CHAPITRE 6 : MODELES CONNEXIONNISTES-PERCEPTRONS MULTICOUCHES

Jean Marie Nicolas

CHAPITRE 6 : MODELES CONNEXIONNISTES-PERCEPTRONS MULTICOUCHES

Jean Marie Nicolas

1 Réseau connexionniste et reconnaissance des formes

1.1 Historique

L'apparition des réseaux de neurones dans le monde scientifique a eu une génèse assez curieuse qu'un peu d'histoire permet de mieux comprendre[1].

L'idée d'utiliser un réseau d'opérateurs élémentaires connectés entre eux est une résurgence d'un courant qui a marqué les travaux en Intelligence Artificielle et en Psychologie des années 60, dont la réalisation la plus marquante a été le Perceptron [Rosenblatt 62]. Elle est fondée sur le comportement de réseaux de processeurs élémentaires interconnectés. Chaque processeur peut être considéré comme une simplification extrême d'un neurone, ce que Mc Culloch et Pitts (1943) ont appelé un "neurone formel".

La première réalisation majeure : Perceptron [Rosenblatt 62] fut sujette à des critiques, en particulier de Minsky et Papert qui, irrités essentiellement par le succès du Perceptron, s'attachèrent à mettre en évidence des limites théoriques et pratiques de celui-ci (problèmes non modélisables par cette approche, difficulté de généralisation à une structure comportant plusieurs niveaux, explosion combinatoire). Suite à cette critique, d'une scientificité étriquée dans la mesure où elle n'envisageait pas des extensions possibles de l'approche, elle fut quasiment abandonnée à la fin des années 60.

Au début des années 80, on assiste à une renaissance de l'approche connexionniste, que l'on peut expliquer par l'intérêt marqué pour le parallélisme d'une part, l'apprentissage de l'autre, ainsi que par le développement technologique qui rend possible la simulation voire la réalisation de machines correspondantes, avec des capacités respectables. Les travaux menés ont permis aux réseaux connectionnistes d'avoir de solides fondements théoriques ainsi que des réalisations informatiques disponibles pour tous : en particulier il existe dans Matlab une *toolbox* consacrée aux réseaux de neurones, démocratisant une fois pour toute ce type d'approche.

[1] Ce paragraphe a été écrit par Alain Grumbach

L'approche connexioniste a donc eu une génèse longue et tortueuse. Les terminologies sont donc variées : on parle de réseaux de neurones, de réseaux neuromimétiques, de réseaux connexionistes. Tout ceci cache en réalité une discipline aux frontières de domaines aussi différents que la physique théorique, la biologie, les sciences informatiques. C'est probablement cet aspect interdisciplinaire qui en fait sa force.

1.2 Les réseaux neuromimétiques en reconnaissance des formes

L'approche connexioniste a très rapidement joué un rôle essentiel en reconnaissance des formes. Le cadre est celui de l'apprentissage supervisé dans lequel on dispose d'une base de données étiquetées (qui deviendra une base servant à l'apprentissage), c'est à dire telle qu'il existe un certain nombre de classes C et qu'à chaque individu de la base est associée sa classe. En reprenant le choix de notations du chapitre 4 "Classification automatique-Méthodes non hiérarchiques", on a :

- un nombre de classes connues : C,
- un espace d'état de dimension N,
- un nombre d'individus donné R,
- chaque individu p, $p \in [1, R]$, est décrit par un vecteur d'état X_p
- un vecteur d'état X_p est décrit par ses composantes $X_{i,p}$, $i \in [1, N]$,
- à chaque individu décrit par X_p est associée une étiquette $d_p, p \in [1, R]$. Cette étiquette est tout simplement le numéro de la classe, c'est à dire une valeur entière entre 1 et C.

L'objectif d'un outil de classification est donc d'associer à un individu étiqueté une grandeur qui est, dans le cas idéal, l'étiquette fournie par la base. De ce fait, pour avoir une première opinion sur les performances d'un outil de classification, il suffit d'analyser les sorties de cet outil sur la totalité de la base et de compter les individus bien classés et ceux qui sont mal classés. On définit ainsi une erreur de classification ε définie par

$$\varepsilon \;=\; \frac{\text{Nombre d'individus mal classés}}{R} \tag{1}$$

On définit de même le taux de bonnes classifications :

$$\tau \;=\; \frac{\text{Nombre d'individus bien classés}}{R} \tag{2}$$

On a bien évidemment :

$$\varepsilon + \tau \;=\; 1$$

Ce sont cette erreur et ce taux qui servent en général à analyser et à comparer les performances de divers outils de classification.

Un outil de classification est donc censé donner une étiquette à une entrée quelconque. Comme toute méthode de classification supervisée, on attend d'un réseau neuromimétique d'être capable de généraliser au mieux ses caractéristiques d'apprentissage. Deux points méritent d'être dès maintenant soulignés :

– La représentativité de la base étiquetée est un élément crucial pour obtenir un outil de classification suffisament général. Il sera illusoire d'obtenir la classification d'un individu trop différent des individus ayant permis l'apprentissage du réseau. Eventuellement on pourrait ajouter une classe supplémentaire, dite classe de rejet qui serait attribuée à tout individu manifestement trop différent de ceux qui ont été appris.
– Le taux de bonne classification est un indicateur insuffisant pour qualifier les performances d'un outil de classification. Prenons le cas de l'apprentissage d'une courbe dans le plan passant par R points : on sait qu'il existe un polynôme de degré R passant exactement par les R points initiaux, mais on sait aussi que son allure fortement oscillatoire ne permet aucune généralisation pour une valeur qui ne serait pas exactement l'une des R valeurs initiales. Classiquement, on recherche plutôt une courbe plus simple (souvent une droite en physique) passant "à peu près" au voisinage des R points initiaux. L'indicateur n'est pas alors seulement le taux de bonne classification (d'autant que le plus souvent la courbe ne passe effectivement sur aucun des points initiaux) : on lui associe aussi un terme d'erreur (le plus souvent une erreur quadratique, dont les propriétés de dérivabilité sont essentielles pour un calcul du type "minimisation au sens des moindres carrés"). Nous verrons donc que l'analyse et la comparaison de méthodes (ou de résultats) utilisent aussi ce type d'erreur.

2 Architecture d'un réseau connexionniste

Un réseau connexioniste associe donc une entrée (vecteur d'état) à une sortie (étiquette scalaire ou vecteur étiquette). Entre ces deux états, le réseau opère un certain nombre d'opérations calculant l'étiquette à partir du vecteur d'entrée, au même titre par exemple qu'un système de décision bayésien associe une étiquette à une entrée.

Ce qui change fondamentalement dans un réseau neuromimétique peut se focaliser sur les deux points suivants :

– Il existe un élément de base, le neurone formel, qui sert de brique élémentaire au système construit entre le vecteur d'entrée et la sortie. Ce neurone formel est capable d'exécuter des opérations simples (suffisament simples pour avoir été implémentées sur des systèmes analogiques dans les années 60).
– Le réseau est un assemblage particulier de neurones formels. Une fois une architecture choisie, on spécifie les neurones formels dans la phase dite d'apprentissage durant laquelle on optimise les neurones pour que la sortie soit la plus proche de la sortie désirée. C'est cette même architecture qui est utilisée en phase d'exploitation (étape de reconnaissance). Aucune hypothèse n'est à faire pour définir les neurones, à la différence d'un système de classification bayésienne qui aura besoin des lois de probabilités a priori des classes pour être construit : c'est simplement à partir de la base de données étiquetées que l'on optimise chaque neurone.

L'objectif de ce paragraphe se focalise sur la définition des neurones et sur l'ar-

chitecture que l'on peut définir autour de ces neurones. La combinaison de neurones élémentaires peut bien entendu s'effectuer d'une multitude de manières. Dans ce cours nous nous limiterons à un type de réseau : les réseaux en couches (layered perceptrons), qui sont bien adaptés à la reconnaissance des formes. Les paragraphes suivants (3 et 4) seront dédiés à l'étape d'apprentissage, c'est à dire comment optimiser les neurones pour que le réseau donne la sortie souhaitée.

2.1 Structure d'un réseau connexionniste

2.1.1 Le neurone formel individuel

Un neurone formel, c'est à dire une unité du réseau, est doté d'une sortie unique et de plusieurs entrées. Il peut être vu comme un processeur élémentaire calculant sa sortie en fonction de ses entrées.

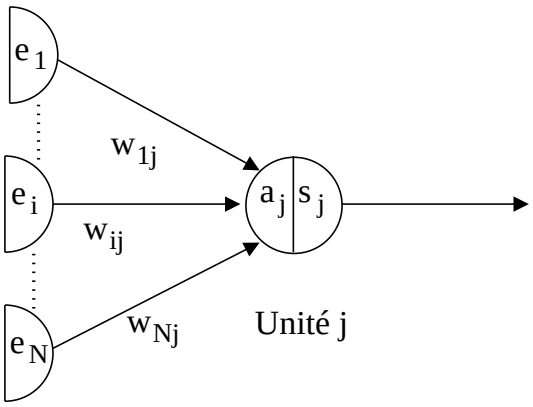

FIG. 1 – Neurone formel (unité j).

Soit le neurone formel j. Il peut être décrit par les éléments suivants :
- les N entrées e_i de l'unité j.
- a_j l'activité de l'unité j qui décrit comment le neurone opère sur les entrées :

$$a_j = \sum_{i=1}^{N} w_{ij}\, e_i$$

  L'activité d'un neurone formel est simplement l'expression d'une somme pondérée (que l'on peut aussi identifier à un filtre linéaire). Le neurone formel j est donc décrit essentiellement par ses poids w_{ij}.
- s_j la sortie de l'unité j : c'est le résultat de l'application d'une fonction f_j, appelée fonction de transition, sur l'activité de l'unité j :

$$\begin{aligned} s_j &= f_j\left(a_j\right) \\ &= f_j\left(\sum_{i=1}^{N} w_{ij}\, e_i\right) \end{aligned} \tag{3}$$

La fonction de transition f_j joue un rôle important dans la génèse des réseaux neuromimétiques. En effet, différents choix peuvent être pris pour cette fonction, conduisant à des réseaux structurellement différents :

- la fonction identité. Dans ce cas la sortie s'exprime simplement comme l'activité du neurone formel :

$$s_j \;=\; \sum_{i=1}^{N} w_{ij}\, e_i$$

  La valeur en sortie appartient à un intervalle sur $\mathbb{R}$. Le choix de cette fonction conduit aux perceptrons "classiques" de Rosenblatt du chapitre 2.
- la fonction seuil, définie par une variable σ_j représentant le seuil :

$$\left\{ \begin{array}{rclcrl} s_j & = & f_{seuil}(a_j) & = & -1 & \text{si} \quad a_j < \sigma_j \\ s_j & = & f_{seuil}(a_j) & = & +1 & \text{si} \quad a_j \geq \sigma_j \end{array} \right.$$

  On voit que les valeurs accessibles se limitent à deux valeurs, ici +1 et -1. C'est en pratique cette fonction qui est utilisée sur un perceptron "à la Rosenblatt" une fois que l'apprentissage est terminé. La classification est dite binaire.
- une fonction sigmoïde, c'est à dire une fonction continue dérivable qui ressemble à une fonction seuil et qui est décrite par un paramètre K_j positif et un seuil σ_j. On peut choisir comme fonction sigmoïde la fonction $f_{sigmoide}$ suivante :

$$f_{sigmoide}\left[K_j, \sigma_j\right](s_j) \;=\; \frac{\mathrm{e}^{K_j(a_j-\sigma_j)} - 1}{\mathrm{e}^{K_j(a_j-\sigma_j)} + 1} \tag{4}$$

  La fonction sigmoïde décrite ici a ses valeurs dans $]-1;+1[$. On montre aisément qu'en faisant tendre K_j vers l'infini, on retrouve la fonction seuil. De plus sa dérivée existe sur $\mathbb{R}$:

$$f'_{sigmoide}\left[K_j, \sigma_j\right](s_j) \;=\; 2\frac{K_j\, \mathrm{e}^{K_j(a_j-\sigma_j)}}{\left(\mathrm{e}^{K_j(a_j-\sigma_j)} + 1\right)^2} \tag{5}$$

  La figure 2 illustre cette fonction sigmoïde ainsi que sa dérivée.
  Nous verrons que l'on utilise aussi une autre fonction sigmoïde ayant ses valeurs dans $]0;+1[$ et définie par :

$$f_{sigmoide,]0,1[}\left[K_j, \sigma_j\right](s_j) \;=\; \frac{\mathrm{e}^{K_j(a_j-\sigma_j)}}{\mathrm{e}^{K_j(a_j-\sigma_j)} + 1}$$

2.1.2 Les neurones en réseau

En raccordant des neurones formels entre eux, on construit forment alors un réseau neuromimétique. La manière dont les neurones se connectent peut se faire de différentes manières. On peut en effet envisager les cas suivants :

- Chaque neurone a pour entrées les sorties de tous les autres neurones. On dit alors que le réseau est totalement interconnecté. On parle alors de réseau de Kohonen. L'étude de ce type de réseaux est en fait du ressort de la physique théorique du fait de grandes similitudes avec les aspects statistiques de la thermodynamique.

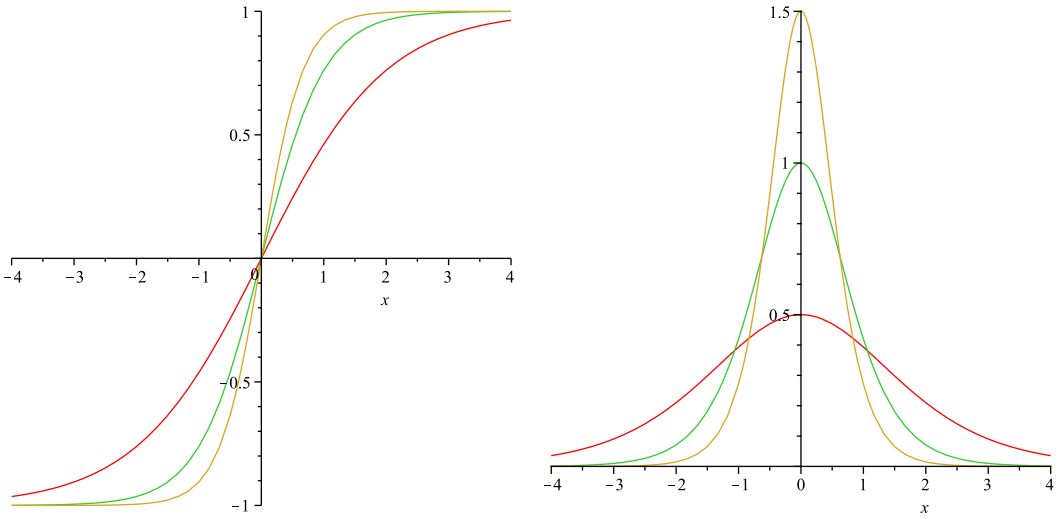

FIG. 2 – A gauche : fonction sigmoïde (équation 4) pour $K_j = 1, 2$ et 3. A droite : dérivée de la fonction sigmoïde (équation 5). On a ici $\sigma_j = 0$.

- Chaque neurone a un petit nombre d'entrées et ne semble pas suivre de règle apparente en terme d'interconnection avec les autres neurones. Les neurones biologiques (comme ceux du cerveau) entrent dans cette catégorie.
- Les neurones sont regroupés en couches. Les entrées d'un neurone d'une couche donnée m sont les sorties des neurones de la couche $m-1$. La sortie d'un neurone de la couche m est aussi une entrée pour les neurones de la couche $m+1$. On parle alors de réseaux multicouches et ce sont ceux là qui entrent dans le cadre de ce chapitre. Nous avons déjà rencontré un tel réseau au chapitre 2 : c'est le perceptron, qui peut être interprété comme un réseau monocouche. Notons que ce type de réseau n'a plus qu'un rapport lointain avec la réalité biologique.

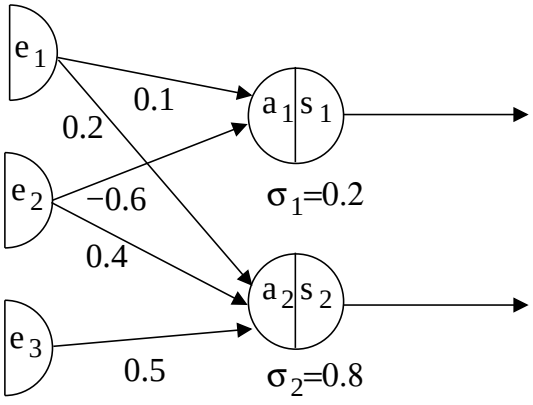

FIG. 3 – Fonctionnement d'une couche. Noter que ce réseau n'est pas totalement connecté entre les entrées et les sorties (formellement, cela revient à prendre $w_{31} = 0$).

La figure 3 montre une couche d'un réseau de 3 entrées et de 2 sorties. Analysé en terme d'information, on peut décrire le fonctionnement de cette couche comme si

chaque neurone était un processeur élémentaire synchronisé avec les autres neurones de la couche :

- L'information en entrée est décrite par les valeurs (e_1, e_2, e_3).
- Elle transite sur les connexions. Chaque unité calcule son activité (somme pondérée), puis sa sortie (fonction sigmoïde associée au seuil σ). Pour cela, à chaque liaison entre entrée et neurone est associé un poids w_{ij} dont la valeur est indiquée sur cette figure.
- Les sorties de cette couche sont les valeurs (s_1, s_2).

Parmi les neurones d'un réseau neuromimétique, se trouvent en général des unités particulières, qui réalisent l'interface avec l'environnement : ce sont les unités d'entrée du réseau et les unités de sortie du réseau. Les unités d'entrée du réseau ont leurs entrées raccordées au "monde extérieur", par exemple, l'espace des données (décrites par leurs vecteurs d'état). Les unités de sortie du réseau ont leurs sorties raccordées au "monde extérieur", par exemple à l'espace des classes. Les autres unités, si elles existent, ne sont pas visibles du monde extérieur : on parle alors de couche cachée.

2.2 Les perceptrons

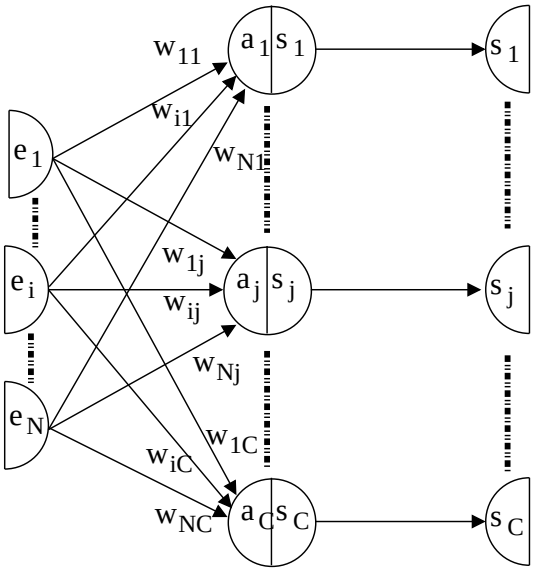

FIG. 4 – Perceptron monocouche. Il y a N entrées et C sorties. Le réseau est totalement interconnecté, c'est à dire que chaque entrée est connectée à chaque sortie (plus précisément, l'activité de chaque unité de sortie dépend des valeurs de chaque entrée).

Depuis les travaux sur les réseaux neuromimétiques des années 80, la famille des perceptrons (au sens large) est associée à celle des réseaux multicouches. Deux grandes familles s'en dégagent :

- Le perceptron simple, figure 4 (perceptron de Rosenblatt déjà rencontré au chapitre 2). C'est un réseau caractérisé par l'identification des unités d'entrée du réseau aux unités d'entrée de l'unique couche et des unités de sortie du réseau

aux unités de sortie de l'unique couche : la couche de neurones est donc totalement "visible" du monde extérieur Dans ce schéma historique, la fonction de transition est la fonction seuil. Les limites d'une telle architecture ont été démontrées dans les années 60 : elle n'est adaptée qu'à des séparations linéaires dans l'espace des données. La détermination des poids s'effectue par la célèbre règle du perceptron décrite au chapitre 2.

– les réseaux à plusieurs couches d'unités que l'on appelle Perceptrons multicouches. Comme le perceptron simple, il communique avec le monde extérieur par les unités d'entrée du réseau et les unités de sortie du réseau. Entre l'entrée et la sortie, il y a donc une ou plusieurs couches dont les sorties ne sont pas connues du monde extérieur et qui sont les couches cachées. Chaque couche (indicée m) est composée de k_{m-1} entrées $e_j^{(m)}$, de k_m unités d'activités $a_j^{(m)}$ et de k_m sorties $s_j^{(m)}$. Pour un réseau à q couches, il y a donc $k_0 = N$ entrées du réseau et $k_q = C$ sorties du réseau.

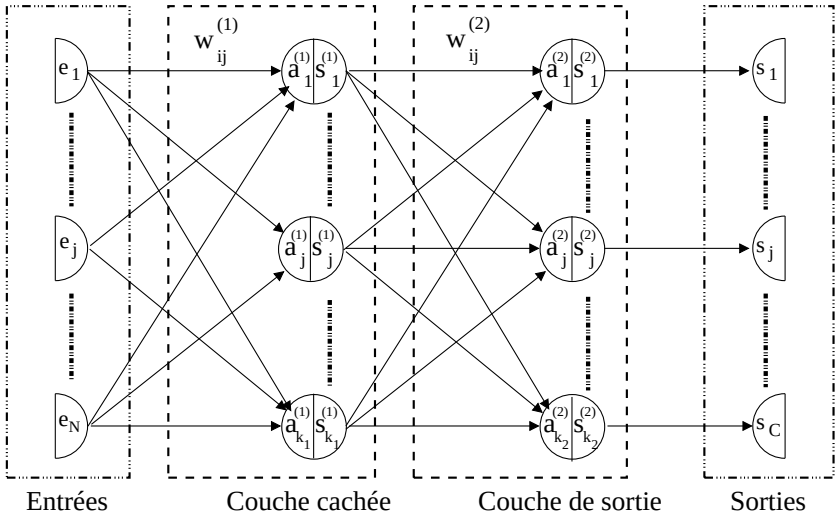

FIG. 5 – Exemple d'un perceptron à 2 couches, donc à une couche cachée. Le système complet a N entrées et C sorties.

Considérons la couche m d'un perceptron multicouches : soit k_m le nombre de neurones de cette couche. Pour tout neurone j de cette couche ($j \in [1, k_m]$), on peut calculer son activité $a_j^{(m)}$:

$$a_j^{(m)} = \sum_{i=1}^{k_{m-1}} w_{ij}^{(m)} e_i^{(m)} = \sum_{i=1}^{k_{m-1}} w_{ij}^{(m)} s_i^{(m-1)}$$

Connaissant sa fonction de transition $f_{m,j}$, on calcule sa sortie $s_j^{(m)}$:

$$s_j^{(m)} = f_{m,j}\left(a_j^{(m)}\right) = f_{m,j}\left(\sum_{i=1}^{k_{m-1}} w_{ij}^{(m)} s_i^{(m-1)}\right)$$

Par simplification, on prend souvent une même fonction d'activation f pour tous les neurones (ce point sera détaillé au paragraphe 5.1).

On voit dans cette expression que le calcul des sorties des neurones de la couche m nécessite d'avoir les sorties de tous les neurones de la couche $m-1$: on parle alors de propagation.

Le perceptron simple (monocouche) a donc été longuement étudié dans les années 60 : on peut en définir les poids grâce, par exemple, à la règle du perceptron. Pour les perceptrons multicouches, des techniques spécifiques d'apprentissage, seulement découvertes dans les années 80, doivent être mises en œuvre (voir paragraphe 4).

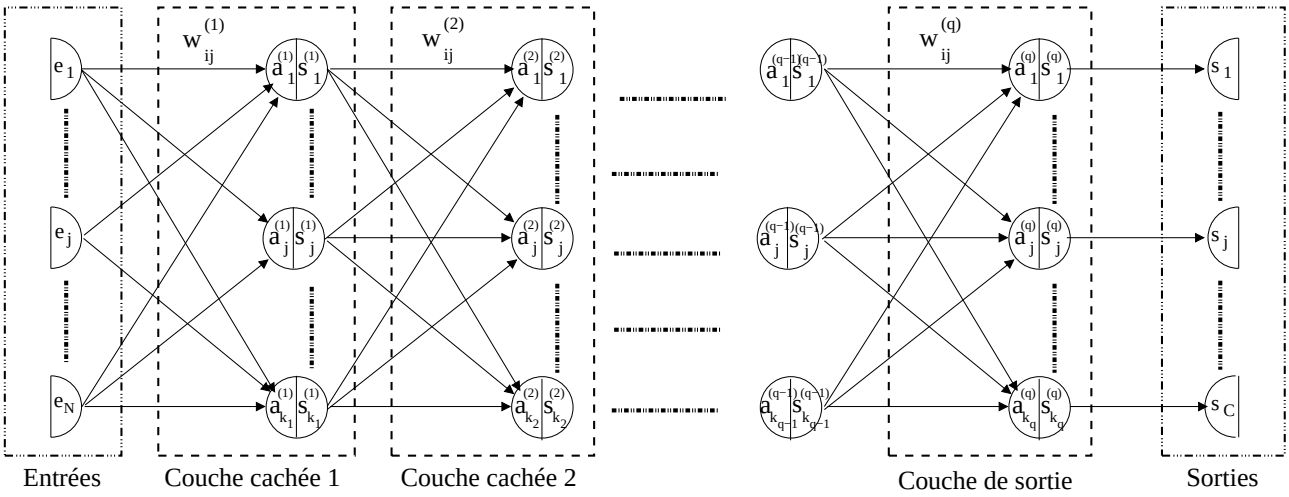

FIG. 6 – Exemple d'un perceptron à q couches, donc à $q-1$ couches cachées.

3 Apprentissage des perceptrons sans couches cachées

Une fois l'architecture d'un réseau choisie (ici, dans ce paragraphe, un réseau sans couche cachée), il reste à choisir les valeurs des poids de chaque neurone : c'est la phase d'apprentissage, qui nécessite une base servant à l'apprentissage[2] composée de R individus décrits dans un espace d'état de dimension N. Chaque individu de cette base est décrit par son vecteur d'état $X_p = (X_{i,p})$, $i \in [1, N]$. Cette base est étiquetée, c'est à dire qu'à chaque vecteur d'état $X_{i,p}$ on associe une étiquette scalaire d_p qui permet de désigner la classe à laquelle appartient l'échantillon p.

Dans le cadre des réseaux neuromimétiques, on a donc R individus dans la base servant à l'apprentissage. Pour une entrée $(e_{i,p}) = (X_{i,p})$ ($i \in [1, N]$ puisque le vecteur d'état est de dimension N), la sortie désirée du réseau est alors $s_{1,p} = d_p$ (un scalaire en sortie).

Par simplification, nous étudions dans ce paragraphe un cas assez simple : celui de la **classification en deux classes**. La sortie du réseau est alors de dimension 1 : on attribue par exemple la valeur +1 comme sortie souhaitée pour les individus de la classe 1 et la valeur -1 comme sortie souhaitée pour les individus de la classe 2. .

[2]La notion exacte de "base d'apprentissage" spécifique aux méthodes neuromimétiques sera introduite ultérieurement.

Nous verrons que l'on peut, de près ou de loin, se ramener à ce cas dans le cas de la classification en C classes (paragraphe 5.2).

3.1 Modification des poids : la règle Delta

Pour apprendre les poids d'un perceptron monocouche, une première approche est d'évaluer la qualité du résultat de classification en analysant les performances globales sur la totalité de la base servant à l'apprentissage. Cette analyse de performances va devoir être plus complète de celle menée au paragraphe 1.2 où l'erreur se limitait à analyser le nombre d'individus bien classés : une fonction de coût (de type "erreur quadratique") va être requise pour permettre une modification des poids débouchant sur de meilleures performances en classification.

Dans ce paragraphe, nous considérons donc le cas où il n'y a que deux classes. On peut sans perte de généralité considérer alors une sortie unique dont les valeurs souhaitées sont les suivantes :

- +1 si l'individu appartient à la classe 1
- -1 si l'individu appartient à la classe 2

Si la fonction de transition est un simple seuil, on aura alors effectivement en sortie les deux valeurs +1 et -1. Puisque la différence entre sortie obtenue et étiquette ne prend que 3 valeurs, -2, 0 et 2, le taux de mauvaise classification est alors aisé à écrire :

$$\varepsilon \;=\; \frac{\sum_{p=1}^{R} |s_{1,p} \;-\; d_p|}{2R}$$

A priori, on ne peut rien déduire de cette expression sur ce qu'il serait possible de faire sur les poids du perceptron pour améliorer ce taux.

Considérons maintenant la sortie d'un réseau dont la fonction de transition peut prendre toutes les valeurs possibles sur un intervalle fixé dans le choix de la fonction de transition, c'est à dire ici l'intervalle $]-1;1[$ (c'est le cas de la fonction sigmoïde du paragraphe 2.1.1). Si l'on compare les valeurs de sortie du réseau à la valeur de l'étiquette, la différence est alors une valeur continue : cette différence peut être décrite par une fonction dérivable si la fonction de transition est dérivable. Introduisons maintenant une fonction de coût J. Il y a bien évidemment un choix très grand pour cette définition. Le choix le plus simple à manipuler est celui de l'erreur quadratique moyenne[3], c'est à dire en considérant la somme des erreurs quadratiques entre étiquette et sortie du réseau pour tous les individus de la base servant à l'apprentissage. On obtient ainsi la fonction de coût J telle que :

$$J \;=\; \sum_{p=1}^{R} (s_{1,p} \;-\; d_p)^2 \tag{6}$$

On retrouve une expression connue en classification automatique puisque c'est celle de la méthode des k-moyennes.

[3]Pour alléger les expressions, on va omettre le terme de division par le nombre d'individus R

En écrivant la sortie s_p en fonction du vecteur d'état (équation 3), et en choisissant une fonction de transition unique pour tous les neurones, on a :

$$J \;=\; \sum_{p=1}^{R} \left(f\left(\sum_{i=1}^{N} w_{i1}\, e_{i,p} \right) \;-\; d_p \right)^2$$

Puisque la sortie est un scalaire (un seul neurone de sortie), on va simplifier les écritures en remplaçant dans ce paragraphe w_{i1} par w_i et $s_{1,p}$ par s_p. On a alors

$$J \;=\; \sum_{p=1}^{R} (s_p \;-\; d_p)^2 \;=\; \sum_{p=1}^{R} \left(f\left(\sum_{i=1}^{N} w_i\, e_{i,p} \right) \;-\; d_p \right)^2 \tag{7}$$

(l'écriture w_{i1} a été ici simplifiée en w_i puisqu'il n'y a qu'une seule sortie).

On voit que cette dernière expression donne une erreur, que l'on cherche a priori à minimiser en jouant sur les paramètres w_i. Si f est une fonction de transition dérivable (ce qui n'est pas le cas de la fonction seuil) et puisque l'erreur quadratique est dérivable, on peut rechercher à minimiser J. Ce minimum sera atteint si

$$\begin{aligned}
\frac{\partial J}{\partial w_i} \;&=\; 0 \quad \forall i \in [1, N] \\
&=\; 2 \sum_{p=1}^{R} \frac{\partial f\left(\sum_{i'=1}^{N} w_{i'}\, e_{i',p} \right)}{\partial w_i} \left(f\left(\sum_{i'=1}^{N} w_{i'}\, e_{i',p} \right) \;-\; d_p \right) \\
&=\; 2 \sum_{p=1}^{R} \frac{\partial f\left(\sum_{i'=1}^{N} w_{i'}\, e_{i',p} \right)}{\partial w_i} (s_p \;-\; d_p) \\
&=\; 2 \sum_{p=1}^{R} e_{i,p}\; (s_p \;-\; d_p)\; \left. \frac{\partial f}{\partial x} \right|_{x=\sum_{i'=1}^{N} w_{i'}\, e_{i',p}}
\end{aligned} \tag{8}$$

Cette dernière expression conduit donc à un système de N équations à N inconnues : les N poids w_i. Deux cas peuvent se présenter :

- la fonction de transition possède une expression assez simple pour que le système 8 puisse s'inverser analytiquement. On a alors directement la valeur des poids. C'est le cas par exemple où la fonction de transition est la fonction identité. On a alors un système linéaire de N équations à N inconnues aisément inversable s'il est bien conditionné, ce qui est rarement le cas. Cette solution est donc très rarement utilisée, voire totalement illusoire.
- la fonction de transition est plus complexe et le système 8 ne peut être inversé simplement de manière analytique. On a bien un système de N équations à N inconnues, mais c'est un système implicite qui doit être résolu de manière numérique. En général, c'est l'algorithme de descente de gradient qui est appliquée. En désignant le gradient selon le poids w_i par la notation $\nabla_i(J)$, on a alors une expression analytique de ce gradient :

$$\nabla_i(J) \;=\; 2 \sum_{p=1}^{R} e_{i,p}\; (s_p \;-\; d_p)\; \left. \frac{\partial f}{\partial x} \right|_{x=\sum_{i'=1}^{N} w_{i'}\, e_{i',p}}$$

$$= \; 2 \sum_{p=1}^{R} e_{i,p} \; (s_p \; - \; d_p) \; \left. \frac{\partial f}{\partial x} \right|_{x=a_{1,p}} \tag{9}$$

cette dernière expression mettant en jeu l'activité $a_{1,p}$ du neurone de sortie.
Il est intéressant de remarquer que le gradient dépend de trois termes :
– $(s_p \; - \; d_p)$: plus la sortie diffère de la sortie désirée, plus le gradient est grand.
– $e_{i,p}$: le gradient est d'autant plus important que l'entrée est grande.
– $\left. \frac{\partial f}{\partial x} \right|_{x=a_{1,p}}$: si la valeur de l'activité du neurone $a_{1,p}$ est autour de la valeur nulle, alors l'effet sera grand sur le gradient puisque la dérivée de la fonction de transition joue un rôle uniquement autour de la valeur nulle.

Puisque l'on a une forme analytique du gradient, il suffit de modifier itérativement les poids en prenant en compte la direction du gradient, c'est à dire en appliquant pour le poids w_i la règle suivante :

$$w_i \; \rightarrow \; w_i' \; = \; w_i \; - \; \eta \nabla_i(J) \tag{10}$$

le coefficient η devant être choisi de sorte à éviter les pièges classiques de tout algorithme de descente de gradient (s'il est trop grand, on va être "éjecté" du minimum absolu ; s'il est trop petit, on risque d'être piégé dans un minimum secondaire).
Appliqué à la fonction de transition identité (dont la dérivée est égale à 1), la relation 9 devient :

$$\nabla_i(J) \; = \; 2 \sum_{p=1}^{R} e_{i,p} \; (s_p \; - \; d_p)$$

Si on ne considére qu'un seul échantillon p, on retrouve la célèbre règle Delta :

$$w_i' \; = \; w_i \; - \; \eta \, e_{i,p} \; (s_p \; - \; d_p) \tag{11}$$

Cette règle est simplissime puisque n'interviennent que la valeur d'entrée, la valeur de sortie calculée et la valeur de sortie désirée. On la trouve aussi dans la littérature sous les noms de règle de Widrow-Hoff, règle adaline ou règle LMS (Least Mean Square).
Dans le cas général, il faut effectivement avoir l'expression analytique de la dérivée de la fonction de transition (voilà pourquoi le choix d'une fonction de transition dérivable a été introduit dans l'introduction du neurone formel au paragraphe 2.1.1). Ensuite il faut en calculer la valeur pour la donnée d'entrée e_p.

3.2 Un cas particulier de la règle Delta : la règle du perceptron

Nous avons déjà vu la règle du perceptron (chapitre 2), que nous allons retrouver ici dans le cadre neuromimétique. En effet, considérons la règle Delta (équation 11, cas d'une sortie scalaire) :

$$w_i' \; = \; w_i \; - \; \eta \, e_{i,p} \; (s_p \; - \; d_p)$$

Prenons $\eta = 1$. On a alors :

$$w'_i \;=\; w_i \;-\; e_{i,p}\;(s_p \;-\; d_p)$$

Or dans ce cadre de classification en deux classes (étiquettes +1 ou -1), et si l'on choisit maintenant la fonction seuil (initialement interdite puisqu'il fallait avoir une fonction de transition dérivable), on peut remarquer que le terme $s_p \;-\; d_p$ peut prendre 3 valeurs :

- 0 si l'élément est bien classé. Dans ce cas, on ne modifie pas les poids.
- +2 ou -2 si l'élément est mal classé. On a alors :
  - +2 si $s_p = 1$ et $d_p = -1$. On a alors

$$w'_i \;=\; w_i \;-\; 2e_{i,p}$$

  - -2 si $s_p = -1$ et $d_p = +1$. On a alors

$$w'_i \;=\; w_i \;+\; 2e_{i,p}$$

Réécrivons maintenant la règle du perceptron (chapitre 2) qui s'applique dès lors que l'échantillon est mal classé :

$$w'_i \;=\; w_i \stackrel{+}{-} e_{i,p}$$

le signe indiqué $\stackrel{+}{-}$ rappelant qu'il faut prendre le signe + si l'individu est étiqueté en classe L^+ et classé en L^-, ou le signe - si l'individu est étiqueté en classe L^- et classé en L^+. On voit qu'à un facteur 2 près, on obtient la même règle. On a bien retrouvé la célèbre règle du perceptron.

Cette règle du perceptron a aussi une autre dimension pratique : c'est en présentant chaque individu de la base servant à l'apprentissage que les poids sont modifiés.

3.3 Apprentissage des perceptrons monocouches

Un des points essentiels des techniques neuromimétiques réside dans le fait que, comme pour la règle du perceptron, on ne calcule pas la fonction de coût sur la totalité de la base servant à l'apprentissage. En pratique, on calcule la fonction de coût pour chaque individu de la base servant à l'apprentissage. Dans ce cas, si on présente l'individu p, le gradient (requis pour la mise en œuvre d'un algorithme de descente de gradient) s'écrit (équation 9) :

$$\nabla_{i,p}(J) \;=\; 2e_{i,p}\;(s_p \;-\; d_p)\;\left.\frac{\partial f}{\partial x}\right|_{x=a_{i,p}} \tag{12}$$

Cet aspect essentiel de l'apprentissage sera abordé au paragraphe 5

4 Apprentissage des perceptrons multi-couches

Face aux limitations aux cas linéairement séparables du perceptron monocouche, l'idée de rajouter des couches au perceptron monocouche et donc d'utiliser un perceptron multicouches semble être une piste prometteuse. Cependant la règle du perceptron n'est pas applicable aux poids d'un perceptron multicouches. Aussi les années 60 ont donc enterré le perceptron faute de pouvoir se pencher dans le calme sur un terme d'erreur quadratique dans un cadre multicouches. C'est cette piste, évidente dès lors que quelqu'un la propose[4], qui n'a été trouvé qu'au milieu des années 80 que nous allons détailler maintenant.

4.1 L'algorithme de rétropropagation du gradient

Considérons un perceptron à 2 couches (donc une couche cachée de k_1 poids, c'est le perceptron de la figure 5), avec N entrées et C sorties. Par simplification, nous prendrons la même forme analytique pour la fonction d'activation f que nous noterons f_1 pour la couche cachée et f_2 pour la couche de sortie (l'objectif de cette notation explicite permettra de bien comprendre les rôles respectifs des couches lors du calcul du gradient).

Détaillons d'abord les activités et les sorties de chaque couche.

Pour la première couche, c'est à dire la couche cachée, on a :

$$a_j^{(1)} = \sum_{i=1}^{N} w_{ij}^{(1)} e_i$$

(avec, par simplification $e_i^{(1)} = e_i$ puisque c'est l'entrée du réseau complet). La sortie des k_1 neurones de la couche cachée $s_j^{(1)}$, $j \in [1, k_1]$, s'écrit :

$$s_j^{(1)} = f\left(a_j^{(1)}\right) = f_1\left(\sum_{i=1}^{N} w_{ij}^{(1)} e_i\right)$$

Propageons ces valeurs dans la seconde couche. On a alors l'activité du neurone j de la seconde couche :

$$\begin{aligned} a_j^{(2)} &= \sum_{q=1}^{k_1} w_{qj}^{(2)} e_q^{(2)} \\ &= \sum_{q=1}^{k_1} w_{qj}^{(2)} s_q^{(1)} \\ &= \sum_{q=1}^{k_1} w_{qj}^{(2)} \left(f_1\left(\sum_{i=1}^{N} w_{iq}^{(1)} e_i\right)\right) \end{aligned}$$

[4] Il y a plusieurs chercheurs qui ont indépendamment proposé cette approche : Yann Le Cun, Rumelhart, Hinton, ...

d'où les sorties du réseau s_j :

$$\begin{aligned} s_j &= f_2\left(a_j^{(2)}\right) \\ &= f_2\left(\sum_{q=1}^{k_1} w_{qj}^{(2)} e_q^{(2)}\right) \\ &= f_2\left(\sum_{q=1}^{k_1} w_{qj}^{(2)} \left(f_1\left(\sum_{i=1}^{N} w_{iq}^{(1)} e_i\right)\right)\right) \end{aligned} \tag{13}$$

On voit dans cette dernière expression comment les valeurs d'entrée se propagent à la première couche (la couche cachée), puis de la première couche à la couche de sortie.

Le terme d'erreur quadratique s'écrit donc (cf équation 6 : on prend en compte la dimension C du vecteur de sortie) :

$$\begin{aligned} J &= \sum_{p=1}^{R}\sum_{j=1}^{C} (s_{j,p} - d_{j,p})^2 \\ &= \sum_{p=1}^{R}\sum_{j=1}^{C} \left(f_2\left(\sum_{q=1}^{k_1} w_{qj}^{(2)} e_{q,p}^{(2)}\right) - d_{j,p}\right)^2 \end{aligned} \tag{14}$$

Il faut donc modifier cette expression en remplaçant les valeurs de sortie de la couche cachée par le résultat de l'action de la couche cachée sur les entrées du réseau, ce qui donne :

$$\begin{aligned} J &= \sum_{p=1}^{R}\sum_{j=1}^{C} (s_{j,p} - d_{j,p})^2 \\ &= \sum_{p=1}^{R}\sum_{j=1}^{C} \left(f_2\left(\sum_{q=1}^{k_1} w_{qj}^{(2)} e_{q,p}^{(2)}\right) - d_{j,p}\right)^2 \\ &= \sum_{p=1}^{R}\sum_{j=1}^{C} \left(f_2\left(\sum_{q=1}^{k_1} w_{qj}^{(2)} \left(f_1\left(\sum_{i=1}^{N} w_{iq}^{(1)} e_{i,p}\right)\right)\right) - d_{j,p}\right)^2 \end{aligned} \tag{15}$$

En quelque sorte, on a propagé de droite à gauche l'erreur dans le réseau : on parle alors de rétropropagation.

Comme dans le cas du perceptron simple, on cherche les poids du réseau minimisant l'erreur quadratique. Le réseau ayant N entrées et C sorties, et la couche cachée ayant k_1 sorties, il y a donc

- $N \times k_1$ poids $w_{iq}^{(1)}$ entre entrée et couche cachée,
- $k_1 \times C$ poids $w_{qj}^{(2)}$ entre couche cachée et sortie.

Pour trouver le minimum de l'erreur quadratique, il suffit de calculer les dérivées de cette erreur vis à vis de chaque poids et de choisir les valeurs de poids visant à annuler cette dérivée. Bien évidemment, ce n'est pas cette approche qui est utilisée : on va plutôt calculer le gradient de l'erreur quadratique et appliquer une descente de gradient sur chaque poids.

Pour ce perceptron à une couche cachée, deux types de calcul sont à mener :

– l'optimisation des poids $w_{qj}^{(2)}$ entre couche cachée et couche de sortie. L'erreur quadratique s'écrit tout simplement (relation 14) :

$$J = \sum_{p=1}^{R} \sum_{j'=1}^{C} \left(f_2 \left(\sum_{q'=1}^{k_1} w_{q'j'}^{(2)} e_{q',p}^{(2)} \right) - d_{j',p} \right)^2$$

C'est une expression parfaitement similaire à celle trouvée dans le perceptron monocouche à sortie unique (relation 7). Dériver cette expression en fonction du poids $w_{qj}^{(2)}$ donne le terme requis par l'algorithme de descente de gradient (équation 9) :

$$\begin{aligned}
\nabla_{qj}^{(2)}(J) &= \frac{\partial J}{\partial w_{qj}^{(2)}} \\
&= 2 \sum_{p=1}^{R} \sum_{j'=1}^{C} (s_{j',p} - d_{j',p}) \frac{\partial f_2 \left(\sum_{q'=1}^{k_1} w_{q'j'}^{(2)} e_{q',p}^{(2)} \right)}{\partial w_{qj}^{(2)}} \\
&= 2 \sum_{p=1}^{R} e_{q,p} (s_{j,p} - d_{j,p}) \left. \frac{\partial f_2}{\partial x} \right|_{x=\sum_{q'=1}^{k_1} w_{q'j}^{(2)} e_{q',p}^{(2)}} \\
&= 2 \sum_{p=1}^{R} e_{q,p} (s_{j,p} - d_{j,p}) \left. \frac{\partial f_2}{\partial x} \right|_{x=a_{j,p}^{(2)}} \qquad (16)
\end{aligned}$$

(on utilise $a_{j,p}^{(2)}$, l'activité du neurone q de la couche de sortie, pour alléger l'expression). On retrouve l'expression menant à la règle Delta (équation 10) : pour la couche de sortie, la minimisation de l'erreur quadratique passe donc par une modification des poids prenant en compte la direction du gradient, c'est à dire

$$w_{qj}'^{(2)} = w_{qj}^{(2)} - \eta \nabla_{qj}^{(2)}(J)$$

– l'optimisation des poids $w_{iq}^{(1)}$ entre couche d'entrée et couche cachée. L'erreur quadratique s'écrit (relation 15) :

$$J = \sum_{p=1}^{R} \sum_{j=1}^{C} \left(f_2 \left(\sum_{q'=1}^{k_1} w_{q'j}^{(2)} \left(f_1 \left(\sum_{i'=1}^{N} w_{i'q'}^{(1)} e_{i',p} \right) \right) \right) - d_{j,p} \right)^2$$

expression qu'il faut dériver par rapport aux poids $w_{iq}^{(1)}$. En appliquant les techniques de dérivation des fonctions composées[5], on peut écrire :

$$\nabla_{iq}^{(1)}(J) = \frac{\partial J}{\partial w_{iq}^{(1)}}$$

[5]Il est curieux de voir que la communauté scientifique soit restée presque 20 ans sans voir que c'était le seul théorème requis pour passer des perceptrons monocouches aux perceptrons multicouches. Mais il fallait aussi utiliser une fonction sigmoïde, ce qui peut expliquer un aveuglement aussi long.

$$
\begin{aligned}
&= \sum_{p=1}^{R}\sum_{j=1}^{C} \frac{\partial \left(f_2\left(\sum_{q'=1}^{k_1} w_{q'j}^{(2)} \left(f_1\left(\sum_{i'=1}^{N} w_{i'q'}^{(1)} e_{i',p}\right)\right)\right) - d_{j,p}\right)^2}{\partial w_{iq}^{(1)}} \\
&= 2\sum_{p=1}^{R}\sum_{j=1}^{C} \frac{\partial \left(f_2\left(\sum_{q'=1}^{k_1} w_{q'j}^{(2)} \left(f_1\left(\sum_{i'=1}^{N} w_{i'q'}^{(1)} e_{i',p}\right)\right)\right) - d_{j,p}\right)}{\partial w_{iq}^{(1)}} (s_{j,p} - d_{j,p}) \\
&= 2\sum_{p=1}^{R}\sum_{j=1}^{C} \left.\frac{\partial f_2}{\partial x}\right|_{x=\sum_{q'=1}^{k_1} w_{q'j}^{(2)} e_{q',p}^{(2)}} \frac{\partial}{\partial w_{iq}^{(1)}} \left(\sum_{q'=1}^{k_1} w_{q'j}^{(2)} \left(f_1\left(\sum_{i'=1}^{N} w_{i'q'}^{(1)} e_{i',p}\right)\right)\right) (s_{j,p} - d_{j,p}) \\
&= 2\sum_{p=1}^{R}\sum_{j=1}^{C} \left.\frac{\partial f_2}{\partial x}\right|_{x=a_{j,p}^{(2)}} \sum_{q'=1}^{k_1} \left(w_{q'j}^{(2)} \frac{\partial f_1\left(\sum_{i'=1}^{N} w_{i'q'}^{(1)} e_{i',p}\right)}{\partial w_{iq}^{(1)}}\right) (s_{j,p} - d_{j,p}) \\
&= 2\sum_{p=1}^{R} e_{i,p} \left.\frac{\partial f_1}{\partial x}\right|_{x=a_{q,p}^{(1)}} \left\{ \sum_{j=1}^{C} w_{qj}^{(2)} (s_{j,p} - d_{j,p}) \left.\frac{\partial f_2}{\partial x}\right|_{x=a_{j,p}^{(2)}} \right\} \qquad (17)
\end{aligned}
$$

Cette expression assez lourde n'a finalement rien de bien compliqué. D'une certaine manière, on voit que l'on propage le gradient de la couche de sortie vers la couche d'entrée, d'où le nom de l'algorithme : rétropropagation du gradient. Le seul élément un peu technique est le fait que modifier un poids de la couche cachée modifie la sortie d'un neurone de la couche cachée : ce neurone étant en entrée de tous les neurones de la couche de sortie, toutes les sorties sont donc modifiées. Voilà pourquoi il demeure une somme sur tous les éléments de la couche de sortie ($\sum_{j=1}^{C}$), mais les termes de cette somme ont une forte ressemblance avec le gradient calculé pour la couche de sortie (expression 16).

Il serait maintenant assez aisé de généraliser ce calcul à des perceptrons avec Q couches : le seul véritable piège serait celui des notations, qui deviennent alors très lourdes à gérer.

4.2 Apprentissage des perceptrons multicouches

Comme dans le cas monocouche, les algorithmes d'apprentissage considèrent individuellement chaque individu de la base servant à l'apprentissage. En pratique, on calcule la fonction de coût pour chaque individu p de la base servant à l'apprentissage et le gradient (requis pour la mise en œuvre d'un algorithme de descente de gradient) s'écrit :

– pour un poids de la couche de sortie (équation 16) :

$$
\nabla_{qj}^{(2)}(J) = 2e_{q,p} (s_{j,p} - d_{j,p}) \left.\frac{\partial f_2}{\partial x}\right|_{x=a_q^{(2)}} \qquad (18)
$$

– pour un poids de la couche cachée (équation 17) :

$$
\nabla_{iq}^{(1)}(J) = 2\sum_{p=1}^{R} e_{i,p} \left.\frac{\partial f_1}{\partial x}\right|_{x=a_{q,p}^{(1)}} \left\{ \sum_{j=1}^{C} w_{qj}^{(2)} (s_{j,p} - d_{j,p}) \left.\frac{\partial f_2}{\partial x}\right|_{x=a_{j,p}^{(2)}} \right\} \qquad (19)
$$

Cet aspect essentiel de l'apprentissage sera abordé au paragraphe 5

5 Mise en œuvre des réseaux neuromimétiques

Dans ce paragraphe, nous allons maintenant aborder rapidement la mise en œuvre d'un réseau neuromimétique (monocouche ou multicouche) plus spécifiquement dans un contexte de classification. Nous nous plaçons dans le cadre assez général où la fonction de transition est la sigmoïde de paramètres K_j et σ_j et prenant ses valeurs dans l'intervalle $]0;1[$:

$$f(x) = \frac{e^{K_j(x-\sigma_j)}}{e^{K_j(x-\sigma_j)}+1}$$

Le paramètre K_j, dont l'inverse peut être amalgamé au concept de température [6], permet de "jouer" sur la raideur de la sigmoïde. En particulier, pour $K_j \to \infty$, on a :

$$\begin{cases} f(x) = 0 & \text{si} \quad x < \sigma_j \\ f(x) = 1 & \text{si} \quad x \geq \sigma_j \end{cases}$$

On retrouve donc la fonction seuil pour $K_j \to \infty$ (à la température nulle, il n'y a que deux états possibles : 0 ou 1).

5.1 "Apprendre" la valeur σ_j de la fonction sigmoïde

On voit qu'à chaque neurone est associée une fonction de transition a priori différente puisque dépendant d'un paramètre σ_j. Reprenons l'expression spécifique de la fonction sigmoïde :

$$\begin{aligned} x - \sigma_j &= \sum_{i=1}^{N} w_{ij}\, e_i \; - \; \sigma_j \\ &= \sum_{i=1}^{N} w_{ij}\, e_i \; + \; (-\sigma_j) \times (1) \end{aligned}$$

Dans cette dernière expression, tout se passe comme si on avait ajouté à la somme pondérée initiale une entrée d'indice $N+1$, de valeur $e_{N+1} = 1$ et de poids $-\sigma_j$. On a alors :

$$x - \sigma_j = \sum_{i=1}^{N+1} w_{ij}\, e_i$$

et il faut trouver $N+1$ poids w_{ij} au lieu de N poids et une valeur σ_j (on a donc exactement le même nombre d'inconnues). La figure 7 illustre cette équivalence sur l'exemple de la figure 3.

Par simplification, nous oublierons cet aspect dans la suite de ce chapitre et nous considérerons les fonctions sigmoïde avec $\sigma = 0$ en supposant que le concepteur du réseau a bien ajouté une entrée unité pour chaque couche (donc pour chaque neurone).

$$f(x) = \frac{e^{Kx}}{e^{Kx}+1}$$

On notera au passage que le paramètre K est le même pour tous les neurones du réseau, ce qui est un choix bien pratique et généralement admis.

[6] Là aussi la thermodynamique statistique peut aider à la compréhension.

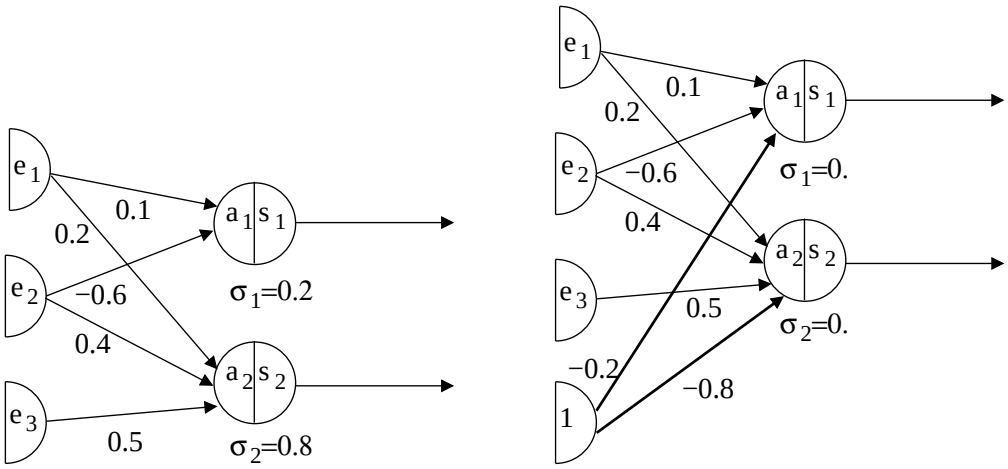

FIG. 7 – A gauche couche de la figure 3 : la fonction sigmoïde est différente pour chaque sortie. A droite, réseau équivalent doté d'une entrée supplémentaire ayant la valeur 1 et tel que la fonction d'activation ait la même valeur nulle pour σ.

5.2 Classification en C classes

La présentation des réseaux neuromimétique a été principalement faite dans les précédents paragraphes dans le cadre d'une classification en 2 classes. Le passage à C classes ne pose aucun problème a priori. On a alors une première architecture de sortie pour un réseau de classification (figure 8) : la valeur de la sortie est alors un entier correspondant au numéro de la classe (valeur entière entre 1 et C).

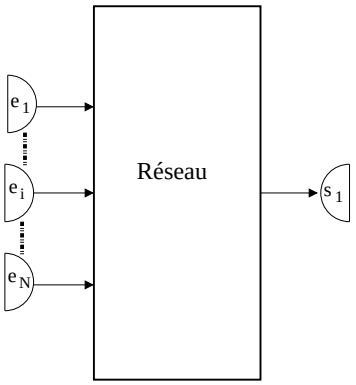

FIG. 8 – Réseau pour la classification avec N entrées et une sortie prenant la valeur de la classe (valeur entière entre 1 et C).

Cependant, il est d'usage d'utiliser plutôt un vecteur de sortie de dimension C. Pour tout individu de la classe q, la sortie désirée (étiquette) est le vecteur $\mathbf{d}$:

$$\mathbf{d} \;=\; (d_i) \;=\; (\delta_{iq}) \tag{20}$$

avec δ_{iq} symbole de kronecker (c'est à dire $\delta_{qq} = 1$ et $\delta_{iq} = 0 \;\forall i \neq q$).

On a donc, par exemple, les C vecteurs suivants comme étiquettes :

$$\begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}, \quad \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{pmatrix}, \quad \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{pmatrix}, \quad \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}$$

Ceci veut donc dire que la couche de sortie du réseau comporte C neurones : s_1 à s_C. L'apprentissage du réseau vise donc à ce que le réseau donne en sortie la valeur suivante pour un échantillon étiquetté en classe q :

$$\begin{cases} s_q = 1 \\ s_r = 0 \quad \forall r \neq q \end{cases}$$

et ce réseau requiert l'utilisation de la sigmoïde $f_{sigmoide,]0,1[}$. On a alors une seconde architecture de sortie pour un réseau de classification (figure 9).

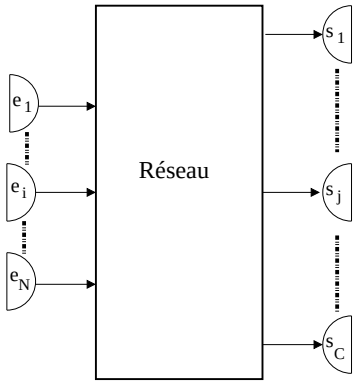

FIG. 9 – Réseau pour la classification en C classes avec N entrées et C sorties.

Dans la réalité, le réseau ne fournira jamais une telle sortie "parfaite" dans la mesure où le paramètre k de la sigmoïde est fini : on a donc des valeurs réelles entre 0 et 1 pour tous les neurones de sortie.

$$\begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_C \end{pmatrix}$$

Il faut prévoir une méthode pour gérer la couche de sortie et décider à quelle classe appartient l'individu. Deux approches peuvent être envisagées :

- Appliquer une valeur infinie au paramètre k de la sigmoïde, c'est à dire prendre la fonction seuil. Dans ce cas le vecteur de sortie sera composé de 0 et de 1. Trois cas peuvent alors se présenter :
  - un seul neurone de sortie est à la valeur 1. Dans ce cas, on connaît la classe de sortie désignée par le réseau.

– aucun neurone de sortie n'a la valeur 1 : le réseau ne peut classer l'échantillon.
– plusieurs neurones de sortie ont la valeur 1 : le réseau ne sait pas classer l'échantillon.

On voit donc poindre la notion d'échantillon non classé, c'est à dire que naturellement apparaît une classe de rejet.

– garder une valeur finie au paramètre k. Dans ce cas les neurones de sortie ont des valeurs quelconques entre 0 et 1 et il faut choisir une méthode pour décider comment associer ces valeurs à une classe. Le plus simple est de choisir l'étiquette la plus proche en calculant la distance euclidienne des sorties avec les étiquettes et en choisissant la distance la plus petite. Vue la définition des étiquettes (relation 20), cela revient à choisir la sortie s_r la plus proche de 1. Là aussi on peut analyser plusieurs cas :
  – une seule valeur s_r est proche de 1 et les autres valeurs sont proches de 0. Il n'y a alors aucun doute possible sur le choix de la classe.
  – les valeurs prenent des valeurs quelconques entre 0 et 1. On peut décider de conserver effectivement la classe r à condition que les autres valeurs s_q soient vraiment plus petites que s_r. On introduit alors un critère de décision γ qui revient à choisir la classe r si

$$s_r \geq s_q + \gamma \quad \forall q \neq r$$

Si ce test n'est pas vérifié, on classe l'échantillon dans la classe de rejet.

On a ainsi rajouté une couche de décision donnant le numéro de la classe et permettant naturellement l'introduction d'une classe de rejet(figure 10).

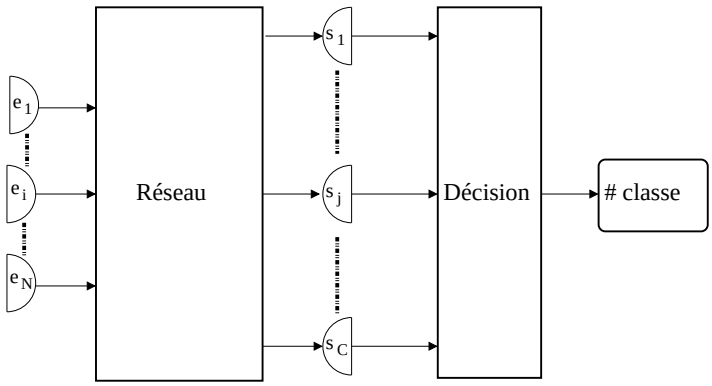

FIG. 10 – Réseau pour la classification avec N entrées et une sortie prenant soit la valeur de la classe (valeur entière entre 1 et C), soit une valeur indiquant que c'est la classe de rejet qui doit être choisie.

5.3 Stratégies d'apprentissage

5.3.1 Rappel sur les termes d'erreur

Un utilisateur d'outil de classification est en général bien informé des performances intrinsèques de cet outil par le taux de bonne classification (équation 2) : cette valeur

résume assez bien certaines qualités d'un classificateur. Bien évidemment, ce serait la matrice de confusion qui serait le meilleur indicateur de la qualité d'un classificateur.

Les réseaux neuromimétiques ont intrinsèquement besoin d'un autre indicateur : le coût, qui est en général l'erreur quadratique moyenne. C'est ce coût qui permet de piloter la phase d'apprentissage des poids du réseau. C'est ce même coût qui permet aussi d'avoir une idée sur le potentiel de généralisation d'un réseau.

A partir de ces deux grandeurs, lors de la mise en œuvre d'un algorithme neuromimétique, il est possible de définir un critère d'arrêt dépendant de l'un ou de l'autre de ces grandeurs, voire des deux.

- L'utilisateur peut fixer un objectif au taux de bonne classification. Si cette valeur est atteinte, alors l'algorithme s'arrête.
- l'utilisateur peut fixer un seuil au coût. Si la valeur du coût est inférieure à ce seuil, alors l'algorithme s'arrête.

5.3.2 Modalités opératoires en phase d'apprentissage

Un des aspects fondamentaux des réseaux neuromimétiques est de requérir une base servant à l'apprentissage, c'est à dire de disposer de R individus dotés de leur étiquette.

Utiliser la base étiquetée en phase d'apprentissage peut se concevoir de deux manières différentes :

- Une modalité d'apprentissage global. Dans ce cas, on évalue l'erreur quadratique moyenne sur la totalité de la base, selon l'équation 6

$$J = \sum_{p=1}^{R} (s_p - d_p)^2$$

  On analyse alors à chaque étape tous les échantillons de la base, puis on calcule l'erreur quadratique moyenne et on corrige les poids selon la règle Delta. Cette méthode peut être utilisée, mais elle ne donne pas toujours de bon résultats car elle débouche souvent sur des minima locaux pour la fonction coût.
- Une modalité d'apprentissage échantillon par échantillon. On choisit un échantillon p dans la base servant à l'apprentissage et on évalue son erreur quadratique.

$$J = (s_p - d_p)^2$$

  Pour les perceptrons monocouches, on peut alors en calculer le gradient (équation 12) :

$$\nabla_{i,p}(J) = 2e_{i,p} (s_p - d_p) \left.\frac{\partial f}{\partial x}\right|_{x=a_{i,p}}$$

  On modifie alors itérativement les poids selon la règle Delta, c'est à dire, dans le cas monocouche, en appliquant la règle (relation 10) :

$$w_{i'} = w_i - \eta \nabla_{i,p}(J)$$

  Les formules pour le perceptron à une couche cachée sont données par les relations 18 et 19.

C'est cette dernière méthode qui est appliquée en général dans les réseaux neuromimétiques : on utilise tous les échantillons de la base servant à l'apprentissage un par un et on modifie les poids pour diminuer l'erreur quadratique.

La mise en œuvre de cette technique requiert quelques précautions :
- Pour utiliser tous les échantillons de la base servant à l'apprentissage, on "visite" cette base en accédant aux échantillons de manière aléatoire (cela devrait éviter de présenter consécutivement au réseau deux individus de la même classe). Pour chaque échantillon on modifie les poids en calculant le gradient pour chaque poids.
- On appelle époque, l'étape complète de visite de la base servant à l'apprentissage. Un apprentissage complet se déroulera donc sur un grand nombre d'époques, ce qui correspond à un certain nombre de "balayages" de la base.

On voit que dans cette démarche, les poids du réseau seront progressivement modifiés et devraient petit à petit tendre vers des valeurs assurant de bons résultats finaux. D'une certaine manière, on peut dire que cette méthode stochastique effectue une sorte de recuit simulé en ce sens que l'on fait tout ce qui semble possible pour éviter de tomber dans des optima locaux.

On peut envisager la modification de la valeur du paramètre k de la fonction sigmoïde tout au long de l'apprentissage. En effet, au début, ce paramètre peut être petit (c'est à dire correspondant à une grande température, ce qui permet des variations fortes des poids si besoin) : ceci permet d'éviter d'être piégé par des minima locaux. En fin d'apprentissage, on peut donner au paramètre k une valeur plus forte, voire tendant vers l'infini (on s'approche alors de la fonction seuil qui ne permet pas d'apprentissage puisque non dérivable).

Le paramètre η lié à l'algorithme de descente de gradient peut lui aussi être modifié tout au long de l'apprentissage.

Dans toute phase d'apprentissage, il faut définir un critère d'arrêt : il n'y a en effet peu de chance d'atteindre une erreur nulle. Plusieurs critères peuvent être associés :
- on fixe le nombre maximal d'époques (ce qui revient à limiter le temps d'apprentissage).
- on fixe une valeur τ de bonne reconnaissance à atteindre.
- on fixe une erreur quadratique minimale à atteindre.

Cependant, cette démarche n'est pas correcte car on peut toujours trouver une architecture permettant d'avoir une valeur de τ quasiment égale à 1 pour une base donnée. Intuitivement, on peut se dire que pour une base étiquetée de R échantillons, un réseau doté de R poids devrait amener un résultat presque parfait (problème classique d'un système de R équations à R inconnues). Donc la qualité d'un apprentissage ne peut se mesurer à l'aune de la base qui sert à construire le réseau : il faut donc avoir aussi une seconde base étiquetée sur laquelle on aura une information crédible sur le réseau obtenu.

5.3.3 Base d'apprentissage et base de reconnaissance

Il est donc indispensable, avant tout apprentissage d'un réseau neuromimétique, d'avoir non seulement une base servant à l'apprentissage (c'est à dire utilisée pour trouver les poids optimaux dans le réseau), mais aussi une base servant à valider et à quantifier les résultats. En pratique, si l'on dispose d'une base étiquetée de $\tilde{R}$ individus, on la découpera en deux bases :

- une base d'apprentissage de R individus
- une base de reconnaissance de R' individus (avec $R + R' = \tilde{R}$).

La base d'apprentissage doit être, en général, mieux fournie que la base de reconnaissance, et l'expérience montre que l'on peut prendre par exemple $R/R' = 4$.

Ainsi découpée, la base étiquetée a donc deux finalités :

- ajuster les poids du réseau par la règle Delta grâce aux échantillons de la base d'apprentissage.
- Evaluer la qualité de la classification grâce aux échantillons de la base de reconnaissance. C'est sur cette base que l'on calcule le taux de bonne classification et l'erreur quadratique moyenne.

5.3.4 Un exemple classique d'algorithme neuromimétique

On a maintenant le déroulement final de la phase d'apprentissage d'un réseau de neurones, qui peut se dérouler selon plusieurs étapes :

- **Etape 1** : Initialiser l'époque à la valeur $t = 0$. Initialiser les poids $w_{ij}^{(q)}$ de manière aléatoire. Choisir une valeur pour η (paramètre de la descente de gradient).
- **Etape 2** : Effectuer successivement les opérations suivantes sur tous les échantillons de la base d'apprentissage
  - Calculer pour chaque individu tous les gradients $\nabla_{ij}^{(q)}$ correspondant à tous les poids $w_{ij}^{(q)}$ du réseau.
  - Modifier tous les poids selon la règle :

$$w_{ij}^{(q)} \quad \rightarrow \quad w_{ij}^{(q)} \; - \; \eta \nabla_{ij}^{(q)}$$

- **Etape 3** : Calculer le coût sur la base d'apprentissage. Si ce coût est inférieur à un seuil donné, passer à l'étape 5.
- **Etape 4** : Incrémenter l'époque : $t \rightarrow t+1$. Si t est inférieur à une valeur donnée (nombre maximal d'époques), retourner à l'étape 2.
- **Etape 5** : Afficher le coût sur la base d'apprentissage **et** sur la base de reconnaissance. Afficher le taux de bonne classification pour la base d'apprentissage **et** pour la base de reconnaissance.

En général, on s'attend à ce que le taux de bonne classification soit meilleur pour la base d'apprentissage que pour la base de reconnaissance. De même on s'attend à ce que le coût moyen calculé sur la base d'apprentissage soit inférieur à celui calculé sur la base de reconnaissance.

Ce protocole d'apprentissage a toutefois ses limites car un phénomène peu prédictible peut apparaître, celui du sur-apprentissage. En effet, dans certains cas, le réseau peut tellement bien apprendre la base d'apprentissage que cela s'effectue au détriment de ses capacités à généraliser cet apprentissage : en pratique, on observe alors une diminution de l'erreur quadratique sur la base d'apprentissage et en même temps l'apparition d'une augmentation de cette erreur quadratique sur la base de reconnaissance. C'est pour cela que l'on interrompt toujours une phase d'apprentissage même si l'erreur quadratique est loin d'être nulle.

5.3.5 Utilisation d'une base de test (ou base d'évaluation)

Un dernier artefact peut apparaître dans une étape d'apprentissage de réseaux neuromimétiques : celui où la base de reconnaissance "colle" trop bien à certains échantillons de la base d'apprentissage. Dans ce cas, tout se passe comme si on effectuait les tests sur la base d'apprentissage, ce qui, comme nous l'avons déjà vu, n'a pas de sens.

Aussi, si la base étiquetée initiale est assez grande, on peut envisager de la découper en trois bases : la base d'apprentissage (qui sert à trouver les bons poids), la base de reconnaissance (qui sert à fixer le test d'arrêt et à donner les performances du réseau) et enfin la base de test sur laquelle on vérifie si les performances affichées sont réellement celles que l'on peut attendre du réseau obtenu.

Cette base de test peut s'avérer très utile lors d'une phase de recherche de l'architecture idéale d'un réseau, mais ne doit en aucun cas être utilisée pour optimiser l'architecture[7].

6 Ce qu'il faut retenir des réseaux neuromimétiques

6.1 Théorème d'existence

L'échec relatif du perceptron monocouche a été scellé dans les années 60 dans les travaux de Minsky et Pappert [2] où il a été montré que seuls les problèmes linéairement séparables pouvaient être résolus par un perceptron. A cette époque, bon nombre de chercheurs ont abandonné cette piste et il a fallu donc attendre les années 80 pour que soient enfin traités les perceptrons multicouches dont le potentiel est très grand puisqu'ils semblent pouvoir résoudre n'importe quel problème de classification réaliste.

La raison du succès des perceptrons multicouche provient d'un théorème mathématique : le théorème de Kolmogorov (1957) qui ne semble avoir aucun lien avec les perceptrons. Ce théorème[8] montre qu'il est possible d'approximer n'importe quelle fonction de plusieurs variables par un ensemble de fonctions à une variable. Appliqué au réseaux

[7] Un industriel à la recherche du meilleur classificateur peut fournir une base étiquettée à un fournisseur de classificateur, mais peut se garder une base de test pour comparer divers fournisseurs.

[8] qui n'a rien à voir avec les approches neuromimétiques puisqu'il visait à résoudre la trentième conjecture d'Hilbert !!

neuromimétiques, ce théorème permet de dire que tout classificateur non linéaire associant des entrées à une sortie peut se modéliser par des fonctions de chaque entrée.

La mise en œuvre du théorème de Kolmogorov n'est pas du tout simple (voir par exemple [1]), voire même totalement déconseillée (les fonctions de transition peuvent n'être plus du tout douces). Cependant, il n'en demeure pas moins que la piste théorique existe et, grâce à elle, on démontre qu'un réseau doté de deux couches cachées peut traiter tous les problèmes. Ceci démontre l'existence du réseau, mais ne donne aucun moyen de savoir à quoi il ressemble.

En pratique, on peut donc attendre d'excellentes performances d'un réseau neuromimétique. Cependant, si le découpage en classes semble a priori compliqué, donc fortement non linéaire, il faut s'attendre à devoir choisir une architecture "lourde" : il n'est pas dit que l'algorithme d'apprentissage converge alors vers un minimum suffisament général pour assurer une bonne classification de la base de reconnaissance.

C'est donc pour cela qu'il ne faut pas hésiter à choisir une architecture simple (une couche cachée, globalement un nombre de neurones cachés du même ordre de grandeur que ceux des entrée et des sorties) et, au vu des résultats, complexifier l'architecture en ajoutant éventuellement une seconde couche cachée ou en jouant sur le nombre de neurones cachés.

6.2 Performances et robustesse des réseaux neuromimétiques

Une fois choisie l'architecture, l'utilisation d'un réseau neuromimétique requiert une base étiquetée découpée en une base d'apprentissage et en une base de reconnaissance (voire aussi en une base de test). Ce découpage pourrait apparemment pénaliser des classes ayant peu de représentants dans la base initiale.

Or, à la différence des techniques bayésiennes, un réseau de neurones peut tout à fait s'adapter à la sous représentation d'une classe. En effet, si on analyse bien l'algorithme de rétropropagation du gradient, disposer d'un grand nombre d'individus d'une classe revient à présenter plus souvent un échantillon de cette classe : le réseau aura donc effectivement tendance à adapter ses poids pour bien classer les individus de cette classe. Néanmoins, les autres représentants des autres classes auront leur rôle à jouer dans la modification des poids (moins souvent, mais tout aussi efficacement). Au final, il n'est donc pas si surprenant que cela d'avoir un taux de bonne classification et une matrice de confusion "corrects".

6.3 Amélioration des capacités de généralisation

En pratique, on attend surtout d'un réseau neuromimétique d'être capable de généraliser son apprentissage, c'est à dire d'être capable d'attribuer une classe à un individu dont la ressemblance avec les individus de la base d'apprentissage n'est guère convaincante. Globalement, les réseaux ont cette propriété. De plus des techniques existent pour améliorer cette capacité de généralisation. Deux méthodes semblent être les plus performantes :

– Ajouter à la fonction de coût un terme portant sur les poids pour éviter que certaisn d'entre eux prennent des valeurs beaucoup trop grandes. Ce processus porte le nom de régularisation.
– Ajouter des individus dans la base d'apprentissage en bruitant les données des individus initiaux de la base. Cette approche est aisée à mettre en œuvre. De plus, l'utilisateur peut avoir une idée sur le niveau de bruit qu'il peut injecter dans cette étape.

On trouvera dans [1] l'équivalence de ces deux approches.

6.4 Le rôle des prétraitements

Dans les années 80, des résultats spectaculaires ont été obtenus visant à prouver l'universalité des méthodes neuromimétiques. Par exemple, pour apprendre à classifier des sinusoïdes pures, un réseau à l'architecture savamment choisie pouvait retrouver la transformée de Fourier : à la fin de l'étape d'apprentissage, les poids du réseau approchaient les vrais coefficients de Fourier requis pour ce travail de classification. Si l'on analyse objectivement cette étonnante expérimentation, on peut dire que c'est beaucoup demander à un réseau neuromimétique que de redécouvrir par l'expérience la théorie de Fourier. A contrario, un connaisseur des méthodes utilisées en traitement du signal pourra "aider" la convergence de son réseau en introduisant des prétraitements avant l'entrée du réseau.

Comme autre exemple de prétraitement "évident", prenons le plan, un cercle centré à l'origine et une base d'apprentissage constitués par deux classes : les points situés à l'intérieur du cercle et les points situés à l'extérieur du cercle. On trouvera toujous un réseau neuromimétique capable d'effectuer correctement cette classification[9]. Mais, si au lieu d'entrer les coordonnées (x, y) des points, on utilise la valeur $d = \sqrt{x^2 + y^2}$, un classificateur linéaire donnera aussi bien le résultat.

Il n'y a aucune règle permettant de définir les meilleurs prétraitements à utiliser : on peut penser que c'est l'expert, qui connaît bien ses données, qui saura trouver les meilleures méthodes de prétraitement.

6.5 Les réseaux de neurones en pratique

Les applications des réseaux neuromimétiques sont à l'heure actuelle multiples et variées. Les questions qui se poseront toujours lors de la mise en œuvre d'une telle méthode se focalisent bien évidemment sur la pertinence d'un algorithme "aveugle" dans le problème posé. Il est néanmoins instructif de se poser les questions suivantes :

– Quelle est la représentativité de ma base de données ?
  – nombre des échantillons et de classes
  – bruit sur les vecteurs d'état
  – qualité de l'étiquetage des données
– Quelle est la dimension de mon vecteur d'état ?

[9] On laisse le soin au lecteur de trouver l'architecture requise pour une telle classification

  - trop grande ⇒ nécessité de la diminuer.
  - faible ⇒ possibilité de l'augmenter en prétraitant les données.
- Quelle est la dimension de ma sortie (nombre de classes, ...)
- Quelle est la connaissance a priori dont je dispose sur le système

Aucun ouvrage ne permettra de répondre directement à ces question en terme de méthode à employer (chaque problème détient ses réponses à travers des prétraitements spécifiques), mais on peut quand même noter les évidences suivantes :

- plus les degrés de liberté d'un système sont élevés, plus il peut "diverger" ce qui se traduira par une faible capacité de généralisation. Il est donc recommandé d'éviter un trop grand nombre de poids.
- plus les degrés de liberté sont faibles, plus le système est contraint dans une représentation trop restritive et non satisfaisante (à commencer sur la base d'apprentissage elle même).

Ne reste alors que la sagesse du concepteur pour assurer un résultat crédible selon les multiples critères qu'il s'est lui même dicté.

6.6 La pratique des réseaux de neurones

Pour ceux qui voudraient utiliser des réseaux neuromimétiques, il n'est heureusement plus requis à l'heure actuelle d'écrire soi-même son propre réseau. Un certain nombre de plateformes parfaitement opérationnelles existent :

- la "Neural Network Toolbox" de MATLAB : une partie de l'outil neuromimétique est dédié à des problèmes de reconnaissance.
- SNNS (Stuttgart Neural Network Simulator) une plateforme "recherche" téléchargeable sur le net (linux, windows, java).
  http ://www.ra.cs.uni-tuebingen.de/SNNS/
- pour les élèves de Télécom ParisTech, l'outil RNSat, développé en 2001 par un groupe d'élèves (en java) et qui permet de tester tous les paramètres décrits dans ce document avec des exemples pris dans le domaine de l'imagerie satellitaire..

Références

[1] C.M. Bishop *Neural Networks for Pattern Recognition* Oxford University Press, 1994

[2] M.L. Minsky, S.A. Pappert *Perceptron* Cambridge, 1969