# Précis de connexionisme

HERVÉ ABDI

*The University of Texas at Dallas & Université de Bourgogne à Dijon*

## RÉSUMÉ

Les trois dernières décades de la psychologie expérimentale ont été marquées par le courant *cognitiviste* qui utilise abondamment comme modèle de la pensée humaine l'ordinateur séquentiel. Récemment, un nouveau courant, le *connexionisme*, est apparu sur la scène. Ce courant s'inspire plus étroitement que son prédécesseur du fonctionnement du cerveau, et des réseaux de neurones. Un modèle connexioniste consiste en un ensemble de processeurs élémentaires (*e.g.*, des neurones) inter-connectés et travaillant en parallèle. Ces processeurs constituent les unités de base du système. Ils sont reliés les uns aux autres par des connexions d'intensité modifiable par apprentissage. La réponse de chaque unité à un moment donné est une fonction de son état d'activation et de l'ensemble de l'information transmis par les connexions (*i.e.*, la réponse de chaque unité ne dépend que de l'information localement disponible pour cette unité). Ces modèles permettent de décrire de nombreux phénomènes psychologiques comme la mémoire et la perception. On utilise des simulations informatiques afin d'étudier le comportement de ces systèmes. La mathématique utilisée dépend de l'aspect linéaire ou non de la réponse des unités. Lorsque les unités sont linéaires, l'algèbre linéaire permet l'étude et la formalisation de ces systèmes, avec comme notion centrale les notions de vecteur et de valeur propres. Cette formalisation met en évidence les rapports étroits entre ces modèles et la statistique multivariée. Lorsque la réponse est non-linéaire, les notions mathématiques utilisées renvoient à la notion *d'attracteur*, et la recherche de minima locaux et globaux. Les techniques correspondent aux techniques d'optimisation utilisées par la physique des solides. Dans ce cas, l'utilisation de simulations informatiques est crucial pour l'étude de ces systèmes. Lorsque la réponse des unités est non-linéaire, on peut alors construire des modèles avec une architecture complexe comportant une couche d'unités d'entrée, des couches intermédiaires (ou cachées), et une couche de sortie. Pour modifier les connexions des unités des couches cachées, on utilise la technique de la *rétro-propagation* de l'erreur évaluée pour la couche de sortie vers les couches cachées. Dans ce cas, ces modèles implémentent la méthode de recherche de minima par la méthode de la *plus profonde descente*.

Ref: Abdi, H. (1993). Précis de Connexionisme. In J.P. Rossi, J.F. Le Ny, J.F. (Eds.) *Intelligence Artificielle et Intelligence Naturelle*. Paris: Presses Universitaires de France.

## 1. INTRODUCTION

La psychologie cognitive représente un courant majeur dans la psychologie comptemporaine. Son but, pour l'essentiel, est l'étude des activités mentales, c'est à dire, l'acquisition, le stockage, la récupération et l'utilisation de l'information ou des connaissances. Les points forts de cette approche portent sur l'étude de la mémoire, de la perception, de la reconnaissance des formes, de la résolution de problèmes, etc. La psychologie cognitive a hérité du courant comportementaliste (« le *behaviorisme* ») une méthodologie expérimentale rigoureuse, mais se sépare de cet ancêtre en acceptant l'étude de processus mentaux inobservables directement. Une source importante de ce changement de perspective fut l'utilisation de l'ordinateur comme *modèle* (Cf. Rumelhart, 1977; Lachman *et al.*, 1979; Abdi, 1986; Quinlan, 1991; Tiberghien, 1991), et la plupart des psychologues définissent la psychologie cognitive comme l'étude du *traitement de l'information* humain. Selon cette approche, nous traitons l'information par *une séquence* d'étapes successives. De telles séquences sont classiquement représentées par des organigrames semblables à ceux utilisés par les informaticiens pour décrire leurs algorithmes (à tel point que certains psychologues ont pu s'amuser de leurs—jeunes—collègues en remarquant que la psychologie semblait dans certains cas conduire à un comportement quasi-obsessionnel de dessin de losanges et de carrés avec des flèches autours, Cf. Underwood, 1975).

Malgré ses excès, la métaphore informatique s'est révélée dans bien des cas fructueuse. Mais certains critiques notent que le cerveau *par sa structure même* diffère d'un ordinateur séquentiel. Cette critique sera la base du courant « *connexioniste* ». En effet, une particularité essentielle du cerveau réside dans son parallélisme et sa lenteur comparé à un ordinateur séquentiel classique. Le cerveau est composé d'un vaste ensemble de neurones (de l'ordre de $10^{11}$), chacun d'eux relié à un grand nombre d'autres neurones (de l'ordre de $10^4$ à $10^5$ connexions par neurone). Chacun de ces neurones fonctionne comme un intégrateur élémentaire (et relativement lent) d'information travaillant « en parallèle ». Ces neurones sont eux-même regroupés en *noyaux* ou centres plus ou moins spécialisés dans certains types de traitement d'information (*e.g.*, centre moteurs, sensitifs, etc.). Encore que certaines fonctions puissent être localisées avec une certaine précision, d'autres échappent à ces tentatives, en particulier, les fonctions « supérieures », ainsi que le notait déjà Lashley en 1950 dans son célèbre essai sur « *la recherche de l'engramme* ». De ce fait, bien que le cerveau

humain et l'ordinateur soient tous deux des « *machines à traiter de l'information* », l'analogie comparant le cerveau à un ordinateur séquentiel composé d'un processeur central et d'une mémoire passive paraît inadéquate. L'observation du *comportement* de ces deux « machines » suffit, d'ailleurs, à s'en convaincre. N'importe quelle calculatrice à 5 francs, surpasse la plupart d'entre nous pour la précision et la rapidité de calcul. Dans l'ensemble, l'ordinateur fait merveille pour résoudre les problèmes de calcul *bien définis*. A l'inverse, le cerveau humain est capable de résoudre avec bonheur des problèmes mal posés ou mal définis. Comme, par exemple, arriver à retrouver le nom d'une personne avec comme indice « je me souviens que ça commence par C, et que j'ai rencontré cette personne en suisse », lorsque la personne s'appelle Paulette et que la rencontre a eu lieu à Paris.

L'école connexioniste préfère poser comme modèle simplifié un ensemble d'unités simples (que l'on peut interpréter comme des neurones ou des ensembles de neurones), avec des connexions entre éléments d'intensité modifiable. Un stimulus correspond dans ces modèles à un *vecteur* de valeurs d'activation, l'ensemble des connexions à une *matrice*, et la réponse du système à un *vecteur*. Le but de ces systèmes est d'*associer* une réponse en sortie au stimulus présenté en entrée, ou de *transformer* le stimulus en réponse.

Pour stocker de l'information, ou pour apprendre, ce système utilise *l'ensemble des connexions* entre unités, de là les termes utilisés pour décrire cette approche: « connexionisme », « traitement distribué et parallèle de l'information », etc. Chaque unité intègre l'ensemble de l'information reçue *via* les connexions, cette intégration étant linéaire ou non (*i.e.,* fonction logistique, réponse avec un plateau, etc.). La réponse de chaque unité est une fonction simple de cette quantité. L'apprentissage est possible par la modification de l'importance (*i.e.,* du « poids ») des connexions. Plusieurs règles d'apprentissage existent (Cf. Duda & Hart, 1973; Rumelhart & Mc Clelland, 1986), avec comme caractéristique commune de n'utiliser pour modifier le poids des connexions que l'information *localement* disponible (*i.e.,* input, réponse de l'unité, poids des connexions pour l'unité). Pour donner une idée intuitive de ces modèles et de leur fonctionnement, on détaille dans un premier temps le modèle classique du premier modèle connexioniste: le *Perceptron*.

Parmi les modèles connexionistes on peut distinguer deux grandes familles: les modèles linéaires, et les modèles non-linéaires. Dans ce qui

suit, on donne une présentation plus formelle d'un membre représentatif de chaque famille. Les modèles linéaires sont illustrés par les *mémoires associatives linéaires*, les modèles non-linéaires par les « *machines de Boltzman* ». Lorsque un modèle non-linéaire comporte plusieurs couches, le problème de l'apprentissage pour les couches internes se pose. Une solution possible est d'utiliser la technique dite de rétro-propagation de l'erreur décrite en fin de ce chapitre.

## 2. LE PERCEPTRON

Le perceptron peut être considéré comme le premier des réseaux de neurones. Il fut mis au point dans les années cinquantes par Rosenblatt (1957, 1961). Comme son nom l'indique, le perceptron se présentait comme un modèle de l'activité perceptive. Le but du perceptron est d'associer des configurations (*i.e.,* des formes, ou des stimuli) présentées en entrée à des réponses. Le perceptron se compose, pour l'essentiel, de deux couches de neurones.

La première couche était appelée, à l'origine, *la rétine* du perceptron. La deuxième couche donne la réponse du perceptron correspondant à la stimulation donnée en entrée (Rosenblatt, 1958, proposait une architecture en apparence plus complexe, mais, de fait, équivalente à celle décrite ici).

Par exemple, imaginons un perceptron composé d'une rétine de $10 \times 10 = 100$ cellules et d'une couche de sortie composée de 26 cellules. La tâche du perceptron sera de reconnaître les lettres présentées sur la rétine.

Chaque lettre devra être associée à une cellule de sortie. Ainsi, lorsque la lettre $A$ est présentée sur la rétine, la première cellule de sortie doit être active, et toutes les autres passives. Lorsque la lettre $E$ est présentée sur la rétine, la cinquième cellule de sortie doit être active, et toutes les autres passives, etc.

Les cellules de la première couche répondent en *oui/non*. La réponse *oui* correspond à une valeur 1 en sortie pour le neurone. La réponse *non* correspond à une valeur 0 en sortie pour le neurone. Les cellules d'entrée sont reliées aux cellules de sortie grâce à des synapses d'intensité variable. L'apprentissage pour le perceptron s'effectue en modifiant l'intensité de ces synapses (la règle d'apprentissage est détaillée plus loin). Les cellules de sortie évaluent l'intensité de la stimulation en provenance des cellules

de la rétine en effectuant la somme des intensités des cellules actives. Avec une formule:

$$a_j = \sum_i^I x_i w_{i,j} \ . \tag{1}$$

Avec:

- $a_j$: activation de la $j^{\text{ème}}$ cellule de sortie.
- $x_i$: valeur de sortie (0 ou 1) de la $i^{\text{ème}}$ cellule de la rétine.
- $w_{i,j}$: intensité de la connexion entre la $i^{\text{ème}}$ cellule d'entrée et la $j^{\text{ème}}$ cellule de sortie.

Les cellules de sortie deviennent actives si leur degré d'activation dépasse un seuil fixé noté $\vartheta_j$. Le plus souvent, le seuil $\vartheta_j$ est fixé à zéro, mais sans que cela soit nécessaire. Avec une formule:

$$o_j = \begin{cases} 0 & \text{pour } a_j \leq \vartheta_j \\ 1 & \text{pour } a_j > \vartheta_j \end{cases} \tag{2}$$

(Le fait d'utiliser un seuil $\vartheta_j$ est équivalent à avoir une cellule d'entrée, notée généralement $x_0$, toujours active). Le terme de *biais de réponse* est parfois utilisé comme un synonyme de seuil.

Un des premiers perceptrons mis au point par Rosenblatt s'appelait Mark 1. Il comportait une rétine de $20 \times 20 = 400$ cellules. Les synapses étaient représentées par des séries de potentiomètres. Mark 1 occupait l'équivalent d'un appartement de deux pièces!

## 2.1. RÈGLE D'APPRENTISSAGE

Le principal problème pour le perceptron est d'arriver à trouver un ensemble de valeurs pour les synapses tel que les configurations d'entrée se traduisent par les réponses voulues.

On peut envisager plusieurs règles d'apprentissage, la plus connue porte plusieurs noms. On l'appelle règle de *Widrow-Hoff* (Cf. Widrow & Hoff, 1960; Sebestyen, 1962; Nilsson, 1965; Duda & Hart, 1973), règle *Delta* (Rumelhart & McClelland, 1986), ou simplement règle d'apprentissage du perceptron (Hecht-Nielsen, 1990). On préférera, ici, le terme de *règle d'apprentissage de Widrow-Hoff*. Pour apprendre, le perceptron doit savoir qu'il a commis une erreur, et il doit connaître la réponse qu'il aurait dû donner. De ce fait, on parle *d'apprentissage supervisé*.

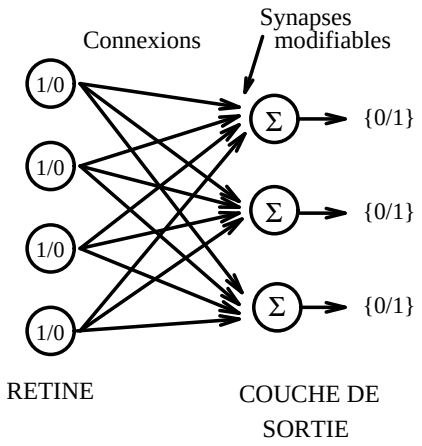


FIG. 1. — Un exemple de perceptron. Le perceptron est composé de deux couches de cellules. La première couche est appelée la *rétine*. Les synapses entre la première et la deuxième couche sont modifiables par apprentissage. Les cellules répondent en « 0/1 ».

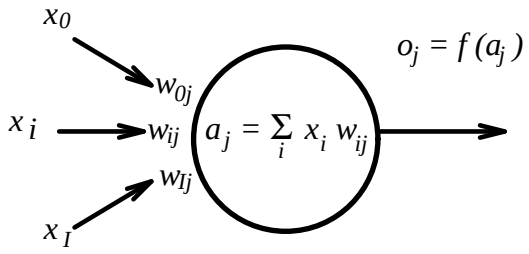


FIG. 2. — Une cellule de sortie d'un perceptron. Les entrées sont notées $x_i$, l'intensité des synapses est notée $w_{i,j}$, l'activation totale de la cellule est notée $a_j$, l'activation en sortie $o_j$ est égale à 0 si $a_j \leq 0$ et 1 si $a_j > 0$.

La règle d'apprentissage est locale dans le sens que chaque cellule de sortie apprend sans avoir besoin de connaître la réponse des autres cellules. Seule importe l'information qui lui est apportée par les cellules de la rétine. La règle en elle-même est très simple. Tout d'abord, une cellule ne modifie l'intensité de ses synapses (*i.e.*, « apprend ») que lorsqu'elle se trompe. Si la cellule de sortie est active quand elle devrait être inactive, alors elle diminue l'intensité des synapses correspondant à des cellules actives de

la rétine. Cela revient à diminuer l'intensité de la stimulation arrivant à la cellule de sortie. Si la cellule de sortie est inactive quand elle devrait être active, alors elle augmente l'intensité des synapses correspondant à des cellules actives de la rétine. Cela revient à augmenter l'intensité de la stimulation arrivant à la cellule de sortie.

On présente, donc, l'ensemble des stimuli à apprendre au perceptron (dans un ordre arbitraire), et on permet au perceptron d'apprendre. Si après une première présentation de l'ensemble des stimuli, le perceptron commet encore des erreurs, on présente de nouveau l'ensemble des stimuli (dans un nouvel ordre arbitraire) en permettant aux cellules du perceptron d'apprendre. La procédure se poursuit ainsi jusqu'à ce que le perceptron soit capable de donner toutes les réponses correctes.

De manière plus générale, la règle d'apprentissage de Widrow-Hoff s'écrit:

$$w_{i,j}^{(t+1)} = w_{i,j}^{(t)} + \eta(t_j - o_j)x_i = w_{i,j}^{(t)} + \Delta w_{i,j} \ . \qquad \textbf{(3)}$$

Avec:

- $\Delta w_{i,j}$: changement à effectuer pour la valeur $w_{i,j}$.
- $x_i$: valeur de sortie (0 ou 1) de la $i^{\text{ème}}$ cellule de la rétine.
- $o_j$: réponse de la $j^{\text{ème}}$ cellule de sortie (0 ou 1).
- $t_j$: réponse théorique (ou désirée) de la $j^{\text{ème}}$ cellule de sortie (0 ou 1).
- $w_{i,j}^{(t)}$: intensité de la connexion entre la $i^{\text{ème}}$ cellule d'entrée et la $j^{\text{ème}}$ cellule de sortie, au temps $t$ (les valeurs $w^{(0)}$ sont généralement choisies au hasard).
- $\eta$: une constante positive (généralement comprise entre 0 et 1). Le problème du choix d'une valeur pour $\eta$ est souvent délicat. Sa valeur influence, en effet, la vitesse d'apprentissage. Sa fonction est détaillée plus loin, dans le cas plus général des mémoires associatives linéaires. Dans un certain nombre de cas, $\eta$ peut varier en fonction de $t$. Dans ce cas, l'apprentissage débute avec une valeur élevée de $\eta$, puis la valeur de $\eta$ diminue avec chaque itération. (Si la notion de valeurs singulières d'une matrice rectangulaire vous est familière, on peut montrer que le choix de $\eta$ dépend uniquement des valeurs singulières de la matrice des stimuli, Cf. Abdi, sous presse).

## 2.2. UN EXEMPLE

Imaginons une version simplifiée de l'exemple décrit plus haut. Un perceptron avec une rétine composée de $5 \times 6 = 30$ cellules, doit reconnaître les chiffres de 0 à 9 lorsqu'ils sont présentés sur la rétine. Ce perceptron comporte 10 cellules de sortie. Il doit activer la cellule de sortie correspondant au chiffre affiché sur la rétine (*e.g.* , lorsque le chiffre 1 est présenté sur la rétine, la première cellule de sortie doit être active et toutes les autres inactives). Les chiffres se présentent comme des valeurs 0/1 sur la rétine. Ils sont « dessinés » comme suit (les valeurs 0 sont remplacées par des tirets pour rendre la figure plus lisible):

```
--1--   11111   11111  -1---   11111   11111  1111-   11111  11111   11111
--1--   ----1   ----1  -1---   1----   1----  ---1-   1---1  1---1   1---1
--1--   11111   11111  -1-1-   1----   1----  ---1-   11111  11111   1---1
--1--   1----   ----1  -1-1-   11111   11111  --111   1---1  ----1   1---1
--1--   1----   ----1  -1111   ----1   1---1  ---1-   1---1  ----1   1---1
--1--   11111   11111  ---1-   11111   11111  ---1-   11111  11111   11111
```

Les connexions entre les 30 cellules de la rétine (*i.e.*, les $6 \times 5$ cellules « dépliées ») et les 10 cellules de sortie peuvent se mettre dans une matrice (si le terme de matrice vous est inconnu, considérez qu'il fonctionne comme un synonyme de « tableau de données ») de 30 lignes et 10 colonnes. A l'intersection de la ligne $i$ et de la colonne $j$ se trouve $w_{i,j}$. Ces valeurs sont initialisées avec un générateur de nombres aléatoires. Voici un ensemble possible de valeurs de départ pour la matrice des valeurs $w_{i,j}$ (les valeurs dans la matrice ont été multipliées par 100 et arrondies pour rendre la lecture plus facile):

```
-28   -45    18    18    43   -12     2    33   -47   -45
  3    17   -49   -12   -43    -8    19     9    43    35
  3   -41    15    -8    20    41    26   -24   -45    24
-17    13    26    49   -13   -25    48    22    25    15
-43    13    38   -23    -6    27    -2   -26   -23   -14
-33    -1    40    41   -44    40     0     2   -18    49
 -1   -23   -41    45   -43     0   -12   -22    41     3
 -4    44   -45    26    27    33   -37   -48    19    37
 13    24    23    50    39   -27   -19   -15     1     9
 35    -9    34   -23    -8     4    -3   -21   -32   -35
  7    30   -47     3     0    46    25     5    39    12
 34   -34   -29    21   -37   -41   -23   -50    -9   -47
 21    44   -26   -32   -18    39    15   -35    18   -11
-11     0   -35     9    35     9    46     6   -35    48
 -9   -36     6   -25    -1    -4    46   -37   -30   -18
 13   -37    15    12    30   -25    -2   -11   -30   -47
 40    -7   -36    45    -9   -37    39   -41   -34   -43
-13   -25   -36    28    -4   -15    -5    31    43    15
-28    18    41   -25    36    -3     1    10    32    26
 -4    45    13    -6    32    19    20    49    45    35
-21     4     1   -40    -9     8    38    -6    23    37
 22    30    21    24   -48    39     2    -4   -43    21
 -1    17    18   -30    42    37    39     4   -36    -5
 49   -28    -5   -18     1    38    -6    -3    31   -13
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| -29 | 50 | -35 | 13 | 12 | -50 | -50 | 27 | 23 | -18 |
| -8 | 18 | 18 | -29 | 34 | 21 | 33 | -41 | -42 | 26 |
| 13 | -29 | -29 | -42 | -11 | 45 | 45 | -11 | -23 | 19 |
| -22 | 28 | 28 | -8 | -22 | -31 | -49 | -31 | 48 | -26 |
| 32 | -36 | -10 | 10 | -32 | 33 | -34 | 49 | -24 | -27 |
| -40 | -28 | 13 | 20 | 29 | 20 | 25 | 17 | 13 | -44 |

Avec cet ensemble de valeurs $w_{i,j}$, le perceptron n'est pas très efficace. Par exemple, lorsque le chiffre 1 est présenté sur la rétine les cellules de la rétine 3, 8, 13, 18, 23, et 28 sont actives. L'activation arrivant à la cellule 1 est donnée par la somme des connexions reliant cette cellule aux cellules actives de la rétine. C'est à dire:

$$a_1 = 3 - 4 + 21 - 13 - 1 - 22 = -16 \ .$$

Comme $a_1$ est négatif, la cellule de sortie 1 n'est pas active. En revanche, un calcul analogue montre que les cellules numéro 2, 5, 6, 9, et 10 sont actives (alors que seule la cellule numéro 1 devrait être active).

On peut décrire la performance du perceptron en notant dans une matrice les réponses des cellules pour chaque stimulus. Chaque colonne correspond aux réponses des 10 cellules du perceptron pour un stimulus donné. Par exemple, la première colonne donne les réponses du perceptron lorsque le chiffre 1 est présenté sur la rétine.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |

On peut considérer qu'une cellule commet une erreur lorsqu'elle est active alors qu'elle devrait être inactive, ou qu'elle est inactive alors qu'elle devrait être active. Dans l'exemple présent, on peut relever 51 de ces erreurs pour l'ensemble des stimuli (ce qui est en accord avec le fait que les valeurs des synapses sont aléatoires, on doit s'attendre à une valeur idéale, ou espérance mathématique, de 50 erreurs). On note en abrégeant que $e = 51$. On peut remarquer aussi que $e$ est une *somme des carrés d'erreur* (parce que $1^2 = 1$; en général, les réseaux cherchent une solution qui minimise une somme de carrés, comme dans le modèle linéaire classique en statistique).

Pour apprendre, le perceptron doit modifier l'intensité des synapses qui entraînent des erreurs. Par exemple, imaginons que le chiffre 1 soit

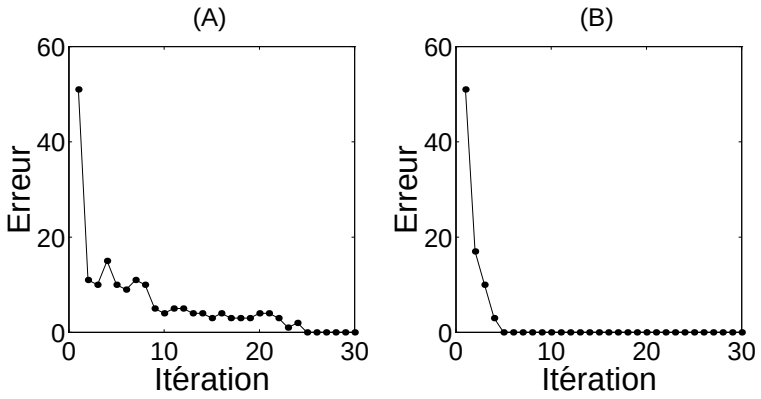


FIG. 3. — Décroissance de la somme des carrés d'erreur du perceptron en fonction du nombre d'itérations. (a) La constante d'apprentissage $\eta = .10$ est fixe. (b) La constante d'apprentissage $\eta$ est égale à .90 pour la première itération puis décroît en fonction du nombre d'itérations.

présenté sur la rétine lors de la phase d'apprentissage, et examinons la première cellule. Elle devrait être active mais ne l'est pas. Il faut donc modifier l'intensité des connexions entre la rétine et cette cellule de sortie. Comme la cellule est inactive et qu'elle devrait être active, il faut augmenter l'intensité des connexions qui la relient à des cellules actives de la rétine. Les cellules actives de la rétine sont les cellules numéro 3, 8, 13, 18, 23 et 28. L'intensité des connexions entre ces cellules actives de la rétine et la première cellule de sortie sont respectivement de .03, $-.04$, .21, $-.13$, $-.01$ et $-.22$. En utilisant la formule donnée plus haut, avec une valeur de $\eta = .10$, on peut trouver qu'il faut changer la connexion de la première cellule en:

$$\begin{aligned} w_{i,j}^{(t+1)} &= w_{i,j}^{(t)} + \eta(t_j - o_j)x_i = w_{i,j}^{(t)} + \Delta_t w_{i,j} \\ &= .03 + .10(1 - 0)1 = .03 + .10 \\ &= .13 \ . \end{aligned}$$

Les autres connexions deviennent .06, .31, $-.03$, .09, et $-.12$.

Après une itération (*i.e.,* après apprentissage de l'ensemble des stimuli), le nombre d'erreurs tombe à 11. Ainsi que le montre la figure 3.a., il faut néanmoins 25 itérations au perceptron pour apprendre parfaitement l'ensemble des stimuli. Si l'on décide de modifier la constante d'apprentissage en fonction de l'itération, on peut diminuer considérablement le temps

d'apprentissage. Par exemple, une simulation débutant avec une valeur de $\eta$ égale à .90 et décroissant $\eta$ proportionnellement au nombre d'itérations permet d'atteindre le résultat désiré en 5 itérations au lieu de 25. La décroissance du nombre d'erreurs est donnée dans la figure 3.b. La matrice de résultats devient maintenant (si vous êtes famillier avec le calcul matriciel, vous reconnaissez la matrice identité):

```
1    0    0    0    0    0    0    0    0    0
0    1    0    0    0    0    0    0    0    0
0    0    1    0    0    0    0    0    0    0
0    0    0    1    0    0    0    0    0    0
0    0    0    0    1    0    0    0    0    0
0    0    0    0    0    1    0    0    0    0
0    0    0    0    0    0    1    0    0    0
0    0    0    0    0    0    0    1    0    0
0    0    0    0    0    0    0    0    1    0
0    0    0    0    0    0    0    0    0    1
```

L'ensemble des connexions est maintenant donné dans la matrice suivante (toujours multipliée par 100 et arrondie à l'entier le plus proche):

```
-28   -35     8    -2    23   -32   -18    -7   -67   -55
-17    27   -59   -22   -63   -28    -1   -31    23    25
 13   -51     5   -28   -10     1     6   -64   -75     4
-17    23    16    29   -33   -45    28   -18     5     5
-43    23    38   -33     4    17   -22   -46   -23   -14
-33   -41   -50    31   -24    50   -10    42   112    49
-21   -23   -41    55   -43     0   -12   -22    41     3
  6    24   -45    26    17    13   -37   -48     9    27
 13    24    13    40     9   -37   -19   -35   -19    -1
 35     1    44   -23   -18   -46   -13   -11   -12     5
  7    40   -47    -7    10    36     5   -15    39    12
 14   -14    11    31   -47   -61   -33    10    31   -97
 31    44    14   -32   -38    -1     5    25    48   -71
-31    20    -5     9    -5   -21    36    46   -15   -12
 -9   -26    16   -25   -11   -54    36   -27   -10    22
 13     3   -45     2    50   -35   -22    79  -110   -37
 20    -7   -46    45    11     3    29   -71   -54   -83
 -3   -45   -56     8   -24    -5   -15   -19    -7   -45
-48    18    21   -35    26    27    -9   -40    -8   -24
 -4   -25    13   -26    12    19    10    39    25    25
-21    44   -49   -40   -49    28    28    94   -37    57
  2    30    21    34   -48    39     2    -4   -43    21
-11    -3    18   -20    32    17    39     4   -46   -15
 29   -28   -15   -18   -29    28    -6   -23    11   -23
-49   -20   -25    13    22   -40   -60    37    23   -18
 -8    28    18   -39    44    11    13   -61   -42    26
 13   -19   -29   -52    -1    35    25   -31   -23    19
-12    18    28   -18   -22   -61   -69   -51    38   -36
 12   -26   -10    10   -22    23   -54    29   -24   -27
-40   -18     3     0     9     0     5   -23    -7   -54
```

On peut montrer que lorsqu'une solution existe, le perceptron arrive à la trouver (Block, 1962; Minsky & Papert, 1969; Abdi, sous presse; voir aussi le paragraphe sur les mémoires associatives).

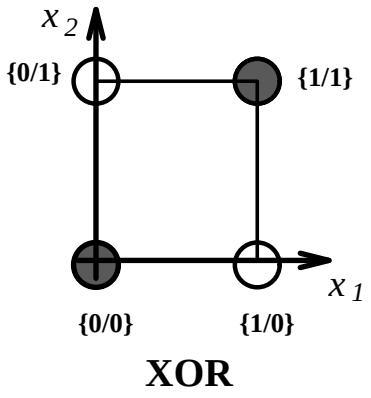


FIG. 4.A. — La fonction logique XOR n'est pas linéairement séparable.

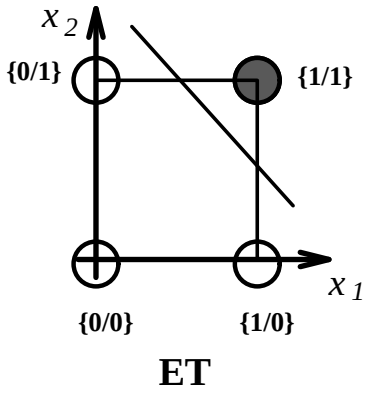


FIG. 4.B. — La fonction logique ET est linéairement séparable.

## 2.3. ÉVALUATION DU PERCEPTRON

L'exemple précédent le montre, le perceptron est capable d'apprendre à reconnaître des formes. Mais quelles sont exactement ses capacités? On peut montrer que le perceptron ne peut apprendre que dans le cas où les catégories à apprendre sont *linéairement séparables*. Le cas particulier le plus connu de catégories ne possédant pas la propriété d'être linéairement séparables est la fonction « XOR » (appelée aussi « OU exclusif »). La fonction

XOR est une fonction utilisée en logique. Elle revient à associer une valeur de sortie pour deux cellules d'entrée de la manière suivante:

$$\begin{array}{ccccc} 0 & 0 & \longmapsto & 0 \\ 1 & 0 & \longmapsto & 1 \\ 0 & 1 & \longmapsto & 1 \\ 1 & 1 & \longmapsto & 0 \end{array} \tag{4}$$

Imaginons un perceptron avec deux cellules d'entrée et une cellule de sortie qui doit apprendre cette fonction. Il possède deux poids $w_1$ et $w_2$ qui relient les cellules d'entrée à la cellule de sortie. Le fait d'associer les valeurs $(1, 0)$ d'entrée à 1 en sortie implique $w_1 > 0$. De même, Le fait d'associer les valeurs $(0, 1)$ d'entrée à 1 en sortie implique $w_2 > 0$. Or, pour donner la réponse 0 pour les valeurs $(1, 1)$ d'entrée, les valeurs de $w_1$ *et* $w_2$ doivent être inférieures ou égal à zéro; ce qui contredit les conditions précédentes. Autrement dit, dès lors que le perceptron a appris à associer la réponse 1 au stimulus $(1, 0)$ ou au stimulus $(0, 1)$, il donnera *toujours* la réponse 1 au stimulus $(1, 1)$. Il est donc impossible de trouver un ensemble de valeurs $w_i$ permettant à un perceptron d'apprendre la fonction XOR.

De manière générale, $k$ catégories d'objets dans un espace à $n$ dimensions (avec $k \leq n$) sont linéairement séparables si on peut trouver $(k - 1)$ hyperplans à $(n - 1)$ dimensions les séparant. Dans le cas particulier d'un perceptron à deux cellules d'entrée, et une cellule de sortie, deux catégories sont linéairement séparables si et seulement si on peut séparer les deux catégories par une droite. Par exemple, comme le montre la figure 4.a., la fonction XOR n'est pas linéairement séparable puisqu'il n'est pas possible de tracer une ligne telle que les sommets $[0, 0]$ et $[1, 1]$ soient d'un même côté de la ligne et les sommets $[1, 0]$ et $[0, 1]$ de l'autre. En revanche, la fonction ET, avec la table de vérité suivante:

$$\begin{array}{ccccc} 0 & 0 & \longmapsto & 0 \\ 1 & 0 & \longmapsto & 0 \\ 0 & 1 & \longmapsto & 0 \\ 1 & 1 & \longmapsto & 1 \end{array} \tag{5}$$

est linéairement séparable, puisque l'on peut tracer une droite qui sépare les objets devant avoir une valeur de sortie de 1 des objets devant avoir

une valeur de sortie de 0. Remarquons, enfin, que le perceptron implémente la méthode statistique de *l'analyse discriminante* pour le cas particulier où les valeurs d'entrées sont des valeurs binaires.

## 2.4. LINÉAIREMENT SÉPARABLE OU PAS ?

La notion de problème linéairement séparable ne mérite peut-être pas l'importance qui lui a été accordée. Une critique, jugée majeure naguère, addressée au perceptron était qu'il ne peut classifier des objets que lorsque les classes sont linéairement séparables. En particulier, comme on vient de le montrer ci-dessus, le perceptron ne peut apprendre la relation XOR.

En fait, le perceptron *peut* apprendre cette relation si elle est *proprement codée.* Il suffit, en effet, d'utiliser 3 cellules d'entrée au lieu de 2, et de présenter à la troisième cellule le produit des deux autres. Le problème revient, ainsi, à apprendre la relation (ternaire) suivante:

$$\begin{array}{ccccc} 0 & 0 & 0 & \longmapsto & 0 \\ 1 & 0 & 0 & \longmapsto & 1 \\ 0 & 1 & 0 & \longmapsto & 1 \\ 1 & 1 & 1 & \longmapsto & 0 \end{array} \tag{6}$$

(la troisième colonne s'obtient par multiplication des deux premières).

Une fois ce recodage effectué, un perceptron avec 3 cellules d'entrée et une cellule de sortie peut résoudre le problème posé. Par exemple, l'ensemble de poids suivant donne l'association correcte (avec un seuil $\vartheta = 0$):

$$w_1 = 1, \qquad w_2 = 1, \qquad w_3 = -2. \tag{7}$$

Cet exemple montre qu'un problème non-linéairement séparable peut, grâce à un recodage approprié, être transformé en un problème linéairement séparable. De ce fait, le problème important en pratique est celui du *codage* du problème à résoudre plus que le problème de l'architecture spécifique du réseau de neurones.

Une autre méthode pour résoudre des problèmes non-linéairement séparables est de créer des réseaux avec plusieurs couches. Rosenblatt avait déjà remarqué, en 1962, qu'un perceptron avec une couche supplémentaire interposée entre la rétine et la cellule de sortie pouvait résoudre le problème

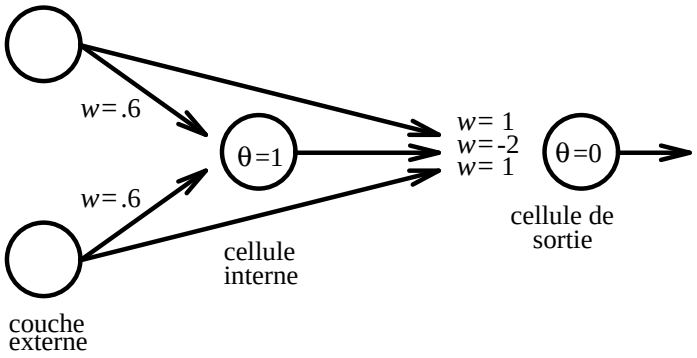


FIG. 5. — Un perceptron avec une cellule « cachée » peut résoudre le problème XOR (voir texte et table 1 pour explication).

XOR. Par exemple, un perceptron avec deux cellules d'entrée, une cellule cachée, une cellule de sortie, et des connexions comme indiquées ci-dessous arrive à implémenter la fonction XOR (on fixe $\vartheta$ égal à 0 pour la cellule de sortie; pour la cellule interne, $\vartheta$ est fixé à 1):

| **Activation des Cellules** | | | |
|---|---|---|---|
| entrée | Activation de la cellule interne ($\vartheta = 1$) | $o$ | Activation de la cellule de sortie ($\vartheta = 0$) |
| 0 0 | $(0 \times .6) + (0 \times .6) = 0.0$ | 0 | $(0 \times 1) + (0 \times 1) + (0 \times -2) = 0$ |
| 0 1 | $(0 \times .6) + (1 \times .6) = 0.6$ | 0 | $(0 \times 1) + (1 \times 1) + (0 \times -2) = 1$ |
| 1 0 | $(1 \times .6) + (0 \times .6) = 0.6$ | 0 | $(1 \times 1) + (0 \times 1) + (0 \times -2) = 1$ |
| 1 1 | $(1 \times .6) + (1 \times .6) = 1.2$ | 1 | $(1 \times 1) + (1 \times 1) + (1 \times -2) = 0$ |

($o$ est la réponse de la cellule interne après « seuillage »)

TABLEAU 1:

Détail des réponses des cellules du perceptron de la figure 5 pour la fonction XOR

Comme on peut le vérifier sur la figure 5 et dans le tableau 1 ci-dessous, la cellule interne (ou cellule cachée) effectue la multiplication (correspondant, donc, au ET logique) des deux cellules d'entrée. Puis, elle transmet le résultat de la multiplication à la cellule de sortie.

Ainsi que le remarquaient, Minsky & Papert (1969), le problème avec les cellules cachées est de trouver un moyen de leur faire modifier l'intensité de leurs connexions par apprentissage. En 1969, lorsque ces auteurs publièrent leur livre « *Perceptrons* », il n'y avait pas de procédure disponible. Ils en conclurent (trop rapidement !) qu'il n'en existait probablement pas. Comme d'habitude quand on prédit que quelque chose n'existe pas, d'autres font exprès de montrer qu'on a tort. Et donc, depuis, on a trouvé plusieurs procédures pour faire apprendre la couche interne en fonction de l'erreur du système. La plus célèbre est la *rétro-propagation* (pour l'anglais « *back-propagation* »), décrite plus loin, en fin de chapitre.

## 3. LES MÉMOIRES ASSOCIATIVES LINÉAIRES

Dans les paragraphes qui suivent, on utilise la notation matricielle qui permet une écriture compacte des formules. Une introduction à ces notions orientée vers les réseaux de neurones se trouve dans Jordan (1986), et dans Abdi (sous presse).

### 3.1. FORMALISATION MATRICIELLE GÉNÉRALE ET NOTATIONS

Les stimuli sont représentés par des vecteurs d'ordre $I \times 1$ notés $\mathbf{f}_k$ dont les composantes codent la valeur des descripteurs des stimulis, ou, de manière équivalente, indiquent la valeur d'entrée pour chaque unité. On admet, en général, que les vecteurs sont de norme unité. Les réponses du système sont représentées par des vecteurs d'ordre $J \times 1$ notés $\mathbf{g}_k$. L'ensemble des $K$ stimuli est représenté par une matrice $\mathbf{F}$ d'ordre $K \times I$ dans laquelle la $k$-ième ligne est $\mathbf{f}_k^T$. L'ensemble des $K$ réponses est représenté par une matrice $\mathbf{G}$ d'ordre $K \times J$ dans laquelle la $k$-ième ligne est $\mathbf{g}_k$. La matrice $J \times I$ de connexions est notée $\mathbf{W}$. La règle d'apprentissage (dite règle de *Hebb*, 1949) dans ces systèmes stipule que les connexions changent de manière proportionnelle au produit de l'entrée et de la sortie. Par exemple, si l'on considère l'association entre le $k$-ième stimulus et la $k$-ième réponse, la connexion $w_{i,j}$ entre la cellule $i$ d'entrée et la cellule $j$ de sortie sera proportionnelle à $f_{k,i} \times g_{k,j}$. Pour simplifier, on admet que la constante de proportionalité est 1. Pour associer le $k$-ième stimulus à la $k$-ième réponse, on construit la matrice:

$$\mathbf{W}_k = \mathbf{g}_k \mathbf{f}_k^T \qquad (8) \ .$$

Pour obtenir la réponse associée au $k$-ième stimulus, on post-multiplie la matrice $\mathbf{W}_k$ par $\mathbf{f}_k$, la réponse obtenue est notée $\hat{\mathbf{g}}_k$. On vérifie que $\hat{\mathbf{g}}_k$ égale $\mathbf{g}_k$:

$$\hat{\mathbf{g}}_k = \mathbf{W}_k\mathbf{f}_k = \mathbf{g}_k\mathbf{f}_k^T\mathbf{f}_k = \mathbf{g}_k \tag{9}$$

Pour associer plusieurs stimuli à plusieurs réponses, on somme les différentes matrices $\mathbf{W}_k$. On obtient, ainsi, la matrice $\mathbf{W}$:

$$\mathbf{W} = \sum \mathbf{W}_k = \sum \mathbf{g}_k\mathbf{f}_k^T = \mathbf{G}^T\mathbf{F} \tag{10}$$

La réponse associée au $k$-ième stimulus s'obtient par:

$$\hat{\mathbf{g}}_k = \mathbf{W}\mathbf{f}_k = \sum_{\ell=1}^{K} \mathbf{g}_\ell\mathbf{f}_\ell^T\mathbf{f}_k = \mathbf{g}_k + \sum_{\ell\neq k} \cos(\mathbf{f}_\ell, \mathbf{f}_k)\mathbf{g}_\ell \tag{11}$$

Note: lorsque $\mathbf{f}_k \perp \mathbf{f}_{k'}$ quels que soient $k \neq k'$ [*i.e.*, $\mathbf{f}_k^T\mathbf{f}_{k'} = 0$, et donc $\cos(\mathbf{f}_k, \mathbf{f}_{k'}) = 0$ alors $\hat{\mathbf{g}}_k = \mathbf{g}_k$. Afin d'estimer la qualité de l'association, il suffit de comparer la réponse obtenue ($\hat{\mathbf{g}}_k$) à la réponse attendue ($\mathbf{g}_k$), la mesure la plus populaire étant $\cos(\hat{\mathbf{g}}_k, \mathbf{g}_k)$.

### 3.1.1. Mémoires auto-associatives linéaires

Les *mémoires auto-associatives* constituent un cas particulier de ce modèle, lorsque stimuli et réponses constituent un seul ensemble (représenté par la matrice $\mathbf{F}$). La matrice $\mathbf{W}$ est alors égale à $\mathbf{F}^T\mathbf{F}$. Elle correspond à une matrice de « produits-croisés » comme celles que l'on rencontre dans les analyses de corrélation. Lorsque l'ensemble de stimuli n'est pas orthogonal, la mémoire sera sujette à des « *fausses reconnaissances* », ou, si l'on préfère, la mémoire reconstituera parfaitement des stimuli qui n'ont pas été stockés. Ces stimuli définis par l'équation:

$$\mathbf{W}\mathbf{u}_k = \lambda\mathbf{u}_k \quad \text{avec: } \mathbf{u}_k^T\mathbf{u}_k = 1 \tag{12}$$

sont les *vecteurs propres* de $\mathbf{W}$. Ils peuvent s'interpréter comme les *composantes principales* des stimuli (*i.e.*, l'analyse duale de celle pratiquée généralement, on rencontre parfois le terme de « *Q-analyse en composantes principales* », Cf. Kerlinger, 1986). Ils peuvent s'interpréter—dans un cadre plus psychologique—comme des « *prototypes* » ou des « *macro-caractéristiques* » (Cf. Anderson, Silverstein, Ritz, & Jones, 1977; Anderson & Mozer, 1981; Abdi, 1988; O'Toole & Abdi, 1989; O'Toole, Deffenbacher, Abdi & Bartlett, 1991).

### 3.1.2. Retour au cas général

Pour améliorer la performance de la mémoire (*i.e.,* pour augmenter le cosinus entre les $\mathbf{f}_k$ et les $\hat{\mathbf{f}_k}$) on peut utiliser différentes procédures d'apprentissage qui n'utilisent que l'information accessible localement par les connexions. La plus populaire d'entre elles, déjà décrite pour le perceptron, est celle de Widrow-Hoff (*alias* règle Delta, Cf. McClelland *et al.*, 1986). Cette procédure est itérative, elle corrige l'écart entre la réponse obtenue et la réponse attendue. Dans le cas général elle s'écrit:

$$\mathbf{W}_{(t+1)} = \mathbf{W}_{(t)} + \eta(\mathbf{g}_k - \mathbf{W}_{(t)}\mathbf{f}_k)\mathbf{f}_k^T \qquad (13)$$

avec $\mathbf{W}_{(t)}$: matrice $\mathbf{W}$ à l'étape $t$; $\eta$: une constante positive arbitraire; et $k$ étant choisi aléatoirement. En général, cette procédure peut demander un nombre considérable d'itérations pour converger vers un état stable. On peut montrer (Abdi, sous presse) que cette procédure d'apprentissage ne dépend que de la matrice des valeurs propres de $\mathbf{F}^T\mathbf{F}$, notée $\mathbf{\Lambda}$, des vecteurs propres de $\mathbf{F}^T\mathbf{F}$ (notés $\mathbf{U}$), des vecteurs propres de $\mathbf{F}\mathbf{F}^T$ (notés $\mathbf{V}$), de $\eta$ et de $t$. Précisément, $\mathbf{W}_{(t)}$ peut s'exprimer comme:

$$\mathbf{W}_{(t)} = \mathbf{G}\mathbf{U}\left\{\mathbf{\Lambda}^{-\frac{1}{2}}\left[\mathbf{I} - (\mathbf{I} - \eta\mathbf{\Lambda})^t\right]\right\}\mathbf{V}^T \ . \qquad (14)$$

On peut voir, d'après cette équation, que lorsque $\eta$ satisfait:

$$0 < \eta < 2\lambda_{max}^{-1} \qquad \text{(avec } \lambda_{max} \text{ étant la plus grande valeur propre),} \quad (15)$$

la procédure d'apprentissage converge vers (Cf. Kohonen, 1984):

$$\widetilde{\mathbf{W}} = \mathbf{G}\mathbf{F}^+ \quad \text{où: } \mathbf{F}^+ \text{ dénote la pseudo-inverse de } \mathbf{F}. \qquad (16)$$

Dans le cas d'une mémoire auto-associative, cette procédure converge vers:

$$\widetilde{\mathbf{W}} = \mathbf{F}\mathbf{F}^+ = \mathbf{U}\mathbf{U}^T \quad \text{avec: } \mathbf{U} \text{ matrice des vecteurs propres de } \mathbf{W}. \quad (17)$$

Ce qui revient à «*sphériciser*» la matrice $\mathbf{W}$ (*i.e.,* à imposer aux valeurs propres non-nulles d'être égales à 1). De ce fait, la notion de vecteur et valeur propres est centrale pour l'analyse de ces systèmes [on notera, au passage, que le seul chapitre de mathématiques dans l'ouvrage—déjà classique—en deux volumes de Rumelhart & McClelland, (1986), est consacré à une introduction à l'algèbre linéaire].

## *4. SYSTÈMES NON-LINÉAIRES*

Les mémoires associatives linéaires fournissent des vecteurs dont les réponses sont des vecteurs à valeurs dans $\mathbb{R}$. Anderson (Anderson *et al.*, 1977; Anderson & Murphy 1987), Hopfield (1982, 1984) étudient le comportement de mémoires (auto-associatives) lorsque les stimuli et les réponses sont des vecteurs à valeurs dans $\{0, 1\}$. Le modèle de Anderson est généralement connu sous le nom de modèle « *Brain State in a Box* » (BSB; Cf., aussi, Proulx, ce volume). Les deux formulations semblent différentes mais sont fondamentalement semblables. L'approche de Hopfield a stimulé l'intérêt de plusieurs théoriciens et son modèle est souvent relié aux concepts de la physique des solides qui traduisent son modèle en termes de *machines de Boltzman* ou de *champs aléatoires markoviens* (Markov Random Fields). Ces mémoires auto-associatives se comportent comme des mémoires auto-adressables. C'est à dire que le fait de ne donner qu'une partie d'un stimulus stocké entraîne comme réponse le stimulus entier (ou reconstitué). Cette propriété distingue les mémoires non-linéaires des mémoires linéaires. En effet, comme l'équation **(11)** le montre, une mémoire linéaire répond en fonction de la proximité (*i.e.,* du cosinus) de l'ensemble des stimuli stockés. A l'inverse, une mémoire non-linéaire répond au stimulus en fonction du *plus proche voisin* (au sens de la distance de la différence symétrique) de ce stimulus parmi les stimuli stockés.

### 4.1. STOCKAGE ET RÉCUPÉRATION DE L'INFORMATION

(Rappel: cet exemple concerne des mémoires auto-associatives pour lesquelles l'ensemble de stimuli est le même que l'ensemble de réponses). Le $k$-ième stimulus est représenté par un vecteur d'ordre $I \times 1$ noté $\mathbf{f}_k$, à valeurs dans $\{0, 1\}$, $\boldsymbol{\vartheta}$ est un $I \times 1$ vecteur à valeurs dans $\mathbb{R}$, avec $\vartheta_i$: seuil pour l'unité $i$. A partir des vecteurs $\mathbf{f}_k$, on définit des vecteurs « recodés » $\mathbf{h}_k$:

$$\mathbf{h}_k = (2\mathbf{f}_k - \mathbf{1}) \qquad (\textit{i.e.,} \text{ les valeurs 0 sont remplacées par } -1). \qquad \mathbf{(18)}$$

La matrice $\mathbf{W}$ s'obtient ici par:

$$\mathbf{W} = \sum_k \mathbf{h}_k \mathbf{h}_k^T \qquad \mathbf{(19)}$$

La récupération de l'information s'obtient par étapes:

**0.** Initialisation: mettre le paramètre $t = 1$ ($t$ donne le numéro de l'itération en cours). Mettre $\bar{\mathbf{f}}_k^{(0)}$ égal à $\mathbf{f}_k$.

**1.** Soit $\hat{\mathbf{f}}_k^{(t)} = \mathbf{W}\bar{\mathbf{f}}_k^{(t-1)}$ (*Nota Bene*, lorsque $t = 1$, cette étape correspond au rappel pour une mémoire auto-associative linéaire).

**2.** Soit

$$\bar{\mathbf{f}}_k^{(t)} = [\bar{f}_{i,k}^{(t)}] \quad \{i = 1 \ldots I\}, \quad \text{avec} \begin{cases} \bar{f}_{i,k}^{(t)} = 1 & \text{ssi } \hat{f}_{i,k}^{(t)} \geq \vartheta_i \\ \bar{f}_{i,k}^{(t)} = 0 & \text{ssi } \hat{f}_{i,k}^{(t)} < \vartheta_i \end{cases} \tag{20}$$

(cette procédure revient à mettre la valeur 1 pour les cellules qui ont une activation supérieure au seuil d'activation, et à mettre la valeur 0 pour les cellules qui ont une activation inférieure au seuil d'activation).

**3.** Comparer $\bar{\mathbf{f}}_k^{(t-1)}$ et $\bar{\mathbf{f}}_k^{(t)}$. Si $\bar{\mathbf{f}}_k^{(t-1)} \neq \bar{\mathbf{f}}_k^{(t)}$, changer $t$ en $t+1$ et itérer la procédure à partir de l'étape **1**. Si $\bar{\mathbf{f}}_k^{(t-1)} = \bar{\mathbf{f}}_k^{(t)}$ (ou $\bar{\mathbf{f}}_k^{(t-1)} \simeq \bar{\mathbf{f}}_k^{(t)}$, si on veut utiliser une approximation), alors on a trouvé une réponse stable du système, et on arrête le calcul.

Les réponses stables constituent les *attracteurs* de la mémoire. On peut montrer (Hopfield, 1982, 1984; Pour le modèle BSB d'Anderson, Cf. Golden, 1986) que cette procédure revient à rechercher le vecteur $\mathbf{s}$ à valeurs dans $\{0, 1\}$ minimisant une *énergie* notée $E$:

$$E = -\mathbf{s}\mathbf{W}\mathbf{s}^T + \boldsymbol{\vartheta}\mathbf{s}^T \tag{21}$$

en partant de $\mathbf{f}_k$ et en utilisant la technique de la *plus profonde descente* (lorsque cette fonction décroît à chaque étape, elle est appelée fonction de *Lyapounov* du système, Cf. Sánchez, 1968; Luenberger, 1979; Beltrami, 1987; Seydel, 1988; Perko, 1991).

Le minimum ainsi obtenu par la mémoire est un *minimum local*. Pour essayer d'augmenter la possibilité d'obtenir un *minimum global* à partir de ces mémoires, on peut utiliser une formulation légèrement différente en termes de *machines de Bolzmann*, ou, ce qui est équivalent (Cf., Moussouri, 1974; Geman & Geman, 1984; Ackley, 1988), en termes de *champs markovien aléatoires*. Ces mémoires reviennent à utiliser les techniques dites de « *recuit simulé* » (Cf. Kirpatrick, Gelatt & Vecchi, 1983; Siarry & Dreyfus, 1988; Aarts & Korst, 1989).

## 4.2. MACHINES DE BOLTZMANN

Les machines de Bolztmann correspondent à une variation sur le thème des mémoires de Hopfield. On esquisse ci-dessous les éléments essentiels. Une discussion détaillée se trouve dans Ackley *et al.* (1985), Hinton & Sejnowski (1986), un modèle similaire est décrit sous le nom de *théorie de l'Harmonie* par Smolensky (1986).

On définit la *différence d'énergie locale à l'unité $i$ pour la configuration* $\mathbf{s}$ par:

$$\Delta E_i = \mathbf{w}_i^T \mathbf{s} - \vartheta_j \qquad \text{(avec } \mathbf{w}_i\text{: } i\text{-ème colonne de } \mathbf{W}) \; . \tag{22}$$

Ce qui revient à calculer, pour un vecteur $\mathbf{s}$ donné, la différence d'énergie lorsque la $i$-ème unité passe de la valeur 0 à la valeur 1. La procédure de récupération peut s'appliquer localement dès l'étape 1. On décide alors de mettre $f_{i,k}^{(t)}$ égal à 1 avec la probabilité:

$$P_i = \frac{1}{1 + e^{-\Delta E_i / T}} \tag{23}$$

où $T$ est une valeur réelle correspondant à la « température » du sytème. Une conséquence de cet algorithme fournit une formule simple pour comparer la probabilité de deux états $\mathbf{s}$ et $\mathbf{s}'$, d'énergie $E_\mathbf{s}$ et $E_{\mathbf{s}'}$. Le rapport des probabilités est donné par la distribution de Gibbs ou de Bolzmann (suivant les préférences pour les appellations):

$$\frac{P_\mathbf{s}}{P_{\mathbf{s}'}} = \frac{e^{-E_\mathbf{s}/T}}{e^{-E_{\mathbf{s}'}/T}} = e^{-(E_\mathbf{s} - E_{\mathbf{s}'})/T} \; . \tag{24}$$

Intuitivement, la fonction de la température permet au système d'éviter les minima locaux. A hautes températures, le système peut facilement passer d'un état à l'autre (et éviter ainsi un minimum local), mais il peut aussi « rebondir » d'un état d'énergie faible à un état d'énergie supérieure (et « laisser passer » un minimum local « meilleur » qu'un autre). La stratégie (cousine du bricolage) est généralement de débuter la procédure à haute température, puis de la réduire progressivement. Cette manière de faire correspond en physique au refroidissement d'un verre ou d'un cristal où les atomes doivent se placer en fonction de l'état de leur voisins. Plus la température est élevée, plus facilement un atome peut changer sa configuration, lorsque la température décroît, les contraintes imposées par les atomes voisins agissent plus fortement sur l'état d'un

atome donné. On emploie souvent—par analogie avec la physique des particules—la notion de « *spin* » lorsque les atomes ne peuvent adopter que deux états: *haut* et *bas* (Cf. les modèles neuro-mimétiques de Little & Shaw, 1974). Les simulations informatiques indiquent que cette manière de faire donne des résultats satisfaisants. Les principales applications de ces techniques sont la reconstitution d'images (Geman & Geman, 1984), ou plus généralement, les problèmes avec de nombreuses contraintes locales à satisfaire simultanément (Cf. Kirpatrick *et al.*, 1983), comme par exemple la vision stéréoscopique (Cf. Schyns, ce volume), ou la reconnaissance de formes (Cf., entre autres, Pao, 1989; Fukunaga; 1990; Carpenter & Grossberg, 1991; Quinlan, 1991; Schalkoff, 1991).

## 5. ARCHITECTURES À COUCHES CACHÉES: RÉTRO-PROPAGATION

A l'origine, un des problèmes du perceptron à plusieurs couches était de ne pouvoir apprendre à résoudre des problèmes non-linéairement séparables, comme, par exemple, la relation binaire XOR. Depuis, plusieurs procédures ont été trouvées permettant aux cellules de la couche interne de modifer leurs connexions en fonction de l'erreur du système. La procédure la plus célèbre est connue sous le nom de *rétro-propagation* (pour l'anglais « *back-propagation* »), ou de *règle de Widrow-Hoff généralisée* ou de *règle Delta*. Elle a été, tout d'abord, trouvée en 1969 par Bryson & Ho, puis (indépendamment) en 1974 par Werbos, mais est restée largement ignorée. Elle a été redécouverte vers la fin des années quatre-vingts par plusieurs chercheurs (Le Cun, 1986; Parker, 1985; Rumelhart, Hinton, & Williams, 1986).

Un réseau utilisant la technique de rétro-propagation de l'erreur comporte une couche d'entrée, une couche de sortie, et au moins une couche (dite *cachée*) intermédiaire entre la couche d'entrée et la couche de sortie. Pour faciliter l'exposé considérons un réseau avec une seule couche cachée (s'il y a plusieurs couches cachées, il suffit de remplacer dans ce qui suit le terme *couche de sortie* par le terme *couche cachée suivante*). Le diagramme d'un réseau avec une couche cachée est donné dans la figure 6 (avec les notations détaillées ci-dessous).

L'idée essentielle de la rétro-propagation est remarquablement simple. Les cellules de la couche de sortie calculent l'erreur entre la réponse donnée et la réponse désirée et ajustent l'intensité des connexions comme pour une mémoire auto-associative normale (mais, non-linéaire), en utilisant la règle de Widrow-Hoff. Ensuite, l'erreur est propagée *en sens inverse* de la

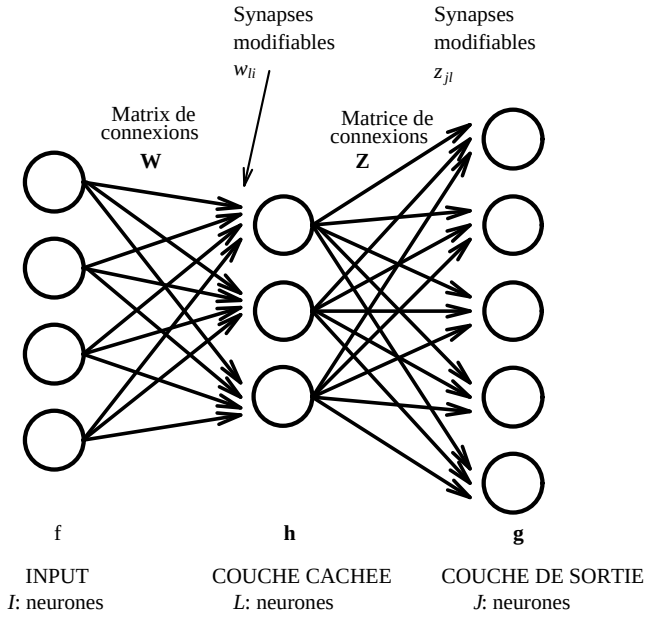


FIG. 6. — Architecture d'un réseau de neurones avec une couche cachée. La couche d'entrée est composée de $I$ cellules. La couche cachée est composée de $L$ cellules, La couche de sortie est composée de $J$ cellules. Les connexions $w_{\ell,i}$ de la couche d'entrée à la couche cachée sont stockées dans la matrice $\mathbf{W}$. Les connexions $z_{j,\ell}$ de la couche cachée à la couche de sortie sont stockées dans la matrice $\mathbf{Z}$. Un stimulus en entrée est noté $\mathbf{f}$, la réponse de la couche cachée est notée $\mathbf{h}$, la réponse de la couche de sortie est notée $\hat{\mathbf{g}}$, la réponse désirée de la couche de sortie est notée $\mathbf{g}$.

couche de sortie à la couche cachée (en utilisant les connexions reliant les cellules de la couche cachée à la couche interne). Puis, les cellules de la couche cachée modifient à leur tour l'intensité des connexions les reliant aux cellules de la couche d'entrée de manière à réduire l'erreur.

De manière plus technique, soit un réseau de neurones avec une couche d'entrée, une couche cachée et une couche de sortie. On note:

- $\mathbf{f}_k$: le vecteur à $I$ éléments représentant le $k$-ième stimulus (*i.e.,* la couche d'entrée comporte $I$ cellules).
- $\mathbf{h}_k$: le vecteur à $L$ éléments représentant la réponse des $L$ cellules de la couche cachée lorsque le $k$-ième stimulus est présenté en entrée (*i.e.,* la couche cachée comporte $L$ cellules).

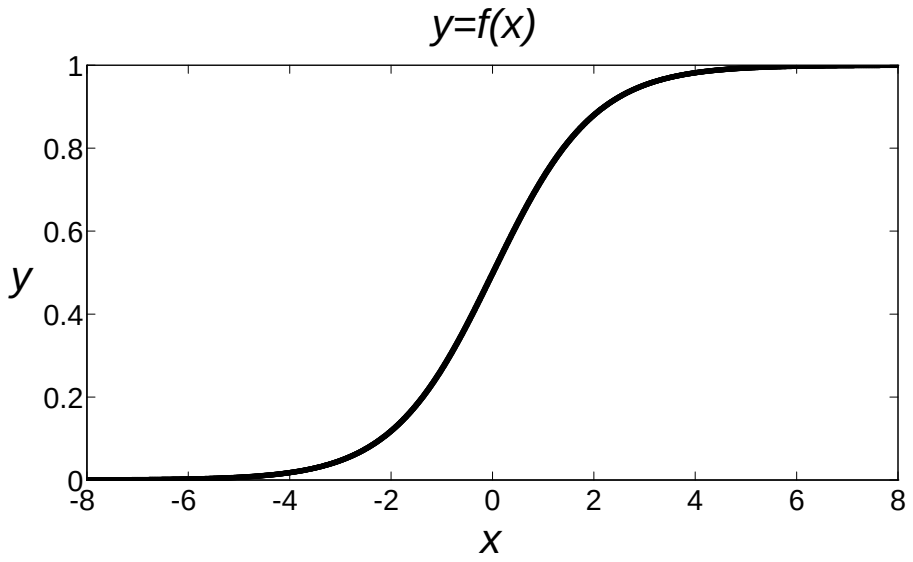


FIG. 7. — Graphe de la fonction logistique $y = f(x)$.

- $\hat{\mathbf{g}}_k$: le vecteur à $J$ éléments représentant la réponse des cellules de la couche de sortie pour le $k$-ième stimulus (*i.e.,* la couche de sortie comporte $J$ cellules).
- $\mathbf{g}_k$: le vecteur à $J$ éléments représentant la réponse désirée (ou théorique) des cellules de la couche de sortie pour le $k$-ième stimulus.
- $\mathbf{W}$: la matrice d'ordre $L \times I$ des valeurs des connexions reliant les cellules de la couche d'entrée aux cellules de la couche cachée; $w_{\ell,i}$ donne la valeur de la connexion entre la $i$-ème cellule d'entrée et la $\ell$-ième cellule de la couche cachée.
- $\mathbf{Z}$: la matrice d'ordre $J \times L$ des valeurs des connexions reliant les cellules de la couche d'entrée aux cellules de la couche cachée; $z_{j,\ell}$ donne la valeur de la connexion entre la $\ell$-ième cellule de la couche cachée et la $j$-ième cellule de sortie.

Pour utiliser la technique de rétro-propagation, la réponse des cellules doit être une fonction non-linéaire de l'état d'activation de la cellule. En notant $a_n$ l'état d'activation de la cellule $n$, la réponse de la cellule (notée $o_n$) sera:

$$o_n = f(a_n) \ . \tag{25}$$

Il existe plusieurs fonctions $f$ satisfaisantes, une des plus fréquentes (et

des plus pratiques) est la fonction *logistique* (Cf. équation **23**):

$$f(x) = \frac{1}{1 + e^{-x}} \ . \tag{26}$$

Comme le montre la figure 7, elle possède la propriété intéressante de rester dans le domaine [0 1]. En outre, sa dérivée (utilisée pour l'apprentissage) est simple à calculer:

$$f'(x) = -\frac{e^{-x}}{(1 + e^{-x})^2} = f(x)[1 - f(x)] \ . \tag{27}$$

Pour répondre à un stimulus, le signal est propagé de la couche d'entrée à la couche de sortie en passant par la couche cachée. A chaque étape, la réponse des cellules s'obtient par la fonction logistique (on peut, évidemment concevoir des réseaux avec des fonctions de réponse différentes pour chaque couche, mais l'analyse en sera rendue plus difficile). Ainsi lorsque le $k$-ième stimulus est présenté en entrée, le vecteur de réponse des cellules de la couche cachée est donné par:

$$\mathbf{h}_k = f(\mathbf{W}\mathbf{f}_k) \ . \tag{28}$$

Puis, la réponse de la cellule de sortie est donnée par le vecteur:

$$\hat{\mathbf{g}}_k = f(\mathbf{Z}\mathbf{h}_k) \ . \tag{29}$$

La technique de rétro-propagation, est une technique d'apprentissage supervisée (*i.e.*, pour apprendre, le réseau doit connaître la réponse qu'il aurait dû donner). Elle modifie l'intensité des connexions de manière à diminuer l'intensité de l'erreur commise par la cellule pour la réponse considérée. La procédure de prise en compte de l'erreur est la même pour toutes les couches, mais l'estimation du signal d'erreur diffère suivant les couches.

Pour les cellules de la couche de sortie, l'erreur est évaluée en comparant la réponse donnée par la cellule avec la réponse théorique. Le vecteur d'erreur pour le $k$-ième stimulus vaut donc

$$\mathbf{e}_k = (\mathbf{g}_k - \hat{\mathbf{g}}_k) \ . \tag{30}$$

Le *signal d'erreur* prend en compte l'erreur commise par la cellule et l'état d'activation de la cellule. Pour la couche de sortie, il est défini comme

$$\boldsymbol{\delta}_{\text{sortie, } k} = f'(\mathbf{Z}\mathbf{h}_k) \circledast (\mathbf{e}_k) = \hat{\mathbf{g}}_k \circledast (\mathbf{1} - \hat{\mathbf{g}}_k) \circledast (\mathbf{g}_k - \hat{\mathbf{g}}_k) \ . \tag{31}$$

avec $\circledast$ indiquant le produit terme à terme des vecteurs (ou produit de Hadamar, Cf. Searle, 1982; Horn & Johnson, 1985), et $\mathbf{1}$ un vecteur unité. La procédure d'apprentissage généralise la procédure déjà vue dans le cas linéaire, la matrice de connexions $\mathbf{Z}$ est corrigée par itérations. A l'étape $t+1$, $\mathbf{Z}$ devient:

$$\mathbf{Z}_{(t+1)} = \mathbf{Z}_{(t)} + \eta \boldsymbol{\delta}_{\text{sortie, } k} \mathbf{h}_k^T = \mathbf{Z}_{(t)} + \Delta_t \mathbf{Z} \ . \tag{32}$$

($k$ étant choisi aléatoirement, et $\eta$ étant un nombre réèl positif, Cf. paragraphe sur la règle de Widrow-Hoff pour le perceptron et pour les mémoires associatives linéaires).

Pour les cellules de la couche cachée, le signal d'erreur ne peut être évalué par comparaison avec une valeur idéale. Il est estimé comme une fonction du signal d'erreur en provenance de la couche de sortie et de l'activation des cellules de la couche cachée. Précisément, le vecteur donnant le signal d'erreur pour les cellules de la couche cachée s'obtient comme:

$$\boldsymbol{\delta}_{\text{cachée},k} = f'(\mathbf{W}\mathbf{f}_k) \circledast (\mathbf{Z}^T \boldsymbol{\delta}_{\text{sortie, } k}) = \mathbf{h}_k \circledast (\mathbf{1} - \mathbf{h}_k) \circledast (\mathbf{Z}^T \boldsymbol{\delta}_{\text{sortie, } k}) \ . \tag{33}$$

Comme on le voit, le signal d'erreur s'obtient en propageant l'erreur de la couche de sortie à la couche cachée (par l'opération $\mathbf{Z}^T \boldsymbol{\delta}_{\text{sortie, } k}$) ce qui correspond au sens inverse de la propagation du signal lorsque la mémoire donne une réponse en réponse à une stimulation. L'apprentissage pour la couche cachée se déroule, ensuite, de manière similaire à celui de la couche de sortie. La matrice de connexions $\mathbf{W}$ est corrigée par itérations. A l'étape $t+1$, $\mathbf{W}$ devient:

$$\mathbf{W}_{(t+1)} = \mathbf{W}_{(t)} + \eta \boldsymbol{\delta}_{\text{cachée, } k} \mathbf{f}_k^T = \mathbf{W}_{(t)} + \Delta_t \mathbf{W} \ . \tag{34}$$

Cette procédure minimise le carré de l'erreur à chaque étape (lorsque $\eta$ est convenablement choisi). Elle converge vers un minimum local. On montre dans le paragraphe suivant que la rétro-propagation correspond à la procédure, classique en optimisation, de recherche de minima d'une fonction par la technique du gradient.

## 5.1. RÉTRO-PROPAGATION ET TECHNIQUE DU GRADIENT

La méthode du gradient (ou methode de « la plus profonde descente ») est une technique relativement classique en optimisation linéaire et non-linéaire (Cf. Pierre, 1969; Ciarlet, 1986; Strang; 1986; Fletcher, 1987).

### 5.1.1. Rappel: la méthode du gradient

L'idée générale de cette technique consiste à chercher le minimum d'une fonction de plusieurs variables ou paramètres par itérations successives. Le gradient d'une fonction matricielle est défini comme la matrice des dérivées partielles de cette fonction. En supposant que les valeurs des paramètres sont stockées dans une matrice $\mathbf{Z}$, et que la fonction à minimiser est $y = g(\mathbf{Z})$. L'algorithme se déroule comme suit:

- **1.** Choisir arbitrairement les valeurs $\mathbf{Z}_{(t)}$ pour $t = 0$ (de manière aléatoire, en règle générale).
- **2.** Calculer le gradient de $g$ (noté $\nabla g$):

$$\frac{\partial g(\mathbf{Z}_{(t)})}{\partial \mathbf{Z}_{(t)}} = \nabla g \ . \tag{35}$$

- **3.** Corriger $\mathbf{Z}_{(t)}$ en direction inverse du gradient de $\mathbf{Z}_{(t)}$ (avec $\eta$ dénotant la constante de proportionnalité):

$$\mathbf{Z}_{(t+1)} = \mathbf{Z}_{(t)} + \Delta_t \mathbf{Z} = \mathbf{Z}_{(t)} - \eta \nabla g = \mathbf{Z}_{(t)} - \eta \frac{\partial g(\mathbf{Z}_{(t)})}{\partial \mathbf{Z}_{(t)}} \ . \tag{36}$$

- **4.** Continuer les étapes **2.** et **3.** tant que l'écart entre $\mathbf{Z}_{(t)}$ et $\mathbf{Z}_{(t+1)}$ est jugé important.

Pour un réseau utilisant la rétro-propagation, la fonction à minimiser est une fonction quadratique de l'erreur. La fonction d'erreur pour la $k$-ième réponse est définie comme:

$$E_k = \tfrac{1}{2}(\mathbf{g}_k - \hat{\mathbf{g}}_k)^T(\mathbf{g}_k - \hat{\mathbf{g}}_k) = \tfrac{1}{2}(\mathbf{g}_k^T\mathbf{g}_k + \hat{\mathbf{g}}_k^T\hat{\mathbf{g}}_k - 2\hat{\mathbf{g}}_k^T\mathbf{g}_k) \ . \tag{37}$$

### 5.1.2. Correction pour la couche de sortie

Pour la couche de sortie, on cherche à modifier les valeurs de $\mathbf{Z}$. Le gradient de $E_k$ par rapport à $\mathbf{Z}$ se calcule en utilisant la règle de dérivation des fonctions composées (ré-écrite en notation matricielle, Cf. Magnus & Neudecker, 1988; Mardsen & Tromba, 1988):

$$\nabla_{\mathbf{Z}} E_k = \frac{\partial E_k}{\partial \mathbf{Z}} = \frac{\partial E_k}{\partial \hat{\mathbf{g}}_k} \frac{\partial \hat{\mathbf{g}}_k}{\partial \mathbf{Z}\mathbf{h}_k} \frac{\partial \mathbf{Z}\mathbf{h}_k}{\partial \mathbf{Z}} \ . \tag{38}$$

En évaluant chacun des termes on trouve

$$\frac{\partial E_k}{\partial \hat{\mathbf{g}}_k} = -(\mathbf{g}_k - \hat{\mathbf{g}}_k)^T \ , \tag{39}$$

et [avec $f$ étant la fonction logistique, et $\hat{\mathbf{g}}_k = f(\mathbf{Z}\mathbf{h}_k)$]:

$$\frac{\partial \hat{\mathbf{g}}_k}{\partial \mathbf{Z}\mathbf{h}_k} = \hat{\mathbf{g}}_k^T \circledast (\mathbf{1} - \hat{\mathbf{g}}_k)^T \ , \tag{40}$$

Puis:

$$\frac{\partial \mathbf{Z}\mathbf{h}_k}{\partial \mathbf{Z}} = \mathbf{h}_k \tag{41}$$

La correction à apporter à $\mathbf{Z}$ à l'étape $t$ est donc proportionnelle à:

$$-\nabla_{\mathbf{Z}} E_k = (\mathbf{g}_k - \hat{\mathbf{g}}_k)^T \circledast \hat{\mathbf{g}}_k^T \circledast (\mathbf{1} - \hat{\mathbf{g}}_k)^T \mathbf{h}_k = \boldsymbol{\delta}_{\text{sortie},\ k}^T \mathbf{h}_k \ . \tag{42}$$

En utilisant $\eta$ comme constante de proportionalité et en transposant on obtient:

$$\Delta_t \mathbf{Z} = \eta \boldsymbol{\delta}_{\text{sortie},\ k} \mathbf{h}_k^T \tag{43}$$

comme indiqué dans l'équation **32**.

### 5.1.3. Correction pour la couche cachée

Pour la couche cachée, on cherche à modifier les valeurs de $\mathbf{W}$. Le gradient de $E_k$ par rapport à $\mathbf{W}$ se calcule en utilisant la règle de dérivation des fonctions composées:

$$\nabla_{\mathbf{W}} E_k = \frac{\partial E_k}{\partial \mathbf{W}} = \frac{\partial E_k}{\partial \hat{\mathbf{g}}_k} \frac{\partial \hat{\mathbf{g}}_k}{\partial \mathbf{Z}\mathbf{h}_k} \frac{\partial \mathbf{Z}\mathbf{h}_k}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}\mathbf{f}_k} \frac{\partial \mathbf{W}\mathbf{f}_k}{\partial \mathbf{W}_k} \ . \tag{44}$$

Les deux premiers termes sont déjà connus (Cf. équations **39** et **40**), ils correspondent à $-\boldsymbol{\delta}^T_{\text{sortie},\,k}$. En évaluant chacun des autres termes on trouve:

$$\frac{\partial \mathbf{Z}\mathbf{h}_k}{\partial \mathbf{h}_k} = \mathbf{Z}^T \ , \tag{45}$$

et [avec $f$ étant la fonction logistique, et avec $\mathbf{h}_k = f(\mathbf{W}\mathbf{f}_k)$]:

$$\frac{\partial \mathbf{h}_k}{\partial \mathbf{W}\mathbf{f}_k} = \mathbf{h}_k^T \circledast (\mathbf{1} - \mathbf{h}_k)^T \ , \tag{46}$$

puis, finalement:

$$\frac{\partial \mathbf{W}\mathbf{f}_k}{\partial \mathbf{W}_k} = \mathbf{f}_k \ . \tag{47}$$

La correction à apporter à $\mathbf{W}$ à l'étape $t$ est donc proportionnelle à:

$$-\nabla_{\mathbf{W}} E_k = \boldsymbol{\delta}_{\text{sortie},\,k} \mathbf{Z}^T \circledast \mathbf{h}_k^T \circledast (\mathbf{1} - \mathbf{h}_k)^T \mathbf{f}_k = \boldsymbol{\delta}^T_{\text{cachée},\,k} \mathbf{f}_k \ . \tag{48}$$

En utilisant $\eta$ comme constante de proportionnalité et en transposant on obtient:

$$\Delta_t \mathbf{W} = \eta \boldsymbol{\delta}_{\text{cachée},\,k} \mathbf{f}_k^T \tag{49}$$

comme indiqué dans l'équation **34**.

### 5.1.4. Méthode du gradient et mémoires linéaires

La règle d'apprentissage de Widrow-Hoff utilisée pour les mémoires linéaires implémente, elle aussi, la méthode du gradient, et, ainsi, la méthode de rétro-propagation représente bien une généralisation du perceptron. En utilisant les notations du paragraphe précédent, la fonction $f$ qui donne la réponse de la cellule en fonction de son état d'activation prend une forme particulièrement simple (Cf. équation **25**):

$$o_n = f(a_n) = a_n \ . \tag{50}$$

La réponse de la mémoire pour le stimulus s'obtient par l'équation **11**, rappelée ci-dessous:

$$\hat{\mathbf{g}}_k = \mathbf{W}\mathbf{f}_k \ .$$

La fonction d'erreur pour la $k$-ième réponse reste la même que celle définie par l'équation **37**, rappelée ci-dessous:

$$E_k = \tfrac{1}{2}(\mathbf{g}_k - \hat{\mathbf{g}}_k)^T(\mathbf{g}_k - \hat{\mathbf{g}}_k) = \tfrac{1}{2}(\mathbf{g}_k^T\mathbf{g}_k + \hat{\mathbf{g}}_k^T\hat{\mathbf{g}}_k - 2\hat{\mathbf{g}}_k^T\mathbf{g}_k) \ .$$

On veut modifier les valeurs de $\mathbf{W}$ de manière à minimiser l'erreur. Le gradient de $E_k$ par rapport à $\mathbf{W}$ se calcule en utilisant la règle de dérivation des fonctions composées:

$$\nabla_{\mathbf{W}} E_k = \frac{\partial E_k}{\partial \mathbf{W}} = \frac{\partial E_k}{\partial \hat{\mathbf{g}}_k}\frac{\partial \hat{\mathbf{g}}_k}{\partial \mathbf{W}} \ . \tag{51}$$

Comme précédemment (Cf. équation **39**), on trouve pour le premier terme à droite de l'équation:

$$\frac{\partial E_k}{\partial \hat{\mathbf{g}}_k} = -(\mathbf{g}_k - \hat{\mathbf{g}}_k)^T \ ,$$

et [avec $\hat{\mathbf{g}}_k = f(\mathbf{W}\mathbf{f}_k) = \mathbf{W}\mathbf{f}_k$], le second terme devient:

$$\frac{\partial \hat{\mathbf{g}}_k}{\partial \mathbf{W}} = \frac{\partial \mathbf{W}\mathbf{f}_k}{\partial \mathbf{W}} = \mathbf{f}_k \ . \tag{52}$$

La correction à apporter à $\mathbf{W}$ à l'étape $t$ est donc proportionnelle à:

$$-\nabla_{\mathbf{W}} E_k = -\frac{\partial E_k}{\partial \mathbf{W}} = (\mathbf{g}_k - \hat{\mathbf{g}}_k)^T\mathbf{f}_k \ . \tag{53}$$

En utilisant $\eta$ comme constante de proportionnalité et en transposant on obtient:

$$\Delta_t \mathbf{W} = \eta(\mathbf{g}_k - \hat{\mathbf{g}}_k)\mathbf{f}_k = \eta(\mathbf{g}_k - \mathbf{W}_{(t)}\mathbf{f}_k)\mathbf{f}_k^T \qquad (54)$$

comme indiqué dans l'équation **13**. Ainsi, la règle d'apprentissage du perceptron est un cas particulier de la rétro-propagation (sans couche cachée! et avec une réponse linéaire).

### 5.1.5. Evaluation de la rétro-propagation

La rétro-propagation a probablement été une des causes récentes du regain d'intérêt pour les réseaux de neurones. Une des principales qualités de ces réseaux est de pouvoir implémenter des classifications de problèmes non-linéaires. En outre, les couches cachées peuvent parfois s'interpréter comme développant une représentation du problème posé, et cette représentation en elle-même peut-être objet d'étude.

La rétro-propagation est probablement à la base du plus grand nombre des applications récentes des réseaux de neurones (Cf. Maren, Harston & Papp, 1990, p.93), comme, par exemple, la reconnaissance de phonèmes, la lecture de textes par ordinateur, la reconnaissance de chiffres écrits, etc.

Néanmoins, la rétro-propagation ne saurait prétendre être une panacée. D'une part, il est difficile de trouver un équivalent neuronal de cette technique. En outre, sa mise en œuvre exige souvent des temps d'apprentissage très long qui rende son application prohibitive pour de nombreux problèmes de taille raisonnable. Enfin, cette technique ne peut s'appliquer que pour des problèmes de classification où les bonnes réponses sont connues (*i.e.*, elle ne s'applique que pour des apprentissages supervisés).

## 6. CONCLUSION

Ce chapitre présente les outils de base de la «boîte à outil» de la modélisation connexioniste. Bien évidemment, je n'ai fait dans ces quelques pages qu'effleurer le sujet. Ce volume permet de compléter cette introduction par de nombreux exemples d'application et par d'intéressants développement théoriques.

En fait, il devient de plus en plus délicat de prétendre couvrir un domaine aussi vaste et en rapide évolution comme la modélisation connexioniste (Cf., néanmoins, pour quelques introductions récentes parmi beaucoup d'autres, Aleksander & Morton, 1990; Hecht-Nielsen, 1990; Kampf & Hasler, 1990; Khanna, 1990; Müller & Reinhardt, 1990; Perez, 1990; Simpson, 1990; Zeidenberg, 1990; Anderson *et al.*, 1991; Bechtel & Abrahmson; Freeman & Skapura, 1991; Hertz, Krogh & Palmer; 1991; Levine, 1991; Quilian, 1991; Abdi, sous presse). Ceci est d'autant plus vrai, que cette approche s'étend de la psychologie à la physique des systèmes complexes (e.g., Fogelman-Soulié, 1985; Serra & Zanarini, 1987; Goles & Martinez, 1990; Weisbuch, 1991), en passant par les sciences de l'ingénieur et le traitement du signal (e.g., Widrow & Stearns, 1985; Catlin, 1989; Souček, 1989; Garner, 1990; Kosko, 1991, 1992), sans oublier les neuro-sciences (e.g., Mac Gregor, 1987; Amit, 1989). Il reste à voir si la communauté psychologique sera capable d'intégrer et de maîtriser ces nouveaux outils conceptuels qu'elle a contribué à créer et de les utiliser pour poser des questions pertinentes de psychologie et pour proposer des solutions.

## BIBLIOGRAPHIE

Aarts, E., & Korst, J. (1989). *Simulated annealing and Boltzmann machines.* New York: Wiley.

Abdi, H. (1986). La mémoire sémantique, une fille de l'intelligence artificielle et de la psychologie. In C. Bonnet, J.M. Hoc & G. Tiberghien (Eds.), *Psychologie et intelligence artificielle.* Bruxelle: Mardaga.

Abdi, H. (1987). Do we really need a contingency model for concept abstraction? *British Journal of Psychology*, **78**, 113–125.

Abdi, H. (1988). Generalized approaches for connectionist auto-associative memories: Interpretation, implication and illustration for face processing. In J. Demongeot (Ed.), *Artificial intelligence and cognitive sciences.* Manchester: Manchester University Press.

Abdi, H. (In press). *Les Réseaux de neurones.* Grenoble: Presses Universitaires de Grenoble.

Ackley, D.H. (1987). *A connectionist machine for genetic Hillclimbing.* Norwell (MA): Kluwer.

Ackley, D.H., Hinton, G.E., & Sejnowski, T.J. (1985). A learning algorithm for Boltzman machines. *Cognitive Science*, **9**, 147–169.

Aleksander, I., & Morton, H. (1990). *An introduction to neural computing.* London: Chapman & Hall.

Amit, D.J. (1989). *Modelling brain function.* Cambridge: C.U.P.

Anderson, J.A., & Murphy, G.L. (1987). Concepts in connectionist models. In J.S. Denker (Ed.), *Neural Network for Computing.* New York: American Institute of Physics.

Anderson, J.A., & Mozer, M.C. (1981). Categorization and selective neurons. In G.E. Hinton & J.A. Anderson (Eds.), *Parallel models of associative memory.* Hillsade: Erlbaum.

Anderson, J.A., Pellionisz, & A., Rosenfeld, E. (1991). *Neurocomputing II.* Cambridge: MIT press.

Anderson, J.A., & Rosenfeld, E. (1987). *Neurocomputing.* Cambridge: MIT press.

Anderson, J.A., Silverstein, J.W., Ritz, S.A., & Jones, R.S. (1977). Distinctive features, categorical perception, and probability learning: some applications of a neural model. *Psychological Review*, **84**, 413–451.

Beltrami, E. (1987). *Mathematics for dynamic modelling.* New York: Academic press.

Bechtel, W., & Abrahamsen, A. (1991) *Connectionim and the mind.* Oxford: Blackwell.

Block, H.D. (1962). The perceptron: a model for brain functioning. *Review of modern physics*, **34**, 123–135.

Bryson, A.E., & Ho, Y.C. (1969). *Applied Optimal Control.* New-York: Blaisdell.

Carpenter, G., & Grossberg, S. (1991). *Pattern recognition by self-organizing neural networks.* Cambridge: M.I.T. Press.

Catlin, D.E. (1989). *Estimation, control, and the discrete Kalman filter.* Berlin: Springer-Verlag.

Ciarlet, P.G., (1989). *Introduction to numerical linear algebra and optimisation.* Cambridge: C.U.P.

Duda, R.O., & Hart, P.E. (1973). *Pattern classification and scene analysis.* New York: Wiley.

Fletcher, R. (1987). *Practical methods of optimization.* New York: Wiley.

Fogelman-Soulié, F. (1985). *Contribution à une théorie du calcul sur réseaux.* Thèse d'état: IMAG, Grenoble.

Freeman, J.A., & Skapura, D.M., *Neural Networks.* Reading (M.A.,): Addison-Wesley.

Fukunaga, K. (1990). *Statistical pattern recognition* New York: Academic Press.

Gardner, W.A. (1990). *Introduction to random processes.* New York: McGraw-Hill.

Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the bayesian restoration of images. *IEEE Proceedings on Artificial and Machine Intelligence*, **8**, 721–741.

Golden, R. (1986). The "brain–state–in–a–box" neural model is a gradient descent algorithm. *Journal of Mathematical Psychology*, **30**, 73–80.

Goles, E., & Martinez, S. (1990). *Neural and automata networks.* Norwell (M.A.): Kluwer.

Hebb, D.O. (1949). *The organization of behavior.* New York: Wiley.

Hecht-Nielsen, R. (1990) *Neurocomputing.* Reading: Addison-Wesley.

Hertz, J.; Krogh, A., & Palmer, R.G. (1991). *Introduction to the theory of neural computing.* Reading: Addison-Wesley.

Hinton, G.E., & Sejnowski, T.J. (1986). Learning & Relearning in Boltzman machine. In D.E. Rumelhart & J.L. McClelland (Eds.), *Parallel distributed processing.* Cambridge: MIT Press.

Hopfield, J.J. (1982). Neural networks and physical system with emergent collective computational abilities. *Proceedings of the National Academy of Science, USA*, **79**, 6871–6874.

Hopfield, J.J. (1984). Neurons with graded responses have collective computational abilities. *Proceeding of the national academy of Sciences, USA*, **81**, 3088–2558.

Horn, R.A., & Johnson, C.R. (1985). *Matrix Analysis.* Cambridge: C.U.P.

Kamp, Y., & Hasler, M. (1990). *Recursive neural networks for associative memory.* New York: Wiley.

Kanerva, P. (1988). *Sparse distributed memory.* Cambridge: M.I.T. press.

Kerlinger, F.N. (1986). *Foundation of Behavioral Research.* New-York: Holt, Rinehart, & Winston.

Khanna, T. (1990). *Foundations of neural networks.* Reading: Addison-Wesley.

Kirkpatrick, S., Gelatt, C.D., & Vecchi, M.P. (1983). Optimization by simulating annealing. *Science*, **220**, 671–680.

Kohonen, T. (1977). *Associative memory: A system theoretical approach.* Berlin: Springer Verlag.

Kohonen, T. (1984). *Self organization and associative memory.* Berlin: Springer Verlag.

Kosko, B. (1991). *Neural network and Fuzzy systems.* Englewwod Cliffs: Prentice-Hall.

Kosko, B. (1992) *Neural network for signal processing.* Englewwod Cliffs: Prentice-Hall.

Lachman R., Lachman, J.L., & Butterfield, E.C. (1979). *Cognitive Psychology and information processing.* Hillsdale: Erlbaum.

Lashley, K.S. (1950). In search of the engram. *Society of Experimental Biology Symposium Nb. 4: Psychological mechanisms in animal behavior.* Cambridge: CUP.

Le Cun, Y. (1986). Learning process in an asymmetric threshold network. In E. Bienenstock, F. Fogelman Soulié & G. Weisbuch (Eds.), *Disordered systems and biological organization.* Berlin: Springer Verlag.

Levine, D.S. (1991). *Introduction to neural and cognitive modelling.* Hillsdale: Erlbaum.

Linsay, P.H., & Norman, D.A. (1980). *Traitement de l'information et comportement humain.* Montréal: Editions Vivantes.

Little, W., & Shaw G. (1974). The existence of persistent states in the brain. *Mathematical Biosciences.* **19**, 101–120.

Luenberger, D.G. (1979). *Introduction to dynamic systems.* New York: Wiley.

McClelland, J.L. (1986). Resource requirements of standard and programable nets. In D.E. Rumelhart & J.L. McClelland (Eds.), *Parallel distributed processing.* Cambridge: MIT Press.

McClelland, J.L., Rumelhart, D.E., & Hinton, G.E. (1986). The appeal of parallel distributed processing. In D.E. Rumelhart & J.L. McClelland (Eds.), *Parallel distributed processing.* Cambridge: MIT Press.

MacGregor, R.J. (1987). *Neural and brain modelling.* New York: Academic Press.

Magnus, J.R., & Neudecker, H. (1988). *Matrix differential calculus with application in Statistics and Econometrics.* New York: Wiley.

Maren, A.J., Harston, G.T., & Pap, R.M. (1990). *Handbook of neural computing applications.* San Diego: Academic Press.

Mardsen, J.E., & Tromba, A.J. (1988). *Vector calculus.* San Francisco: Freeman.

Minsky, M.L., & Papert, S.A. (1969). *Perceptron.* Cambridge: Mit Press.

Moussouris, J. (1974). Gibbs and Markov random systems with constraints. *Journal of Statistical Physics.* **10**, 11–13.

Muller, B., & Reinhardt, J. (1990). *Neural networks.* Berlin: Springer Verlag.

Newell, A., & Simon, H.A. (1972). *Human problem solving.* Englewood Cliffs: Prentice–Hall.

Nilsson, N.J. (1965). *Learning machines.* New York: MacGraw-Hill.

O'Toole, A.J., & Abdi, H. (1989). Connectionist approaches to visually based feature extraction. In G. Tiberghien (Ed.), *Advances in cognitive psychology (Vol. 2).* London: Wiley.

O'Toole, A.J., Deffenbacher, K., Abdi, H. & Bartlett, J.C. (1991). Simulating the 'other-race' effect as a problem in perceptual learning, *Connection Science*, **3**, 163–178.

Pao, Y.H. (1989). *Adaptative pattern recognition and neural networks.* Reading: Addison-Wesley.

Parker, D.B. (1985). Learning logic. Technical report TR-47, Center for Computational Research

in Economics and Management Science, Massachusetts Institute of Technology, Cambridge, MA.

Pierre, D.A. (1969). *Optimization theory with applications.* New York: Wiley.

Perez, J.C. (1990). *La révolution des ordinateurs neuronaux.* Paris: Hermès.

Perko, L. (1991). *Differential equations and dynamical systems.* Berlin: springer-Verlag.

Quinlan, P.T., (1991). *Connectionism and Psychology.* Chicago: University of Chicago Press.

Rosenblatt, F. (1957). The perceptron: a perceiving and recognizing automation (projet PARA), Cornell Aeronautical Laboratory Report, 85-460-1.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organisation in the brain. *Psychological Review*, **65**, 386-408.

Rosenblatt, F. (1961). *Principles of neurodynamics.* Washington: Spartan Books.

Rumelhart, D.E. (1977). *Human information processing.* New York: Wiley.

Rumelhart, D.E., & McClelland, J.L. (1986). *Parallel distributed processing.* Cambridge: MIT Press.

Rumelharts, D.E., Hinton, G.E., & Williams, R.J. (1986). Learning internal representations by error propagation. In D.E. Rumelhart & J.L. McClelland (Eds.), *Parallel distributed processing.* Cambridge: MIT Press.

Sánchez, D.A. (1968). *Ordinary differential equations and stability theory: An introduction.* San-Francisco: Freeman.

Schalkoff, R. (1991). *Pattern recognition: Statistical, structural, and neural approaches.* New York: Wiley.

Searle S.R. (1982) *Matrix algebra useful for statistics.* New York: Wiley.

Sebestyen, G.S. (1962). *Decision making processes in pattern recognition.* New York: Macmillan.

Serra, R., & Zanarini, G. (1990). *Complex systems and cognitive processes.* Berlin: Springer-Verlag.

Seydel, R. (1988). *From equilibrium to chaos.* New York: Elsevier.

Siarry, P., Dreyfus, G. (1988). *La méthode du recuit simulé.* Paris: I.D.S.E.T.

Simpson, P.K. (1990). *Artificial neural systems.* New York: Pergamon press.

Smolenski, P. (1986). Information processing in dynamical system: foundations of harmony theory. In D.E. Rumelhart & J.L. McClelland (Eds.), *Parallel distributed processing.* Cambridge: MIT Press.

Souček, B. (1989). *Neural and concurrent real-time systems.* New York: Wiley.

Strang, G. (1986) *Introduction to applied mathematics.* Cambridge: Wellesley-press.

Tiberghien, G. (1991). Psychologie de la mémoire humaine. In M. van der Linden & R. Bruyer (Eds.), *Neuropsychologie de la mémoire humaine.* Grenoble: P.U.G.

Underwood, B.J. (1975). Individual differences as a crucible in theory construction. *American Psychologist*, **30**, 128–34.

Weisburg, G. (1991). *Complex systems dynamics.* Reading: addison-Wesley.

Werbos, P.J. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* Doctoral Dissertation Thesis: Havard University.

Widrow, B., & Hoff, M.E. (1960). Adaptative switching circuits. *1960 IRE WESCON Convention Records*, 96–104.

Widrow, B., & Stearns, S. (1985). *Adaptive signal processing.* New-York: Prentice-Hall.

Zeidenberg, M. (1990). *Neural network in artificial intelligence.* Chichester: Ellis Horwood.