# Chapitre 3 Apprentissage automatique : les réseaux de neurones



## 3.1 Introduction

Comment l'homme fait-il pour raisonner, parler, calculer, apprendre, ...? Comment s'y prendre pour créer une ou de l'intelligence artificielle ? Deux types d'approches ont été essentiellement explorées :

- procéder d'abord à l'analyse logique des tâches relevant de la cognition humaine et tenter de les reconstituer par programme. C'est cette approche qui a été privilégiée par l'Intelligence Artificielle et la psychologie cognitive classiques. Cette démarche est étiquetée sous le nom de *cognitivisme*.
- puisque la pensée est produite par le cerveau ou en est une propriété, commencer par étudier comment celui-ci fonctionne. C'est cette approche qui a conduit à l'étude de réseaux de neurones formels. On désigne par *connexionnisme* la démarche consistant à vouloir rendre compte de la cognition humaine par des réseaux de neurones.

La seconde approche a donc menée à la définition et l'étude de réseaux de neurones formels qui sont des réseaux complexes d'unités de calcul élémentaire interconnectées. Il existe deux courants de recherche sur les réseaux de neurones : un premier motivé par l'étude et la modélisation des phénomènes naturels d'apprentissage à l'aide de réseaux de neurones, la pertinence biologique est importante ; un second motivé par l'obtention d'algorithmes efficaces ne se préoccupant pas de la pertinence biologique. Nous nous plaçons du point de vue du second groupe. En effet, bien que les réseaux de neurones formels aient été définis à partir de considérations biologiques, pour la plupart d'entre eux, et en particulier ceux étudiés dans ce cours, de nombreuses caractéristiques biologiques (le temps, la mémoire, ...) ne sont pas prises en compte. Toutefois, nous donnons, dans la suite de cette introduction, un bref aperçu de quelques propriétés élémentaires de neurophysiologie qui permettent au lecteur de relier neurones réels et neurones formels. Nous donnons ensuite un rapide historique des réseaux de neurones. Enfin, nous donnons une classification des différents types de réseau et les principales applications.

### 3.1.1 Quelques éléments de physiologie du cerveau

La physiologie du cerveau montre que celui-ci est constitué de cellules (les *neurones*) interconnectées. Quelques étapes de cette découverte :

- Van Leuwenhook (1718) : première description fidèle de ce qu'on appellera plus tard les *axones*,
- Dutrochet (1824) : observation du corps cellulaire des neurones
- Valentin : découverte des *dendrites*,
- Deiters (1865) : image actuelle de la cellule nerveuse
- Sherington (1897) : les *synapses*,
- les neuro-transmetteurs (première moitié du siècle).

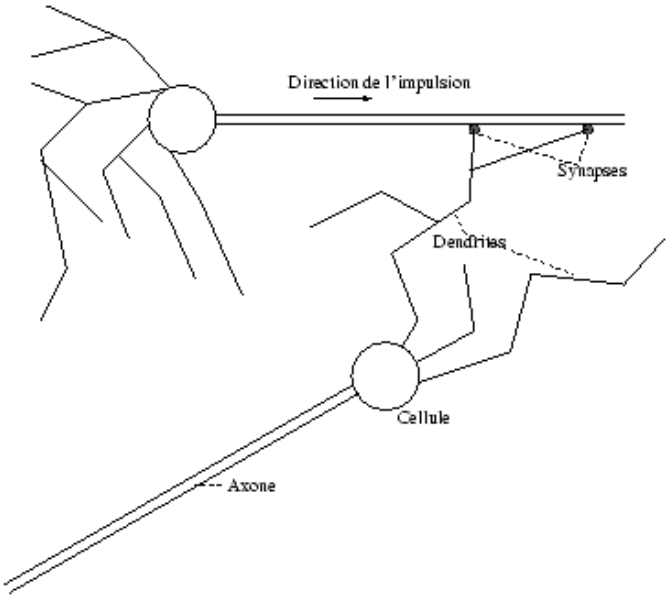


Figure 3.1 : Les neurones schématisés

---

Les *neurones* reçoivent les signaux (impulsions électriques) par des extensions très ramifiées de leur corps cellulaire (les *dendrites*) et envoient l'information par de longs prolongements (les *axones*). Les impulsions électriques sont régénérées pendant le parcours le long de l'axone. La durée de chaque impulsion est de l'ordre d'1 ms et son amplitude d'environ 100 mvolts.

Les contacts entre deux neurones, de l'axone à une dendrite, se font par l'intermédiaire des *synapses*. Lorsqu'un potentiel d'action atteint la terminaison d'un axone, des neuromédiateurs sont libérés et se lient à des récepteurs post-synaptiques présents sur les dendrites. L'effet peut être excitateur ou inhibiteur.

Chaque neurone intègre en permanence jusqu'à un millier de signaux synaptiques. Ces signaux n'opèrent pas de manière linéaire (effet de *seuil*).

Quelques informations en vrac :

- le cerveau contient environ 100 milliards de neurones.
- on ne dénombre que quelques dizaines de catégories distinctes de neurones.
- aucune catégorie de neurones n'est propre à l'homme (cela serait trop beau !).
- la vitesse de propagation des influx nerveux est de l'ordre de 100m/s. C'est à dire bien inférieure à la vitesse de transmission de l'information dans un circuit électronique.
- on compte de quelques centaines à plusieurs dizaines de milliers de contacts synaptiques par neurone. Le nombre total de connexions est estimé à environ $10^{15}$.
- la connectique du cerveau ne peut pas être codée dans un << document biologique >> tel l'ADN pour de simples raisons combinatoires. La structure du cerveau provient donc en partie des contacts avec l'environnement. L'apprentissage est donc indispensable à son développement.
- le nombre de neurones décroit après la naissance. Cependant, cette affirmation semble remise en question.
- on observe par contre une grande plasticité de l'axone, des dendrites et des contacts synaptiques. Celle-ci est surtout très importante après la naissance (on a observé chez le chat un accroissement des contacts synaptiques de quelques centaines à 12000 entre le 10ème et le 35ème jour). Cette plasticité est conservée tout au long de l'existence.
- les synapses entre des neurones qui ne sont pas simultanément actifs sont affaiblis puis éliminés.
- il semble que l'apprentissage se fasse par un double mécanisme : des connections sont établies de manière redondantes et aléatoires puis seules les connexions entre des neurones simultanément actifs sont conservés (phase de sélection) tandis que les autres sont éliminés. On parle de stabilisation sélective.

Nous conseillons sur le sujet deux livres accessibles au profane et passionnants : *L'homme neuronal* de Jean-Pierre Changeux et *La biologie de la conscience* de Gerald Edelman. Pour un survol de quelques pages, voir [*Kor97*]. Pour ceux que l'anglais n'effraie pas et qui aiment les bandes dessinées, voir [*GZ98*]. Les recherches sur la physiologie du cerveau sont actuellement

très actives.

### 3.1.2 Le connexionnisme et les réseaux de neurones formels

La question fondamentale du *connexionnisme* est :

> *comment rendre compte des processus cognitifs à partir d'un ensemble d'unités, dotées chacunes d'une faible puissance de calcul et interconnectées en réseau ?*

La définition de *réseaux de neurones formels* et l'expérimentation menée sur ces réseaux permettent d'étudier et de tester cette hypothèse. Citons quelques étapes dans la formalisation des réseaux de neurones :

- Première définition d'un neurone formel par McCulloch et Pitts en 1943
- Les percepts ou concepts sont physiquement représentés dans le cerveau par l'entrée en activité (simultanée) d'une *assemblée* de neurones (Donald Hebb, 1949). L'hypothèse concurrente est la spécialisation de certains neurones dans des tâches cognitives complexes (cf le fameux neurone << grand-mère >>).
- deux neurones entrant en activité simultanément vont être associés (c'est-à-dire que leur contacts synaptiques vont être renforcés). On parle de *loi de Hebb* et d'*associationnisme*
- Le *perceptron* de Frank Rosenblatt (1958) : le premier modèle pour lequel un processus d'apprentissage a pu être défini. De cette période, date également les travaux de Widrow et Hoff.
- Le livre de Minski et Papert "Perceptrons" (1969). Cet ouvrage contient une étude critique très complète des perceptrons. On lui reproche parfois violemment d'avoir sonné le glas des recherches sur les réseaux neuronaux dans les années 70, ce que nient leurs auteurs. Ce livre a été réédité en 1980, avec des ajouts et corrections manuscrites dans les marges, sans doute pour qu'on ne puisse pas les accuser de camoufler la première version du texte !

- l'algorithme de rétropropagation du gradient dans les réseaux multi-couches découvert au début des années 80 par Rumelhart et McClelland, Parker, Hinton, Le Cun. Les << inventeurs >> sont nombreux car l'idée de descente de gradient est naturelle. La plupart de ces travaux étaient associés à des études empiriques montrant la puissance du modèle.
- le modèle de Hopfield (1982) qui utilise des réseaux totalement connectés basés sur la règle de Hebb qui ont permis de définir la notion d'attracteurs et de mémoire associative.
- les cartes de Kohonen (1984) avec un algorithme non supervisé basé sur l'auto-organisation.
- la machine de Boltzman (1985), autre type de réseaux à attracteurs avec une dynamique de Monte-Carlo.

### 3.1.3 Classification des réseaux de neurones

Un réseau de neurones formels est constitué d'un grand nombre de cellules de base interconnectées. De nombreuses variantes sont définies selon le choix de la cellule élémentaire, de l'architecture du réseau et de la dynamique du réseau.

Une cellule élémentaire peut manipuler des valeurs binaires ou réelles. Les valeurs binaires sont représentées par 0 et 1 ou -1 et 1. Différentes fonctions peuvent être utilisées pour le calcul de la sortie. Le calcul de la sortie peut être déterministe ou probabiliste.

L'architecture du réseau peut être sans rétroaction, c'est à dire que la sortie d'une cellule ne peut influencer son entrée. Elle peut être avec rétroaction totale ou partielle.

La dynamique du réseau peut être synchrone : toutes les cellules calculent leurs sorties respectives simultanément. La dynamique peut être asynchrone. Dans ce dernier cas, on peut avoir une dynamique asynchrone séquentielle : les cellules calculent leurs sorties chacune à son tour en séquence ou avoir une dynamique asynchrone aléatoire.

Par exemple, si on considère des neurones à sortie stochastique -1 ou 1 calculée par une fonction à seuil basée sur la fonction sigmoïde, une interconnection complète et une dynamique synchrone, on obtient le modèle de Hopfield et la notion de mémoire associative.

Si on considère des neurones déterministes à sortie réelle calculée à l'aide de la fonction sigmoïde, une architecture sans rétroaction en couches successives avec une couche d'entrées et une couche de sorties, une dynamique asynchrone séquentielle, on obtient le modèle du Perceptron multi-couches (PMC) qui sera étudié dans les paragraphes suivants.

### 3.1.4 Applications des réseaux de neurones

Les principales applications des réseaux de neurones sont l'optimisation et l'apprentissage. En apprentissage, les réseaux de neurones sont essentiellement utilisés pour :

- l'apprentissage supervisé ;
- l'apprentissage non supervisé ;
- l'apprentissage par renforcement.

Pour ces trois types d'apprentissage, il y a également un choix traditionnel entre :

- l'apprentissage << off-line >> : toutes les données sont dans une base d'exemples d'apprentissage qui sont traités simultanément ;
- l'apprentissage << on-line >> : Les exemples sont présentés les uns après les autres au fur et à mesure de leur disponibilité.

Nous nous limitons, dans ce cours, à l'apprentissage supervisé à partir d'une base d'exemples. Dans ce cadre, l'apprentissage à l'aide de réseaux de neurones est bien adapté pour l'apprentissage à partir de données complexes (images sur une rétine, sons, ...) mais aussi à partir de données symboliques. Les entrées peuvent être représentées par de nombreux attributs à valeurs réelles ou symboliques, les attributs pouvant être dépendants ou non. La ou les sorties peuvent être réelles ou discrètes. L'apprentissage à l'aide de réseaux de neurones est tolérant au bruit et aux erreurs. Le temps d'apprentissage peut être long, par contre, après apprentissage, le calcul des sorties à partir d'un vecteur d'entrée est rapide. La critique principale est que le résultat de l'apprentissage, c'est-à-dire le réseau de neurones calculé par l'algorithme d'apprentissage, n'est pas interprétable par l'utilisateur : on ne peut pas donner d'explication au calcul d'une sortie sur un vecteur d'entrée. On parle de << boîte noire >>. Ceci est la principale différence entre réseaux de neurones et arbres de décision. Si l'utilisateur a besoin de pouvoir interpréter le résultat de l'apprentissage, il choisira un système basé sur les arbres de décision, sinon les deux méthodes sont concurrentes.

Nous n'étudions que le perceptron, brique de base des modèles plus complexes, et le perceptron multi-couches (PMC). L'accent sera mis sur les algorithmes d'apprentissage pour ces deux modèles, en particulier sur l'algorithme de rétropropagation du gradient appliqué aux PMC. Cet algorithme est, en effet, le premier algorithme d'apprentissage convaincant dans un modèle suffisamment puissant et cet algorithme a de nombreuses applications.

# 3.2 Le Perceptron

Le *perceptron* est un modèle de réseau de neurones avec algorithme d'apprentissage créé par Frank Rosenblatt en 1958. La version ci-dessous est simplifiée par rapport à l'originale. Vous trouverez une description de cette dernière dans l'exercice ??.

## 3.2.1 Définition du Perceptron

**Définition 5** *Un* perceptron linéaire à seuil *(voir figure ??) prend en entrée n valeurs* $x_1,..., x_n$ *et calcule une sortie o. Un perceptron est défini par la donnée de n+1 constantes : les* coefficients synaptiques $w_1, ..., w_n$ *et le* seuil *(ou le* biais*)* $\theta$*. La sortie o est calculée par la formule :*

$$o = \begin{cases} 1 \text{ si } \sum_i w_i x_i > \theta \\ 0 \text{ sinon} \end{cases}$$

---

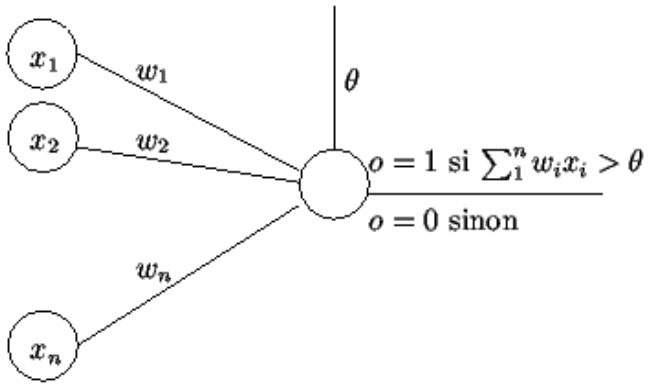


Figure 3.2 : Le perceptron avec seuil

---

Les entrées $x_1,...,x_n$ peuvent être à valeurs dans {0,1} ou réelles, les poids peuvent être entiers ou réels. Une variante très utilisée de ce modèle est de considérer une fonction de sortie prenant ses valeurs dans {-1,1} plutôt que dans {0,1}. Il existe également des modèles pour lesquels le calcul de la sortie est probabiliste. Dans la suite de cette partie sur le perceptron, nous considérerons toujours le modèle déterministe avec une sortie calculée dans {0,1}.

Pour simplifier les notations et certaines preuves, nous allons remplacer le seuil par une entrée supplémentaire $x_0$ qui prend toujours comme valeur d'entrée la valeur $x_0$=1. À cette entrée est associée un coefficient synaptique $w_0$. Le modèle correspondant est décrit dans la figure ??. On peut décomposer le calcul de la sortie *o* en un premier calcul de la quantité $\sum_i w_i x_i$ appelée *potentiel post-synaptique* ou l'*entrée totale* suivi d'une application d'une *fonction d'activation* sur cette entrée totale. La fonction d'activation est la fonction de Heaviside définie par :

$$f(x) = \begin{array}{l} 1 \; x > 0 \\ 0 \; sinon \end{array}$$

---

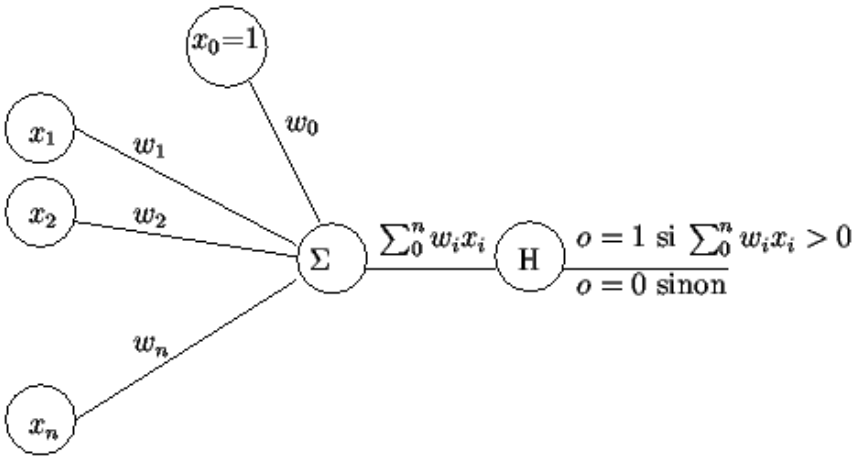


Figure 3.3 : Le perceptron avec entrée supplémentaire

---

Bien que considérant une entrée supplémentaire $x_0$, un perceptron est toujours considéré comme associant une sortie *o* aux *n* entrées $x_1,...,x_n$. L'équivalence entre le modèle avec seuil et le modèle avec entrée supplémentaire à 1 est immédiate : le coefficient $w_0$ est l'opposé du seuil $\theta$. Nous considérerons toujours ce dernier modèle de perceptron linéaire à seuil par la suite.

Pour passer du modèle avec sorties à valeurs dans {0,1} au modèle à valeurs dans {-1,1}, il suffit de remplacer la fonction de Heaviside *f* par la fonction *g* définie par : $g(x) = 2f(x) - 1$. D'autres fonctions d'activation peuvent également être utilisées.

**Exemple 10** *Un perceptron qui calcule le OU logique avec les deux versions : seuil ou entrée supplémentaire est présenté*

*dans la figure ??.*

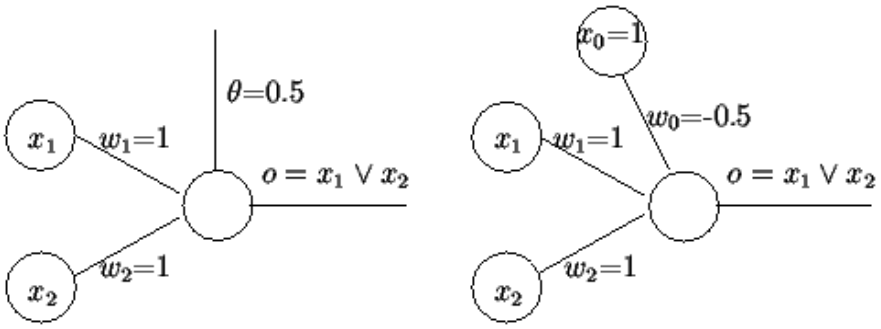


Figure 3.4 : perceptrons qui calculent le OU

On voit que quelques uns des traits principaux des neurones réels ont été retenus dans la définition du perceptron : les entrées modélisent les dendrites, les impulsions en entrée sont pondérées par les coefficients synaptiques et l'impulsion émise, c'est-à-dire la sortie, obéit à un effet de seuil (pas d'impulsion si l'entrée totale est trop faible).

Un perceptron à *n* entrées réelles (respectivement binaires) est une fonction de $R^n$ (respectivement $\{0,1\}^n$) dans $\{0,1\}$. Si l'on veut faire le lien avec les chapitres précédents, on peut voir les neurones d'entrées comme décrivant un espace de description avec des attributs réels (respectivement binaires) et le perceptron comme une procédure de classification binaire (c'est-à-dire en deux classes) sur cet espace. Un système d'apprentissage à base de perceptrons doit générer, à partir d'un ensemble d'apprentissage, une hypothèse qui est un perceptron. Nous nous intéressons, dans la section suivante, à cet espace d'hypothèses, c'est-à-dire à l'étude des fonctions calculables par perceptron.

### 3.2.2 Interprétation géométrique et limitations

**Définition 6** *Soit S un ensemble d'exemples dans* $R^n \times \{0,1\}$ *. On note* $S_0 = \{ s \in R^n \mid (s,0) \in S \}$ *et* $S_1 = \{ s \in R^n \mid (s,1) \in S \}$ *. On dit que S est* linéairement séparable *s'il existe un hyperplan H de* $R^n$ *tel que les ensembles* $S_0$ *et* $S_1$ *soient situés de part et d'autre de cet hyperplan.*

**Théorème 2** *Un perceptron linéaire à seuil à n entrées divise l'espace des entrées* $R^n$ *en deux sous-espaces délimités par un hyperplan. Réciproquement, tout ensemble linéairement séparable peut être discriminé par un perceptron.*

**Démonstration :** Il suffit pour s'en convaincre de se rappeler que l'équation d'un hyperplan dans un espace de dimension *n* est de la forme :

$$\alpha_1 x_1 + \ldots + \alpha_n x_n = \beta$$

Un perceptron est donc un discriminant linéaire. On montre facilement qu'un échantillon de $R^n$ est séparable par un hyperplan si et seulement si l'échantillon de $R^{n+1}$ obtenu en rajoutant une entrée toujours égale à 1 est séparable par un hyperplan passant par l'origine.

Toute fonction de $R^n$ dans $\{0,1\}$ est-elle calculable par perceptron ? La réponse est évidemment non. De même, toute fonction booléenne peut-elle être calculée par un perceptron ? La réponse est également non. Le contre-exemple le plus simple est le << OU exclusif >> (XOR) sur deux variables.

**Théorème 3** *Le XOR ne peut pas être calculé par un perceptron linéaire à seuil.*

**Démonstration :**

**Démonstration algébrique :** Supposons qu'il existe un perceptron défini par les coefficients synaptiques $(w_0, w_1, w_2)$ calculant le XOR sur deux entrées booléennes $x_1$ et $x_2$. On devrait avoir :

$$w_0 + 0\,w_1 + 0\,w_2 = w_0 \leq 0 \quad (3.1)$$
$$w_0 + 0\,w_1 + 1\,w_2 = w_0 + w_2 > 0 \quad (3.2)$$
$$w_0 + 1\,w_1 + 0\,w_2 = w_0 + w_1 > 0 \quad (3.3)$$
$$w_0 + 1\,w_1 + 1\,w_2 = w_0 + w_1 + w_2 \leq 0 \quad (3.4)$$

Il suffit d'additionner l'équation ?? et l'équation ?? d'une part, l'équation ?? et l'équation ?? d'autre part pour se rendre compte que l'hypothèse est absurde.

**Démonstration géométrique :** on << voit >> bien qu'aucune droite ne peut séparer les points de coordonnées (0,0) et (1,1) des points de coordonnées (0,1) et (1,0) (voir Figure ??). Si on considère une entrée $x_0$=1, il n'existe pas de plan passant par l'origine qui sépare les points de coordonnées (1,0,0) et (1,1,1) des points de coordonnées (1,0,1) et (1,1,0).

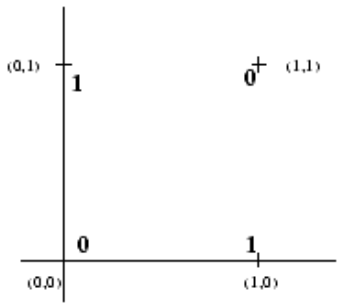


Figure 3.5 : Comment trouver une droite séparant les points (0,0) et (1,1) des points (0,1) et (1,0) ?

### 3.2.3 Algorithme d'apprentissage par correction d'erreur

#### Présentation de l'algorithme

Étant donné un échantillon d'apprentissage $S$ de $R^n \times \{0,1\}$ (respectivement $\{0,1\}^n \times \{0,1\}$), c'est-à-dire un ensemble d'exemples dont les descriptions sont sur $n$ attributs réels (respectivement binaires) et la classe est binaire, il s'agit de trouver un algorithme qui infère à partir de $S$ un perceptron qui classifie correctement les éléments de $S$ au vu de leurs descriptions si c'est possible ou au mieux sinon.

**Exemple 11** *Pour apprendre la notion de chiffre pair ou impair, on peut considérer un échantillon composé des 10 chiffres écrits sur une rétine à 7 leds.*

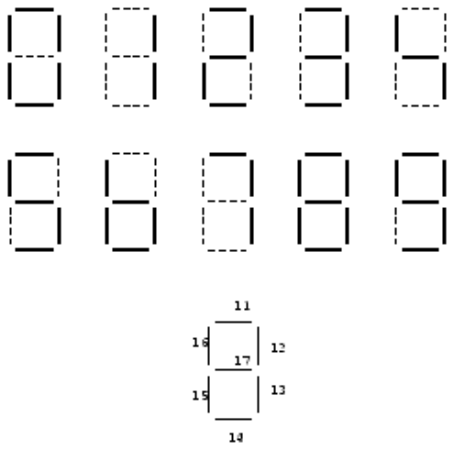

*Figure 3.6 : Les 10 chiffres sur une rétine à 7 leds*

---

*En représentant chaque chiffre par le symbole qui le désigne habituellement, un échantillon d'apprentissage complet est :*

*S = {(1111110,0), (0110000,1), (1101101,0), (1111001,1), (0010011,0), (1011011,1), (0011111,0), (1110000,1), (1111111,0), (1111011,1)}. Le but sera d'inférer, à partir de S, un perceptron qui prend ses entrées dans $\{0,1\}^7$ et qui retourne la classe 0 si le vecteur d'entrée correspond à un chiffre pair et 1 sinon. Sur cet exemple, l'échantillon S est complet (toutes les entrées possibles sont décrites). Il est fréquent, pour les problèmes concrets, d'avoir un échantillon non complet.*

L'algorithme d'apprentissage peut être décrit succinctement de la manière suivante. On initialise les poids du perceptron à des valeurs quelconques. A chaque fois que l'on présente un nouvel exemple, on ajuste les poids selon que le perceptron l'a correctement classé ou non. L'algorithme s'arrête lorsque tous les exemples ont été présentés sans modification d'aucun poids.

Dans la suite, nous noterons $x$ une description qui sera un élément de $R^n$ ou $\{0,1\}^n$. La i-ème composante de $x$ sera notée $x_i$. Un échantillon $S$ est un ensemble de couples $(x,c)$ où $c$ est la classe de $x$. Lorsqu'il sera utile de désigner un élément particulier de $S$, nous noterons $(x^s,c^s)$ le $s$-ième élément de $S$. $x_i^s$ désignera la i-ème composante du vecteur d'entrée $x^s$. Si une entrée $x^s$ est présentée en entrée d'un perceptron, nous noterons $o^s$ la sortie binaire calculée par le perceptron. Nous rappelons qu'il existe une $n$+1-ième entrée $x_0$ de valeur 1 pour le perceptron.

L'algorithme d'apprentissage par correction d'erreur du perceptron linéaire à seuil est :

**Algorithme par correction d'erreur:**
**Entrée :** un échantillon $S$ de $R^n \times \{0,1\}$ ou $\{0,1\}^n \times \{0,1\}$
Initialisation aléatoire des poids $w_i$ pour $i$ entre 0 et $n$
**Répéter**
Prendre un exemple $(x,c)$ dans $S$
Calculer la sortie $o$ du perceptron pour l'entrée $x$
- - Mise à jour des poids - -
**Pour** $i$ de 0 à $n$
$w_i \leftarrow w_i + (c-o)x_i$
**finpour**
**finRépéter**
**Sortie :** Un perceptron $P$ défini par $(w_0, w_1, ..., w_n)$

La procédure d'apprentissage du perceptron est une procédure de *correction d'erreur* puisque les poids ne sont pas modifiés lorsque la sortie attendue $c$ est égale à la sortie calculée $o$ par le perceptron courant. Étudions les modifications sur les poids lorsque $c$ diffère de $o$ :

- si $o$=0 et $c$=1, cela signifie que le perceptron n'a pas assez pris en compte les neurones actifs de l'entrée (c'est-à-dire les neurones ayant une entrée à 1) ; dans ce cas, $w_i \leftarrow w_i + x_i$ ; l'algorithme ajoute la valeur de la rétine aux poids synaptiques (*renforcement*).
- si $o$=1 et $c$=0, alors $w_i \leftarrow w_i - x_i$ ; l'algorithme retranche la valeur de la rétine aux poids synaptiques (*inhibition*).

Remarquons que, en phase de calcul, les constantes du perceptron sont les poids synaptiques alors que les variables sont les entrées. Tandis que, en phase d'apprentissage, ce sont les coefficients synaptiques qui sont variables alors que les entrées de l'échantillon $S$ apparaissent comme des constantes.

Certains éléments importants ont été laissés volontairement imprécis. En premier lieu, il faut préciser comment est fait le choix d'un élément de $S$ : aléatoirement ? En suivant un ordre prédéfini ? Doivent-ils être tous présentés ? Le critère d'arrêt de la boucle principale de l'algorithme n'est pas défini : après un certain nombre d'étapes ? Lorsque tous les exemples ont été présentés ? Lorsque les poids ne sont plus modifiés pendant un certain nombre d'étapes ? Nous reviendrons sur toutes ces questions par la suite. Tout d'abord, examinons le comportement de l'algorithme sur deux exemples :

**Exemple 12** ***Apprentissage du OU*** *: les descriptions appartiennent à $\{0,1\}^2$, les entrées du perceptron appartiennent à $\{0,1\}^3$, la première composante correspond à l'entrée $x_0$ et vaut toujours 1, les deux composantes suivantes correspondent aux variables $x_1$ et $x_2$ . On suppose qu'à l'initialisation, les poids suivants ont été choisis : $w_0$=0 ; $w_1$ = 1 et $w_2$ = -1. On suppose que les exemples sont présentés dans l'ordre lexicographique.*

| *étape* | $w_0$ | $w_1$ | $w_2$ | *Entrée* | $\sum_0^2 w_i x_i$ | *o* | *c* | $w_0$ | $w_1$ | $w_2$ |
|---|---|---|---|---|---|---|---|---|---|---|
| *init* | | | | | | | | *0* | *1* | *-1* |
| *1* | *0* | *1* | *-1* | *100* | *0* | *0* | *0* | *0+0x1* | *1+0x0* | *-1+0x0* |
| *2* | *0* | *1* | *-1* | *101* | *-1* | *0* | *1* | *0+1x1* | *1+1x 0* | *-1+1x1* |
| *3* | *1* | *1* | *0* | *110* | *2* | *1* | *1* | *1* | *1* | *0* |
| *4* | *1* | *1* | *0* | *111* | *2* | *1* | *1* | *1* | *1* | *0* |
| *5* | *1* | *1* | *0* | *100* | *1* | *1* | *0* | *1+(-1)x1* | *1+(-1)x0* | *0+(-1)x0* |
| *6* | *0* | *1* | *0* | *101* | *0* | *0* | *1* | *0+1x1* | *1+1x0* | *0+1x1* |
| *7* | *1* | *1* | *1* | *110* | *2* | *1* | *1* | *1* | *1* | *1* |
| *8* | *1* | *1* | *1* | *111* | *3* | *1* | *1* | *1* | *1* | *1* |
| *9* | *1* | *1* | *1* | *100* | *1* | *1* | *0* | *1+(-1)x1* | *1+(-1)x0* | *1 +(-1)x0* |
| *10* | *0* | *1* | *1* | *101* | *1* | *1* | *1* | *0* | *1* | *1* |

*Aucune entrée ne modifie le perceptron à partir de cette étape. Vous pouvez aisément vérifier que ce perceptron calcule le OU logique sur les entrées $x_1$ et $x_2$.*

**Exemple 13** ***Apprentissage d'un ensemble linéairement séparable*** *: les descriptions appartiennent à $R^2$, le concept cible est défini à l'aide de la droite d'équation y=x/2. Les couples (x,y) tels que y>x/2 sont de classe 1 ; Les couples (x,y) tels que y ≤ x/2 sont de classe 0. L'échantillon d'entrée est S={((0,2),1), ((1,1),1), ((1,2.5),1), ((2,0),0), ((3,0.5),0)}. On suppose qu'à l'initialisation, les poids suivants ont été choisis : $w_0$=0 ; $w_1$ = 0 et $w_2$ = 0. On choisit de présenter tous les exemples en alternant exemple positif (de classe 1) et exemple négatif.*

| *étape* | $w_0$ | $w_1$ | $w_2$ | *Entrée* | $\sum_0^2 w_i x_i$ | *o* | *c* | $w_0$ | $w_1$ | $w_2$ |
|---|---|---|---|---|---|---|---|---|---|---|
| *init* | | | | | | | | *0* | *0* | *0* |
| *1* | *0* | *0* | *0* | *(1,0,2)* | *0* | *0* | *1* | *1* | *0* | *2* |
| *2* | *1* | *0* | *2* | *(1,2,0)* | *1* | *1* | *0* | *0* | *-2* | *2* |
| *3* | *0* | *-2* | *2* | *(1,1,1)* | *0* | *0* | *1* | *1* | *-1* | *3* |
| *4* | *1* | *-1* | *3* | *(1,3,0.5)* | *-0.5* | *0* | *0* | *1* | *-1* | *3* |
| *5* | *1* | *-1* | *3* | *(1,1,2.5)* | *7.5* | *1* | *1* | *1* | *-1* | *3* |

*Aucune entrée ne modifie le perceptron à partir de cette étape car ce perceptron classifie correctement tous les exemples de S. Le perceptron de sortie associe la classe 1 aux couples (x,y) tels que y>x/3 - 1/3.*

---

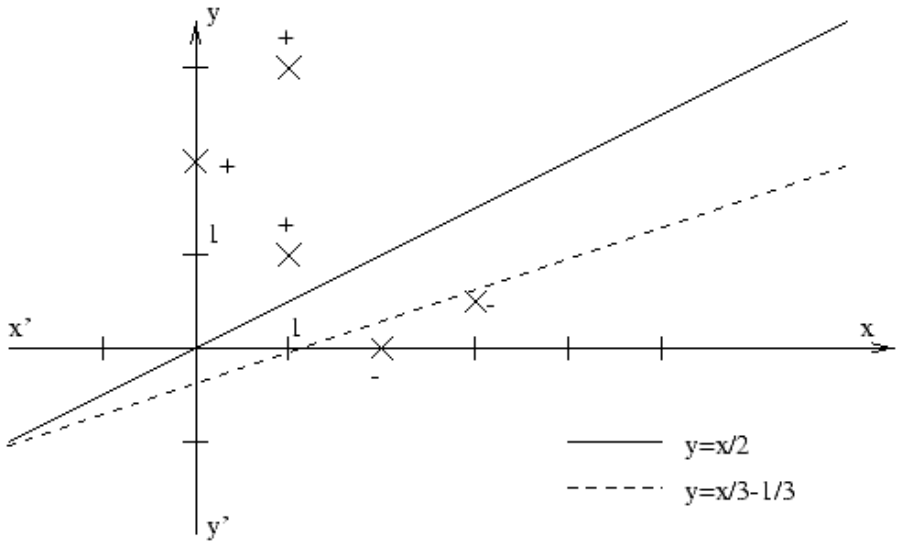


*Figure 3.7 : échantillon S ; hyperplans séparateurs cible et appris*

---

Dans les deux exemples, l'échantillon d'apprentissage est un ensemble linéairement séparable. Lors de la phase d'apprentissage, tous les exemples sont présentés jusqu'à la convergence, c'est-à-dire jusqu'à ce qu'une présentation complète des exemples n'entraîne aucune modification de l'hypothèse en cours. Nous démontrons, dans la section suivante, que ceci est un résultat général.

### Théorème d'apprentissage par correction d'erreur

**Théorème 4** *Si l'échantillon S est linéairement séparable et si les exemples sont présentés équitablement (c'est-à-dire que la procédure de choix des exemples n'en exclut aucun), la procédure d'apprentissage par correction d'erreur converge vers un perceptron linéaire à seuil qui calcule S.*

**Démonstration :**

Soit un échantillon d'entrée sur $n$ variables réelles (le cas de variables binaires s'en déduit), soit $S$ l'échantillon obtenu en ajoutant une $n$+1-ième entrée $x_0$ toujours égale à 1, par hypothèse l'échantillon est linéairement séparable, donc il existe un hyperplan de $R^{n+1}$ passant par l'origine qui sépare $S$, soit encore, il existe un vecteur $v=(v_0,\ldots, v_n)$ de $R^{n+1}$ tel que :

$$\forall (x,1)\in S \quad x.v=\sum_{i=0}^{n} x_i v_i >0 \text{ et } \forall (x,0)\in S \quad x.v<0 \qquad (3.5)$$

Comme $S$ est fini, cela implique qu'il existe un réel $d$ strictement positif tel que :

$$\forall (x,1)\in S \quad x.v>d \text{ et } \forall (x,0)\in S \quad x.v<-d \qquad (3.6)$$

Soit $w_0$ le vecteur des poids synaptiques choisi à l'initialisation de l'algorithme, soit $(w_i)_{i\in I}$ la suite des valeurs successives différentes des vecteurs de poids au cours de l'exécution de l'algorithme, c'est-à-dire que l'on suppose que, pour tout $i$, $w_i \neq w_{i+1}$ (le lecteur remarquera qu'il se peut que le vecteur $w_i$ reste inchangé pour un certain nombre d'exemples). Supposons que l'algorithme d'apprentissage ne s'arrête pas, la présentation des exemples étant équitable, ceci implique qu'il y a une infinité de $w_i$. Nous allons montrer que cette hypothèse est absurde.

Soit $M$ un majorant de $\{x^2 \mid (x,c)\in S\}$, nous allons calculer des bornes pour $w_i^2$ et $w_i.v$. Lors du passage de $w$ à $w'$, deux cas peuvent se produire :

$i$ $i+1$

premier cas :

l'exemple ($x$ ,$c$) vérifie $c$=1 et la sortie calculée sur cette entrée par le perceptron courant est $o$=0. $o$=0 donc $x$ .$w_i \leq 0$ ; $c$=1 donc $x$ .$v$ > 0. D'après la règle de mise à jour des poids, on a $w_{i+1} = w_i + x$ . On en déduit que :

- $w_{i+1}^2 = (w_i + x)^2 = w_i^2 + 2\, x.w_i + x^2 \leq w_i^2 + x^2 \leq w_i^2 + M$ ;
- $w_{i+1}.v = (w_i + x).v = w_i.v + x.v > w_i.v + d$.

soit

$$w_{i+1}^2 \leq w_i^2 + M \text{ et } w_{i+1}.v > w_i.v + d \quad (3.7)$$

deuxième cas :

l'exemple ($x$ ,$c$) vérifie $c$=0 et la sortie calculée sur cette entrée par le perceptron courant est $o$=1. $o$=1 donc $x$ .$w_i > 0$ ; $c$=0 donc $x$ .$v \leq 0$. D'après la règle de mise à jour des poids, on a $w_{i+1} = w_i - x$ . On montre de même que :

$$w_{i+1}^2 < w_i^2 + M \text{ et } w_{i+1}.v \geq w_i.v + d \quad (3.8)$$

Par conséquent, à l'aide des inégalités ?? et ??, nous déduisons que, pour tout $i \geq 0$, on a :

$$w_{i+1}^2 \leq w_i^2 + M \text{ et } w_{i+1}.v \geq w_i.v + d \quad (3.9)$$

D'où, pour tout $i \geq 1$, nous obtenons :

$$w_i^2 \leq w_0^2 + iM \text{ et } w_i.v \geq w_0.v + id \quad (3.10)$$

Comme $d$>0, il existe un entier $i_0$ tel que , pour tout $i \geq i_0$, $w_0.v + id$>0, en utilisant les inégalités de ??, nous obtenons que, pour tout $i \geq i_0$,

$$(w_0.v + id)^2 \leq (w_i.v)^2 = w_i^2 v^2 \leq (w_0^2 + iM)v^2 \quad (3.11)$$

Cette inégalité nous amène à la contradiction recherchée. En effet, le terme de gauche est un polynôme de degré 2 en $i$ majoré par le terme de droite qui est linéaire en $i$.

### Critiques sur la méthode par correction d'erreur

Nous venons de démontrer que si l'échantillon est linéairement séparable, si tous les exemples sont présentés équitablement et que le critère d'arrêt est la stabilité de l'hypothèse après une présentation complète de l'échantillon alors l'algorithme s'arrête avec un perceptron qui classifie correctement l'échantillon d'apprentissage.

Que se passe-t-il si l'échantillon d'entrée n'est pas linéairement séparable ? L'inconvénient majeur de cet algorithme est que si l'échantillon présenté n'est pas linéairement séparable, l'algorithme ne convergera pas et l'on aura aucun moyen de le savoir. On pourrait penser qu'il suffit d'observer l'évolution des poids synaptiques pour en déduire si l'on doit arrêter ou non l'algorithme. En effet, si les poids et le seuil prennent deux fois les mêmes valeurs sans que le perceptron ait appris et alors que tous les exemples ont été présentés, cela signifie d'après le théorème précédent que l'échantillon n'est pas séparable. Et l'on peut penser que l'on peut borner les poids et le seuil en fonction de la taille de la rétine. C'est vrai mais les résultats de complexité énoncés ci-dessous (sans démonstration) montrent que cette idée n'est pas applicable en pratique.

**Théorème 5**

1. *Toute fonction booléenne linéairement séparable sur n variables peut être implantée par un perceptron dont les poids synaptiques entiers $w_i$ sont tels que* $w_i \leq (n+1)^{n+1/2}$.

2. *Il existe des fonction booléennes linéairement séparables sur n variables qui requièrent des poids entiers supérieurs à* $2^{n+1/2}$.

Ces résultats sont assez décevants. Le premier montre que l'on peut borner les poids synaptiques en fonction de la taille de la rétine, mais par un nombre tellement grand que toute application pratique de ce résultat semble exclue. Le second résultat montre en particulier que l'algorithme d'apprentissage peut nécessiter un nombre exponentiel d'étapes (en fonction de la taille de la rétine) avant de s'arrêter. En effet, les poids ne varient qu'au plus d'une unité à chaque étape.

Même lorsque l'algorithme d'apprentissage du perceptron converge, rien ne garantit que la solution sera *robuste*, c'est-à-dire qu'elle ne sera pas remise en cause par la présentation d'un seul nouvel exemple. Pour s'en persuader, il suffit de se reporter à l'exemple ??. Supposons qu'on ajoute l'exemple ((3,1),0), cet exemple remet en cause l'hypothèse générée car le perceptron sorti par notre algorithme associe la classe 1 à la description (3,1). Un << bon >> algorithme d'apprentissage devrait produire une solution robuste. Graphiquement, si on considère un échantillon linéairement séparable, une solution robuste serait << à mi-chemin >> entre les points de classe 1 et de classe 0 comme le montre la Figure ??.

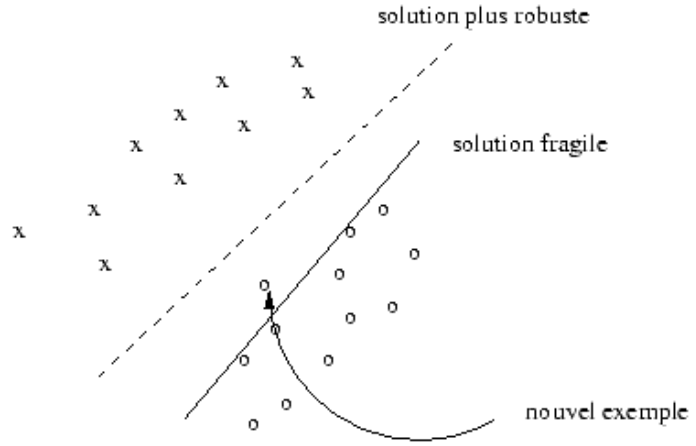


Figure 3.8 : Un nouvel exemple peut remettre en cause le perceptron appris.

Pire encore, cet algorithme n'a aucune tolérance au << bruit >> : si du bruit, c'est-à-dire une information mal classée, vient perturber les données d'entrée, le perceptron ne convergera jamais. En effet, des données linéairement séparables peuvent ne plus l'être à cause du bruit. En particulier, les problèmes *non-déterministes*, c'est-à-dire pour lesquels une même description peut représenter des éléments de classes différentes ne peuvent pas être traités à l'aide d'un perceptron. Si on considère les données de l'exemple présenté dans la Figure ??, les données ne sont pas linéairement séparables, mais un << bon >> algorithme d'apprentissage pour le perceptron devrait être capable de produire un séparateur linéaire comme celui qui est présenté dans cette même figure, ce qui n'est pas le cas de l'algorithme par correction d'erreur.

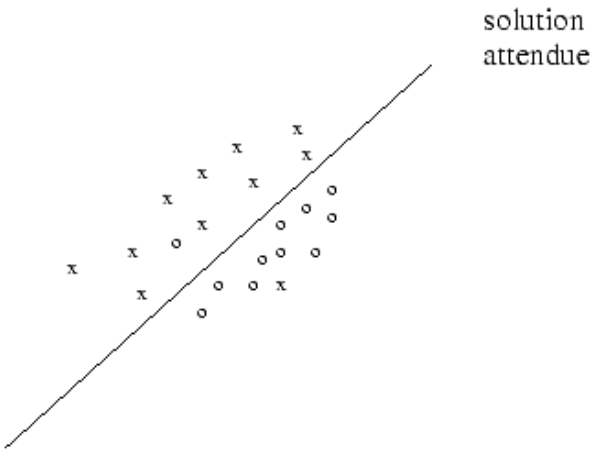


Figure 3.9 : Apprentissage en présence de bruit

---

Le but des sections suivantes est de présenter des algorithmes d'apprentissage du perceptron qui produisent des solutions robustes pour des échantillons linéairement séparables et des solutions << approximatives >> pour des échantillons non linéairement séparables.

## 3.2.4 Apprentissage par descente de gradient

### Introduction

Plutôt que d'obtenir un perceptron qui classifie correctement tous les exemples, il s'agira maintenant de calculer une erreur et d'essayer de minimiser cette erreur. Pour introduire cette notion d'erreur, on utilise des poids réels et on élimine la notion de seuil (ou d'entrée supplémentaire), ce qui signifie que la sortie sera égale au potentiel post-synaptique et sera donc réelle.

**Définition 7** *Un* perceptron linéaire *prend en entrée un vecteur* $x$ *de n valeurs* $x_1,...,x_n$ *et calcule une sortie o. Un perceptron est défini par la donnée d'un vecteur* $w$ *de n constantes : les* coefficients synaptiques $w_1, ..., w_n$. *La sortie o est définie par :*

$$o = x.w = \sum_{i=1}^{n} w_i x_i \quad (3.12)$$

L'erreur d'un perceptron P défini par $w = (w_1,...,w_n)$ sur un échantillon d'apprentissage $S$ d'exemples $(x^s, c^s)$ est définie en utilisant la fonction erreur quadratique par :

$$E(w) = 1/2 \sum_{(x^s,c^s)\in S} (c^s - o^s)^2 \quad (3.13)$$

où $o^s$ est la sortie calculée par P sur l'entrée $x^s$. L'erreur mesure donc l'écart entre les sorties attendue et calculée sur l'échantillon complet. On remarque que $E(w) = 0$ si et seulement si le perceptron classifie correctement l'échantillon complet.

On suppose $S$ fixé, le problème est donc de déterminer un vecteur $w$ qui minimise $E(w)$. Une méthode qui permet de rechercher le minimum d'une fonction est d'utiliser la méthode du gradient. Cette méthode est rappelée maintenant :

### Méthode du gradient

Soit $f$ une fonction d'une variable réelle à valeurs réelles, suffisamment dérivable dont on recherche un minimum. La méthode du gradient construit une suite $x$ qui doit en principe s'approcher du minimum. Pour cela, on part d'une valeur

quelconque $x_0$ et l'on construit la suite récurrente par : pour tout $n > 0$, $x_{n+1} = x_n + \Delta x_n$ avec $\Delta x_n = -\varepsilon f'(x_n)$ où $\varepsilon$ est une valeur << bien >> choisie.

On a : $f(x_{n+1}) = f(x_n - \varepsilon f'(x_n)) \approx f(x_n) - \varepsilon (f'(x_n))^2$ d'après le théorème des approximations finies si $\varepsilon f'(x_n)$ est << suffisamment >> petit. On voit que, sous réserve de la correction de l'approximation, $f(x_{n+1})$ est inférieur à $f(x_n)$.

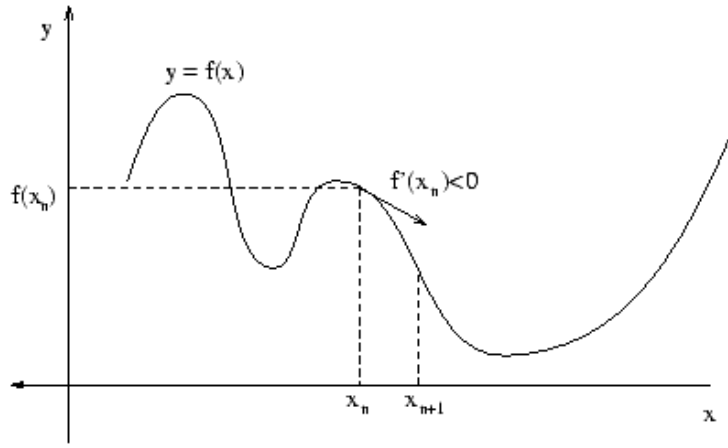


Figure 3.10 : La méthode du gradient

On remarque que $x_{n+1}$ est d'autant plus éloigné de $x_n$ que la pente de la courbe en $x_n$ est grande. On peut décider d'arrêter l'itération lorsque cette pente est suffisamment faible. Les inconvénients bien connus de cette méthode sont :

1. le choix de $\varepsilon$ est empirique,
2. si $\varepsilon$ est trop petit, le nombre d'itérations peut être très élevé,
3. si $\varepsilon$ est trop grand, les valeurs de la suite risquent d'osciller autour du minimum sans converger,
4. rien ne garantit que le minimum trouvé est un minimum global.

**Algorithme d'apprentissage par descente de gradient**

$E$ est une fonction des $n$ variables $w_i$. La méthode du gradient a été rappelée dans le cas d'une variable réelle, mais cette méthode peut être étendue au cas de fonctions de plusieurs variables réelles. Pour mettre en oeuvre la méthode appliquée à la fonction erreur quadratique $E$, nous allons, tout d'abord, évaluer la dérivée partielle de $E$ par rapport à $w_i$, pour tout $i$. On a :

$$\frac{\partial E(w)}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_S (c^s - o^s)^2 = \frac{1}{2} \sum_S \frac{\partial}{\partial w_i} (c^s - o^s)^2 = \frac{1}{2} \sum_S 2\,(c^s - o^s) \frac{\partial}{\partial w_i} (c^s - o^s) = \sum_S (c^s - o^s) \frac{\partial}{\partial w_i} (c^s - w.x^s) = \sum_S (c^s - o^s)\,(-x_i^s)$$

où $x_i^s$ est la $i$ème composante du vecteur $x^s$. L'application de la méthode du gradient nous invite donc à modifier le poids $w_i$ après une présentation complète de $S$ d'une quantité $\Delta w_i$ définie par :

$$\Delta w_i = -\varepsilon \times \frac{\partial E(w)}{\partial w_i} = \varepsilon \sum_S (c^s - o^s)\, x_i^s \qquad (3.14)$$

L'algorithme d'apprentissage par descente de gradient du perceptron linéaire peut maintenant être défini :

**Algorithme par descente de gradient :**

**Entrée :** un échantillon $S$ de $R^n \times \{0,1\}$ ; $\eta$

Initialisation aléatoire des poids $w_i$ pour $i$ entre 1 et $n$

**Répéter**

**Pour** tout $i$ $\Delta w_i \leftarrow 0$ **finPour**

**Pour** tout exemple $(x^s, c^s)$ de $S$

calculer la sortie $o^s$

**Pour** tout $i$ $\Delta w_i \leftarrow \Delta w_i + \eta(c^s - o^s) x_i^s$ **finPour**

**finPour**

**Pour** tout $i$ $w_i \leftarrow w_i + \Delta w_i$ **finPour**

**finRépéter**

**Sortie :** Un perceptron $P$ défini par $(w_1, ..., w_n)$

La fonction erreur quadratique ne possède qu'un minimum (la surface est une paraboloïde). L'algorithme précédent est assuré de converger, même si l'échantillon d'entrée n'est pas linéairement séparable, vers un minimum de la fonction erreur pour un $\eta$ bien choisi suffisamment petit. Si $\eta$ est trop grand, on risque d'osciller autour du minimum. Pour cette raison, une modification classique est de diminuer graduellement la valeur de $\eta$ en fonction du nombre d'itérations. Le principal défaut est que la convergence peut être très lente et que chaque étape nécessite le calcul sur tout l'ensemble d'apprentissage.

### Algorithme d'apprentissage de Widrow-Hoff

Cet algorithme est une variante très utilisée de l'algorithme précédent. Au lieu de calculer les variations des poids en sommant sur tous les exemples de $S$, l'idée est de modifier les poids à chaque présentation d'exemple. La règle de modification des poids donnée dans l'équation ?? devient :

$$\Delta w_i = \eta(c^s - o^s) x_i^s \qquad (3.15)$$

Cette règle est appelée *règle delta*, ou *règle Adaline*, ou encore *règle de Widrow-Hoff* d'après le nom de ses inventeurs. L'algorithme s'écrit alors :

**Algorithme de Widrow-Hoff :**

**Entrée :** un échantillon $S$ de $R^n \times \{0,1\}$ ; $\eta$

Initialisation aléatoire des poids $w_i$ pour $i$ entre 1 et $n$

**Répéter**

Prendre un exemple $(x, c)$ dans $S$

Calculer la sortie $o$ du perceptron pour l'entrée $x$

- - Mise à jour des poids - -

**Pour** $i$ de 1 à $n$

$w_i \leftarrow w_i + \eta(c-o)x_i$

**finpour**

**finRépéter**

**Sortie :** Un perceptron $P$ défini par $(w_0, w_1, ..., w_n)$

En général, on parcourt l'échantillon dans un ordre prédéfini. Le critère d'arrêt généralement choisi est : pour un passage complet de l'échantillon, toutes les modifications de poids sont en dessous d'un seuil prédéfini.

Au coefficient $\eta$ près dans la règle de modification des poids, on retrouve l'algorithme d'apprentissage par correction d'erreur. Pour l'algorithme de Widrow-Hoff, il y a correction chaque fois que la sortie totale (qui est un réel) est différente de la valeur attendue (égale à 0 ou 1). Ce n'est donc pas une méthode d'apprentissage par correction d'erreur puisqu'il y a modification du perceptron dans (presque) tous les cas. Rappelons également que l'algorithme par correction d'erreur produit en sortie un perceptron linéaire à seuil alors que l'algorithme par descente de gradient produit un perceptron linéaire. L'avantage de l'algorithme de Widrow-Hoff par rapport à l'algorithme par correction d'erreur est que, même si l'échantillon d'entrée n'est pas linéairement séparable, l'algorithme va converger vers une solution << optimale >> (sous réserve du bon choix du paramètre $\eta$). L'algorithme est, par conséquent, plus robuste au bruit (voir Figure ??).

L'algorithme de Widrow-Hoff s'écarte de l'algorithme du gradient sur un point important : on modifie les poids après présentation de *chaque* exemple en fonction de l'erreur locale et non de l'erreur globale. Rien ne prouve donc que la diminution de l'erreur en un point ne va pas être compensée par une augmentation de l'erreur pour les autres points. La justification empirique de cette manière de procéder est commune à toutes les méthodes adaptatives : le champ d'application des méthodes adaptatives est justement l'ensemble des problèmes pour lesquels des ajustements locaux vont finir par converger vers une solution globale.

L'algorithme de Widrow-Hoff est très souvent utilisé en pratique et donne de bons résultats. La convergence est, en général, plus rapide que par la méthode du gradient. Il est fréquent pour cet algorithme de faire diminuer la valeur de en fonction du nombre d'itérations comme pour l'algorithme du gradient.

### 3.2.5 Conclusion

En conclusion, l'apprentissage par perceptron ou par la méthode du gradient ne sont rien d'autre que des techniques de séparation linéaire qu'il faudrait comparer aux techniques utilisées habituellement en statistiques. Ces méthodes sont non paramétriques, c'est-à-dire qu'elles n'exigent aucune autre hypothèse sur les données que la séparabilité.

On peut montrer que << presque >> tous les échantillons de moins de $2n$ exemples sont linéairement séparables lorsque $n$ est le nombre de variables. Une classification correcte d'un petit échantillon n'a donc aucune valeur prédictive. Par contre, lorsque l'on travaille sur suffisamment de données et que le problème s'y prête, on constate empiriquement que le perceptron appris par un des algorithmes précédents a un bon pouvoir prédictif.

Il est bien évident que la plupart des problèmes d'apprentissage qui se posent naturellement ne peuvent pas être résolus par des méthodes aussi simples : il n'y a que très peu d'espoir que les exemples << naturels >> se répartissent << sagement >> de part et d'autre d'un hyperplan. Une manière de résoudre cette difficulté serait soit de mettre au point des séparateurs non-linéaires, soit (ce qui revient à peu près au même) de complexifier l'espace de représentation de manière à linéariser le problème initial. C'est ce que permettent de faire les réseaux multicouches que nous étudions maintenant.

## 3.3 Les réseaux multi-couches

### 3.3.1 Introduction et définition de l'architecture

Un perceptron linéaire à seuil est bien adapté pour des échantillons linéairement séparables. Cependant, dans la plupart des problèmes réels, cette condition n'est pas réalisée. Un perceptron linéaire à seuil est constitué d'un seul neurone. On s'est très vite rendu compte qu'en combinant plusieurs neurones le pouvoir de calcul était augmenté. Par exemple, dans le cas des fonctions booléennes, il est facile de calculer le XOR en utilisant deux neurones linéaires à seuil. Cet exemple est présenté dans la figure ??.

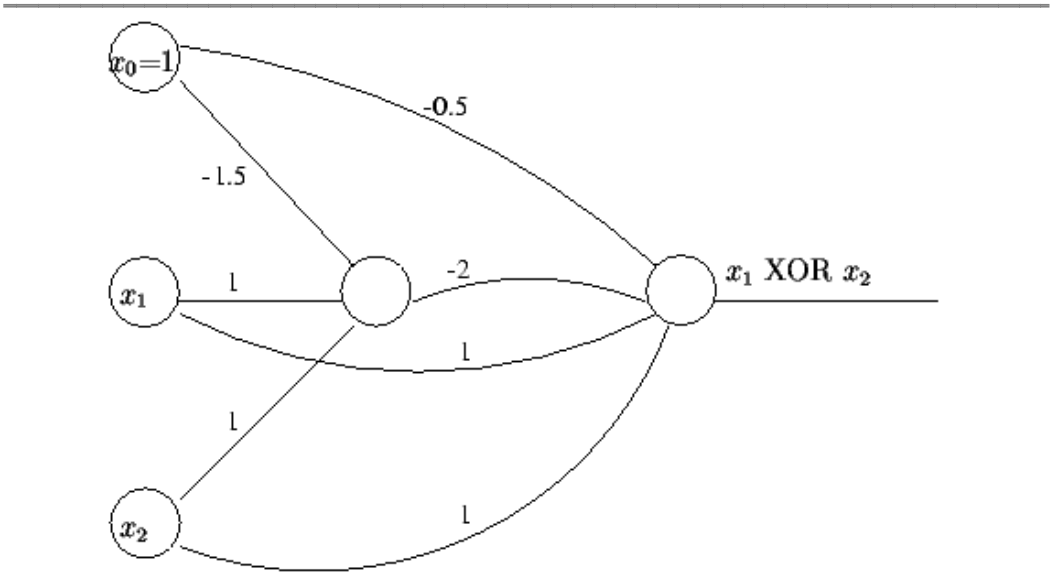


Figure 3.11 : Il suffit de rajouter un neurone intermédiaire entre la rétine et la cellule de décision pour pouvoir calculer le XOR

La notion de perceptron multi-couches (PMC) a ainsi été définie. On considère une couche d'entrée qui correspond aux variables d'entrée, une couche de sorties, et un certain nombre de couches intermédiaires. Les liens n'existent qu'entre les cellules d'une couche avec les cellules de la couche suivante. Le XOR peut être calculé par un perceptron multi-couches présenté dans la figure ?? en transformant légèrement le réseau présenté dans la figure ??.

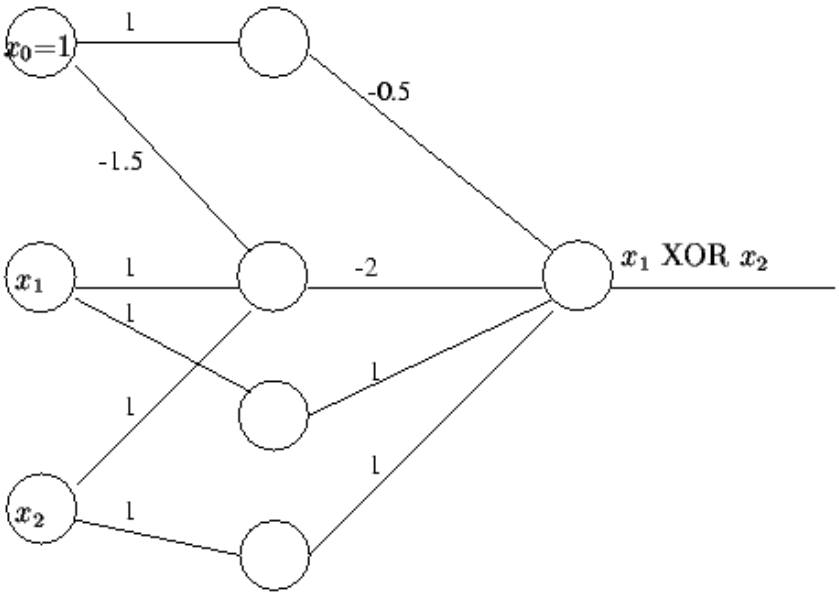


Figure 3.12 : PMC pour le XOR ; les liens avec poids nul ne sont pas représentés

**Définition 8** *Un réseau de neurones à couches cachées est défini par une architecture vérifiant les propriétés suivantes :*

- *les cellules sont réparties de façon exclusive dans des couches $C_0$, $C_1$, ..., $C_q$,*
- *la première couche $C_0$ est la rétine composée des cellules d'entrée qui correspondent aux n variables d'entrée ; les couches $C_1$, ..., $C_{q-1}$ sont les couches* cachées *; la couche $C_q$ est composée de la (ou les) cellule(s) de décision,*
- *Les entrées d'une cellule d'une couche $C_i$ avec $i \geq 1$ sont toutes les cellules de la couche $C_{i-1}$ et aucune autre cellule.*

*La dynamique du réseau est synchrone.*

Le réseau présenté dans la figure ?? pour le calcul du XOR est un réseau à une couche cachée. L'architecture d'un réseau à couches cachées est sans rétroaction. Dans notre définition, nous avons supposé qu'une cellule avait pour entrée toutes les cellules de la couche précédente, ce peut être un sous-ensemble des cellules de la couche précédente. Ce qui est primordial dans la définition, c'est que les entrées appartiennent uniquement à la couche précédente, c'est-à-dire que la structure en couches est respectée et qu'il n'y a pas de rétroaction.

Supposons que les cellules élémentaires soient des perceptrons linéaires à seuil, on parle alors de perceptrons multi-couches (PMC) linéaire à seuil. Soit $n$ variables binaires, il est facile de montrer que le OU $n$-aire est calculable par un perceptron linéaire à seuil et que toute conjonction sur les littéraux définis à partir des $n$ variables est calculable par un perceptron linéaire à seuil. Étant donné une fonction booléenne sur $n$ variables, cette fonction peut être mise sous forme normale disjonctive, il suffit alors que chaque cellule de la couche cachée calcule une conjonction et que la cellule de sortie calcule la disjonction des résultats. Nous avons ainsi démontré que :

**Proposition 1** *Toute fonction booléenne peut être calculée par un PMC linéaire à seuil comprenant une seule couche cachée.*

Cependant, si l'on utilise cette méthode pour construire un réseau de neurones pour calculer une fonction booléenne quelconque, la couche cachée pourra contenir jusqu'à $2^n$ neurones (où $n$ est la taille de la rétine), ce qui est inacceptable en pratique. On peut montrer par ailleurs que cette solution est loin d'être la meilleure (voir le cas de la fonction parité dans

l'exercice ??).

Pour pouvoir utiliser les réseaux multi-couches en apprentissage, deux choses sont indispensables :

- une méthode indiquant comment choisir une *architecture* de réseau pour résoudre un problème donné. C'est-à-dire, pouvoir répondre aux questions suivantes : combien de couches cachées ? combien de neurones par couches cachées ?
- une fois l'architecture choisie, un algorithme d'apprentissage qui calcule, à partir de l'échantillon d'apprentissage, les valeurs des coefficients synaptiques pour construire un réseau adapté au problème.

Le premier point est encore un sujet de recherche actif : quelques algorithmes d'apprentissage *auto-constructifs* ont été proposés. Leur rôle est double :

- apprentissage de l'échantillon avec un réseau courant,
- modification du réseau courant, en ajoutant de nouvelles cellules ou une nouvelle couche, en cas d'échec de l'apprentissage.

Il semble assez facile de concevoir des algorithmes auto-constructifs qui classent correctement l'échantillon, mais beaucoup plus difficile d'en obtenir qui aient un bon pouvoir de généralisation.

Il a fallu attendre le début des années 80 pour que le deuxième problème trouve une solution : *l'algorithme de rétropropagation du gradient*, découvert simultanément par des équipes française et américaine. Cet algorithme est, comme son nom l'indique, basé sur la méthode du gradient. Il est donc nécessaire de considérer des fonctions d'erreur dérivables. Ceci implique qu'il n'est pas possible de considérer comme cellule élémentaire un perceptron linéaire à seuil. L'idée est alors de prendre comme cellule élémentaire un perceptron linéaire. Malheureusement, dans ce cas, l'introduction de cellules supplémentaires n'augmente pas l'expressivité. En effet, une combinaison linéaire de fonctions linéaires est une fonction linéaire ! Nous allons donc avoir besoin de considérer une nouvelle cellule élémentaire. La sortie de cette cellule sera une fonction de variable réelle dérivable qui est une approximation de la fonction de Heaviside. Nous donnons dans la section suivante la définition d'une telle cellule et présentons l'algorithme de rétropropagation du gradient.

### 3.3.2 L'algorithme de rétropropagation du gradient

#### Définition du PMC

La fonction sigmoïde de paramètre $k>0$ est définie par :

$$\sigma_k(x) = \frac{e^{kx}}{e^{kx}+1} = \frac{1}{1+e^{-kx}} \quad (3.16)$$

Cette fonction est une approximation indéfiniment dérivable de la fonction à seuil de Heaviside, d'autant meilleure que $k$ est grand. Nous prendrons $k=1$ dans la suite, soit la fonction $\sigma$ (voir figure ??) définie par :

$$\sigma(x) = \frac{e^{x}}{e^{x}+1} = \frac{1}{1+e^{-x}} \quad (3.17)$$

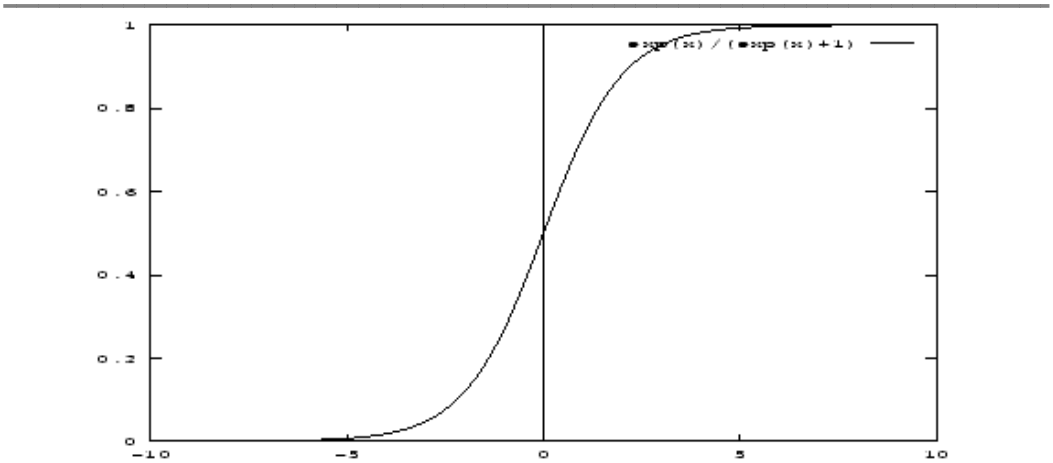

Figure 3.13 : La fonction sigmoïde

---

On peut remarquer que la dérivée de la fonction $\sigma$ est simple à calculer :

$$\sigma'(x) = \frac{e^x}{(1+e^x)^2} = \sigma(x)(1-\sigma(x)) \quad (3.18)$$

Il est essentiel que ce calcul soit simple car la dérivée de cette fonction sera utilisée dans la règle de mise à jour des poids par l'algorithme de rétropropagation du gradient. On utilise aussi parfois, pour cette raison, la fonction $th(x)$ qui est une approximation de la fonction de Heaviside dont la dérivée est égale à $1-th^2(x)$. Nous pouvons maintenant définir les réseaux considérés dans la suite de ce cours :

**Définition 9** *Une cellule élémentaire à n entrées réelles $x = (x_1,...,x_n)$ est définie par les poids synaptiques réels $w = (w_1,...,w_n)$ et la sortie o est calculée par la formule suivante :*

$$o(x) = \frac{1}{1+e^{-y}} \text{ où } y = x.w = \sum_{i=1}^{n} w_i x_i \quad (3.19)$$

*un perceptron multi-couches (PMC) est un réseau de neurones à couches cachées avec les cellules élémentaires ainsi définies.*

**Introduction de l'algorithme**

Le principe de l'algorithme est, comme dans le cas du perceptron linéaire, de minimiser une fonction d'erreur. Il s'agit ensuite de calculer la contribution à cette erreur de chacun des poids synaptiques. C'est cette étape qui est difficile. En effet, chacun des poids influe sur le neurone correspondant, mais, la modification pour ce neurone va influer sur tous les neurones des couches suivantes. Ce problème est parfois désigné sous le nom de << Credit Assignment Problem >>.

Soit un PMC défini par une architecture à $n$ entrées et à $p$ sorties, soit $w$ le vecteur des poids synaptiques associés à tous les liens du réseau. L'erreur du PMC sur un échantillon d'apprentissage $S$ d'exemples $(x^s, c^s)$ est définie par :

$$E(w) = 1/2 \sum_{(x^s, c^s) \in S} \sum_{k=1}^{p} (c_k^s - o_k^s)^2 \quad (3.20)$$

où $o_k^s$ est la $k$-ième composante du vecteur de sortie $o^s$ calculé par le PMC sur l'entrée $x^s$. L'erreur mesure donc l'écart entre les sorties attendue et calculée sur l'échantillon complet. On suppose $S$ fixé, le problème est donc de déterminer un vecteur $w$ qui minimise $E(w)$. Cependant, de la même façon que pour le perceptron avec la règle de Widrow-Hoff, plutôt que de chercher à minimiser l'erreur globale sur l'échantillon complet, on cherche à minimiser l'erreur sur chaque présentation individuelle d'exemple. L'erreur pour un exemple est :

$$E_{(x,c)}(w) = 1/2 \sum_{k=1}^{p} (c_k - o_k)^2 \quad (3.21)$$

Nous notons $E$ la fonction $E_{(x,c)}$, $E$ est une fonction des poids synaptiques, pour appliquer la méthode du gradient, il nous faut évaluer les dérivées partielles de cette fonction $E$ par rapport aux poids synaptiques. Les calculs qui suivent sont faciles. La seule complication provient de la complexité des notations et des indices utilisés, complication due à la structure du PMC. Nous utilisons les notations suivantes (voir figure ??) :

- chaque cellule est définie par un indice,
- le réseau comporte $p$ cellules de sortie,
- si $i$ est l'indice d'une cellule de sortie, $c_i$ est la sortie attendue pour cette cellule sur l'entrée $x$ ,
- $w_{ij}$ est le poids synaptique associé au lien entre cellule $j$ vers la cellule $i$, ce qui implique qu'elles se trouvent sur deux couches successives par définition de l'architecture,
- $x_{ij}$ est l'entrée associée au lien entre cellule $j$ vers cellule $i$,
- $Pred(i)$ est l'ensemble des cellules dont la sortie est une entrée de la cellule $i$ ; ceci implique que la cellule n'est pas une cellule d'entrée et que tous les éléments de $Pred(i)$ appartiennent à la couche précédente de celle à laquelle appartient la cellule $i$,
- $y_i$ l'entrée totale de la cellule $i$, soit $y_i = \sum_{j \in Pred(i)} w_{ij} x_{ij}$,
- $o_i$ est la sortie de la cellule $i$, soit $o_i = \sigma(y_i)$,
- $Succ(i)$ est l'ensemble des cellules qui prennent comme entrée la sortie de la cellule $i$, ceci implique la cellule n'est pas une cellule de sortie et que tous les éléments de $Succ(i)$ appartiennent à la couche suivante de celle à laquelle appartient la cellule $i$.

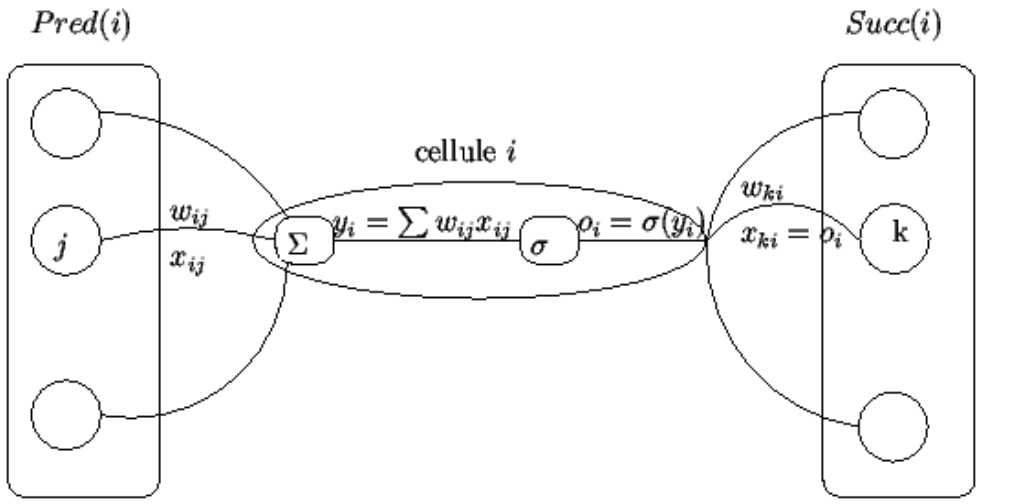


Figure 3.14 : notations utilisées

Il nous reste maintenant à évaluer $\partial E(w)/\partial w_{ij}$ que nous noterons $\partial E/\partial w_{ij}$. Tout d'abord remarquons que $w_{ij}$ ne peut influencer la sortie du réseau qu'à travers le calcul de la quantité $y_i$, ce qui nous autorise à écrire que :

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial w_{ij}} = \frac{\partial E}{\partial y_i} x_{ij} \qquad (3.22)$$

Il nous suffit donc de calculer $\partial E/\partial y_i$, pour cela, nous allons distinguer deux cas : le cas où la cellule $i$ est une cellule de sortie et le cas où c'est une cellule interne.

La cellule $i$ est une cellule de sortie

Dans ce cas, la quantité $y_i$ ne peut influencer la sortie du réseau que par le calcul de $o_i$. Nous avons donc :

$$\frac{\partial E}{\partial y_i} = \frac{\partial E}{\partial o_i}\frac{\partial o_i}{\partial y_i} \qquad (3.23)$$

Nous allons maintenant calculer chacune des deux dérivées partielles apparaissant dans l'équation ??. Pour la première de ces deux dérivées nous avons :

$$\frac{\partial E}{\partial o_i} = \frac{\partial}{\partial o_i}\frac{1}{2}\sum_{k=1}^{p}(c_k - o_k)^2$$

$k=1$

Seul le terme correspondant à $k=i$ a une dérivée non nulle, ce qui nous donne finalement :

$$\frac{\partial E}{\partial o_i} = \frac{\partial}{\partial o_i}\frac{1}{2}(c_i - o_i)^2 = -(c_i - o_i) \quad (3.24)$$

Pour la seconde des deux dérivées de l'équation ??, en utilisant la définition du calcul de la sortie d'une cellule élémentaire et la formule de calcul de la dérivée de la fonction sigmoïde donnée en ??, nous avons :

$$\frac{\partial o_i}{\partial y_i} = \frac{\partial \sigma(y_i)}{\partial y_i} = \sigma(y_i)(1-\sigma(y_i)) = o_i(1-o_i) \quad (3.25)$$

En substituant les résultats obtenus par les équations ?? et ?? dans l'équation ??, nous obtenons :

$$\frac{\partial E}{\partial y_i} = -(c_i - o_i)o_i(1-o_i) \quad (3.26)$$

La cellule $i$ est une cellule interne

Dans ce cas, la quantité $y_i$ va influencer le réseau par tous les calculs des cellules de l'ensemble $Succ(i)$. Nous avons alors :

$$\frac{\partial E}{\partial y_i} = \sum_{k \in Succ(i)} \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial y_i} = \sum_{k \in Succ(i)} \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial o_i}\frac{\partial o_i}{\partial y_i} = \sum_{k \in Succ(i)} \frac{\partial E}{\partial y_k} \times w_{ki} \times o_i(1-o_i)$$

Soit encore :

$$\frac{\partial E}{\partial y_i} = o_i(1-o_i) \sum_{k \in Succ(i)} \frac{\partial E}{\partial y_k} \times w_{ki} \quad (3.27)$$

Par l'étude de ces deux cas, nous avons obtenu deux équations ?? et ?? qui nous permettent de calculer les dérivées partielles $\partial E/\partial y_i$ pour toute cellule $i$. Le calcul devra être fait pour les cellules de sortie puis des cellules de l'avant-dernière couche jusqu'aux cellules de la première couche. C'est pour cette raison que l'on parle de << rétropropagation >>. Grace à l'équation ??, nous pouvons calculer toutes les dérivées partielles $\partial E(w)/\partial w_{ij}$. Enfin, pour en déduire la modification à effectuer sur les poids synaptiques, il nous reste simplement à rappeler que la méthode du gradient nous indique que :

$$\Delta w_{ij} = -\eta \frac{\partial E(w)}{\partial w_{ij}} \quad (3.28)$$

Tous les éléments sont donc en place pour nous permettre de définir l'algorithme de rétropropagation du gradient.

**Algorithme de rétropropagation du gradient**

Pour écrire l'algorithme, nous allons simplifier quelques notations. Nous appelons $\delta_i$ la quantité $-\partial E/\partial y_i$. En utilisant les équations ?? , ?? , ?? et ??, nous obtenons les formules suivantes :

pour une cellule $i$ de sortie, nous avons :

$$\delta_i = o_i(1-o_i)(c_i - o_i) \quad (3.29)$$

pour une cellule $i$ interne, nous avons :

$$\delta_i = o_i(1-o_i) \sum_{k \in Succ(i)} \delta_k w_{ki} \quad (3.30)$$

la modification du poids $w_{ij}$ est alors définie par :

$$\Delta w_{ij} = \eta x_{ij} \delta_i \quad (3.31)$$

On peut faire les remarques suivantes :

- la règle de modification des poids pour le perceptron linéaire est : $w_i \leftarrow w_i + \eta(c-o)x_i$. Dans le cas du PMC, cette règle est : $w_{ij} \leftarrow w_{ij} + \eta\delta_i x_{ij}$. Ces deux règles sont très similaires, le terme d'erreur $c$-$o$ est remplacé par un terme plus compliqué $\delta_i$,
- pour une cellule $i$ de sortie, la quantité $\delta_i$ correspond à l'erreur usuelle $c_i$-$o_i$ multipliée par la dérivée de la fonction sigmoïde,
- pour une cellule $i$ interne, le calcul de $\delta_i$ dépend de la somme pondérée des erreurs des cellules de la couche suivante,
- après présentation de l'entrée $x$ et calcul de la sortie $o$, le calcul des erreurs $\delta_i$ sera effectué de la couche de sortie vers la couche d'entrée.

Il ne reste qu'à écrire l'algorithme.

**Algorithme de rétropropagation du gradient :**
**Entrée :** un échantillon $S$ de $R^n \times R^p$ ; $\eta$
un PMC avec une couche d'entrée $C_0$, $q$-1 couches cachées $C_1$, ..., $C_{q-1}$,
une couche de sortie $C_q$, $n$ cellules.
Initialisation aléatoire des poids $w_i$ dans [-0.5,0.5] pour $i$ entre 1 et $n$
**Répéter**
Prendre un exemple ($x$ ,$c$ ) de $S$ et calculer $o$
- - calcul des $\delta_i$ par rétropropagation
**Pour** toute cellule de sortie $i$ $\delta_i \leftarrow o_i(1-o_i)(c_i-o_i)$ **finPour**
**Pour** chaque couche de $q$-1 à 1
**Pour** chaque cellule $i$ de la couche courante
$\delta_i = o_i(1-o_i) \sum_{k \in Succ(i)} \delta_k w_{ki}$
**finPour**
**finPour**
- - mise à jour des poids
**Pour** tout poids $w_{ij}$ $w_{ij} \leftarrow w_{ij} + \eta\delta_i x_{ij}$ **finPour**
**finRépéter**
**Sortie :** Un PMC défini par la structure initiale choisie et les $w_{ij}$

**Remarques**

- L'algorithme de rétroprogation du gradient est une extension de l'agorithme de Widrow-Hoff. En effet, dans les deux cas, les poids sont mis à jour à chaque présentation d'exemple et donc on tend à minimiser l'erreur calculée pour chaque exemple et pas l'erreur globale.
- La méthode donne de bons résultats pratiques. Dans la plupart des cas, on rencontre peu de problèmes dus aux minima locaux, peut être grâce au fait que l'on minimise une erreur locale.
- Cependant, les problèmes de minima locaux existent. Pour améliorer l'algorithme vis à vis de ce problème, mais aussi pour essayer d'améliorer la vitesse de convergence, une variante couramment utilisée consiste à pondérer la modification des poids en fonction du nombre d'itérations déja effectué. Plus formellement, on fixe une constante $\alpha \in$ [0,1[ appelée *moment* (momentum) ; soit $t$ un compteur du nombre d'itérations de la boucle principale, la règle de modification des poids devient :

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}(t)$$
$$\Delta w_{ij}(t) = \eta\delta_i x_{ij} + \alpha \Delta w_{ij}(t-1)$$

L'intérêt de cette règle est de prendre en compte les modifications antérieures des poids dans le but d'éviter des

oscillations perpétuelles.
- Le critère d'arrêt n'est pas précisé dans l'algorithme. Ce critère peut être : << arrêter dès que l'erreur estimée passe sous un seuil prédéfini >>. On retrouve alors le problème toujours présent en apprentissage de la sur-spécialisation. En effet, le critère d'arrêt dépend de l'erreur observée mesurée sur l'ensemble d'apprentissage et non de l'erreur réelle. Les méthodes pour palier à ce problème sont identiques à celles utilisées pour les arbres de décision. Par exemple, utiliser un ensemble test, quand c'est possible, pour estimer l'erreur réelle. Ou encore, utiliser des techniques d'élagage qui tentent de diminuer la taille du réseau.
- Le mode de présentation des exemples est également absent de notre algorithme. En règle générale, on choisit une présentation équitable.
- Le choix de l'architecture initiale du réseau reste un problème difficile. Ce choix peut être fait par l'expérience. Des méthodes dites << autoconstructives >> existent : il s'agit d'ajouter des cellules au cours de l'apprentissage pour que l'apprentissage se fasse bien. Mais ces méthodes rencontrent souvent le problème de la sur-spécialisation. L'architecture peut aussi être choisie à l'aide de méthodes basées sur les algorithmes génétiques.
- Le modèle et l'architecture étant choisie, un problème est également de choisir les << bonnes >> valeurs pour les paramètres $\alpha$ et $\beta$. Pour cela, on découpe l'ensemble d'apprentissage en un ensemble d'apprentissage, un ensemble de validation et un ensemble test. Lors de la phase d'apprentissage, on arrête périodiquement l'apprentissage, on estime l'erreur réelle sur l'ensemble de validation, on met à jour les paramètres $\alpha$ et $\beta$ en fonction de la variation de cette erreur estimée. L'ensemble test sert à estimer l'erreur réelle à la fin de l'apprentissage.
- L'algorithme de rétropropagation du gradient a pu être étendu à certaines classes de réseaux récurrents pour lesquels il y a rétroaction. Ces réseaux sont très utiles pour la prédiction de séries temporelles.
- Enfin, terminons par une dernière remarque sur la puissance de représentation des réseaux multi-couches. Nous avons vu que toute fonction booléenne peut être calculée par un PMC linéaire à seuil comprenant une seule couche cachée dont on rappelle qu'elle peut être de taille exponentielle. Ce résultat a été généralisé par Hornik en 1989 au cas des fonctions réelles et des PMC : *la plupart des fonctions numériques peuvent être approximées avec une précision arbitraire par des réseaux à une seule couche cachée*. Mais comme dans le cas booléen, cette couche cachée peut être démesurément grande et le théorème de Hornik est essentiellement un résultat *théorique* sur l'expressivité des réseaux multi-couches.

## 3.3.3 Applications

De nombreuses applications de l'algorithme de rétropropagation du gradient ont été réalisées. Parmi les plus souvent citées, nous en mentionnons deux : *NetTalk* de Sejnowski et les familles italo-américaines de Hinton

#### NetTalk

NetTalk est un réseau qui a appris à transformer un texte (en anglais) en une suite de phonèmes correspondant à sa lecture. Couplé en entrée à un scanner et à un OCR et en sortie à un synthétiseur de paroles, ce réseau est donc capable de lire un texte à haute voix.

Description de l'architecture du réseau :

- la couche d'entrée comprend 7 groupes de 29 neurones. Chaque groupe correspond à un caractère codé directement (non compressé). Les 7 caractères en entrée forment un contexte local de trois caractères entourant de part et d'autre un caractère central. Par exemple, le caractère 'c' se prononce différemment dans 'cygne' et dans 'carte'. Les ambiguïtés les plus courantes semblent pouvoir être levés avec un tel contexte.
- la couche cachée contient 80 neurones
- la couche de sortie comprend 26 neurones servant à coder les caractéristiques des phonèmes : la *zone vibratoire* (labiale, dentale, ...), le *type* de phonème (arrêt, nasale, fricative, ...), la *hauteur* des voyelles, la *ponctuation* (silence, pause, élision, arrêt net), l'*accentuation*, ...

---

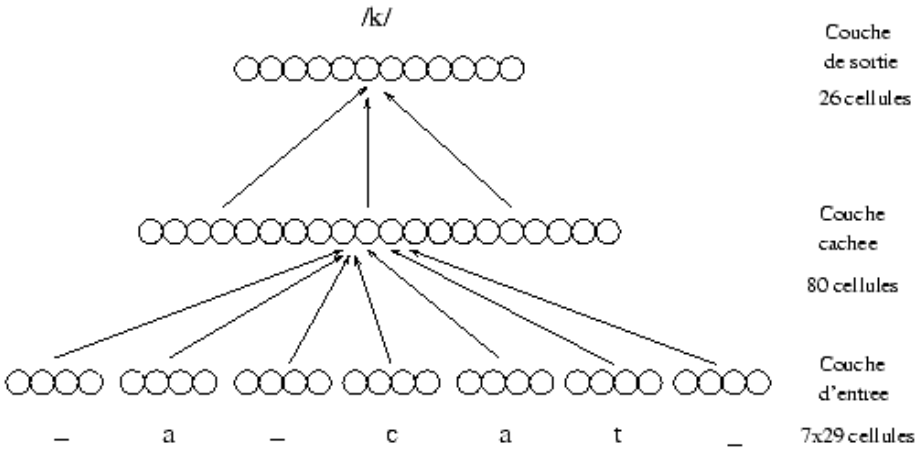


Figure 3.15 : NetTalk

---

Le réseau comprend donc au total 309 neurones et comme les connexions sont complètes d'une couche à l'autre, 18320 connexions.

D'après les résultats publiés : 50000 mots appartenant à un corpus de 1000 mots ont été présentés au réseau. Le temps d'apprentissage a été : une nuit sur un VAX 780. Les performances : 95% pour l'ensemble d'apprentissage et 75% pour les nouveaux mots.

Citons à ce propos Jean-Pierre Nadal : << Dans ses conférences, T. Sejnowski faisait entendre à l'auditoire un enregistrement sonore pris à divers moments au cours de la phase d'apprentissage. On pouvait alors entendre le réseau d'abords balbutier, puis on distinguait un découpage du texte en phrases, jusqu'à finalement une lecture raisonnable du texte. L'effet est évidemment spectaculaire, et il n'y a pas de doute, qu'à la suite de ces démonstrations, nombreux sont ceux qui se sont convertis au connexionnisme, si je puis dire ...On a ainsi vu, et ceci principalement aux Etats-Unis, se développer la vague, née en 1985, d'une activité impressionnante de ceux pour qui, << ça y était >> : pour résoudre n'importe quel problème, il suffit de mettre dans une boîte noire quelques neurones artificiels, d'injecter une base de données et de laisser tourner la << backprop >> pendant une nuit ; au matin, miracle, on retrouve une machine intelligente. Comme l'a dit Y. Le Cun, l'un des inventeurs de l'algorithme, l'usage de la RPG (rétropropagation du gradient) est à la fois *wide* et *wilde* (large et sauvage) ...

En fait, les performances de NetTalk étaient loin d'être exceptionnelles, si on les compare à ce qui se fait de mieux dans ce domaine de la lecture automatique. Il n'empêche que c'est une très jolie application, qu'on peut considérer comme le prototype de l'utilisation de la RPG pour un problème réel. Cette simulation démontre le pouvoir potentiel des réseaux de neurones : un temps de calcul raisonnable, une mise en oeuvre facile, et des performances acceptables. Mais elle montre aussi les limitations de l'approche : les performances ne sont *que* acceptables. >>

**Les familles italo-américaines**

Les deux arbres généalogiques ci-dessous présentent les relations entre les membres de deux familles comprenant chacune 12 personnes. On remarque que ces arbres sont isomorphes. Les relations sont : père, mère, mari, femme, fils, fille, oncle, tante, frère, soeur, neveu et nièce.

On souhaite faire apprendre ces relations à un réseau de neurones, c'est-à-dire que pour tout triplet de la forme (<personne1>,<relation>,<personne2>) décrit dans l'un des deux arbres, et toute entrée égale à (<personne1>,<relation>), le réseau calcule la réponse (<personne2>).

---

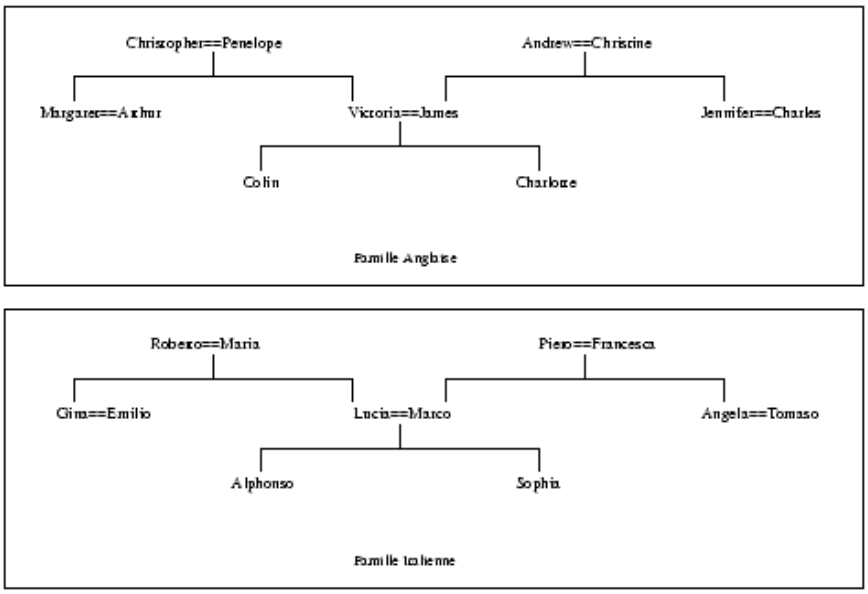


Figure 3.16 : Les familles américaines et italiennes de Hinton

Pour cela, Hinton utilise un réseau à 3 couches cachées dont l'architecture est décrite ci-dessous. Un groupe de 24 cellules d'entrée sert à coder les 24 personnes possibles. Un deuxième groupe de 12 cellules d'entrée sert à coder les relations. Chacun de ces groupes est connecté à un groupe de 6 cellules. Le rôle de cette couche est de coder l'information en entrée de manière optimale relativement au problème posé. La couche centrale contient 12 cellules ; c'est à ce niveau que la liaison persone-relation doit s'effectuer. L'avant dernière couche contient 6 cellules qui devra contenir une version codée de la sortie.

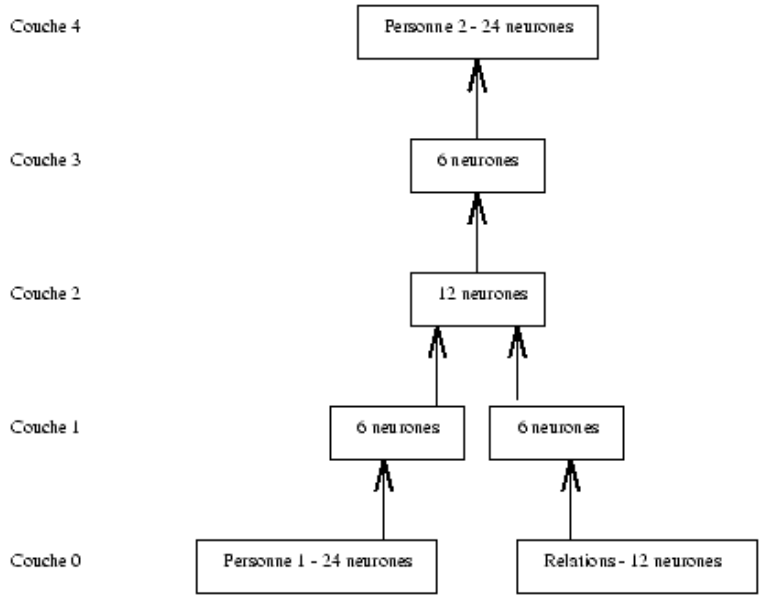


Figure 3.17 : Le réseau

Le réseau a été entrainé sur 100 des 104 relations possibles et après apprentissage prolongé, il a été capable de généraliser correctement sur les 4 exemples restants. Citons Hinton à ce propos :

<<It generalized correctly because during the training it learned to represent each of the people in terms of important features such as age, nationality, and the branch of the family tree that they belonged to, even these << semantic >> features were not at all explicit in the input or output vectors. Using these underlying features, much of the information about family relationships can be captured by a fairly small number of << micro-inferences >> between features. For example, the father of

a middle-aged person is an old person, and the father of an Italian person is an Italian person. So the features of the output person can be derived from the features of the input person and of relationship. The learning procedure can only discover these features by searching for a set of features that make it easy to express the associations. Once these features have been discovered, the *internal* representation of each person (in the first hidden layer) is a distributed pattern of activity and similar people are represented by similar patterns. Thus the network constructs its own internal similarity metric. This is a significant advance over simulations in which good generalization is achieved because the experimenter chooses representations that already have an appropriate similarity metric>>.

### 3.3.4 Conclusion

Autant le perceptron est un dispositif très rudimentaire d'apprentissage, autant des algorithmes comme la rétropropagation du gradient appliqué à des réseaux multicouches permettent d'aborder des problèmes déjà très complexes. Parmi les applications les plus fréquentes de ces réseaux, on peut noter :

- *la reconnaissance des formes*. Il semble que ce soit là un des domaines où les réseaux neuronaux sont les plus performants. On peut signaler comme exemple un réseau reconnaissant les visages (voir ouvrage de Mitchell [*Mit97*]). c'est un exemple de solution connexionniste d'un problème pour lequel les méthodes classiques de l'intelligence artificielle ont été très peu performantes.
- *concurrence avec les méthodes statistiques*. Les réseaux neuronaux sont de plus en plus utilisés en marketing, scoring, ...avec des succès divers. D'après certains statisticiens, si ces nouvelles méthodes sont intéressantes et parfois plus performantes que les techniques statistiques usuelles, elles sont aussi moins robustes, moins bien fondées et partant, plus dangereuses.
- *la cognition*. L'espoir qu'ont suscité les techniques connexionnistes dans la communauté des sciences cognitives provient du fait que l'on a pensé avoir trouvé avec elles un dispositif expliquant ou montrant comment le << symbolique >> pouvait émerger spontanément de l'expérience. Le compte-rendu des familles de Hinton vont dans ce sens. Il me semble que les travaux et expérimentations visant à étudier ce phénomène n'avancent que très lentement.

---