Les réseaux de neurones bouclés

Contrairement aux réseaux de neurones non bouclés dont le graphe de connexions est acyclique, les réseaux de neurones bouclés peuvent avoir une topologie de connexions quelconque, comprenant notamment des boucles qui ramènent aux entrées la valeur d'une ou plusieurs sorties. Pour qu'un tel système soit causal, il faut évidemment qu'à toute boucle soit associé un *retard* : un réseau de neurones bouclé est donc un système *dynamique,* régi par des équations différentielles ; comme l'immense majorité des applications sont réalisées par des programmes d'ordinateurs, on se place dans le cadre des systèmes à temps discret, où les équations différentielles sont remplacées par des équations aux différences.

Il s'agit donc de réseaux de neurones avec retour en arrière (feedback network or recurrent network), (Figure 10).

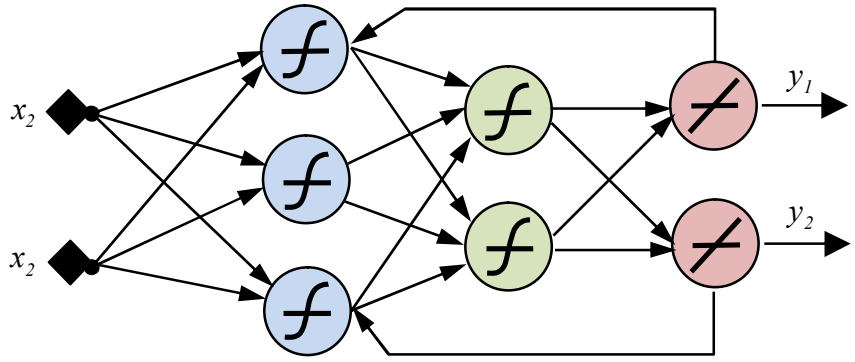


Fig.10 - Schéma de réseau de neurones bouclé.

Un réseau de neurones bouclé à temps discret est donc régi par une (ou plusieurs) équations aux différences non linéaires, résultant de la composition des fonctions réalisées par chacun des neurones et des retards associés à chacune des connexions.

La forme la plus générale des équations régissant un réseau de neurones bouclé est appelée *forme canonique ;*

$$x(k+1) = \varphi\left[x(k), u(k)\right] \tag{8}$$

$$y(k) = \psi\left[x(k), u(k)\right] \tag{9}$$

Où φ et ψ sont des fonctions non linéaires réalisées par un réseau de neurones non bouclé (mais pas obligatoirement, un perceptron multicouche), et où k désigne le temps (discret). La forme canonique est représentée sur la figure 11. Tout réseau de neurones, aussi compliqué soit-il, peut être mis sous cette forme canonique, de manière complètement automatique.

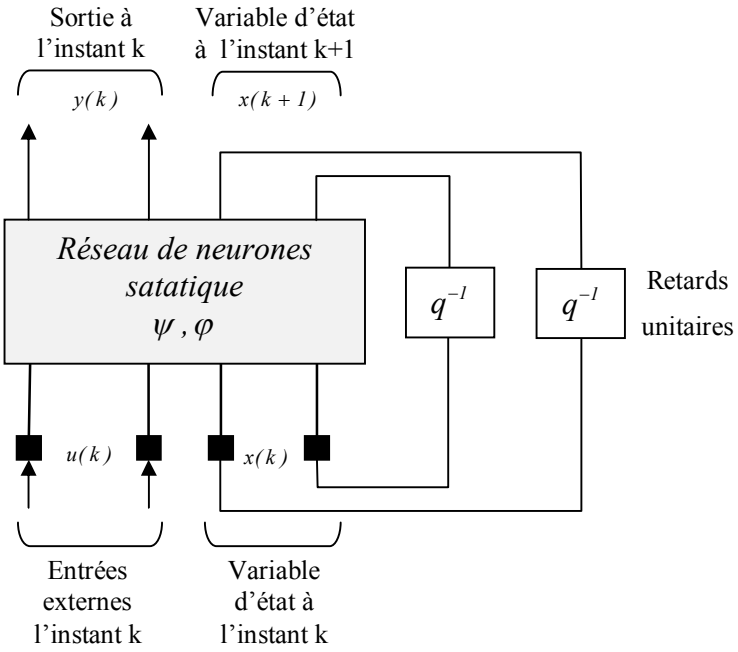


Fig.11 - Forme canonique d'un réseau de neurones bouclé.

Les réseaux de neurones bouclés sont utilisés pour effectuer des tâches de modélisation de systèmes dynamiques, de commande de processus, ou de filtrage.

VI - Modèles des réseaux de neurones

VI.1 - Modèle de Hopfield

Le modèle de Hopfield fut présenté en 1982. Ce modèle très simple est basé sur le principe des mémoires associatives. C'est d'ailleurs la raison pour laquelle ce type de réseau est dit associatif (par analogie avec le pointeur qui permet de récupérer le contenu d'une case mémoire).

Le modèle de Hopfield utilise l'architecture des réseaux entièrement connectés et récurrents (dont les connexions sont non orientées et ou chaque neurone n'agit pas sur lui-même). Les sorties sont en fonction des entrées et du dernier état pris par le réseau.

VI.2 - Modèle de Kohonen

Ce modèle a été présenté par T. Kohonen en 1982 en se basant sur des constatations biologiques. Il a pour objectif de présenter des données complexes et appartenant généralement à un espace discret de grandes dimensions dont la topologie est limitée à une ou deux dimensions. Les cartes de Kohonen sont réalisées à partir d'un réseau à deux couches, une en entrée et une en sortie. Notons que les neurones de la couche d'entrée sont entièrement connectés à la couche de sortie (figure 12).

Les neurones de la couche de sortie sont placés dans un espace d'une ou de deux dimensions en général, chaque neurone possède donc des voisins dans cet espace. Et qu'enfin, chaque neurone de la couche de sortie possède des connexions latérales récurrentes dans sa couche (le neurone inhibe, les neurones éloignés et laisse agir les neurones voisins).

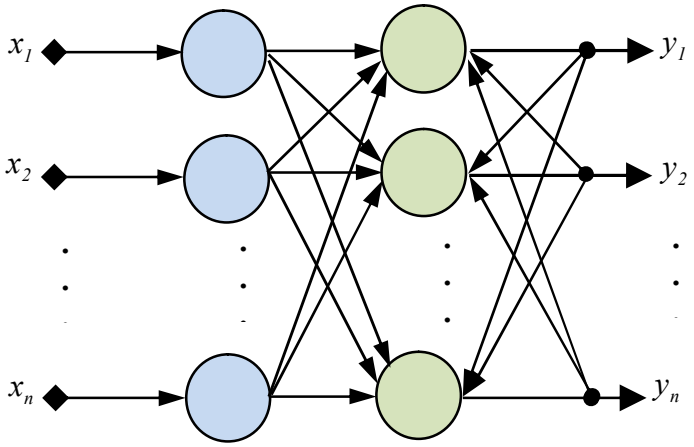


Fig. 12 - Le modèle de Kohonen.

VI.3 - Le modèle de Perceptron

Le mécanisme perceptron fut inventé par le psychologue F. Rosenblat à la fin des années 50. Il représentait sa tentative d'illustrer certaines propriétés fondamentales des systèmes intelligents en générale.

Le réseau dans ce modèle est formé de trois couches : Une couche d'entrée, fournissant des donnés à une couche intermédiaire, chargée des calculs, cela en fournissant la somme des impulsions qui lui viennent des cellules auxquelles elle est connectée, et elle répond généralement suivant une loi définie avec un seuil, elle-même connectée à la couche de sortie (couche de décision), représentant les exemples à mémoriser. Seule cette dernière couche renvoie des signaux à la couche intermédiaire, jusqu'à ce que leurs connexions se stabilisent (voir figure 8).

VI.4 - Le modèle ADALINE

L'ADALINE de Widrow et Hoff est un réseau à trois couches : une d'entrée, une couche cachée et une couche de sortie. Ce modèle est similaire au modèle de perceptron, seule la fonction de transfert change, mais reste toujours linéaire. Les modèles des neurones utilisés dans le perceptron et l'ADALINE sont des modèles linéaires. Séparation linéaire : on dit que deux classes A et B, sont linéairement séparables si on arrive à les séparer par une droite coupant le plan en deux (figure 13). Le problème est résolu avec les réseaux multicouches, car il peut résoudre toute sorte de problèmes qu'ils soient linéairement séparables ou non.

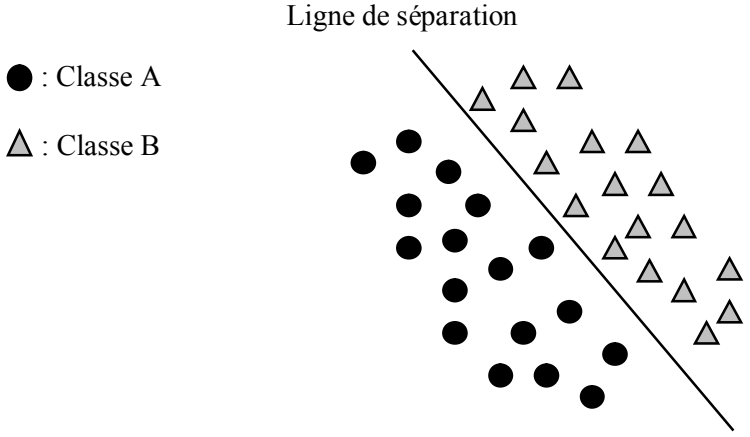


Fig.13 - La séparation linéaire entre la classe A et B.

VII - L'apprentissage

Parmi les propriétés désirables pour un réseau de neurones, la plus fondamentale est sûrement la capacité d'apprendre de son environnement, d'améliorer sa performance à travers un processus d'apprentissage. Mais qu'est-ce donc que l'apprentissage ?

Malheureusement, il n'existe pas de définition générale, universellement acceptée, car ce concept touche à trop de notions distinctes qui dépendent du point de vue que l'on adopte. Dans le contexte des réseaux de neurones artificiels, nous adopterons un point de vue pragmatique en proposant la définition suivante :

" L'apprentissage est un processus dynamique et itératif permettant de modifier les paramètres d'un réseau en réaction avec les stimuli qu'il reçoit de son environnement. Le type d'apprentissage est déterminé par la manière dont les changements de paramètre surviennent ".

Cette définition implique qu'un réseau se doit d'être stimulé par un environnement, qu'il subisse des changements en réaction avec cette stimulation, et que ceux-ci provoquent dans le futur une réponse nouvelle vis-à-vis de l'environnement. Ainsi, le réseau peut s'améliorer avec le temps

VII.1 - Stratégies d'apprentissage

L'apprentissage au sein des différentes architectures dépend de l'architecture du réseau et de l'environnement du problème. Les deux règles d'apprentissage pour mettre corriger les poids d'un neurone (règle de Hebb et de Widrow) ne concernent qu'un neurone seul. Ces règles peuvent servir pour mettre à jour les poids d'un neurone, de certains réseaux de neurones, mais ne peuvent être généralisées et s'appliquer à n'importe quelle architecture. Chaque architecture possède ses spécificités et nécessite une règle d'adaptation des poids qui lui est propre.

L'apprentissage n'est pas modélisable dans le cadre de la logique déductive : celle-ci en effet procède à partir de connaissances déjà établies dont on tire des connaissances dérivées. Or il s'agit ici de la démarche inverse : par observations limitées tirer des généralisations plausibles.

La notion d'apprentissage recouvre deux réalités (figure 14) :

- La mémorisation : le fait d'assimiler sous une forme dense des exemples éventuellement nombreux ;
- La généralisation : le fait d'être capable, grâce aux exemples appris, de traiter des exemples distincts, encore non rencontrés, mais similaires.

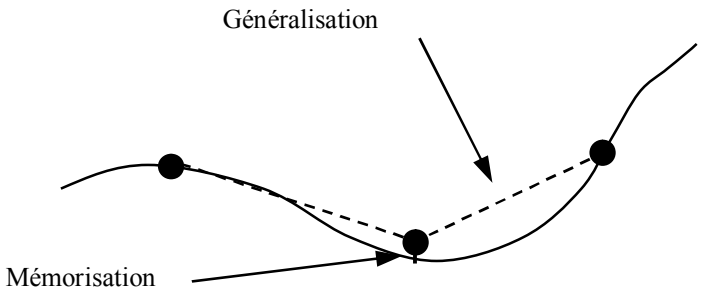


Fig.14 - Principe de l'apprentissage.

Le concept de la mémorisation et celui de la généralisation sont partiellement en opposition. Si on privilégie l'un, on élaborera un système qui ne traitera pas forcément de façon très efficace l'autre.

Dans le cas des systèmes d'apprentissage statistique, utilisés pour optimiser les modèles statistiques classiques, réseaux de neurones et automates markoviens, c'est la généralisation qui est préférée. Il faut trouver un compromis en choisissant un coefficient d'apprentissage satisfaisant (par essai-erreur).

VII.1.1 - Données et processus

Analyser un processus, c'est à dire comprendre son fonctionnement, caractériser ses états, savoir ce qui est significatif, quelle est sa structure, requiert l'observation d'un certain nombre (parfois grand) de variables issues de ce processus et liées à son fonctionnement et son environnement (Figure 15).

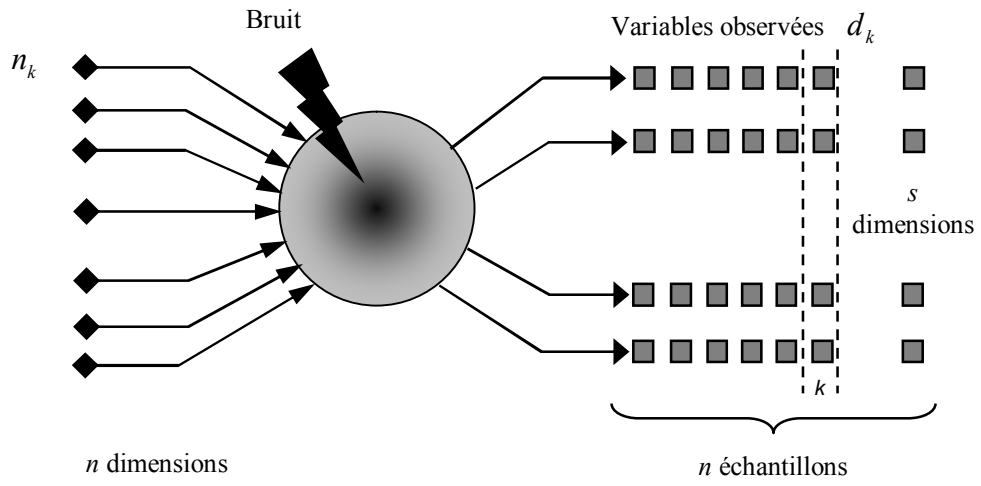


Fig.15 - Principe de fonctionnement et d'analyse des processus.

Si les données ne sont pas représentatives du processus, on ne pourra l'apprendre que partiellement. Le choix des données est crucial pour un bon apprentissage. Il faut bien choisir par exemple: Le type des données, leur nature, leur pertinence/importance, la quantité, l'ordre dans lequel elles sont prises en compte.

VII.1.2 - Apprentissage en ligne et l'apprentissage hors ligne "batch"

L'apprentissage est un moyen flexible et efficace d'extraire une structure stochastique d'un environnement. Deux types différents d'apprentissage sont utilisés, à savoir l'apprentissage hors ligne ou en paquet et l'apprentissage en ligne.

- La procédure d'apprentissage en paquet utilise tous les exemples d'entraînement de façon répétée de sorte que sa performance est comparable à celle d'une procédure d'estimation statistique.
- L'apprentissage en ligne est plus dynamique, en mettant à jour l'estimation courante par l'observation des nouvelles données une par une. C'est une procédure itérative. L'apprentissage en ligne est en général lent mais est recommandé dans des environnements changeants.

Il est possible de combiner les deux, dans l'ordre :

1. On récupère une série d'observation à partir du processus à modéliser (en quantité et pertinence suffisante).
2. On effectue un "préapprentissage" : les données récoltées servent à faire converger les poids vers des valeurs proches de la solution finale.
3. On utilise le réseau de neurones tout en utilisant un apprentissage en ligne pour affiner la valeur des poids et pourvoir tenir compte de toutes variations éventuelles (dérives, perturbations, changements de mode de fonctionnement, etc.)

On peut faire une comparaison entre les deux types d'apprentissages, les résultats obtenus sont classés dans le tableau 3.

Apprentissage hors ligne	Apprentissage en ligne
• requiert souvent de charger en mémoire l'ensemble des poids et des données d'apprentissage (entrées et éventuellement sorties désirées correspondantes) ; • ne peut satisfaire une contrainte de temps réel ; • autorise de réinitialiser sans risque l'apprentissage.	• prend en compte les observations itérativement, au fur et à mesure et demande de ce fait moins de mémoire, moins de calculs ; • est compatible au temps réel ; • "subit" l'ordre dans lequel les observations sont accessibles.

Tab.3 - Comparaison entre l'apprentissage hors ligne et l'apprentissage en ligne.

VII.1.3 - Apprentissage global et apprentissage local

- L'apprentissage global est le processus qui met à jour l'intégralité des poids d'un réseau. Dans ce cas, tous les poids sont modifiés à chaque itération.
- L'apprentissage local vise à ne mettre à jour que certains poids à chaque itération. Plusieurs techniques existent : à travers le concept de voisinage, à travers des règles spécifiques d'apprentissage, etc.

VII.2 - Algorithmes d'apprentissages

Dans la plupart des architectures que nous étudierons, l'apprentissage se traduit par une modification de l'efficacité synaptique, c'est à dire par un changement dans la valeur des poids qui relient les neurones d'une couche à l'autre. Soit le poids w_{ij} reliant le neurone i à son entrée j. Au temps t, un changement $\Delta w_{ij}(t)$ de poids peut s'exprimer simplement de la façon suivante :

$$\Delta w_{ij}(t) = w_{ij}(t+1) - w_{ij}(t) \tag{10}$$

Et, par conséquent, $w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$, avec $w_{ij}(t+1)$ et $w_{ij}(t)$ représentant respectivement les nouvelle et ancienne valeurs du poids w_{ij}.

Un ensemble de règles bien définies permettant de réaliser un tel processus d'adaptation des poids constitue ce qu'on appelle l'algorithme d'apprentissage du réseau.

Dans la suite de ce chapitre, nous allons passer en revue différents algorithmes d'apprentissages ainsi que différents principes pouvant guider l'apprentissage d'un réseau de neurone.

VII.2.1 - Apprentissage par correction d'erreur

La première règle que l'on peut utiliser est fondée sur la correction de l'erreur observée en sortie. Soit $y_i(t)$ la sortie que l'on obtient pour le neurone i au temps t. Cette sortie résulte d'un stimulus $x(t)$ que l'on applique aux entrées du réseau dont un des neurones correspond au neurone i. Soit $d_i(t)$ la sortie que l'on désire obtenir pour ce même neurone i au temps t. Alors, $y_i(t)$ et $d_i(t)$ seront généralement différents et il est naturel de calculer l'erreur $e_i(t)$ entre ce qu'on obtient et ce qu'on voudrait obtenir :

$$e_i(t) = d_i(t) - y_i(t) \tag{11}$$

Et de chercher un moyen de réduire autant que possible cette erreur. Sous forme vectorielle, on obtient :

$$e(t) = d(t) - y(t) \tag{12}$$

Avec $e(t) = \left[e_1(t) \quad e_2(t) \quad \ldots \quad e_i(t) \quad \ldots \quad e_S(t)\right]^T$ qui désigne le vecteur des erreurs observées sur les S neurones de sortie du réseau. L'apprentissage par correction des erreurs consiste à minimiser un indice de performance F basé sur les signaux d'erreur $e_i(t)$, dans le but de faire converger les sorties du réseau avec ce qu'on voudrait qu'elles soient. Un critère très populaire est la somme des erreurs quadratiques :

$$F\big(e(t)\big) = \sum_{i=1}^{S} e_i^2(t) = e(t)^T e(t) \tag{13}$$

Maintenant, il importe de remarquer que les paramètres libres d'un réseau sont ses poids. Prenons l'ensemble de ces poids et assemblons les sous la forme d'un vecteur $w(t)$ au temps t. Pour minimiser $F\big(e(t)\big) = F\big(w(t)\big) = F(t)$, nous allons commencer par choisir des poids initiaux $(t=0)$ au hasard, puis nous allons modifier ces poids de la manière suivante:

$$w(t+1) = w(t) + \eta\, X(t) \tag{14}$$

Où le vecteur $X(t)$ désigne la direction dans laquelle nous allons chercher le minimum et η est une constante positive déterminant l'amplitude du pas dans cette direction (la vitesse d'apprentissage). L'objectif est de faire en sorte que $F(t+1) < F(t)$. Mais comment peut-on choisir la direction X pour que la condition précédente soit respectée ? Considérons la série de Taylor de 1er ordre autour de $w(t)$:

$$F(t+1) = F(t) + \nabla F(t)^T \Delta w(t) \tag{15}$$

Où $\nabla F(t)$ désigne le gradient de F par rapport à ses paramètres libres (les poids w) au temps t, et $\Delta w(t) = w(t+1) - w(t)$. Or, pour que $F(t+1) < F(t)$ il faut que la condition suivante soit respectée :

$$\nabla F(t)^T \Delta w(t) = \eta \nabla F(t)^T X(t) < 0 \tag{16}$$

N'importe quel vecteur $X(t)$ qui respecte l'inégalité de l'équation (16) pointe donc dans une direction qui diminue F. On parle alors d'une direction de «descente». Pour obtenir une descente maximum, étant donnée $\eta > 0$, il faut que le vecteur $X(t)$ pointe dans le sens opposé au gradient car c'est dans ce cas que le produit scalaire sera minimum :

$$X(t) = -\nabla F(t) \tag{17}$$

Ce qui engendre la règle dite de «descente du gradient» :

$$\Delta w(t) = -\eta \nabla F(t) \tag{18}$$

Illustrée à la figure 16. Dans l'espace des poids, cette figure montre les courbes de niveau de F représentées par des ellipses hypothétiques. La flèche en pointillés montre la direction optimale pour atteindre le minimum de F. La flèche pleine montre la direction du gradient qui est perpendiculaire à la courbe de niveau en $w(t)$.

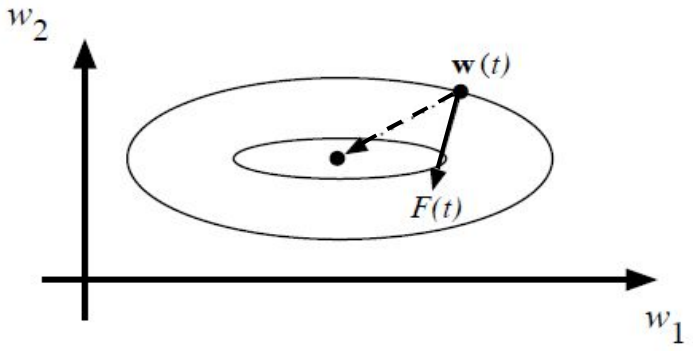


Fig.16 - Trajectoire de la descente du gradient.

La règle de la correction des erreurs est utilisée pour beaucoup de réseaux de neurones artificiels, bien qu'elle ne soit pas plausible biologiquement. En effet, comment le cerveau pourrait-il connaître a priori les sorties qu'il doit produire ? Cette règle ne peut être utilisée que dans un contexte d'apprentissage supervisé sur lequel nous reviendrons bientôt.

VII.2.2 - Apprentissage par la règle de Hebb

Dans cette section nous abordons une règle qui s'inspire des travaux du neurophysiologiste Donald Hebb :

«*When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B, is increased.*»

Dans un contexte neurobiologique, Hebb cherchait à établir une forme d'apprentissage associatif au niveau cellulaire. Dans le contexte des réseaux artificiels, on peut reformuler l'énoncé de Hebb sous la forme d'une règle d'apprentissage en deux parties :

1. Si deux neurones de part et d'autre d'une synapse (connexion) sont activés simultanément (d'une manière synchrone), alors la force de cette synapse doit être augmentée ;
2. Si les mêmes deux neurones sont activés d'une manière asynchrone, alors la synapse correspondante doit être affaibli ou carrément éliminé.

Une telle synapse est dite «synapse Hebbien». Elle utilise un mécanisme interactif, dépendant du temps et de l'espace, pour augmenter l'efficacité synaptique d'une manière proportionnelle à la

corrélation des activités pré- et post-synaptiques. De cette définition ressort les propriétés suivantes :

1. **Dépendance temporelle:** Les modifications d'une synapse Hebbien dépendent du moment exact des activités pré- et post-synaptique ;
2. **Dépendance spatiale:** Etant donné la nature même de la synapse qui constitue un lieu de transmission d'information, l'apprentissage Hebbien se doit de posséder une contiguïté spatiale. C'est cette propriété qui, entre autres, permet l'apprentissage dit non-supervisé sur lequel nous reviendrons bientôt ;
3. **Interaction:** L'apprentissage Hebbien dépend d'une interaction entre les activités de part et d'autre de la synapse.
4. **Conjonction ou corrélation:** Une interprétation de l'énoncé de Hebb est que la condition permettant un changement dans l'efficacité synaptique est une conjonction des activités pré et post-synaptiques. C'est la co-occurrence des activités de part et d'autre de la synapse qui engendre une modification de celui-ci. Une interprétation plus statistique réfère à la corrélation de ces activités. Deux activités positives simultanées (corrélation positive) engendrent une augmentation de l'efficacité synaptique, alors que l'absence d'une telle corrélation engendre une baisse de cette efficacité.

Mathématiquement, on peut exprimer la règle de Hebb sous sa forme la plus simple par la formule suivante :

$$\Delta w_j(t-1) = \eta\, x_j(t)\, y(t) \tag{19}$$

Où η est une constante positive qui détermine la vitesse de l'apprentissage, $x_j(t)$ correspond à l'activité pré- synaptique (l'entrée j du neurone) au temps t, et $y(t)$ à l'activité post synaptique (sortie du neurone) à ce même temps t. Cette formule fait ressortir explicitement la corrélation entre le signal qui entre et celui qui sort. Sous une forme vectorielle, on écrit :

$$\Delta w(t-1) = \eta\, x(t)\, y(t) \tag{20}$$

Un problème immédiat avec la règle de l'équation (20) est que les changements de poids $\Delta w_j(t)$ peuvent croître de façon exponentielle si, par exemple, l'entrée et la sortie demeurent constantes dans le temps. Pour pallier à cette croissance exponentielle qui provoquerait invariablement une saturation du poids, on ajoute parfois un facteur d'oubli qui retranche de la variation de poids, une fraction α du poids actuel. On obtient ainsi :

$$\Delta w_j(t-1) = \eta\, x_j(t)\, y(t) - \alpha\, w_j(t-1) \tag{21}$$

Où $0 \le \alpha \le 1$ est une nouvelle constante. Sous forme vectorielle, on écrit :

$$\Delta w\,(t-1) = \eta\, x\,(t)\, y(t) - \alpha\, w\,(t-1) \tag{22}$$

La règle de Hebb avec oubli, énoncée à l'équation (22), contourne efficacement le problème des poids qui croissent (ou décroissent) sans limite. Supposons que $x_j(t) = y(t) = 1$ et que nous ayons atteint le régime permanent où $\Delta w_j = 0$. Alors, la valeur maximale w_j^{max} que peut atteindre le poids $w_j(t)$ est donnée par :

$$w_j^{max} = (1-\alpha) w_j^{max} + \eta$$
$$= \frac{\eta}{\alpha} \quad (23)$$

Mais cette règle ne résout pas tous les problèmes. À cause du terme d'oubli, il est primordial que les stimuli soient répétés régulièrement, sinon les associations apprises grâce à la règle de l'équation (22) seront éventuellement perdues car complètement oubliées. Une autre variante de la règle de Hebb s'exprime donc de la manière suivante :

$$\Delta w_j(t-1) = \eta\, x_j(t) y(t) - \alpha\, y(t) w_j(t-1) \quad (24)$$

Et si l'on fixe $\alpha = \eta$ pour simplifier (on pose un rythme d'apprentissage égale à celui de l'oubli), on obtient la règle dite «instar» :

$$\Delta w_j(t-1) = \eta\, y(t)\left[x_j(t) - w_j(t-1)\right] \quad (25)$$

Que l'on peut réécrire sous sa forme vectorielle de la façon suivante :

$$\Delta w(t-1) = \eta\, y(t)\left[x(t) - w(t-1)\right] \quad (26)$$

Une façon d'aborder cette règle, est de regarder ce qui se passe lorsque $y(t) = 1$:

$$w(t) = w(t-1) + \eta\, \left[x(t) - w(t-1)\right]$$
$$= (1-\eta)\, w(t-1) + \eta\, x(t) \quad (27)$$

Alors, on constate qu'en présence d'une activité post-synaptique positive, le vecteur de poids est déplacé dans la direction du vecteur d'entrée $x(t)$, le long du segment qui relie l'ancien vecteur de poids avec le vecteur d'entrée, tel qu'illustré à la figure 17. Lorsque $\eta = 0$, le nouveau vecteur de poids est égal à l'ancien (aucun changement). Lorsque $\eta = 1$, le nouveau vecteur de poids est égal au vecteur d'entrée. Finalement, lorsque $\eta = \frac{1}{2}$, le nouveau vecteur est à mi-chemin entre l'ancien vecteur de poids et le vecteur d'entrée.

Une propriété particulièrement intéressante de la règle instar est qu'avec des entrées normalisées, suite au processus d'apprentissage, les poids w convergeront également vers des vecteurs normalisés. Mais nous y reviendrons lorsque nous traiterons du réseau «instar».

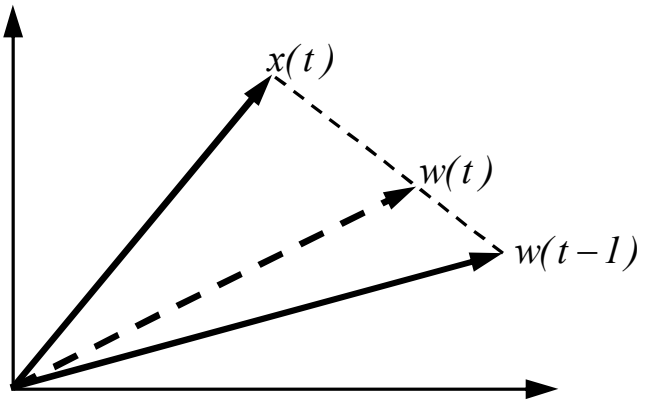


Fig.17 - Représentation graphique de la règle «instar» lors d'une activité post-synaptique positive.

VII.2.3 - Apprentissage compétitif « competitive learning »

L'apprentissage compétitif, comme son nom l'indique, consiste à faire compétitionner les neurones d'un réseau pour déterminer lequel sera actif à un instant donné. Contrairement aux autres types d'apprentissage où, généralement, tous les neurones peuvent apprendre simultanément et de la même manière, l'apprentissage compétitif produit un «vainqueur» ainsi que, parfois, un ensemble de neurones «voisins» du vainqueur, et seuls ce vainqueur et, potentiellement, son voisinage bénéficient d'une adaptation de leur poids. On dit alors que l'apprentissage est local car limité à un sous-ensemble des neurones du réseau.

Une règle d'apprentissage compétitif comporte les éléments suivants :

- Un ensemble de neurones identiques (même type) sauf pour les valeurs de leurs poids synaptiques ;
- Une limite imposée à la «force» d'un neurone ;
- Un mécanisme permettant aux neurones de compétitionner pour le droit de répondre à un certain sous-ensemble des stimuli d'entrée, de manière à ce qu'un seul neurone de sortie soit actif à la fois.

Ainsi, les neurones individuels peuvent apprendre à se spécialiser sur des sous-ensembles de stimuli similaires pour devenir des détecteurs de caractéristiques.

Dans leur forme la plus simple, les réseaux de neurones qui utilisent l'apprentissage compétitif sont souvent constitués d'une seule couche de neurones de sortie, totalement connectée sur les entrées. Un neurone vainqueur modifiera ses poids synaptiques en les rapprochant (géométriquement) d'un stimulus d'entrée x pour lequel il a battu tous les autres neurones lors de la compétition :

$$\Delta w = \begin{cases} \eta(x - w) \text{ si le neurone est vainqueur} \\ 0 \quad \text{autrement} \end{cases} \tag{28}$$

Où $0 < \eta < 1$ correspond à un taux d'apprentissage. Un neurone qui ne gagne pas la compétition ne modifiera aucunement ses poids. Il ne sera donc pas affecté par le stimulus en question. Parfois, on définit également un voisinage autour du neurone gagnant et on applique une règle similaire sur les voisins, mais avec un taux d'apprentissage différent :

$$\Delta w = \begin{cases} \eta_1(x - w) \text{ si le neurone est vainqueur} \\ \eta_2(x - w) \text{ si le neurone est voisin vainqueur} \\ 0 \quad \text{autrement} \end{cases} \tag{29}$$

Avec $\eta_2 \leq \eta_1$

Comme nous le verrons plus loin dans ce chapitre, l'apprentissage compétitif est surtout utilisé dans le contexte d'un apprentissage dit non-supervisé, c'est-à-dire lorsqu'on ne connaît pas les valeurs désirées pour les sorties du réseau.

VII.2.4 - Problème de l'affectation du crédit

Dans le domaine général de l'apprentissage, il existe un problème qui tourne autour de la notion de «affectation du crédit» (Credit assignment). Essentiellement, il s'agit d'affecter le crédit d'un résultat global, par exemple l'adéquation des sorties d'un réseau face à un certain stimulus d'entrée, à l'ensemble des décisions internes prises par le système (le réseau) et ayant conduit à ce résultat global. Dans le cas de l'exemple d'un réseau, les décisions internes correspondent aux

sorties des neurones situés sur les couches qui précèdent la couche de sortie. Ces couches sont habituellement qualifiées de «couches cachées» car on ne dispose pas, a priori, d'information sur l'adéquation de leurs sorties.

Le problème de l'affectation du crédit est donc bien présent dans l'apprentissage des réseaux de neurones. Par exemple, si l'on adopte une règle basée sur la correction des erreurs, comment fera-t-on pour calculer cette erreur sur les couches cachées, si l'on ne possède pas l'information, à propos de leurs sorties désirées? De même, que fera-t-on si l'on dispose uniquement d'une appréciation générale de performance du réseau face à chaque stimulus, et non des sorties désirées pour chaque neurone de la couche de sortie ? Nous apporterons certains éléments de réponse à ces questions dans les sous-sections suivantes.

VII.2.5 - Apprentissage supervisé « supervised learning »

L'apprentissage dit "supervisé" est caractérisé par la présence d'un «professeur» qui possède une connaissance approfondie de l'environnement dans lequel évolue le réseau de neurones. En pratique, les connaissances de ce professeur prennent la forme d'un ensemble de Q couples de vecteurs d'entrée et de sortie que nous noterons $\{(x_1,d_1),(x_2,d_2),...,(x_Q,d_Q)\}$, où x_i désigne un stimulus (entrée) et d_i la cible pour ce stimulus, c'est-à-dire les sorties désirées du réseau. Chaque couple (x_i,d_i) correspond donc à un cas d'espèce de ce que le réseau devrait produire (la cible) pour un stimulus donné. Pour cette raison, l'apprentissage supervisé est aussi qualifié d'apprentissage par des exemples.

L'apprentissage supervisé est illustré d'une manière conceptuelle à la figure 18. L'environnement est inconnu du réseau. Celui-ci produit un stimulus x qui est acheminé à la fois au professeur et au réseau. Grâce à ses connaissances intrinsèques, le professeur produit une sortie désirée $d(t)$ pour ce stimulus. On suppose que cette réponse est optimale.

Elle est ensuite comparée (par soustraction) avec la sortie du réseau pour produire un signal d'erreur $e(t)$ qui est réinjecté dans le réseau pour modifier son comportement via une procédure itérative qui, éventuellement, lui permet de simuler la réponse du professeur. Autrement dit, la connaissance de l'environnement par le professeur est graduellement transférée vers le réseau jusqu'à l'atteinte d'un certain critère d'arrêt. Par la suite, on peut éliminer le professeur et laisser le réseau fonctionner de façon autonome.

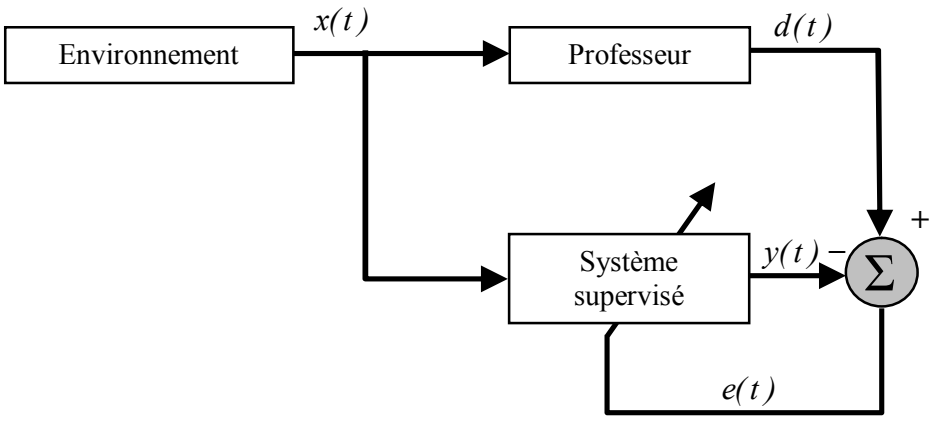


Fig.18 - Schéma bloc de l'apprentissage supervisé.

Le lecteur attentif aura remarqué qu'un apprentissage supervisé n'est rien d'autre qu'un synonyme de l'apprentissage par correction des erreurs (voir §VII.2.1). Il possède donc les mêmes limitations, à savoir que sans professeur pour fournir les valeurs cibles, il ne peut d'aucune façon apprendre de nouvelles stratégies pour de nouvelles situations qui ne sont pas couvertes par les exemples d'apprentissage.

VII.2.6 - Apprentissage par renforcement « reinforcement learning »

L'apprentissage par renforcement permet de contourner certaines des limitations de l'apprentissage supervisé. Il consiste en une espèce d'apprentissage supervisé, mais avec un indice de satisfaction scalaire au lieu d'un signal d'erreur vectoriel. Ce type d'apprentissage est inspiré des travaux en psychologie expérimentale de Thorndike (1911) :

« *Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur; those which are accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater, the satisfaction or discomfort, the greater the strengthening or weakening of the bond* ».

Même si cet énoncé ne peut expliquer à lui seul le comportement animal au niveau biologique, sa simplicité et son pragmatisme peuvent nous permettre de composer des règles d'apprentissage utiles.

Dans le contexte des réseaux de neurones artificiels, nous pouvons reformuler l'énoncé de Thorndike de la façon suivante :

« Lorsqu'une action (décision) prise par le réseau engendre un indice de satisfaction positif, alors la tendance du réseau à prendre cette action doit être renforcée. Autrement, la tendance à prendre cette action doit être diminuée ».

En pratique, l'usage de l'apprentissage par renforcement est complexe à mettre en œuvre, de sorte que nous n'aborderons aucun réseau qui l'emploie. Il importe cependant de bien comprendre la différence entre ce type d'apprentissage et l'apprentissage supervisé.

L'apprentissage supervisé dispose d'un signal d'erreur qui non seulement permet de calculer un indice de satisfaction (p.ex. l'erreur quadratique moyenne), mais permet aussi d'estimer le gradient local qui indique une direction pour l'adaptation des poids synaptiques. C'est cette information fournie par le professeur qui fait toute la différence. Dans l'apprentissage par renforcement, l'absence de signal d'erreur rend le calcul de ce gradient impossible. Pour estimer le gradient, le réseau est obligé de tenter des actions et d'observer le résultat, pour éventuellement inférer une direction de changement pour les poids synaptiques. Pour ce faire, il s'agit alors d'implanter un processus d'essais et d'erreurs tout en retardant la récompense offerte par l'indice de satisfaction. Ainsi, on introduit deux étapes distinctes : une d'exploration où l'on essaie des directions aléatoires de changement, et une d'exploitation où l'on prend une décision. Ce processus en deux étapes peut ralentir considérablement l'apprentissage. De plus, il introduit un dilemme entre le désir d'utiliser l'information déjà apprise à propos du mérite des différentes actions, et celui d'acquérir de nouvelles connaissances sur les conséquences de ces décisions pour, éventuellement, mieux les choisir dans le futur.

VII.2.7 - Apprentissage Par la règle de perceptron

Dans le neurone du perceptron on utilise la fonction d'activation à seuil, qui permet de classer les vecteurs d'entrée dans un hyperplan. La règle d'adaptation permet de modifier la position de l'hyperplan séparateur dont l'équation dans l'espace d'entrée est définie par les poids du neurone, afin de réduire l'erreur $\big(d(t)-y(t)\big)$. Ainsi, à chaque présentation d'un couple (x,y) mal classé (c'est-à-dire dont la sortie proposée $y(t)$ par le neurone ne correspond pas à la sortie désirée $d(t)$, on modifie le vecteur poids de manière à ramener le point mal classé du bon côté de l'hyperplan (figure 19). Si le point est bien classé, on ne fait rien.

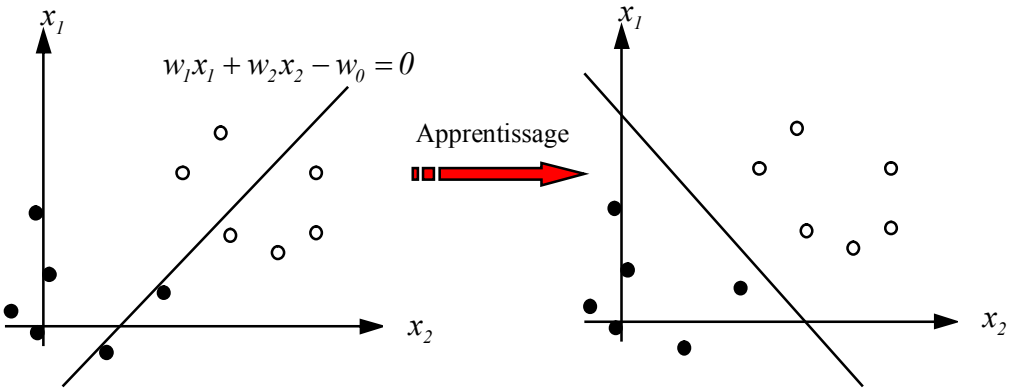


Fig.19 - Apprentissage par la règle de perceptron.

- Règle du Perceptron:

$$w_i(t+1) = w_i(t) + \alpha(t)\big(d(t)-y(t)\big)x_i \tag{30}$$

Où

$$y(t) = sign\big(s(t)\big) \Leftrightarrow \begin{cases} x(t) > 0 \Rightarrow y(t) = 1 \\ x(t) < 0 \Rightarrow y(t) = -1 \end{cases} \tag{31}$$

Et

$$d(t) \in [-1,1]$$

Le perceptron ne converge que si les classes sont linéairement séparables, on prend pour exemple la fonction logique *OU-Exclusif* qui ne peut pas être simulée par un perceptron.

- Le problème de Ou-Exclusif :

L'apprentissage ne converge pas, i.e l'erreur ne tend jamais vers zéro, les poids sont sans cesse modifiés car le perceptron n'arrive jamais à classer correctement tous les vecteurs d'entrées (Figure 20).

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

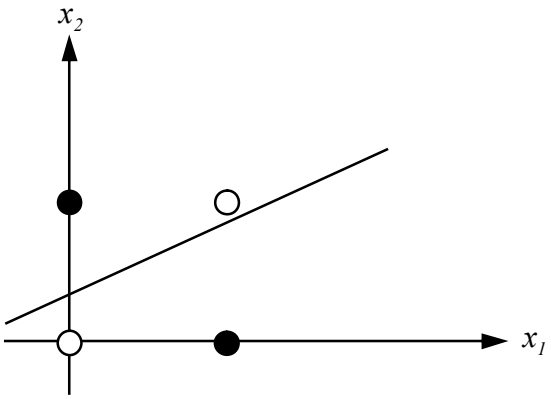


Fig. 20 - Apprentissage par la règle de perceptron.

VII.2.8 - Apprentissage par L'ADALINE

Soit un neurone dont la fonction de sortie est l'Identité:

$$y(t) = \sigma\big(s(t)\big) = s(t) \tag{32}$$

La sortie de neurone dépend linéairement des entrées d'où son nom .Il s'agit de résoudre un système d'équations linéaire (car σ est l'identité pour l'ADALINE).

Exemple pour n vecteurs d'apprentissage à deux dimensions présentés lors de la phase d'entraînement:

$$\begin{cases} \sigma\big(x_1(1)w_1 + x_2(1)w_2 - w_0\big) = y(1) \\ \sigma\big(x_1(2)w_1 + x_2(2)w_2 - w_0\big) = y(2) \\ \vdots \\ \sigma\big(x_1(n)w_1 + x_2(n)w_2 - w_0\big) = y(n) \end{cases} \tag{33}$$

Faire l'apprentissage du neurone consiste donc à trouver les poids w_i qui sont solutions de ce système .On constate que ce système n'a en générale pas de solution, i.e. si le nombre d'exemples présentés (ici n) est supérieur à la dimension de l'espace d'entrée plus un (pour les seuils) (ici *2+1 = 3*) donc au nombre d'inconnues (w_0, w_1 et w_2) .

Comme le système n'a souvent pas de solution, on va essayer de trouver une droite telle que la *somme des écarts* entre la distance $d(k)$ que l'on *désire* associer au vecteur d'entrée $x(k)$ et la distance *réelle* $y(k)$ de ce vecteur à la droite, soit minimale (Figure 21).

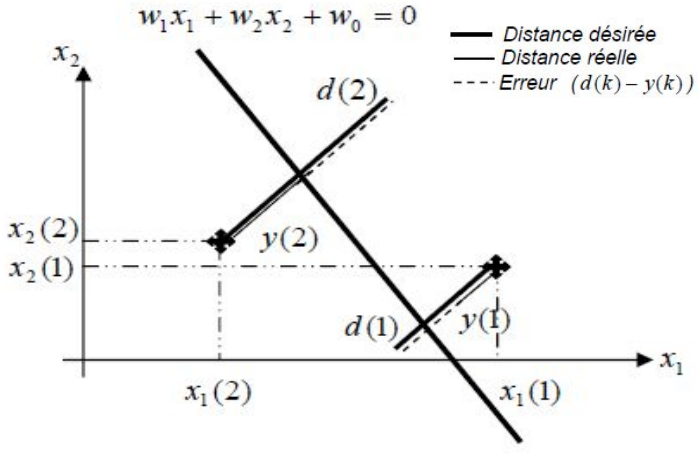


Fig.21: Présentation des écarts entre $d(k)$ et $y(k)$.

On va chercher par une méthode itérative (méthode du gradient stochastique) à minimiser un critère d'erreur de notre choix. Le critère des moindres carrés est très souvent utilisé :

$$\varepsilon(\vec{w},k)=\frac{1}{2}\big(d(k)-y(k)\big)^2 \tag{34}$$

Où $d(k)$ est la sortie désirée et $y(k)$ la sortie réellement fournie par le neurone pour chaque vecteur d'apprentissage, et w est le vecteur poids.

On note que lorsqu'une solution existe, elle annule cette fonction coût. On va donc calculer le gradient de la fonction d'erreur ε.

La règle d'apprentissage s'obtient alors par:

$$w(t+1)=w(t)-\alpha(t)\overrightarrow{grad}\Big(\varepsilon(\vec{w})\Big) \tag{35}$$

Où la coordonnée i du vecteur *grad* est donnée par:

$$\frac{\partial}{\partial w_i}\varepsilon(\vec{w})=-\big(d(t)-y(t)\big)\sigma'\big(s(t)\big)\,x_i(t) \tag{36}$$

Soit la règle du delta pour l'apprentissage de l'ADALINE :

$$w_i(t+1)=w_i(t)-\alpha(t)\big(d(t)-y(t)\big)x_i(t) \tag{37}$$

Et la règle du delta généralisée pour l'apprentissage d'un ADALINE non linéaire:

$$w_i(t+1)=w_i(t)-\alpha(t)\big(d(t)-y(t)\big)\,\sigma'\big(s(t)\big)x_i(t) \tag{38}$$

On note que dans ce dernier cas, la fonction de sortie σ doit être dérivable (par exemple une sigmoïde (*tanh*), mais pas un échelon). On peut constater que ces règles ressemblent à la règle de Hebb mais la sortie du neurone est remplacée par l'erreur de sortie (écart entre la sortie réelle et la sortie désirée).

L'ADALINE converge toujours vers la solution des moindres carrés, que les classes d'entrée soient ou non linéairement séparables, mais la solution obtenue n'est pas forcément celle qui sépare correctement les classes.

L'ADALINE non linéaire sépare toujours les classes linéairement séparables car les éléments éloignés de la zone de séparation ont peu d'influence (zone de saturation de la fonction d'activation).

L'information stockée par un neurone au niveau de ses poids est apprise des données. Les poids du réseau sont modifiés pour que la sortie qu'il propose soit le plus proche possible de la sortie désirée que les données impose.

L'apprentissage est en fait une phase d'optimisation itérative d'une fonction de coût dont le minimum est la solution recherchée, i.e. la configuration des poids telle que la sortie obtenue soit le plus proche possible de la sortie désirée au sens des moindres carrés.

VII.2.9 - Apprentissage non-supervisé « unsupervised learning »

La dernière forme d'apprentissage que nous abordons est dite «non-supervisée» ou encore «auto-organisée». Elle est caractérisée par l'absence complète de professeur, c'est-à-dire qu'on ne dispose ni d'un signal d'erreur, comme dans le cas supervisé, ni d'un indice de satisfaction, comme dans le cas par renforcement. Nous ne disposons donc que d'un environnement qui fournit des stimuli, et d'un réseau qui doit apprendre sans intervention externe. En assimilant les stimuli de l'environnement à une description de son état interne, la tâche du réseau est alors de modéliser

cet état le mieux possible. Pour y arriver, il importe d'abord de définir une mesure de la qualité pour ce modèle, et de s'en servir par la suite pour optimiser les paramètres libres du réseau, c'est-à-dire ses poids synaptiques. À la fin de l'apprentissage, le réseau a développé une habilité à former des représentations internes des stimuli de l'environnement permettant d'encoder les caractéristiques de ceux-ci et, par conséquent, de créer automatiquement des classes de stimuli similaires.

L'apprentissage non-supervisé s'appuie généralement sur un processus compétitif (voir §VII.2.3) permettant d'engendrer un modèle où les poids synaptiques des neurones représentent des prototypes de stimuli. La qualité du modèle résultant doit s'évaluer à l'aide d'une métrique permettant de mesurer la distance entre les stimuli et leurs prototypes. C'est le processus de compétition qui permet de sélectionner le prototype associé à chaque stimulus en recherchant le neurone dont le vecteur de poids synaptiques est le plus proche (au sens de la métrique choisie) du stimulus en question.

VIII - Les réseaux multicouches et l'approximation des fonctions

VIII.1 - Le Perceptron Multicouches (MLP)

VIII.1.1 - Structure du réseau MLP

Une seule couche de neurones ne pouvant réaliser que des séparations linéaires, l'idée vient alors de rajouter des couches dites cachées pour réaliser un réseau de neurone multicouche. Dans une couche, les neurones ne sont pas connectés entre eux.

Un réseau à couches est une extension du célèbre perceptron avec une ou plusieurs couches intermédiaires appelées "cachées". Le perceptron multicouches (**M**ulti **L**ayered **P**erceptron - MLP) sont les réseaux de neurones les plus connus. Un perceptron est un réseau de neurones artificiel du type « ***feedforward »***, c'est-à-dire à propagation directe. Le schéma de la figure 22 montre un réseau à trois couches possédant trois entrées et une sortie. La première est celle des entrées (elle n'est cependant pas considérée comme couche neuronale par certains auteurs car elle est linéaire et ne fait que distribuer les variables d'entrées). La deuxième est dite couche cachée et constitue le cœur du réseau de neurones. La troisième, constituée ici par un seul neurone est la couche de sortie, on numérote les couches de l'entrée vers la sortie donc sur le schéma de gauche à droite, la couche cachée a trois neurones aussi on notera par la suite un tel réseau 3 – 3 – 1.

Nous pouvons remarquer sur la figure 22, des termes x_0^m en entrée des neurones (le terme en exposant représente, non pas la fonction puissance, mais plutôt l'indice (^{m}) de la couche du réseau de neurones).

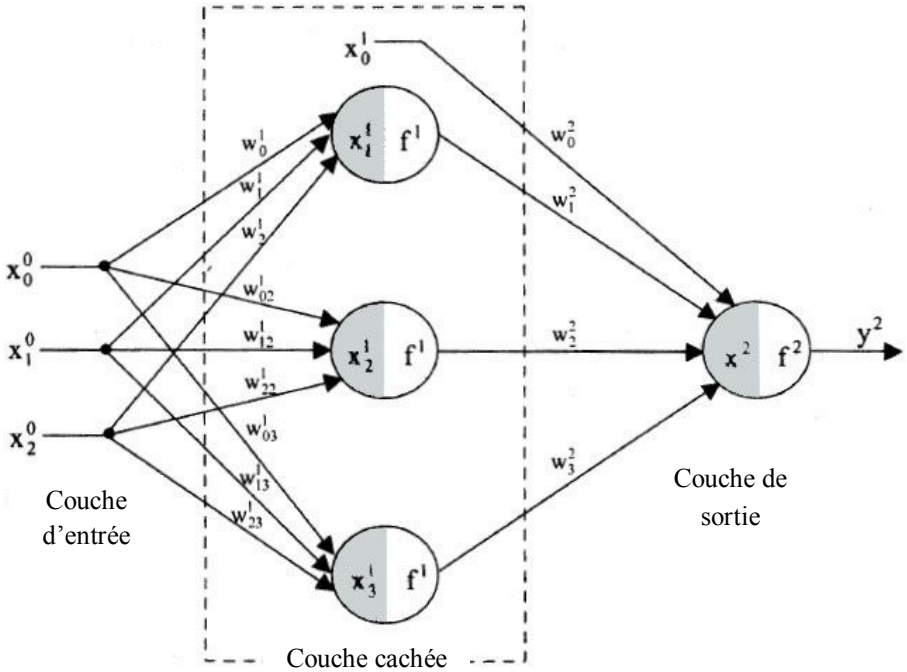


Fig.22 – Réseau de neurones de type perceptron à une couche cachée.

En fait, sur chaque neurone, en plus de ses entrées qui le lient avec les neurones précédents, on ajoute une entrée particulière que l'on appelle polarisation du neurone, elle correspond à un biais qui joue un rôle de translation du domaine d'activité du neurone. Sa valeur est donc liée à la fonction d'activation puisqu'elle permet le déplacement de cette fonction.

Afin de garder une notation généralisée, nous représentons ces biais comme le produit d'une entrée x_0^m par les poids w_{0j}^m. Nous fixons l'entrée x_0^m à l'unité, le poids porte alors l'information sur la polarisation du neurone.

L'un des problèmes de l'utilisation des réseaux multicouches (MLP) consiste dans le choix de son architecture. La détermination du nombre de couches nécessaires est fondamentale et à rendre minimale (pour des raisons évidentes de vitesse de calcul mais aussi de capacité de généralisation).

Le perceptron multicouche (MLP) est très utilisé en identification et en contrôle. Avec une couche cachée, il constitue un "approximateur universel". De récentes recherches montrent qu'il peut entrainer de manière à approximer n'importe quelle fonction entrées/sorties sous réserve de mettre suffisamment de neurones dans la couche cachée et d'utiliser des sigmoïdes pour les fonctions d'activation. Bien entendu, les théorèmes mathématiques ne démontrent pas qu'un réseau à une seule couche cachée est optimal.

Malheureusement, il n'existe pas de règle générale qui donne le nombre de neurones à retenir pour la couche cachée. Le résultat le plus important est certainement le théorème de Hecht-Nielsen : Toute fonction continue peut être implémentée exactement comme un réseau de neurones à trois couches ayant *n* cellules en entrée, *2n+1* cellule en couche cachée et *m* cellules de sortie. Il faut bien noter que ce théorème ne donne aucune indication quant au nombre de connexion (le réseau n'est pas toujours totalement connecté), et ne garantit pas que ce nombre de

neurones est optimal mais suffisant. Enfin, le réseau de neurones fait partie de Réseaux Adaptatifs Non-linéaires, cela signifie que ses agents (neurones) s'organisent et modifient leurs liens mutuels lors d'une procédure fondamentale qu'est l'apprentissage. Pour une tache précise, l'apprentissage du réseau de neurones consiste donc à adapter les différents poids.

VIII.1.2 - L'algorithme de la rétropropagation du gradient d'erreur (back-propagation)

L'un des algorithmes les plus répandus est celui de la "rétropropagation" ou *« back-propagation »*. Cet algorithme change les poids d'un réseau dont l'architecture est fixée par l'opérateur, à chaque fois qu'un exemple $y_i = f(x_i)$ est présenté. Ce changement est fait de telle sorte à minimiser l'erreur entre la sortie désirée et la réponse du réseau à une entrée x_i. Ceci est réalisé grâce à la méthode de descente de gradient. A chaque itération le signal d'entrée se propage dans le réseau dans le sens entrée-sortie, une sortie est ainsi obtenue, l'erreur entre cette sortie et la sortie désirée est calculée puis par rétropropagation *« error back-propagation »*, des erreurs intermédiaires, correspondant à la couche cachée sont ainsi calculées et permettent l'ajustement des poids $w_{ij}(t)$ de la couche cachée.

L'algorithme de rétropropagation du gradient comporte donc 2 phases :

1- Propagation : à chaque étape, on présente au réseau un exemple en entrée. Cette entrée est propagée jusqu'à la couche de sortie.

2- Correction : A coup sur, le réseau ne fournira pas exactement ce que l'on attendait. On calcule donc une erreur (en général la somme quadratique moyenne des erreurs pour tous les neurones de sortie) que l'on rétro-propage dans le réseau. Ce processus est interrompu dès que l'erreur globale est estimée suffisante.

La figure 23 montre le processus de rétropropagation.

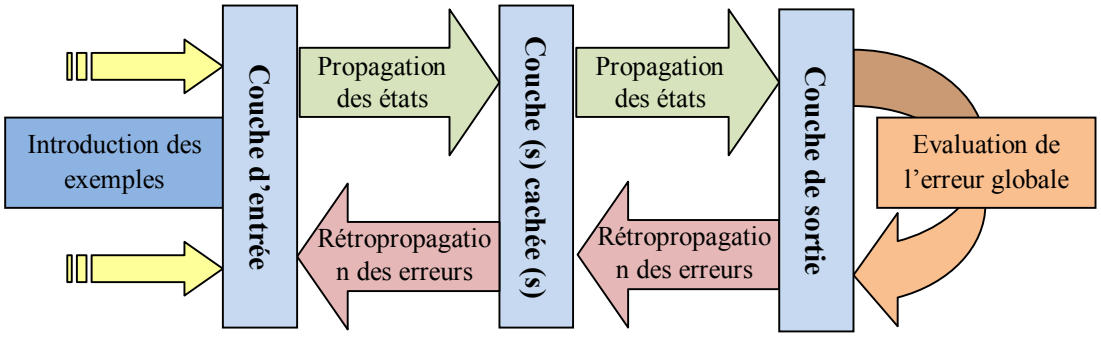


Fig.23 - Apprentissage des réseaux de neurone par l'algorithme de rétropropagation

VIII.1.3 - Présentation de l'algorithme de rétropropagation

Pour réaliser l'apprentissage d'un réseau multicouche, on utilise la règle d'apprentissage du delta généralisé pour chaque neurone i :

$$w_{ij}(t+1) = w_{ij}(t) + \alpha(t)\delta_j(t)x_i \qquad (39)$$

Où $\delta_j(t)$ est l'erreur faite par le neurone i.

Un exemple avec un réseau à deux entrées, trois neurones dans une couche cachée, et deux neurones dans la couche de sortie est illustré dans la figure 24.

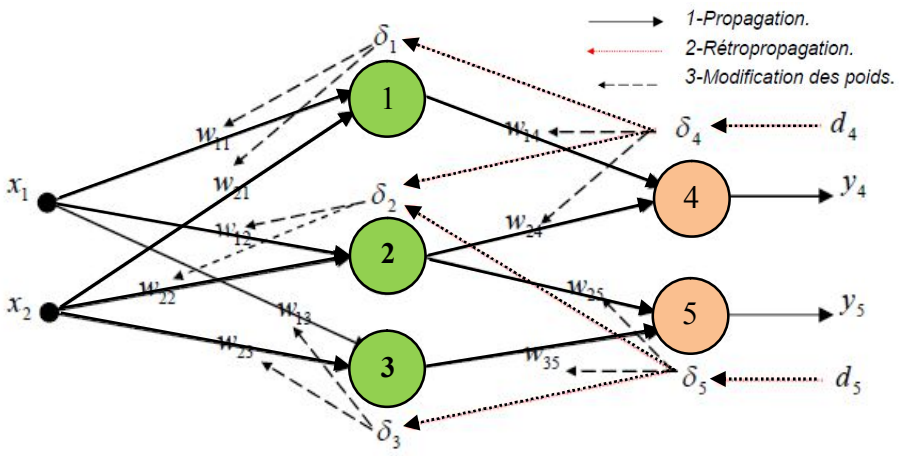


Fig.24 - Algorithme de rétropropagation.

Pour pouvoir modifier les poids synaptiques reliant la couche d'entrée à la couche cachée $(w_{11};w_{12};w_{13}\ et\ w_{21};w_{22};w_{23})$, il faut connaître les sorties désirées d_1, d_2 et d_3 qui permettent d'appliquer la règle du delta généralisé, i.e. connaître les erreurs: δ_1, δ_2 et δ_3 que font les neurones 1, 2 et 3.

L'idée consiste alors à propager les erreurs δ_4 et δ_5 vers les neurones 1, 2 et 3, au travers des poids $w_{14}, w_{24}, w_{25}\ et\ w_{35}$, d'où le nom de rétropropagation du gradient d'erreur de l'algorithme proposé indépendamment par Rumelhart, Le Cun et Hinton en 1984.

❖ **Exemple de calcule**

1. On calcule y_1, y_2, y_3 (on n'oublie pas le seuil w_0 qui n'est jamais représenté):

$$\begin{cases} y_1 = \sigma(x_1 w_{11} + x_2 w_{21} - w_{01}) \\ y_2 = \sigma(x_1 w_{12} + x_2 w_{22} - w_{02}) \\ y_3 = \sigma(x_1 w_{13} + x_2 w_{23} - w_{03}) \end{cases} \quad (40)$$

Puis y_4 et y_5

$$\begin{cases} y_4 = \sigma(y_1 w_{14} + y_2 w_{24} - w_{04}) \\ y_5 = \sigma(y_2 w_{25} + x_3 w_{35} - w_{05}) \end{cases} \quad (41)$$

2. On calcule les erreurs de la couche de sortie:

$$\begin{cases} \delta_4 = (d_4 - y_4)\sigma'(y_1 w_{14} + y_2 w_{24} - w_{04}) \\ \delta_5 = (d_5 - y_5)\sigma'(y_2 w_{25} + x_3 w_{35} - w_{05}) \end{cases} \quad (42)$$

Et les erreurs de la couche cachée:

$$\begin{cases} \delta_1 = w_{14}\delta_4\,\sigma'(x_1 w_{11} + x_2 w_{21} - w_{01}) \\ \delta_2 = (w_{24}\delta_4 + w_{25}\delta_5)\sigma'(x_1 w_{12} + x_2 w_{22} - w_{02}) \\ \delta_3 = w_{35}\delta_5\sigma'(x_1 w_{13} + x_2 w_{23} - w_{03}) \end{cases} \quad (43)$$

3. On calcule la nouvelle valeur de chaque poids entre la couche d'entrée et la couche cachée:

$$\begin{cases} w_{11}(t+1) = w_{11}(t) + \alpha(t)\delta_1(t)x_1 \\ w_{21}(t+1) = w_{21}(t) + \alpha(t)\delta_1(t)x_2 \\ w_{12}(t+1) = w_{12}(t) + \alpha(t)\delta_2(t)x_1 \\ w_{22}(t+1) = w_{22}(t) + \alpha(t)\delta_2(t)x_2 \\ w_{13}(t+1) = w_{13}(t) + \alpha(t)\delta_3(t)x_1 \\ w_{23}(t+1) = w_{23}(t) + \alpha(t)\delta_3(t)x_2 \end{cases} \quad (44)$$

Et entre la couche cachée et la couche de sortie:

$$\begin{cases} w_{14}(t+1) = w_{14}(t) + \alpha(t)\delta_4(t)y_1 \\ w_{24}(t+1) = w_{24}(t) + \alpha(t)\delta_4(t)y_2 \\ w_{25}(t+1) = w_{25}(t) + \alpha(t)\delta_5(t)y_2 \\ w_{35}(t+1) = w_{35}(t) + \alpha(t)\delta_5(t)y_3 \end{cases} \quad (45)$$

On vient de réaliser un pas d'apprentissage. Il faut recommencer ces opérations pour tous les vecteurs d'apprentissage, puis tester la qualité de l'apprentissage avec les vecteurs de test qui n'ont pas servi à l'apprentissage: ce qui permet de tester les capacités de la généralisation du réseau.

L'algorithme de rétropropagation du gradient consiste à effectuer une descente de gradient sur la fonction de coût déjà utilisée pour le neurone seul:

$$\varepsilon(\vec{w},k) = \frac{1}{2}\big(d(k) - y(k)\big)^2 \quad (46)$$

Où y est la sortie du réseau (donc une somme pondérée de sigmoïdes) et non la sortie d'un neurone seul. En dérivant cette expression par rapport à chaque poids w_{ij} jusqu'à l'obtention de $\varepsilon < \varepsilon_{seuil}$, et cela pour chaque paire entrée sortie (x_i, y_i).

Le mode d'adaptation des poids tel qu'il est présenté par les l'exemple précédent, s'appelle le *« batch mode »*. La mise à jour des poids se fait après avoir passée en revue tous les exemples d'apprentissage. Ce mode d'apprentissage est encore appelé déterministe, *« off-line »* ou « *by-epoch* ». Le réajustement des poids se fait suivant la moyenne de tous les exemples ce qui rend la méthode moins sensible aux bruits que peuvent contenir ces entrées.

Une autre approche consiste à modifier les poids après chaque présentation d'un exemple d'apprentissage, c'est l'apprentissage en mode *« on-line »* ou *« by pattern »*. De cette manière, le processus devient sensible à chaque exemple individuellement, ce qui le rend donc facilement influençable par les bruits que peuvent contenir ces entrées durant l'entrainement. Cette technique n'est donc utilisée que pour un apprentissage en temps réel.

VIII.2 - Problèmes et propriétés

VIII.2.1 - Problèmes liés à l'algorithme de rétropropagation

L'apprentissage par l'algorithme rétropropagation pose plusieurs problèmes, les principaux qu'on peut citer sont comme suit :

- ***L'architecture du réseau :*** il n'existe pas de règle générale pour déterminer la structure des réseaux (le nombre de couches cachées et le nombre de neurones par couche), sachant que le problème critique pendant l'apprentissage est de trouver un réseau assez large pour bien apprendre mais également assez petit pour bien généraliser. On ne sait (presque) pas

dimensionner correctement le réseau. Les couches entrées et sorties sont bien sur imposées, puisque le nombre de neurones qu'elles admettent dépend du problème posé. Mais que dire surtout du nombre de neurones des couches cachées ?

Si chaque neurone a une fonction de sortie de type sigmoïde (ex: *tanh*), et plus généralement si cette fonction est non linéaire (sauf polynomiale) et dérivable, alors Cybenko a démontré en 1989 qu'un réseau multicouche est capable d'approximer n'importe quelle fonction, à condition que le nombre de neurones dans la couche cachée soit suffisant.

- Une méthode par élagage consiste à initialiser le réseau avec un très grand nombre de neurones dans la couche cachée et à supprimer ceux d'entre eux dont les poids synaptiques sont très faibles et n'influencent pas trop le comportement du réseau.
- Une méthode par construction consiste à rajouter des neurones dans la couche cachée au fur et à mesure des "besoins", i.e. lorsque l'erreur globale du réseau ne diminue plus.

× ***Problème des valeurs initiales des poids du réseau :*** un autre problème est le temps de convergence de l'algorithme de rétropropagation. En effet, plus la somme pondérée des entrées d'un neurone est forte, plus le neurone se trouve dans la zone de saturation de sa fonction d'activation $\sigma(tanh)$, donc plus la dérivée σ' est faible (i.e. la pente de la fonction σ en zone de saturation), et moins les poids du neurone sont modifiés. Il faut donc démarrer l'apprentissage en initialisant les poids du réseau à des valeurs suffisamment faibles qui placent la fonction d'activation dans sa zone linéaire: on choisit donc en général des valeurs initiales inférieures à 0.1.

× ***Le temps d'apprentissage :*** le temps d'apprentissage augmente avec le nombre de couples d'apprentissage, ce qui diminue la vitesse de convergence.

× ***La convergence de l'algorithme :*** aucune preuve mathématique sur la convergence de cet algorithme vers un minimum global n'existe du fait que cette méthode utilise la décente du gradient. La recherche d'un minimum global sur la surface de l'erreur dans le domaine des poids, peut présenter un problème si cette surface possède des minimums locaux qui peuvent ralentir (voire stopper) l'algorithme. Le choix d'apprentissage variable permet dans certains cas d'accélérer la convergence. Il arrive cependant qu'on reste au dessus du critère d'arrêt sans jamais l'atteindre. C'est souvent le signe que le mécanisme d'apprentissage est inadapté ou que l'architecture du réseau ne permet d'atteindre ce degré de précision. Dans ce cas il faut augmenter le nombre de neurones de la couche cachée ou changer de structure.

× ***Le pas de correction des poids :*** si le pas de correction des poids est très petit, l'apprentissage nécessite alors un temps très important. Par contre si ce même pas est très grand le réseau devient oscillatoire, ce qui compromet sa convergence. La solution à ce problème est de choisir un pas variable initialisé à une grande valeur comprise entre 0 et 1, et qui sera diminuer jusqu'à une valeur minimale positive fixée au préalable.

× ***Le pas d'apprentissage :*** le réglage du pas d'apprentissage $\alpha(t)$ joue aussi un rôle important dans la vitesse de convergence. Ainsi il est préférable qu'il soit grand au début de l'apprentissage, et diminue au fur et à mesure que le réseau se rapproche de la solution. (La valeur du pas d'apprentissage est de l'ordre de 0.1 à 0.001).

- ***La saturation du réseau :*** si les poids prennent des grandes valeurs, les sorties deviennent grandes et se rapprochent de la zone de saturation de la fonction d'activation. La solution à ce problème est un compromis à faire entre :
  1- Le choix d'un pas de correction petit ;
  2- L'initialisation des poids à de très petites valeurs.
- ***Problème de sur-apprentissage :*** il faut aussi donner suffisamment d'exemples bien répartis (i.e. *représentatifs*) pour que le réseau généralise correctement, mais pas trop pour qu'il ne fasse pas de sur-apprentissage (i.e. de l'apprentissage par cœur) au détriment des capacités de généralisation. Un moyen simple de vérifier qu'il n'y a pas sur-apprentissage consiste à comparer l'erreur quadratique globale du réseau qui décroît toujours, et l'erreur faite par le réseau sur la base de test qui diminue puis augmente lorsqu'il y a sur-apprentissage. La base de test ne doit jamais servir à l'apprentissage.
- ***Le paramètre η :*** appelé taux d'apprentissage**,** joue un rôle important. S'il est trop faible, la convergence est lente, et s'il est trop grand l'algorithme oscille entre des points différents à cause de l'existence de vallées et de plateaux à la surface de la fonction cout. Pour stabiliser la recherche des poids optimisant la fonction coût, une méthode consiste à ajouter un terme dit de "moment" ou *« momentum »* à l'expression d'adaptation des poids, l'idée est de donner une certaine "inertie" pour chaque poids, de sorte que sa mise à jour ne se fasse pas de manière brutale. Ceci permet alors d'utiliser un taux d'apprentissage relativement important sans pour autant augmenter les oscillations de la trajectoire sur la surface d'erreur. Le choix de ce facteur est cependant délicat, on peut d'ailleurs aboutir à des effets inverses, des oscillations ou un ralentissement de la convergence.
- ***Le nombre d'itération :*** On ne peut pas prévoir le nombre d'itération nécessaire à l'apprentissage.
- L'algorithme de rétropropagation introduit la dérivée première des fonctions d'activation. Il est cependant tout à fait envisageable d'utiliser d'autres algorithmes qui ne nécessitent pas de dérivation comme par exemple les algorithmes génétiques.

VIII.2.2 - Propriétés de généralisation et de validation

La généralisation concerne la tache accomplie par le réseau une fois son apprentissage est achevé. Elle peut être évaluée en testant le réseau sur des données qui n'ont pas servi à l'apprentissage. Elle est influencée essentiellement par quatre facteurs :

1. La complexité du problème (sa nature) ;
2. L'algorithme d'apprentissage (son aptitude à trouver un minimum local assez profond, sinon le minimum global) ;
3. La complexité de l'échantillon (le nombre d'exemples et la manière dont ils représentent le problème ;
4. La complexité du réseau (nombre de poids).

Généralement, le modèle possédant un nombre de paramètres modéré réalise un bon compromis entre précision d'apprentissage et bonne généralisation.

La validation croisée consiste à extraire de la base d'apprentissage, une partie des exemples qui serviront non pas à l'apprentissage, mais à l'évaluation après apprentissage de l'erreur commise par le réseau. Ils constituent la base de vérification.

Durant le processus itératif, l'erreur commise sur la base d'apprentissage est réduite à chaque itération. Si l'on procède à chaque itération à une validation croisée sur une base de l'erreur sur la base d'apprentissage conduit à une augmentation de l'erreur sur la base de vérification. Ce phénomène est bien connu en modélisation. Il correspond schématiquement au cas où l'on introduirait trop de variables explicatives dans un modèle. Il arrive alors que l'on n'explique plus le comportement global du système, mais aléas spécifiques aux données de l'apprentissage : on modélise les résidus ! On dit dans ce cas qu'il y'a "sur-apprentissage" ou "apprentissage par cœur". Dans ce cas, le modèle perd sa capacité de généralisation.

L'algorithme de rétropropagation dans sa forme de base utilise la technique de descente du gradient, celle-ci est parmi la plus simple, mais elle n'est pas très efficace dans le cas général car elle utilise peu d'information sur la surface de l'erreur. Dans la littérature, on trouve une grande quantité de techniques plus sophistiquées, dont on peut citer quelque unes :

- ✓ Gradient descendant avec taux d'apprentissage variable ;
- ✓ Rétropropagation résiliente ;
- ✓ L'algorithme du gradient conjugué ;
- ✓ L'algorithme de Fletcher-Reeves ;
- ✓ Algorithme de Quasi-Newton ;
- ✓ Algorithme de Levenberg-Marquardt.

IX - Conclusion

Les réseaux de neurones formels, tels que nous les avons définis, possèdent une propriété remarquable qui est à l'origine de leur intérêt pratique dans des domaines très divers : ce sont des approximateurs universels parcimonieux.

La propriété d'approximation peut être énoncée de la manière suivante : toute fonction bornée suffisamment régulière peut être approchée avec une précision arbitraire, dans un domaine fini de l'espace de ses variables, par un réseau de neurones comportant une couche de neurones cachés en nombre fini, possédant tous la même fonction d'activation, et un neurone de sortie linéaire. Cette propriété de parcimonie est précieuse dans les applications industrielles.

L'apprentissage est vraisemblablement la propriété la plus intéressante des réseaux neuronaux, cependant il existe plusieurs algorithmes utilisés pour faire l'apprentissage des réseaux multicouches, en général, les méthodes du second ordre assurent une convergence plus rapide que celle du premier ordre pour les réseaux dont l'apprentissage est par paquets, pour les réseaux dont l'apprentissage est incrémental les méthodes du premier ordre assurent une convergence plus rapide que celle du second ordre. Pour les réseaux dont l'apprentissage est par paquets :

L'algorithme de Levenberg-Marquardt est le plus rapide et assure la meilleure convergence vers un minimum de l'erreur quadratique, pour les problèmes d'approximation des fonctions où le nombre des poids du réseau est inférieur à cents. Quand le nombre de poids augmente l'efficacité de l'algorithme LM diminue, car la taille du Hessien augmente et nécessite une très grande place dans la mémoire, cet algorithme est pauvre pour les problèmes de classification.

X. Références bibliographiques

H.S. Tsoukalas, R. E. Uhrig, « Fuzzy and neural approaches in engineering »; New York: Wiley and Sons, 1997.

B.K. Bose, « Artificial neural network applications in power electronics »; Proceedings of the IEEE Industrial Electronics Society (IECON'2001), pp.1631– 1638, Denver, CO, November-December 2001.

K.J. Hunt, D. Sbarbaro, R. Zbikowski, P.J. Gawthrop, « Neural networks for control systems – A survey »; Automatica, Vol.28, No.6, pp.1083–1112, 1992.

P. Wira, « Approches neuromimétiques pour l'identification et la commande »; Habilitation à Diriger des Recherches, Université de Haute Alsace, France, Novembre 2009.

M. Parizeau, « Réseaux de neurones »; Notes de cours (GIF-21140 et GIF-64326), Université Laval, Canada, Automne 2004.

G. Dreyfus, J. Martinez, M. Samuelides, M. Gordon, F. Badran, S. Thiria, L. Hérault, « Réseaux de neurones : méthodologie et application »; Eyrolles, Paris, France, 2002.

B. Widrow, M. A. Lehr, « 30 years of adaptive neural networks: Perceptron, Adaline, and Back-propagation »; Proceedings of the IEEE, Vol.78, No.9, pp.1415-1442, 1990.

S. Haykin, « Neural Networks : A comprehensive foundation »; Prentice Hall PTR, Upper Saddle River, New Jersey, USA, 1994.

G. Irwin, K. Warwick, K. Hunt, « Neural network applications in control »; The Institution of Electrical Engineers, London, 1995.

P. Bourret, « Réseaux neuronaux : une Approche Connexionniste de l'IA »; Edition TEKNA, TOULOUSE, France, 1991.