CHAPITRE 6

# Réseaux multicouches

*L'apprentissage, c'est-à-dire la capacité d'aquérir de nouveaux comportements ou d'en modifier d'anciens par expérience, est une propriété de presque tous les animaux, y compris ceux que l'on peut difficilement qualifier d'intelligents.*

ERIC KANDEL*

ON appelle apprentissage toute transformation d'un système cognitif ayant pour effet de conférer à ce système une nouvelle disposition (Daniel Andler Paris VII Intellectica)

## 6.1 Systèmes paramétriques et non paramétriques

Lorsque l'on veut résoudre un problème concernant un certain phénomène, la démarche fréquente est de faire un modèle du phénomène et de résoudre ensuite le problème grâce à ce modèle. C'est ce qui est généralement fait en Physique, par exemple lorsque l'on veut étudier la chute d'un corps particulier. Dans ce cas on modélise le mouvement d'un point pesant dans le champ de gravité $g$ en résolvant les équations du mouvement et on obtient un modèle analytique dépendant de $g$, de la vitesse initiale etc. qui sont les paramètres du modèle..

Une autre démarche est de chercher une fonction de transfert qui sera construite à partir d'un certain nombre d'exemples contenant à la fois les conditions initiales et la grandeur que l'on recherche. Par exemple la vitesse initiale $\mathbf{v_0}$ (son module et son angle) et le temps $t$ mis par le point pesant pour atteindre une altitude donnée $h$. Si le nombre d'exemples $\{\mathbf{v_0}, \mathbf{t}, \mathbf{h}\}$ est suffisamment grand, on peut espérer que la fonction réalisée sera capable de fournir une réponse satisfaisante lorsqu'elle sera appliquée à des données différentes de celles qui ont permis de la construire. Cette démarche est non paramétrique et conduit à la réalisation d'approximaeurs universels (FIG. 6.1).

## 6.2 Un rapide survol avant de commencer

Les réseaux de neurones multicouches sont utilisés pour trouver la meilleure application fonctionnelle permettant de passer d'un ensemble de données d'entrée à un ensemble de données de sortie. L'ensemble des couples $\{entree, sortie\}$ s'appelle l'ensemble des données. C'est en modifiant

*Cellular Basis of Behaviour

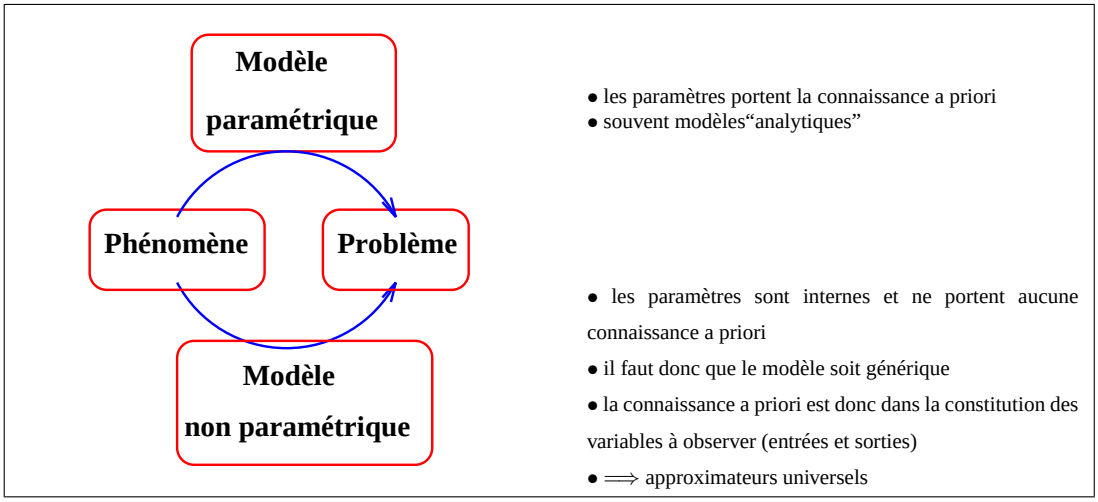



FIG. 6.1 – Modèles paramétrique et non paramétrique

les poids du réseau que l'application est construite. Mais cette construction dépend de notre façon de percevoir ce que veut dire *meilleure* dans ce contexte. D'un côté, nous voulons que le réseau approxime les données le mieux possible, mais d'un autre côté, nous voulons que le réseau soit capable de *généralisation*, c'est-à-dire que la sortie produite par le réseau sur des données inconnues soit une sorte d'interpolation effectuée sur les données connues. Nous voyons qu'une bonne généralisation est un objectif contradictoire avec une bonne régression.

Le problème de l'apprentissage c'est donc la généralisation. Voici une définition : c'est la capacité à étendre une compétence à des exemples non appris. Une autre définition : « C'est la transformation d'une présentation partielle en une représentation complète ... c'est un processus d'inférence déductive » (Daniel Andler)

Pour nous, nous ramènerons un système d'apprentissage au problème suivant : Soit une fonction $\phi : X \rightarrow Y$, apprendre ce sera produire $\hat{\phi} : X \rightarrow Y$ qui estime $\phi$.

Le cas de l'**apprentissage supervisé** sera celui où on dispose de $M$ couples $(x_i, \phi(x_i))$. Est-il alors possible de trouver à tout coup un réseau de neurones pouvant approximer $\phi$ aux $M$ points ?

Arrive alors un problème de **Complexité** : Combien d'instructions sont nécessaires pour trouver les poids idéaux d'un réseau de neurones ?

La **Généralisation** : si on trouve une fonction $\hat{\phi}$ qui estime $\phi$ aux $M$ points donnés, qu'en est-il pour d'autres points $\in X$ ?

L'apprentissage des poids d'un neurone ou d'un réseau de neurones formels est un problème d'optimisation qui consiste à trouver l'ensemble des poids minimisant une certaine fonction de coût.

## 6.3 Notion de neurone formel

En 1943 McCullogh et Pitts définirent un neurone formel comme étant une fonction de plusieurs entrées dont la sortie $s$ peut prendre les valeurs 0 ou 1. Chaque entrée $x_i$ est pondérée par un coefficient $w_i$, qui peut être positif ou négatif (voir Fig. 6.2).

L'opération effectuée par le neurone est la suivante :

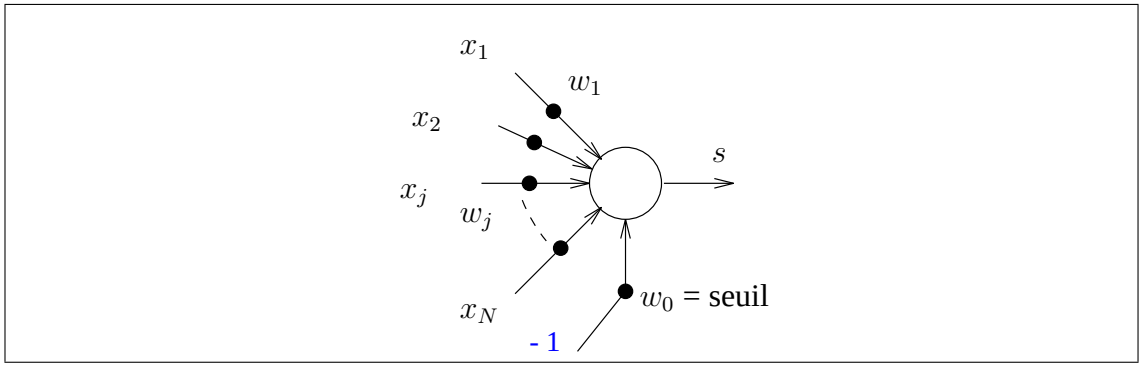



FIG. 6.2 – Un neurone.

$$s = \begin{cases} 1 & \text{si} \quad \sum_{i=1}^{N} w_i x_i > \theta \quad (\theta \text{ est un seuil}) \\ 0 & \text{sinon} \end{cases} \tag{6.1}$$

En fait on pose plutôt de manière plus compacte :

$$s = f\left(\sum_{i=0}^{N} w_i x_i\right) \tag{6.2}$$

avec $w_0$ = seuil $\theta$ et f = fonction signe.

On appelle *entrée totale* du neurone la quantité $a = \sum_{i=0}^{N} w_i x_i$, c'est la somme pondérée des entrées. On sous-entend toujours que le seuil est compris dans cette somme pondérée.

La terminologie utilisée "neurone formel" vient de McCullogh et Pitts qui se référaient à la biologie. Si on se réfère à la théorie des automates on parlera plutôt d'*automate à seuil*, en reconnaissance des formes on parle de *classifieur linéaire*, si l'on est physicien on parlera de *spin*, et dans certains ouvrages on parlera d'*élément linéaire à seuil*. Nous appellerons souvent le neurone formel : *neurone*, *cellule* ou encore *unité*.

## 6.4 Propriétés d'un neurone

La propriété essentielle d'un neurone formel de McCulloch et Pitts est d'être un séparateur linéaire, c'est-à-dire qu'il réalise une partition de l'espace des entrées en deux sous-espaces telle que les réponses qu'il donne pour deux vecteurs d'entrée appartenant à ces deux sous-espaces soient opposées. La séparatrice a donc pour équation $a = 0$.

Prenons comme exemple une fonction booléenne à deux variables $e_1$ et $e_2$, le **ET** logique en l'occurence (Fig 6.3).

On recherche les coefficients $w_0, w_1, w_2$ tels que le point (1,1) du plan soit d'un côté de la séparatrice. Une solution évidente est : $w_0 = 1.5, \quad w_1 = 1, \quad w_2 = 1.$

Si on voulait faire la fonction logique **OU**, il faudrait prendre $w_0 = 0.5, \quad w_1 = 1, \quad w_2 = 1.$ pour que le point (0,0) soit d'un côté de la séparatrice.

Voyons maintenant ce qui se passe si on veut faire un **XOR** ou **OU** exclusif (voir Fig. 6.4).

On voit que ce n'est pas possible. C'est pourquoi on dit souvent que les fonctions calculables par un neurone sont des fonctions linéairement séparables.

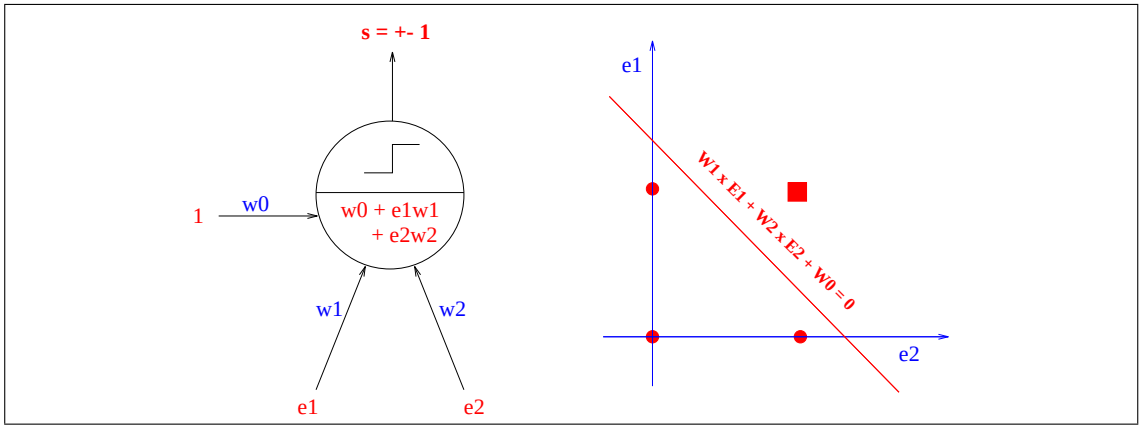



FIG. 6.3 – Un neurone est un séparateur linéaire

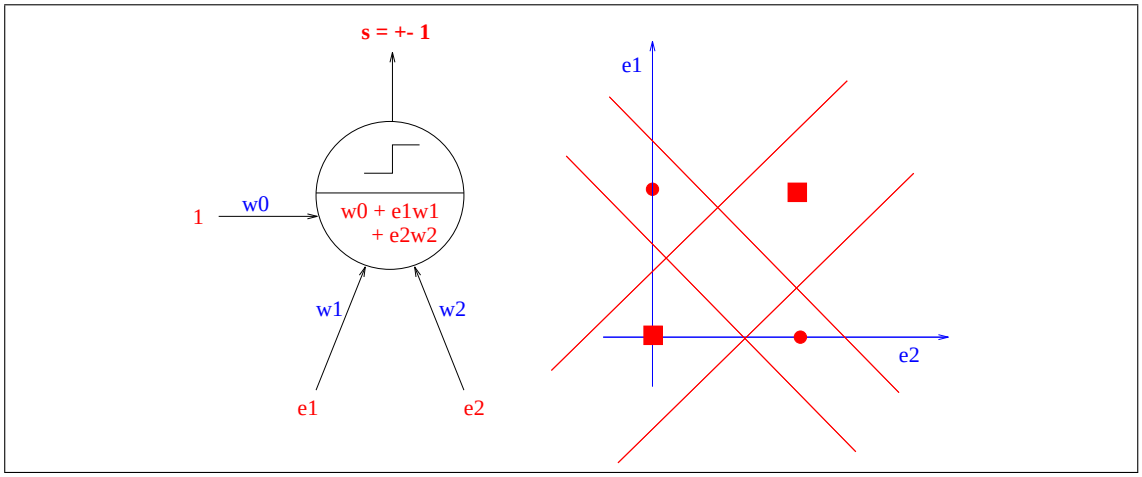



FIG. 6.4 – Un neurone peut-il réaliser le **OU** exclusif ?

## 6.5 Le Perceptron

Le Perceptron est une machine inventée par Rosenblatt en 1957. On a pensé pendant quelques temps qu'il s'agissait d'une machine à calculer universelle. Il se présente sous la forme de 4 éléments (Fig. 6.5) :

**une rétine** constituée d'un ensemble de capteurs binaires. Un "image" y apparaît composée de points noirs ou blancs.

**des cellules d'association** qui sont des prédicats prenant des valeurs binaires, réalisant des extractions de primitives sur des sous-ensembles de la rétine. Ces prédicats réalisent des fonctions booléennes quelconques et leurs valeurs de vérité $\phi_i$ sont aussi binaires.

**une couche de poids** modifiables pondérant les $\phi_i$ précédents,

**au moins une cellule de décision** qui est un élément linéaire à seuil, c'est-à-dire un neurone formel.

Lorsqu'il n'y a qu'une cellule de décision on parle de perceptron simple.

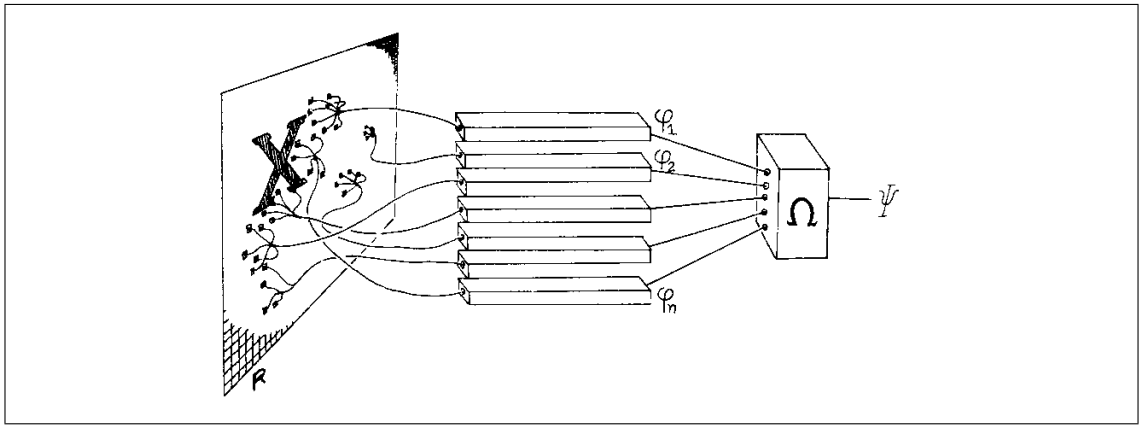



FIG. 6.5 – Tiré de minsky :1969 :book

### 6.5.1 Le talon d'Achille du Perceptron

En 1969, Minsky et Papert dans un livre célèbre [MP69], car il mis à mal la cybernétique et les systèmes à apprentissage, démontrèrent que les Perceptrons ne pouvaient résoudre certaines classes de problèmes et n'étaient donc pas des machines à calculer universelles. Ils démontrèrent en particulier que :

**Théorème** : le prédicat de connexité n'est pas calculable par un Perceptron de diamètre limité.

Nous allons "démontrer" intuitivement ce théorème et pour cela nous avons besoin de quelques définitions :

**Connexité** : une image d'entrée est connexe si elle est en un seul morceau

**Perceptron de diamètre D** : chaque prédicat prend ses valeurs sur un disque de diamètre D.

On cherche donc à construire un Perceptron qui répond $+1$ si l'image d'entrée est connexe et 0 sinon.

Soit alors les quatre images représentées sur la figure 6.6. On veut que la sortie du Perceptron

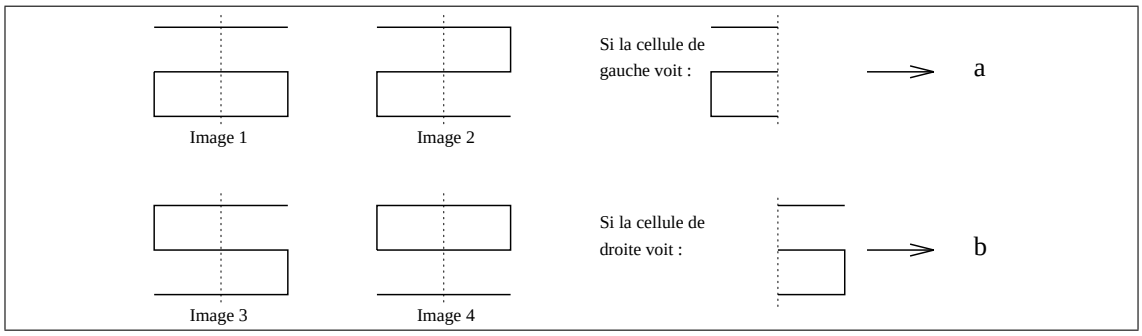



FIG. 6.6 – Problème : trouver un Perceptron de diamètre limité donnant 1 en sortie sur les images 3 et 4, et 0 sur les images 1 et 2

soit 1 sur les images connexes (2 et 3) et 0 sur les images 1 et 4. Le Perceptron aura deux cellules d'association, une première examinant le côté gauche de la figure et une seconde examinant le côté droit.

Chaque prédicat travaillera donc sur un demi espace, situé à gauche ou à droite par rapport au centre de la figure.

Posons que, selon ce que voit la cellule d'association de gauche (cf. Fig. 6.6), sa sortie vaut $a$ ou $\bar{a}$ et que de manière similaire, la sortie de la cellule d'association de droite vaut soit $b$ soit $\bar{b}$.

Le perceptron doit donc réaliser la table de vérité suivante :

| # figure | Gauche | Droite | Désiré |
|---|---|---|---|
| 1 | $a$ | $b$ | 0 |
| 2 | $a$ | $\bar{b}$ | 1 |
| 3 | $\bar{a}$ | $b$ | 1 |
| 4 | $\bar{a}$ | $\bar{b}$ | 0 |

Si maintenant on remplace $a$ et $b$ par 1 et $\bar{a}$ et $\bar{b}$ par 0 on obtient la table de vérité suivante :

| # figure | Gauche | Droite | Désiré |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
| 2 | 1 | 0 | 1 |
| 3 | 0 | 1 | 1 |
| 4 | 0 | 0 | 0 |

C'est celle du **OU** exclusif !

Le **XOR** n'étant pas linéairement séparable, il ne peut donc pas être réalisé par un Perceptron.

■

## 6.6 Apprentissage des poids d'un neurone formel

Nous allons faire ici une présentation intuitive et informelle dont le but est de **montrer** comment se passe un apprentissage supervisé par essai et erreur où la mesure de l'erreur commise lors d'une itération permet de minimiser l'erreur commise à l'itération suivante.

Soit donc un ensemble de $M$ "motifs" ou vecteurs d'entrée $\{f_1, \ldots, f_M\}$ où chacun des motifs appartient soit à une classe notée $\epsilon^+$ soit à une autre classe $\epsilon^-$. On cherche à trouver les poids $w$ du neurone formel tels que le neurone réponde $+1$ lorsque le motif présenté $\in \epsilon^+$ et $-1$ lorsque le motif présenté $\in \epsilon^-$.

On procédera de manière itérative : durant la session d'apprentissage, l'ensemble des motifs sera présenté et à chaque présentation d'un motif, le vecteur de poids sera modifié. L'opération sera répétée autant de fois que nécessaire jusqu'à l'obtention d'un résultat satisfaisant.

Soit l'ensemble d'apprentissage $\mathcal{A} = \{(\mathbf{x}^1, d_1), \ldots, (\mathbf{x}^M, d_M)\}$, où les $M$ données d'entrées $\mathbf{x}^1, \ldots, \mathbf{x}^M$ sont des vecteurs de dimension $N$ et où les sorties désirées sont (pour simplifier) des nombres réels $d_1, \ldots, d_M$.

4 cas sont alors possibles :

$$\textbf{bon} \rightarrow \begin{cases} d_m = +1 \text{ et } s_m = +1 \\ d_m = -1 \text{ et } s_m = -1 \end{cases} \qquad \textbf{faux} \rightarrow \begin{cases} d_m = +1 \text{ et } s_m = -1 \\ d_m = -1 \text{ et } s_m = +1 \end{cases}$$

Tant que la sortie réelle est égale à la sortie désirée, on ne fait rien. S'il y a une erreur on modifie chacun des poids de façon à ce que l'entrée totale se rapproche de la sortie désirée. Par exemple si

$$d_m = +1 \text{ et } s_m = -1$$

il faut augmenter (un peu) les poids positifs qui correspondent aux entrées positives ainsi que les poids négatifs qui correspondent aux entrées négatives.
De manière générale on voit qu'il faut que :

$$w_i^{t+1} \leftarrow w_i^t + \lambda(d-s)e_i \tag{6.3}$$

c'est ce qu'on appelle *la règle du Perceptron*.

### 6.6.1 Limitations

Cette règle ne fonctionne que pour des problèmes linéairement séparables.

### 6.6.2 Perspectives

Néanmoins, on sait que toute proposition logique peut se mettre sous la forme de conjonctions de disjonctions (des **OU** de **ET**) et on a vu qu'un neurone peut réaliser un **OU** et un **ET**. Pourquoi ne pas faire un réseau de neurones avec une couche de **ET** suivi d'une couche de **OU** ?
Regardons la figure 6.7. Toutes les entrées, et tous les poids participent à l'erreur commise sur

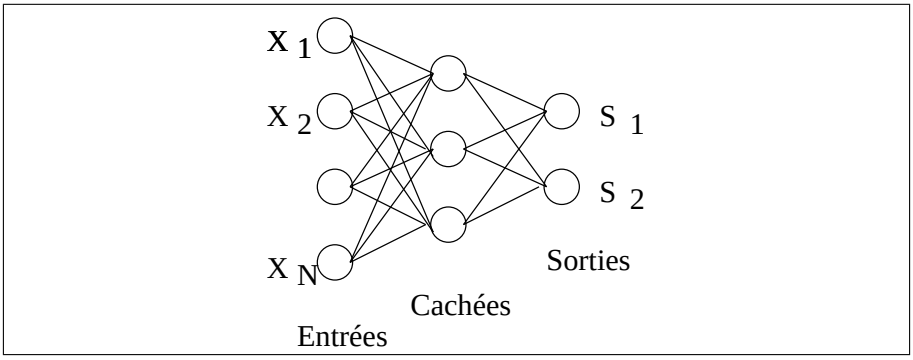



FIG. 6.7 – principe d'un réseau multi-couches

un neurone de sortie et on ne sait par conséquent pas quelles sorties désirées assigner aux cellules cachées. C'est ce qu'on appelle en anglais le *credit assignment problème* qu'on peut traduire par *problème du bouc émissaire*.

## 6.7 Minimisation d'une fonction de coût

L'approche intuitive que nous venons de voir ne permet pas de résoudre ce problème du bouc émissaire. Il nous faut formaliser le problème, ce que nous ferons en introduisant la notion de fonction de coût. Trouver les poids qui minimisent cette fonction de coût devient alors un problème d'optimisation.
La solution du problème comporte donc deux étapes : une recherche de la bonne fonction de coût puis une procédure de minimisation de cette fonction.

### 6.7.1 exemples de fonction de coût

- nombre d'exemples mal classés :

- $C(W) = \sum_m (d_m - s_m)$
- Cette fonction est bien minimale quand tous les exemples $m$ sont bien classés. Cependant elle n'a pas les bonnes propriétés : elle n'est pas continue, pas dérivable.

- Perceptron
  - $C(W) = \sum_m (s_m - d_m) a_m$
  - $= 0$ si pas d'erreur,
  - elle est toujours positive,
  - continue, dérivable par morceaux
- moindre carrés
  - $C(W) = \sum_m (d_m - a_m)^2$
  - elle possède de bonnes propriétés.

### 6.7.2 procédure de minimisation

La fonction de coût des moindres carrés, si elle dépendait d'un seul paramètre $w$, pourrait se représenter comme la fonction dessinée sur la figure XX. Une idée naturelle est d'en calculer

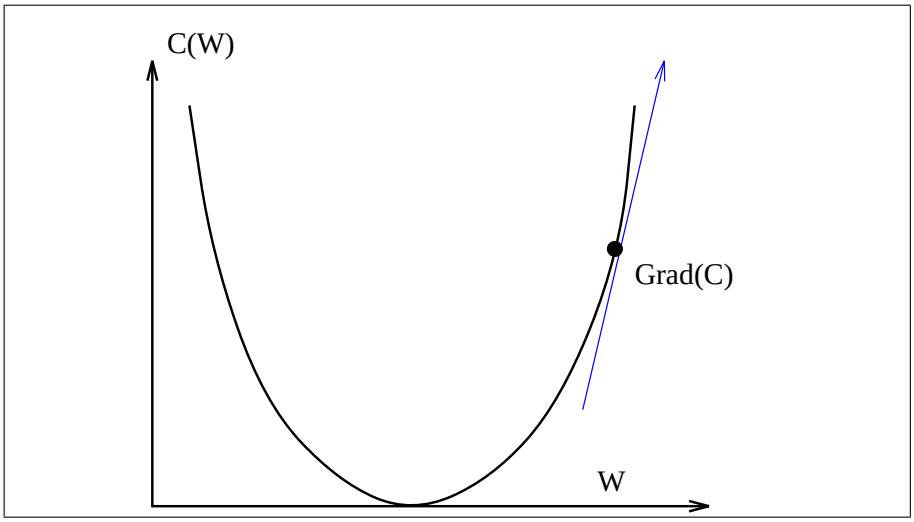



FIG. 6.8 – principe

son gradient et de modifier le paramètre $w$ dans le sens opposé au gradient. L'algorithme peut se décrire de la façon suivante :

- on part d'un $W^0$
- on calcule itérativement :
  - $W^{t+1} = W^t - \lambda(t) \cdot Grad\{C[W(t)]\}$
    où $\lambda$ est un pas d'itération "petit"

Nous verrons un peu plus loin comment calculer effectivement le gradient.

## 6.8 Bref historique

En utilisant la descente de gradient les chercheurs se rendirent compte que des réseaux de neurones à deux couches de poids étaient capables de résoudre de nombreux problèmes difficiles.

La question se posait alors de savoir s'il existait des raisons fondamentales expliquant les performances de ces réseaux.

En 1988, Gallant et White [GW88] montrèrent qu'un réseau de neurones particulier, à une seule couche de poids et avec une fonction de transfert en cosinus (la partie monotone croissante : $\Psi(x) = (1 + \cos[x + 3\pi/2])$) était capable de donner la série de Fourier d'une fonction. Un tel réseau possède donc toutes les propriétés d'approximation des séries de Fourier. En particulier celle d'approximer, à la précision que l'on veut, toute fonction de carré sommable sur un compact en utilisant un nombre fini de neurones cachés.

En 1989, Hornik [HSW89] démontra rigoureusement qu'un réseau de neurones multicouche avec une seule couche cachée de neurones utilisant n'importe quelle fonction de transfert pouvait approximer toute fonction mesurable (au sens de Borel) d'un espace de dimension finie à un autre, d'aussi près que l'on voulait, pourvu qu'il y ait suffisamment de neurones cachés. En ce sens les réseaux multicouches sont une classe d'approximateurs universels.

Les fonctions qui ne sont pas mesurables au sens de Borel existent mais sont pathologiques. Donc, à chaque fois qu'un réseau de neurones ne fonctionne pas comme il est prévu sur une application, il faut incriminer soit l'apprentissage, soit le nombre de neurones cachés, soit enfin le fait qu'il peut ne pas exister de relation déterministe entre les entrées et les sorties désirées.

## 6.9 Réseaux de neurones : complexité

- Résultat de Judd [Jud87]

  Étant donnés un réseau de neurones artificiels arbitraire $R$ et une tâche arbitraire $T$ devant être résolue par $R$, le problème consistant à décider si dans l'espace de tous les paramètres de $R$ (ses poids, sa structure) il existe une solution qui résout adéquatement $T$, est équivalent au problème de satisfiabilité, et est donc *NP-complet*. Ainsi, la recherche d'une telle solution (c'est-à-dire l'apprentissage) est *NP-ardue* (*NP-ardue* veut dire que le problème est au moins aussi difficile qu'un problème *NP-complet*).

## 6.10 NOTES

**********

- Capacité d'un système d'apprentissage Vapnik :1982 $\Longleftrightarrow$ nombre de couples $(x_i, \phi(x_i))$ pouvant être appris à tout coup par le système.

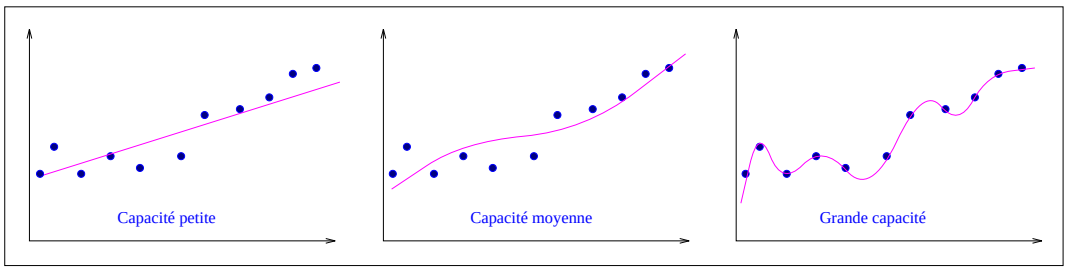



FIG. 6.9 – Apprentissage

********** Problème de la généralisation

- Soit $A$ l'ensemble des exemples d'apprentissage $x_i$, et $D(\cdot, \cdot)$ une mesure de distance,
- *erreur d'apprentissage* $= \sum_{x_i \in A} D(\phi(x_i), \hat{\phi}(x_i))$

- *erreur de généralisation* $= \sum_{x_i \notin A} D(\phi(x_i), \hat{\phi}(x_i))$

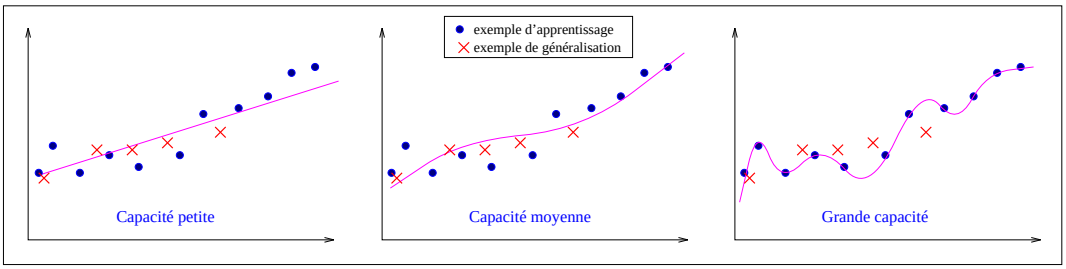



FIG. 6.10 – Généralisation

**********

## 6.11 Régression linéaire

Considérons un ensemble d'apprentissage $\mathcal{A} = \{(\mathbf{x}^1, d_1), \ldots, (\mathbf{x}^M, d_M)\}$, où les $M$ données d'entrées $\mathbf{x}^1, \ldots, \mathbf{x}^M$ sont des vecteurs de dimension $N$ et où les sorties désirées sont (pour simplifier) des nombres réels $d_1, \ldots, d_M$.

Cherchons à contruire le neurone (voir Fig. 6.11) dont les poids $w_0, \ldots, w_N$ sont tels que :

$$d_m = s_m + \epsilon_m = w_0 + w_1 x_1^m + w_2 x_2^m + \cdots + w_N x_N^m + \epsilon_m \tag{6.4}$$

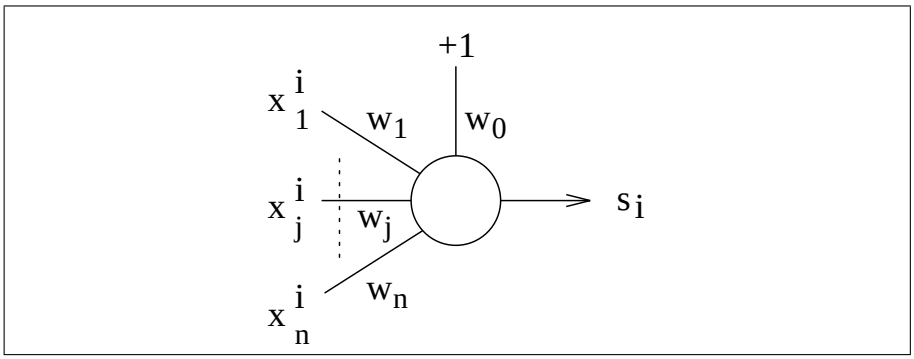



FIG. 6.11 – Régression linéaire multiple avec un neurone linéaire

Le problème de trouver les poids optimaux est connu en statistique sous le nom de *régression linéaire multiple*. Ici, les poids doivent minimiser l'erreur totale quadratique $\sum_{m=1}^{M} \epsilon_m^2$.

Ce problème peut être résolu par des méthodes algébriques de la manière suivante :

Soit $\mathbf{X}$ la matrice $M \times (N+1)$ suivante :

$$\mathbf{X} = \begin{pmatrix} 1 & x_1^1 & \cdots & x_N^1 \\ 1 & x_1^2 & \cdots & x_N^2 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^M & \cdots & x_N^M \end{pmatrix} \tag{6.5}$$

et soient $\mathbf{d}$, $\mathbf{w}$ et $\epsilon$ les vecteurs colonnes :

$$\mathbf{d} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_M \end{pmatrix} \quad \mathbf{w} = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_N \end{pmatrix} \quad \epsilon = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_M \end{pmatrix} \tag{6.6}$$

Le vecteur $\mathbf{w}$ doit satisfaire l'équation $\mathbf{d} = \mathbf{X}\mathbf{w} + \epsilon$ sous la contrainte que la norme de $\epsilon$ soit minimisée. Puisque :

$$\|\epsilon\|^2 = (\mathbf{d} - \mathbf{X}\mathbf{w})^t(\mathbf{d} - \mathbf{X}\mathbf{w}) \tag{6.7}$$

le minimum de la norme peut être trouvé en calculant la dérivée de cette expression par rapport à $\mathbf{w}$ et en la posant égale à 0 :

$$\frac{\partial}{\partial \mathbf{w}}(\mathbf{d} - \mathbf{X}\mathbf{w})^t(\mathbf{d} - \mathbf{X}\mathbf{w}) = -2\mathbf{X}^t\mathbf{d} + 2\mathbf{X}^t\mathbf{X}\mathbf{w} = \mathbf{0} \tag{6.8}$$

On a donc $\mathbf{X}^t\mathbf{X}\mathbf{w} = \mathbf{X}^t\mathbf{d}$. Si la matrice $\mathbf{X}^t\mathbf{X}$ est inversible, alors la solution du problème est donnée par :

$$\mathbf{w} = (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t\mathbf{d} \tag{6.9}$$

## 6.12 Unités non linéaires

Le calcul ne peut plus être aussi direct et on aura recours à des techniques itératives de descente de gradient par exemple. Mais nous verrons cela plus loin.

Pour l'instant nous allons nous intéresser à une forme de régression non linéaire utilisée depuis de nombreuses années en biologie ou en économie. Il s'agit de la *régression logistique*.

Soit une unité non linéaire (munie d'une sigmoïde), on cherche le vecteur $\mathbf{w}$ à $N$ composantes qui minimise l'erreur quadratique :

$$E = \sum_{m=1}^{M} (d_m - s(\mathbf{w} \cdot \mathbf{x}^m))^2 \tag{6.10}$$

où $s$ est maintenant la fonction sigmoïde telle que celle-ci par exemple :

$$s = \frac{1}{1 + \exp^{-\sum_{n=1}^{N} w_n . x_n}}. \tag{6.11}$$

Nous verons donc plus loin comment la back-prop résout le problème, mais nous pouvons d'ores et déjà trouver une solution approximative par des techniques de régression linéaire en inversant la sigmoïde et en minimisant la nouvelle erreur :

$$E' = \sum_{m=1}^{M} (s^{-1}(d_m) - \mathbf{w} \cdot \mathbf{x}^m)^2 \tag{6.12}$$

Les $d_m$ étant connus cette inversion peut se faire facilement si bien que l'on peut utiliser toutes les techniques de régression linéaires pour résoudre :

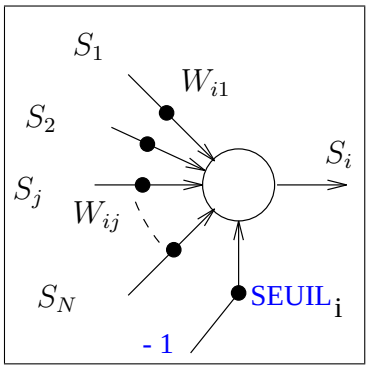
$$s^{-1}(d_m) = \sum_{n=1}^{N} w_n x_n^m = \ln\left(\frac{d_m}{1 - d_m}\right), \text{ pour } m = 1, \ldots, M \tag{6.13}$$

ce qui est bien l'équation de référence de la régression logistique (que l'on appelle couramment la *transformation Logit*). Les valeurs de $\mathbf{w}$ qui minimisent cette expression sont estimées classiquement par des méthodes numériques de maximisation de la vraisemblance.

Mais la précision n'est pas des meilleures du fait du logarithme. La back-prop, qui résoud le problème dans l'espace de départ, est plus précise.

**********

## 6.13 Méthode historique : première version




L'entrée totale et la sortie du neurone $i$ valent respectivement :
$a_i = \sum_{j=0}^{N} W_{ij} s_j$
$s_i = f(a_i)$
(Notez l'ordre des indices).

FIG. 6.12 – Un neurone effectue une transformation non linéaire, $f$, de la somme pondérée des signaux arrivant sur ses entrées. $f$ peut être la fonction Signe, de Heaviside ou une sigmoïde.

Un réseau apprend en résolvant plusieurs fois un même problème, par approximations successives, en minimisant l'erreur à chaque itération. La fonction d'erreur, ou *fonction de coût*, la plus communément utilisée est la somme du carré des erreurs des unités de sortie :

$$C = \frac{1}{2} \sum_{i \in sortie} (s_i - d_i)^2 \tag{6.14}$$

où $d_i$ est la sortie désirée de la cellule $i$. Il s'agit de l'erreur faite par le réseau lors de la présentation d'un seul motif.

La sortie effective est $s_i = f(a_i)$, où f est une fonction sigmoïde ; par exemple :

$$f(x) = \frac{1}{1 + \exp^{-x}} \tag{6.15}$$

Pour minimiser l'erreur il faut prendre la dérivée de l'erreur par rapport aux $w_{ij}$, le poids de la connexion allant de l'unité $j$ vers l'unité $i$. Nous aurons donc à calculer :

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial a_i} \cdot \frac{\partial a_i}{\partial w_{ij}} = s_j \cdot \frac{\partial C}{\partial a_i} = s_j \cdot Y_i \tag{6.16}$$

L'expression de $Y_i$ pour les cellules de sortie est donnée par :

$$Y_i = (s_i - d_i) \cdot f'(a_i) \tag{6.17}$$

et pour les cellules cachées :

$$Y_i = \sum_k \frac{\partial C}{\partial a_k} \cdot \frac{\partial a_k}{\partial s_j} \cdot \frac{\partial s_j}{\partial a_i} = f'(a_i) \cdot \sum_k Y_k \cdot w_{ki} \tag{6.18}$$

Remarque : avec l'expression de $f$ donnée plus haut nous avons : $f' = f \cdot (1 - f)$.

Comme vous pouvez le voir, il est facile de calculer l'erreur pour les liens qui vont vers les unités de sortie. Mais pour les unités cachées le gradient partiel $Y$ dépend de toutes les unités situées dans la couche après elles (La somme selon $k$ est effectuée sur les unités situées dans la couche supérieure). On voit que la valeur de $Y$ doit être rétro-propagée au travers du réseau pour calculer toutes les dérivées ; d'où le nom donné à la procédure : *rétro-propagation du gradient*.

On a donc :

$$\boxed{\begin{array}{c} w_{ij}^{t+1} \leftarrow w_{ij}^{t} - \lambda \cdot \frac{\partial C}{\partial w_{Ij}} \\ w_{ij}^{t+1} \leftarrow w_{ij}^{t} - \lambda \cdot s_j \cdot Y_i \end{array}} \tag{6.19}$$

## 6.14 Méthode historique : deuxième version

## 6.15 Méthode du Lagrangien

## 6.16 Amélioration : autre fonction de coût

## 6.17 Amélioration : ajout d'un terme d'inertie

Discussion sur la liste de diffusion `connectionists@cs.cmu.edu` du 17 mars 1999 :

```
>       I show that in the limit of continuous time, the momentum
parameter
>       is analogous to the mass of Newtonian particles that move
through a
> viscous medium in a conservative force field.

At the risk of sounding like Jim Bower, I would like to opint
out that
mechanical model was the original motivation for the momentum
method.

Geoff Hinton
```

## 6.18 Méthode de Gauss-Newton

## 6.19 Méthode de Levenberg-Marquard

## 6.20 Amélioration : pentes optimisées

Les neurones effectuent "en quelque sorte" une séparation linéaire de l'espace d'entrée, mais cette séparation n'est pas brutale, d'où les guillemets. En effet il existe par construction une zone de transition où la cellule n'est ni à +1, ni à -1. La dimension caractéristique de cette zone (par exemple celle comprise entre -90% et +90% de l'excursion de la sigmoïde) dépend de la pente

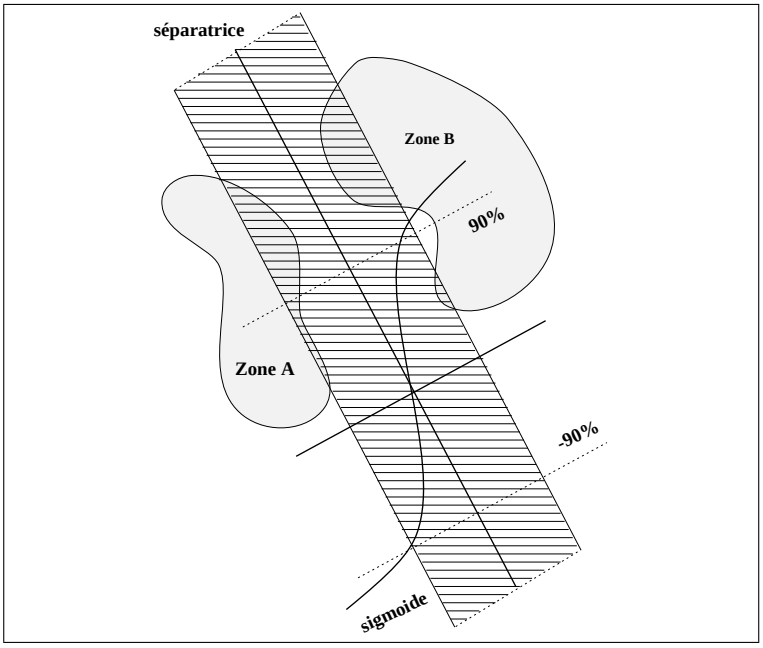



FIG. 6.13 – Position du problème

à l'origine de la fonction d'activation (voir figure 6.14). Lorsque les zones que l'on cherche à séparer sont proches l'une de l'autre, elles vont se trouver en partie dans cette zone de transition. Il existe donc pour chaque neurone effectuant une séparation linéaire, une pente optimale de la fonction d'activation. Il est exclu de la fixer *a priori* puisque cela reviendrait à avoir résolu le problème. Mais il est possible, ainsi que nous allons le voir, de trouver la pente optimale au cours de l'apprentissage en même temps que l'on trouve les poids optimaux.

Supposons que la fonction de coût dépende d'un poids $w_{ij}$ et d'une pente $p_i$ et qu'elle ait une forme quadratique (parabolique) (voir figure 6.14). La différentielle totale de cette fonction s'écrira :

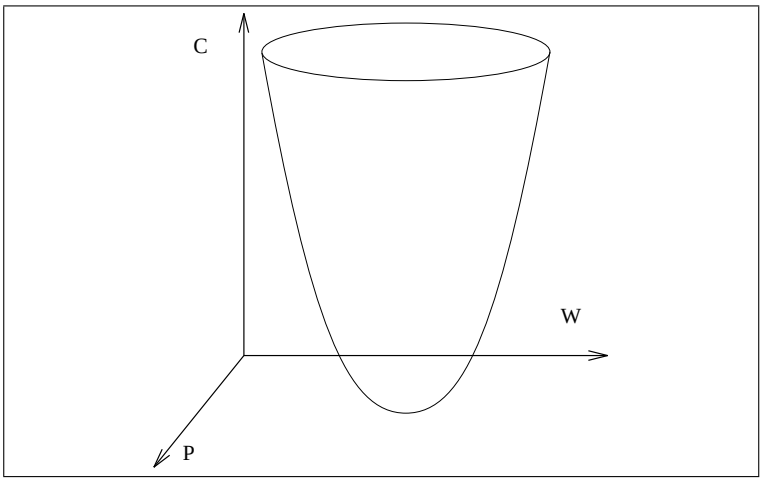



FIG. 6.14 – principe

$$dC_i = \frac{\partial C_i}{\partial w_{ij}} dw_{ij} + \frac{\partial C_i}{\partial p_i} dp_i \tag{6.20}$$

Nous avons donc deux quantités à calculer : $\frac{\partial C_i}{\partial w_{ij}}$ et $\frac{\partial C_i}{\partial p_i}$, qu'il faudra exprimer pour les cellules de sortie et les cellules cachées.

Afin d'obtenir les expressions de ces deux gradients nous poserons un coût quadratique et une fonction d'activation de la forme $f(\cdot) = th(\cdot)$ avec :

$$\begin{aligned} a_i &= \sum_j w_{ij} s_j \\ s_i &= th(p_i a_i) = th(p_i \sum_j w_{ij} s_j) \\ C &= \frac{1}{2} \sum_i (s_i - d_i)^2 \end{aligned}$$

Pour le premier gradient, dépendant des poids, nous procéderons comme d'habitude mais en faisant apparaître les pentes $p_i$ :

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial p_i a_i} \cdot \frac{\partial p_i a_i}{\partial w_{ij}} = \frac{\partial C}{\partial p_i a_i} p_i s_j = Y_i \cdot p_i s_j \tag{6.21}$$

Le gradient partiel $Y_i$ pour les cellules de sortie s'écrit :

$$\begin{aligned} Y_i &= \frac{\partial C}{\partial p_i a_i} \\ &= \frac{\partial C}{\partial s_i} \cdot \frac{\partial s_i}{\partial p_i a_i} \\ &= (s_i - d_i) f'(p_i a_i) \end{aligned} \tag{6.22}$$

Pour les cellules cachées, nous aurons :

$$\begin{aligned} Y_i &= \frac{\partial C}{\partial p_i a_i} \\ &= \sum_k \frac{\partial C}{\partial p_k a_k} \cdot \frac{\partial p_k a_k}{\partial s_i} \cdot \frac{\partial s_i}{\partial p_i a_i} \\ &= \sum_k Y_k \cdot p_k w_{ki} \cdot f'(p_i a_i) \\ &= f'(p_i a_i) \sum_k Y_k p_k w_{ki} \end{aligned} \tag{6.23}$$

Pour le deuxième gradient, dépendant des pentes, nous devons calculer :

$$\frac{\partial C}{\partial p_i} = \frac{\partial C}{\partial p_i a_i} \cdot \frac{\partial p_i a_i}{\partial p_i} = \frac{\partial C}{\partial p_i a_i} a_i \tag{6.24}$$

Pour les cellules de sortie nous aurons alors :

$$\begin{aligned} \frac{\partial C}{\partial p_i a_i} &= \frac{\partial C}{\partial s_i} \cdot \frac{\partial s_i}{\partial p_i a_i} \\ &= (s_i - d_i) f'(p_i a_i) = Y_i \quad \text{comme çi-dessus} \end{aligned} \tag{6.25}$$

De la même façon, pour les cellules cachées, nous aurons :

$$\frac{\partial C}{\partial p_i a_i} = f'(p_i a_i) \sum_k Y_k p_k w_{ki} \quad \text{comme çi-dessus} \tag{6.26}$$

En résumé, si nous procédons de manière itérative en calculant les gradients à chaque itération nous aurons :

$$w^{t+1} \leftarrow w^t + \lambda_w \Delta w^t \tag{6.27}$$

$$p^{t+1} \leftarrow p^t + \lambda_p \Delta p^t \tag{6.28}$$

avec, pour les sorties
$$\begin{cases} \Delta w_{ij} &= (d_i - s_i) \cdot f'(p_i a_i) \cdot p_i \cdot s_j = Y_i \cdot p_i \cdot s_j \\ \Delta p_i &= Y_i \cdot a_i \end{cases} \tag{6.29}$$

et, pour les cachées
$$\begin{cases} \Delta w_{ij} &= f'(p_i a_i) \cdot p_i \cdot s_j \cdot \sum_k Y_k p_k w_{ki} \\ \Delta p_i &= f'(p_i a_i) \cdot a_i \cdot \sum_k Y_k p_k w_{ki} \end{cases} \tag{6.30}$$

## 6.21 Relations entre pente et pas

Si nous utilisons une sigmoïde dont la pente à l'origine est $\beta$ :

$$f(x) = \frac{1}{1 + \exp^{-\beta x}} = \tilde{f}(\beta x) \tag{6.31}$$

Soient alors deux réseaux. Le premier est défini par ses poids $w$, par $a_{i,l}$ l'entrée totale d'un neurone $i$ de la couche $l$, par $f(a)$ la fonction de transfert d'un neurone et par le pas $\lambda$. Le second est défini par $\tilde{w}$, $\tilde{a}_{i,l}$, $\tilde{f}(\tilde{a})$, $\tilde{\lambda}$.

Nous allons rechercher dans quelles conditions deux réseaux fonctionneront de la même façon [TMF96].

Une condition suffisante est que pour tous les neurones on ait :

$$\begin{array}{rrcl} & f(a_{i,l}) &=& \tilde{f}(\tilde{a}_{i,l}) \\ \text{soit :} & \tilde{f}(\beta a_{i,l}) &=& \tilde{f}(\tilde{a}_{i,l}) \\ \text{donc :} & \beta a_{i,l} &=& \tilde{a}_{i,l} \\ & \beta \cdot \mathbf{w}_{i,l} \cdot \mathbf{f}_{l-1} &=& \tilde{\mathbf{w}}_{i,l} \cdot \tilde{\mathbf{f}}_{l-1} \end{array} \tag{6.32}$$

donc :

$$\beta \cdot \mathbf{w}_{i,l} = \tilde{\mathbf{w}}_{i,l} \tag{6.33}$$

L'équation 6.33 signifie que l'accroissement des poids dans chaque réseau est le même à chaque itération. On a donc :

$$\Delta \tilde{w}_{i,l} = \beta \Delta w_{i,l} \tag{6.34}$$

En tenant compte des équations 6.19 nous avons donc :

$$\begin{array}{rrcl} & \tilde{\lambda}.\tilde{f}.\tilde{Y}_i &=& \beta.\lambda.f.Y_i \\ \text{puisque :} & Y_i &=& \beta.\tilde{Y}_i \\ \text{on a donc :} & \tilde{\lambda} &=& \beta^2.\lambda \end{array} \tag{6.35}$$

En conclusion, passer d'une fonction d'activation de gain 1 à une fonction d'activation de gain $\beta$ revient à multiplier tous les poids par $\beta$ et à multiplier le pas d'apprentissage par $\beta^2$.

## 6.22 Régularisation : weight elimination

Trop de poids permettent au réseau de coder les idiosyncrasies de la base de données d'apprentissage (le bruit en quelque sorte) conduisant à une mauvaise généralisation. Si un poids ne sert pas il peut prendre au cours de l'apprentissage n'importe quelle valeur et en particulier des valeurs élevées, seulement pour améliorer la performance globale sur des détails non significatifs de la base. Ce problème de *sur-apprentissage* se rencontre aussi dans d'autres problèmes d'inférence déductive comme l'ajustement de courbes par des polynômes. Il existe de nombreuses méthodes pour circonvenir ce problème, mais nous ne nous occuperons ici que de celles relatives à la grandeur des poids.

Il est alors intéressant de trouver une procédure qui le ramènera à une valeur faible ce qui devrait améliorer la généralisation. L'idée sous-jacente est que, si plusieurs réseaux différents donnent les mêmes performances sur les données, le meilleur est celui qui est le plus simple. Il s'agit donc d'un avatar du rasoir d'Occam.

On peut en effet voir cette diminution de la valeur des poids comme la solution à un problème de complexité. On sait en effet (voir par exemple [Cas89]) que la complexité d'un algorithme peut se mesurer par la longueur de sa description minimale dans un certain langage. On peut le formaliser par un critère de « longueur minimale de description » issu de la théorie de l'information [Ris87].

Le modèle le plus probable est donc celui qui minimise : la longueur de description totale = la longueur de description des données, étant donné le modèle + la longueur de description du modèle.

Le risque global que l'on cherchera à minimiser comportera alors deux termes, un premier relatif à l'erreur d'approximation faite et mesurant la performance (fonction de coût habituelle) et un deuxième relatif à la description du modèle.

Le coût dû au modèle sera grossièrement proportionnel au nombre de poids multiplié par le nombre de bits utilisés pour décrire chacun de ces poids.

#### Interprétation de la régularisation en termes probabilistes

Soit $\mathcal{M}$ le modèle (en fait l'ensemble des paramètres qui caractérisent le modèle) et $\mathcal{D}$ les données. Ce que l'on cherche, c'est optimiser le modèle pour les données disponibles. Il est donc naturel d'essayer de maximiser $\Pr(\mathcal{M}|\mathcal{D})$. Or d'après le théorème de Bayes on a :

$$\Pr(\mathcal{M}|\mathcal{D}) = \Pr(\mathcal{D}|\mathcal{M}) \cdot \Pr(\mathcal{M}) / \Pr(\mathcal{D}) \tag{6.36}$$

Dans cette équation :
- $\Pr(\mathcal{M}|\mathcal{D})$ est la probabilité a posteriori du modèle ou encore la vraisemblance des données,
- $\Pr(\mathcal{D}|\mathcal{M})$ est la vraisemblance du modèle,
- $\Pr(\mathcal{M})$ est la probabilité a priori du modèle,
- $\Pr(\mathcal{D})$ est la probabilité a priori des données.

Maximiser la probabilité du modèle pour un jeu de données particulier revient à minimiser l'expression suivante (on passe en $\log$ et on multiplie par -1) :

$$-\log(\Pr(\mathcal{M}|\mathcal{D})) = -\log(\Pr(\mathcal{D}|\mathcal{M})) - \log(\Pr(\mathcal{M})) + \text{Cste} \tag{6.37}$$

En identifiant avec la longueur de description totale on voit que :
- $-\log(\Pr(\mathcal{D}|\mathcal{M}))$, la log-vraisemblance du modèle, est l'erreur de reconstruction
- $-\log(\Pr(\mathcal{M}))$, le log de la probabilité a priori du modèle, est le coût de complexité.

(Notons que $-\log_2(\Pr(E))$ est, par définition la quantité d'information associée à la réalisation de l'événement $E$ (voir l'annexe sur la théorie de l'information), mesurée en bit (sinon il s'agit de Neper ou de Hartley)).

On va donc choisir une probabilité a priori des poids qui forcera ceux-ci à se trouver au voisinage de 0 tout en autorisant cependant l'existence de poids de forte valeur. [WHR92] propose la distribution suivante :

$$\Pr(\mathcal{M}) \propto \left[\exp\left(-\frac{w_i^2/w_0^2}{1+w_i^2/w_0^2}\right)\right]^{\lambda} \tag{6.38}$$

où $w_0$ est un paramètre à ajuster.

Le risque qu'il faut alors minimiser est :

$$\sum_{\text{motifs k}} (desir_k - sortie_k)^2 + \lambda \sum_i \frac{w_i^2/w_0^2}{1+w_i^2/w_0^2} \tag{6.39}$$

Le second terme mesure la taille du réseau ; la somme s'étend à tous les poids du réseau sauf les poids de biais (en effet ces derniers mesurent la distance à l'origine de la droite séparatrice que réalise un neurone, et ils doivent être aussi grands que nécessaire). $\lambda$ représente l'importance relative du terme de complexité par rapport au terme de performance.

Sur la figure 6.15-a nous avons représenté comment varie ce terme de complexité lorsque $w$ varie

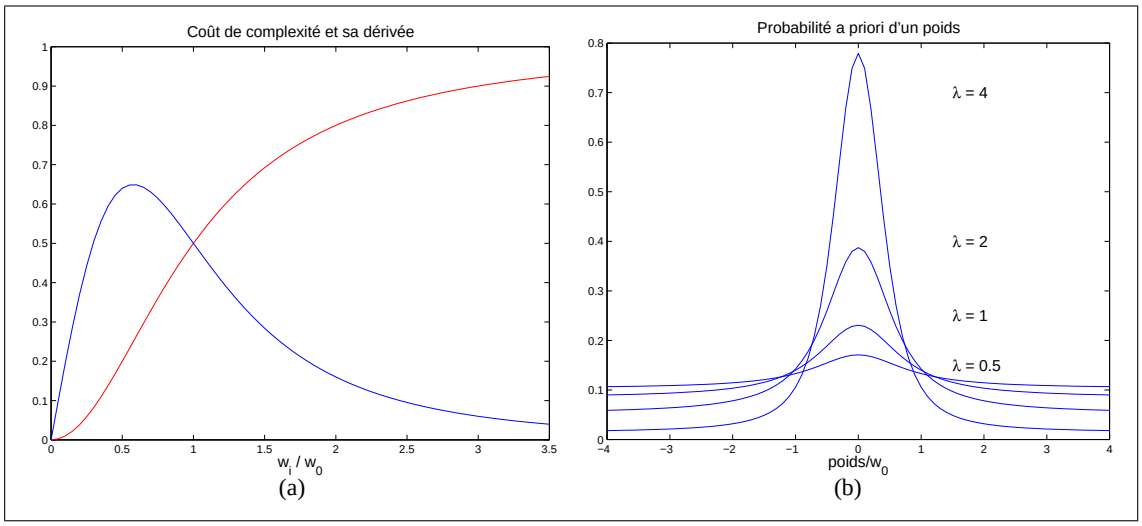



FIG. 6.15 – Variation du coût du module d'un poids (et de sa dérivée) (a) et variation de la probabilité a priori d'un poids (b)

(en unité $w_0$). Lorsque $w$ est nul, le coût correspondant est nul. Lorsque $|w|$ est grand, le coût correspondant vaut 1, le terme de complexité compte ainsi le nombre de poids non nuls, ce qui est bien ce que nous recherchons.

Sur la figure 6.15-b, nous avons représenté la probabilité a priori du modèle pour différentes valeurs de $\lambda$, la normalisation de la probabilité étant réalisée en prenant pour borne de $|w|$ la valeur $4w_0$. Pour les faibles valeurs de $\lambda$, on remarque que tout se passe comme si les poids étaient tirés de 2 distributions : une distribution plutôt plate d'où devraient être tirés les poids qui servent à quelque chose et une distibution piquée autour de 0 d'où devraient être tirés les poids qui ne servent à rien.

Si nous approximons le pic de distribution autour de 0 par une gaussienne, nous obtenons :

$$\lim_{w \to 0} \exp\left(-\lambda \frac{w_i^2/w_0^2}{1 + w_i^2/w_0^2}\right) = \exp\left(-\frac{w_i^2}{w_0^2/\lambda}\right) \tag{6.40}$$

La variance de cette gaussienne est donc $\sigma^2 = w_0^2/(2\lambda)$, ce qui signifie que le paramètre $\lambda$, qu'on aurait pu penser n'être là que pour peser le terme de complexité, a une influence plus subtile : la variance du bruit lui est inversement proportionnelle. Plus $\lambda$ est grand, plus un poids doit être proche de 0 pour qu'il ait une probabilité raisonnable de faire partie de la distribution de bruit. De plus, plus $\lambda$ est grand, plus les poids seront tous petits.

Pour clarifier la signification de $w_0$ [WRH91] proposent de considérer un neurone relié - de manière redondante - à la même source. Est-il plus économique d'avoir deux poids petits ou un poids grand et un poids petit ? La réponse est très intéressante et est illustrée sur la figure 6.16 où l'on

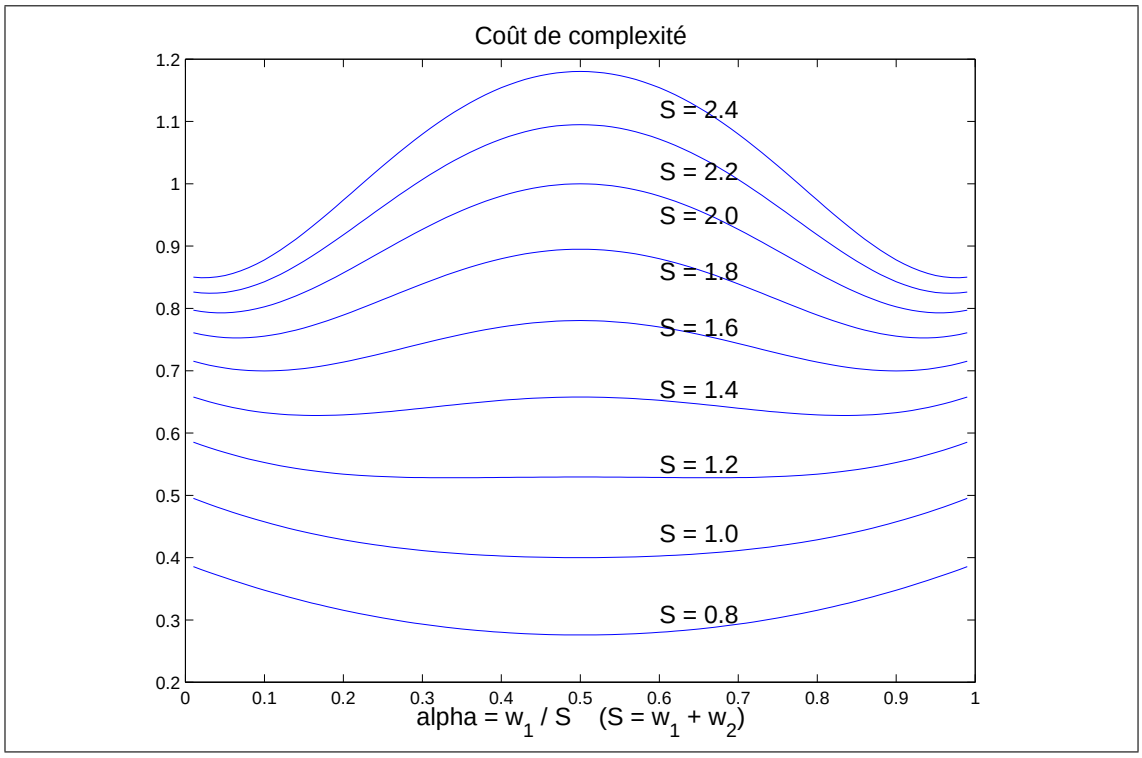



FIG. 6.16 – Variation du coût de complexité pour un neurone connecté à la même source par deux poids $w_1$ et $w_2$

voit que cela dépend de la somme des poids $S = w_1 + w_2$. Pour des valeurs de $S/w_0$ allant jusqu'à 1.1, il n'existe qu'un minimum à $\alpha = w_1/S = 0.5$, ce qui signifie que les deux poids ont des valeurs égales. Lorsque $S/w_0$ augmente, la symétrie est brisée et il est moins coûteux d'avoir un poids $\approx S$ et l'autre $\approx$ nul

Choisir un grand $w_0$ conduit à beaucoup de petits poids, choisir un faible $w_0$ permet à quelques poids de rester grand. Une valeur de $w_0$ de l'ordre de 1 est expérimentalement une bonne valeur lorsque les activations des neurones sont de l'ordre de 1.

### Implémentation

Dans son esprit la procédure d'élimination des poids proposé par Weigend *et all* est simple, mais en pratique elle est très sensible au choix de $\lambda$. Si $\lambda$ est trop petit, elle n'aura aucun effet et si $\lambda$ est trop grand, tous les poids seront tirés vers 0. De plus, une valeur de $\lambda$ bonne pour un problème donné ne le sera plus pour un autre problème. Pire encore, pour un problème donné, l'apprentissage peut être difficile dans une région (au début par exemple) et nécessiter une valeur faible de $\lambda$, et plus facile ensuite (et nécessiter une plus grande valeur).

La procédure s'appliquera à tous les poids sauf les biais.

On part de $\lambda = 0$, le réseau peut donc utiliser toutes ses ressources et on modifie $\lambda$ après chaque époque en se basant sur l'erreur $E_t$ obtenu à la fin de l'époque $t$ ($E_t$ est la partie "mesure de performance" de la fonction de coût globale). $E_t$ peut augmenter ou diminuer puisque la descente de gradient minimise la somme des deux termes de la fonction de coût.

On compare $E_t$ à trois termes :

- $E_{t-1}$, l'erreur à l'époque précédente,
- $A_t$, l'erreur moyenne pondérée et définie par $A_t = \gamma A_{t-1} + (1-\gamma)E_t$, avec $\gamma$ voisin de 1,
- $D$, un critère, une erreur désirée imposée. Elle pose problème car il est nécessaire de l'estimer en mesurant l'erreur obtenue sur l'ensemble de test, juste avant que la sur-estimation se produise, et en posant $D$ inférieur à cette erreur.

Après chaque époque, on évalue si $E_t$ est au-dessus ou au-dessous de chacune de ces quantités, ce qui donne 8 possibilités.

1. $(E_t < E_{t-1})$ et $(E_t < A_t)$ et $(E_t < D)$
2. $(E_t < E_{t-1})$ et $(E_t < A_t)$ et $(E_t > D)$
3. $(E_t < E_{t-1})$ et $(E_t > A_t)$ et $(E_t < D)$
4. $(E_t < E_{t-1})$ et $(E_t > A_t)$ et $(E_t > D)$
5. $(E_t > E_{t-1})$ et $(E_t < A_t)$ et $(E_t < D)$
6. $(E_t > E_{t-1})$ et $(E_t < A_t)$ et $(E_t > D)$
7. $(E_t > E_{t-1})$ et $(E_t > A_t)$ et $(E_t < D)$
8. $(E_t > E_{t-1})$ et $(E_t > A_t)$ et $(E_t > D)$

3 actions sont alors possibles :

- $\lambda \leftarrow \lambda + \Delta\lambda$
  On augmente légèrement $\lambda$ dans les 6 cas (1,2,3,4,5,7) où tout se passe bien : l'erreur est plus petite que le critère ($E_t < D$) ou elle décroit ($E_t < E_{t-1}$). Augmenter $\lambda$ signifie attacher plus d'importance au terme de complexité et rend la gaussienne plus pointue. La valeur initiale de $\Delta\lambda$ est petite, de l'ordre de $10^{-6}$.
- $\lambda \leftarrow \lambda - \Delta\lambda$
  Dans les deux cas restants l'erreur a augmenté et elle est plus grande que le critère. Lorsqu'elle reste plus faible que l'erreur moyenne ($E_t < A_t$, cas 6) on diminue un peu $\lambda$.
- $\lambda \leftarrow 0.9\lambda$
  Mais si elle est plus grande que la valeur moyenne ($E_t > A_t$, cas 8), alors on la diminue fortement.

## 6.23 Régularisation : weight decay

La procédure de weight decay proposé par Hinton et Le Cun en 1987 consiste à poser un coût de complexité $\propto w_i^2$. Dans la communauté statistique cela s'appelle *ridge regression*[2], il s'agit d'un cas particulier de la méthode proposée par Weigend, celui où $w_0$ est grand et où donc on se place dans la partie gauche de la figure 6.15-a.

## 6.24 A propos des données

Partant de l'observation qu'un prédicteur ou un classifieur donné fera des erreurs sur certaines parties de l'espace d'entrée et qu'un autre prédicteur ou classifieur fera des erreurs sur d'autres parties, il est naturel de combiner les résultats de chacun pour obtenir une classification ou une prédiction qui, en moyenne par exemple, sera meilleure (FIG. 6.17). La décision collective doit être meilleure que chacune des décisions individuelles.

Si la sortie désirée est numérique (régression), la combinaison peut simplement consister à prendre la moyenne de toutes les sorties. Si la sortie désirée est un label (classification), un mécanisme de vote majoritaire donnera le résultat souhaité. Notons qu'il est auusi possible d'utiliser une combinaison pondérée des sorties de chacun des réseaux.

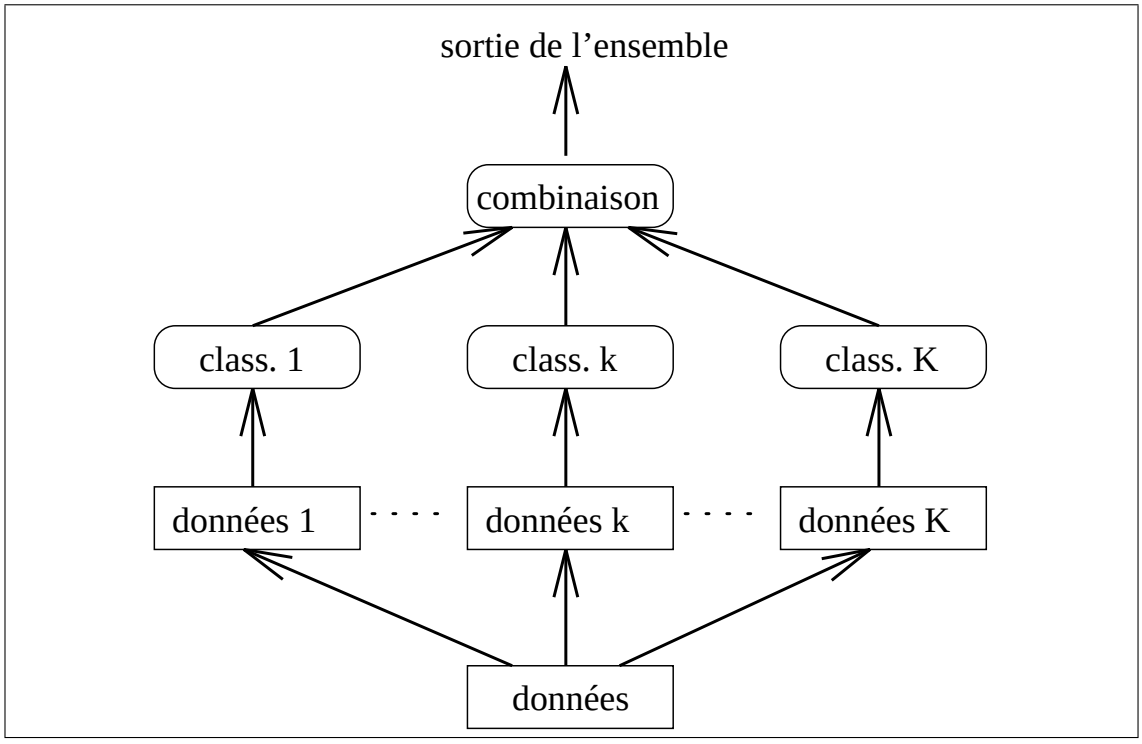



FIG. 6.17 – Architecture de principe d'un ensemble de $K$ classifieurs

Hansen et Salamon [HS90] ont montré que si les classifieurs sont indépendants et si chacun donne la réponse correcte plus de 50 % du temps, alors l'erreur réalisée par l'ensemble tend vers zéro quand le nombre de classifieurs tend vers l'infini.

[2] Paraît-il. Qui peut me dire à quoi cela correspond ?

Krogh et Vedelsby [KV95] ont de plus montré que l'erreur réalisée par un ensemble se décompose en deux termes, le premier mesurant l'erreur de généralisation moyenne de chacun des classifieurs et le second mesurant le désaccord entre les classifieurs.

Cette construction d'*ensembles* (le mot semble être issu de la physique) de classifieurs est basée sur des techniques de rééchantillonnage de la base de données parmi lesquelles les plus populaires sont le *bagging* et le *boosting*. Dans le boosting, l'ensemble d'apprentissage dépend des résultats des classifieurs précédents, dans le bagging non.

On montre que le bagging est efficace quel que soit le type de classifieur, arbre de décision ou réseau de neurones par exemple, mais que le boosting peut conduire à des résultats très variables selon les problèmes.

Dans le cas des réseaux de neurones une alternative très efficace au bagging est de constituer un ensemble de réseaux identiques où chacun est entraîné sur la totalité de la base d'apprentissage, mais à partir de poids initiaux différents.

### 6.24.1 Bagging

Bagging est un acronyme proposé par [Bre94] et vient de "**b**ootstrap **agg**regat**ing**". Il s'agit d'une méthode de bootstrap qui consiste à générer de multiples versions d'un prédicteur et à "aggréger" ceux-ci d'une certaine façon pour constituer un *ensemble*.

Soit une base d'apprentissage contenant $M$ individus. Chaque classifieur de l'ensemble est entraîné sur une base d'apprentissage contenant aussi $M$ individus mais ou chaque individu est tiré aléatoirement, avec remise, dans la base d'origine.

Un exemple donné a la probabilite $1 - (1 - 1/M)^M$ d'être au moins une fois dans la base d'apprentissage. Si $M$ est grand cette probabilité vaut approximativement $1 - 1/e$ = 63%. C'est-à-dire que chaque base d'apprentissage contient environ 63% des exemples de la base de départ.

La base résultante peut donc contenir des exemples identiques et ne pas contenir certains exemples de la base d'origine. Chaque classifieur de l'ensemble est ainsi construit sur un rééchantillonnage aléatoire de la base d'origine.

La table 6.1 tirée de [OM99] montre comment sont obtenues les diverses bases d'apprentissage sur un exemple jouet qui suppose que l'ensemble d'apprentissage d'origine contient les individus 1, 2, 3, 4, 5, 6, 7, 8.

| base d'origine | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| base n$^o$ 1 | 2 | 7 | 8 | 3 | 7 | 6 | 3 | 1 |
| base n$^o$ 2 | 7 | 8 | 5 | 6 | 4 | 2 | 7 | 1 |
| base n$^o$ 3 | 3 | 6 | 2 | 7 | 5 | 6 | 2 | 2 |
| base n$^o$ 4 | 4 | 5 | 1 | 4 | 6 | 4 | 3 | 8 |

TAB. 6.1 – Exemple d'ensemble d'apprentissage pour le bagging

Breiman [Bre94] prétend que les réseaux de neurones et les arbres de décision sont des algorithmes d'apprentissage instables (c'est-à-dire pour lesquels un petit changement dans la base d'apprentissage peut conduire à de grandes variations dans les résultats) et que, sur ces algorithmes, les techniques de bagging sont très efficaces. En revanche, sur un algorithme stable (les k plus proches voisins par exemple), le bagging peut dégrader les performances.

**Pourquoi le bagging marche ?**

Soient une base d'apprentissage $L$ et $(\mathbf{x}, y)$ un exemple tiré dans $L$ suivant la loi de probabilité $P(\mathbf{x}, y)$ indépendante de $L$. Supposons que $y$ est numérique et soit $\phi(\mathbf{x}, L)$ le prédicteur. Alors le prédicteur réalisé par l'ensemble est :

$$\phi_e(\mathbf{x}, P) = E_L[\phi(\mathbf{x}, L)] \tag{6.41}$$

L'erreur réalisée par l'ensemble est :

$$\epsilon_e = E_{\mathbf{x},y}[(y - \phi_e)^2] \tag{6.42}$$

L'erreur moyenne de prédiction de chacun des prédicteurs est :

$$\begin{aligned}
\epsilon &= E_L\left[E_{\mathbf{x},y}[(y-\phi_e)^2]\right] && (6.43)\\
&= E_L\left[E_{\mathbf{x},y}[y^2] - 2E_{\mathbf{x},y}[y\phi] + E_{\mathbf{x},y}[\phi^2]\right]\\
&= E_{\mathbf{x},y}[y^2] - 2E_{\mathbf{x},y}[yE_L[\phi]] + E_{\mathbf{x},y}[E_L[\phi^2]]\\
&= E_{\mathbf{x},y}[y^2] - 2E_{\mathbf{x},y}[y\phi_e] + E_{\mathbf{x},y}[E_L[\phi^2]]\\
&\geq E_{\mathbf{x},y}[y^2] - 2E_{\mathbf{x},y}[y\phi_e] + E_{\mathbf{x},y}[E_L[\phi]^2] \quad \text{car} \quad E[z^2] \geq (E[z])^2\\
&\geq E_{\mathbf{x},y}[(y-\phi_e)^2] = \epsilon_e && (6.44)
\end{aligned}$$

L'erreur quadratique moyenne de l'ensemble $\phi_e$ est plus petite que celle de $\phi$. L'amélioration dépend de l'inégalité :

$$(E_L[\phi(\mathbf{x}, L)])^2 \leq E_L[\phi(\mathbf{x}, L)^2] \tag{6.45}$$

Plus $\phi$ varie selon les ensembles $L$, c'est-à-dire plus l'algorithme qui les produit est instable, plus importante sera l'amélioration.

### 6.24.2 Boosting

La technique de boosting regroupe toute une famille de méthodes. Elles ont été étudiées par Schapire et Freund. Elle visent à améliorer, « booster », la précision de n'importe quelle méthode d'apprentissage.

Le but de ces méthodes est de construire une *série* de classifieurs où chaque classifieur de la série est construit sur une base d'apprentissage qui dépend des performances du classifieur *précédent*. C'est-à-dire que les exemples mal prédits du classifieur précédent sont retirés plus souvent que les autres exemples. Au départ, les probabilités de tirer chaque exemple sont égales à $1/M$, $M$ étant le nombre d'exemples.

Il y a deux grandes techniques de boosting dépendant de la façon dont les exemples sont tirés.

**AdaBoost**

*AdaBoost* (**Ada**ptive **Boost**ing) [FS96] [Sch96] construit successivement $K$ classifieurs où chacun tire ses exemples avec une certaine probabilité dépendant des erreurs faites par le classifieurs précédent. Soit $D_k(i)$ la probabilité de tirer l'exemple $i$ au cours de l'itération $k$ avec $D_1(i) = 1/M$.

- Soit $\epsilon_k$ la somme des probabilités des exemples mal classés par le classifieur $C_k$.

- On pose alors :

$$\alpha_k = \frac{1}{2} \ln \left( \frac{1 - \epsilon_k}{\epsilon_k} \right) \tag{6.46}$$

- On actualise :

$$\begin{aligned} D_{k+1}(i) &= \frac{D_k(i)}{Z_k} \times \begin{cases} \exp(-\alpha_t) & \text{si} \quad \phi_k(x_i) = y_i \\ \exp(\alpha_t) & \text{si} \quad \phi_k(x_i) \neq y_i \end{cases} \\ &= \frac{D_k(i).\exp(-\alpha_k y_i \phi_k(x_i))}{Z_k} \end{aligned} \tag{6.47}$$

  où $Z_k$ est un facteur de normalisation calculé sur tous les exemples de façon à ce que $D_{t+1}$ soit une probabilité.
- le résultat final est :

$$\Phi(x) = \text{sign}(\sum_{k=1}^{K} \alpha_k \phi_k(x)) \tag{6.48}$$

AdaBoost combine ainsi les classifieurs $C_1, \ldots, C_K$ en utilisant un vote pondéré. Cette façon de fonctionner présente l'intérêt que tous les exemples de la base d'apprentissage seront sélectionnés.

En reprenant l'exemple ci-dessus et en supposant que le motif 1 est difficile à apprendre, nous obtenons le tableau 6.2.

| base d'origine | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| base n$^o$ 1 | 2 | 7 | 8 | 3 | 7 | 6 | 3 | 1 |
| base n$^o$ 2 | 1 | 4 | 5 | 4 | 1 | 5 | 6 | 4 |
| base n$^o$ 3 | 7 | 1 | 5 | 8 | 1 | 8 | 1 | 4 |
| base n$^o$ 4 | 1 | 1 | 6 | 1 | 1 | 3 | 1 | 5 |

TAB. 6.2 – Exemple d'ensemble d'apprentissage pour le boosting

### Arcing

*Arcing* (**A**daptively **r**esample and **c**ombine) regarde les exemples mal prédits des $K$ précedents classifieurs et tire avec remise les exemples qui serviront au classifieur $K+1$ avec une probabilité qui dépend du nombre de fois $m_i$ que ces exemples $x_i$ ont été mal classés par les $K$ précédents classifieurs.

$$p_i = \frac{1 + m_i^4}{\sum_{k=1}^{K} (1 + m_i^4)} \tag{6.49}$$

Breiman [Bre96] trouve empiriquement la puissance quatrième (c'est la raison pour laquelle on appelle souvent arcing *arc-x4*). La combinaison est réalisée par un vote pondéré. Arcing permet de réduire la variance plus efficacement que bagging.

### Comparaison

Une comparaison de bagging, boosting, arcing et d'autre variantes se trouve dans [BK99].

### 6.24.3 Biais et variance

En se basant sur l'article originel de Geman et al. [GBD92] concernant le dilemme biais-variance, Breiman [Bre96], Kohavi et Wolpert [KW96], ont proposé que l'erreur de classification (donc pour des systèmes où les sorties sont binaires ou sont des labels) pouvait se decomposer en deux termes :

1. Un terme de biais qui mesure la distance entre le classifieur moyen et le classifieur idéal,
2. un terme de variance qui mesure la distance entre les classifieurs,
3. un terme dû au bruit intrinsèque (terme de Bayes).

Bagging et boosting réduisent la variance, c'est assez intuitif, mais ils semblent aussi réduire le biais dans quelques problèmes réels. En fait, comme pour estimer le biais et la variance il faut connaître le bon classifieur, ces considérations sont de portée limitée.

## 6.25 Apprentissage statistique et généralisation dans les MLP

Notons $x \in X \subset \boldsymbol{R}^p$ l'entrée d'un réseau de neurones caractérisé par un point $w$ dans l'espace des poids et $y \in Y \subset \boldsymbol{R}^q$ la sortie. Soit alors $\xi^{(m)} = \{\xi_i,\ 1 \leq i \leq m\}$, où $\xi = (x, y)$.

On cherche à apprendre, à partir d'exemples, une relation inconnue $F$ qui lie les entrées aux sorties. Cette fonction peut être vue comme la densité de probabilité définie sur l'espace des paires entrée-sortie $X \otimes Y \subset \boldsymbol{R}^{p+q}$ :

$$\Pr_F(\xi) = \Pr_F(x, y) = \Pr_F(x).\Pr_F(y|x) \tag{6.50}$$

où $\Pr_F(x)$ définit la région d'intérêt des entrées et $\Pr_F(y|x)$ définit la relation statistique reliant les entrées aux sorties.

L'apprentissage du réseau de neurones sera alors vu comme un problème d'optimisation en introduisant une mesure de la qualité de l'approximation $F_w$ de la fonction $F$ réalisée par le réseau.

La fonction d'erreur à minimiser, qui mesure la dissemblance entre $F$ et $F_w$ sur l'ensemble restreint d'apprentissage est :

$$\mathbf{E}_w^{(m)} = \mathbf{E}(\xi^{(m)}|w) = \sum_{i=1}^{m} \mathbf{e}(y_i|x_i, w) \tag{6.51}$$

La fonction d'erreur $\mathbf{e}(y_i|x_i, w)$ est une mesure de distance sur $\boldsymbol{R}^q$ entre la sortie désirée $y$ et la sortie du réseau sachant qu'on a $x$ en entrée. On peut l'écrire

$$\mathbf{e}(y_i|x_i, w) = \mathbf{d}(y, F_w(x)) \tag{6.52}$$

### 6.25.1 Minimisation de l'erreur et maximum de vraisemblance

Le problème de l'apprentissage optimal d'une règle peut être vue selon un angle bayésien [WR93], mais plus souvent, on le voit [LTS90] comme le problème de trouver l'ensemble des $w$ qui maximise la vraisemblance de l'ensemble d'apprentissage :

$$\max_{w \in \mathbf{w}} \Pr(\xi^{(m)}|w) = \prod_{i=1}^{m} \Pr_F(x_i) \cdot \prod_{i=1}^{m} \Pr_F(y_i|x_i, w) \tag{6.53}$$

On veut que la minimisation de l'erreur (6.51) soit équivalente à la maximisation de la vraisemblance (6.53) pour tous les ensembles d'apprentissage indépendants $\xi^{(m)}$. Ces deux critères d'optimisation ne peuvent être équivalents que s'ils sont reliés par une fonction monotone $\phi$ telle que :

$$\prod_{i=1}^{m} \Pr(y_i|x_i, w) = \phi\left(\sum_{i=1}^{m} \mathbf{e}(y_i|x_i, w)\right) \tag{6.54}$$

On montre [TTL84] que la seule solution à cette équation fonctionnelle est donnée par :

$$\Pr(y|x, w) = \frac{1}{z(\beta)} \exp(-\beta \mathbf{e}(y|x, w)) \tag{6.55}$$

$$\text{avec } z(\beta) = \int_Y \exp(-\beta \mathbf{e}(y|x, w)) \tag{6.56}$$

où $\beta$ est une constante positive qui détermine la sensibilité de la probabilité $\Pr(y|x, w)$ à l'erreur. L'erreur moyenne est :

$$\bar{\mathbf{e}} = \int \mathbf{e}(y|x, w) \Pr(y|x, w)\, dy \tag{6.57}$$

elle est reliée à $\beta$ par la relation :

$$\bar{\mathbf{e}} = -\frac{\partial \log(z)}{\partial \beta}; \quad \text{avec} \quad \frac{\partial \bar{\mathbf{e}}}{\partial \beta} < 0 \tag{6.58}$$

Lorsque la fonction d'erreur est quadratique $e(y|x, w) = (y - F_w(x))^2$, la probabilité $\Pr(y|x, w)$ est gaussienne :

$$\Pr(y|x, w) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp[-\frac{(y - F_w(x))^2}{2\sigma^2}] \tag{6.59}$$

où $\beta = 1/(2\sigma^2)$. Dans ce cas on a : $z(\beta) = \sqrt{\pi/\beta}$ et $\bar{\mathbf{e}} = \sigma^2$ indépendants du réseau $w$ ainsi que de l'entrée $x$.

### 6.25.2 Distribution de Gibbs

Il est maintenant possible d'inverser la vraisemblance (6.53) en utilisant la formule de Bayes pour obtenir la probabilité des poids connaissant toutes les paires d'entrée-sortie $\xi^{(m)}$ :

$$\rho^{(m)}(w) \equiv \Pr(w|\xi^{(m)}) \tag{6.60}$$

$$= \frac{\Pr(\xi^{(m)}|w) \Pr(w)}{\int_{\mathbf{w}} \Pr(\xi^{(m)}|w) \Pr(w)} \tag{6.61}$$

$$= \frac{\rho^{(0)}(w) \prod_{i=1}^{m} \Pr(y_i|x_i, w)}{\int_{\mathbf{w}} \rho^{(0)}(w) \prod_{i=1}^{m} \Pr(y_i|x_i, w)\, dw} \tag{6.62}$$

où $\rho^{(0)}$ est une distribution a priori de l'espace de configuration.

On peut réécrire (6.62) en faisant apparaître l'erreur d'apprentissage $\mathbf{E}(\xi^{(m)}|w)$ sous la forme de la distribution canonique de Gibbs :

$$\rho^{(m)}(w) = \frac{1}{\mathbf{Z}^{(m)}} \rho^{(0)}(w) \exp[-\beta \mathbf{E}^{(m)}(w)] \tag{6.63}$$

où l'intégrale de normalisation :

$$\mathbf{Z}^{(m)}(\beta) = \int_{\mathbf{w}} \rho^{(0)}(w) \exp[-\beta \mathbf{E}^{(m)}(w)] \, dw \tag{6.64}$$

est appelée *fonction de partition* en physique statistique[3] et mesure le volume de l'espace de configuration accessible au système. Par analogie, l'erreur se comporet comme une énergie.

L'équation 6.63 montre que l'apprentissage modifie la distribution de probabilité des poids. En effet, l'apprentissage d'un exemple supplémentaire $\xi_{m+1}$ revient à multiplier la distribution $\rho^{(m)}$ par $\exp[-\beta \mathbf{e}(y_{m+1}|x_{m+1}, w)]$ et à renormaliser, ce qui conduit à :

$$\begin{aligned} \mathbf{Z}^{(m+1)}(\beta) &= \int_{\mathbf{w}} \rho^{(0)}(w) \exp[-\beta \mathbf{E}^{(m)}(w) - \beta \mathbf{e}(y_{m+1}|x_{m+1}, w)] \, dw & (6.65) \\ &\leq \int_{\mathbf{w}} \rho^{(0)}(w) \exp[-\beta \mathbf{E}^{(m)}(w)] \, dw \quad = \quad \mathbf{Z}^{(m)} & (6.66) \end{aligned}$$

L'apprentissage conduit donc à une diminution du volume occupé par les poids dans l'espace de configuration ou, ce qui revient au même, à une augmentation monotone de l'énergie libre $\beta \mathbf{F} = -\log \mathbf{Z}^{(m)}$ lorsque la taille $m$ de l'ensemble d'apprentissage augmente. C'est cette énergie libre qui détermine l'erreur d'apprentissage moyenne :

$$\left\langle \mathbf{E}^{(m)} \right\rangle = \int_{\mathbf{w}} \rho^{(m)}(w) \mathbf{E}^{(m)}(w) \, dw = -\frac{\partial \log \mathbf{Z}^{(m)}}{\partial \beta} \geq 0 \tag{6.67}$$

aussi bien que les fluctuations de l'ensemble autour de cette erreur :

$$\frac{\partial < \mathbf{E} >}{\partial \beta} = -\frac{\partial^2 \log \mathbf{Z}}{\partial \beta^2} = -\left\langle (\mathbf{E} - < \mathbf{E} >)^2 \right\rangle \quad < 0 \tag{6.68}$$

L'erreur moyenne sur l'ensemble d'apprentissage est donc une fonction décroissante de $\beta$ comme on s'y attendait.

Une caractérisation importante de l'apprentissage est la quantité d'information gagnée durant celui-ci. Une mesure simple de cette quantité est donnée par la distance de Kullback-Leibler entre la distribution des poids avant et après l'apprentissage :

$$\mathbf{S}^{(m)} \equiv D[\rho^{(m)}|\rho^{(0)}] = \int_{\mathbf{w}} \rho^{(m)}(w) \log \frac{\rho^{(m)}(w)}{\rho^{(0)}(w)} \, dw \quad \geq 0 \tag{6.69}$$

qui est l'entropie en thermodynamique et qui peut s'écrire aussi :

$$\mathbf{S}^{(m)} = -\log \mathbf{Z}^{(m)} - \beta \left\langle \mathbf{E}^{(m)} \right\rangle \tag{6.70}$$

Au cours de l'apprentisage cette entropie décroît à la fois parce que le volume $\mathbf{Z}^{(m)}$ décroît et que l'erreur d'apprentissage $< \mathbf{E}^{(m)} >$ décroît.

On peut voir $\beta$ comme le multiplicateur de Lagrange utilisé dans la minimisation de l'entropie relative 6.69 sous la contrainte de la minimisation de l'erreur moyenne $< \mathbf{E}^{(m)} >$.

[3] Vient de l'allemand ...xxx .

# Bibliographie


[BK99] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms : Bagging, Boosting and variants. *Machine Learning*, 36(1/2) :105–142, 1999.

[Bre94] Leo Breiman. Bagging predictors. Technical Report 421, Department of statistics, Univ. California, Berkeley, sept. 1994.

[Bre96] Leo Breiman. Bias, variance, and arcing classifiers. Technical Report 460, Department of statistics, Univ. California, Berkeley, april 1996. available at `http ://www.stat.berkeley/edu/users/breiman`.

[Cas89] M. Casdagli. Nonlinear prediction of chaotic time series. *physica D*, 35 :335–356, 1989.

[FS96] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Machine learning : Proceedings of the thirteenth International Conference*, pages 148–156, 1996. available at : http ://www.research.att.com/˜schapire.

[GBD92] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias-variance dilemma. *Neural Computation*, 4(1) :1–58, 1992.

[GW88] A. R. Gallant and H. White. There exists a neural network that does not make avoidable mistakes. In SOS Printing, editor, *IEEE Second International Conference on Neural Networks*, volume 1, pages 657–664, San Diego, 1988.

[HS90] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE Trans. on PAMI*, 12(10) :993–101, 1990.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2 :359–366, 1989.

[Jud87] Stephen Judd. Learning in networks in hard. In *IEEE First International Conference on Neural Networks*, volume II, pages 685–692, San Diego, Californie, 21-24 juin, 1987.

[KV95] Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. In Tesauro, Touretzky, and Leen, editors, *NIPS 7*, volume 7, pages 231–238. MIT press, 1995.

[KW96] Ron Kovahi and David H. Wolpert. Bias plus variance decomposition for zero-one loss function. In L. Saita, editor, *Machine learning proceedings of the Thirteenth International Conference*. Morgan Kaufmann, 1996. available at `http ://robotics.stanford.edu/users/ronnyk`.

[LTS90] Esther Levin, Naftali Tishby, and Sara A. Solla. A statistical approach to learning and generalization in layered neural networks. *Proceedings of the IEEE, special issue on Neural Networks II*, 78(10) :1568–1574, 1990.

[MP69] M.L. Minsky and S.A. Papert. *Perceptrons*. MIT Press, Cambridge, 1969.

[OM99] David Opitz and Richard Maclin. Popular ensemble methods : An empirical study. *Journal of Artificial Intelligence Research*, 11 :169–198, 1999. available at `http ://www.jair.org/contents/v11.html`.

[Ris87] Jorma Rissanen. Stochastic complexity. *Journal Royal Statistical Society B*, 49 :223–239, 1987. with discussion : 252–265.

[Sch96] Robert E. Schapire. A brief introduction to Boosting. In *Proceedings of the sixteenth Joint Conference on Artificial Intelligence*, pages 148–156, 1996. available at : http ://www.research.att.com/˜schapire.

[TMF96] Georg Thimm, Perry Moerland, and Emile Fiesler. The interchangeability of learning rate and gain in backpropagation neural networks. *Neural Computation*, 8 :451–460, 1996.

[TTL84] Y Tikochinsky, Naftali Tishby, and R. D. Levine. Alternative approach to maximum entropy inference. *Phys. Rev. A*, 30 :2638–2644, 1984.

[WHR92] Andreas S. Weigend, Bernado A. Huberman, and David E. Rumelhart. Predicting sunspots and exchange rates with connectionist networks. In Martin Casdagli and Stephen Eubank, editors, *Nonlinear modeling and forecasting*, pages 395–431. Addison Wesley, 1992.

[WR93] Timothy L. H. Watkin and Albrecht Rau. The statistical mechanics of learning a rule. *Rev. Mod. Phys.*, 65(2) :499–555, 1993.

[WRH91] Andreas S. Weigend, David E. Rumelhart, and Bernado A. Huberman. Generalization by weight-elimination with application to forecasting. In Moody Lippmann and Touretzky, editors, *Advance in NIPS, 3*, pages 875–882. Morgan Kaufmann, 1991.