I

Les Réseaux de Neurones

2. Présentation des réseaux neuronaux

On pense communément que "l'ordinateur de l'avenir" sera massivement parallèle et tolérera les erreurs. Toutefois la conception d'une telle machine s'étant avérée étonnamment difficile, nous aurions abandonné depuis longtemps si le cerveau n'était pas une preuve vivante que le traitement parallèle et tolérant les erreurs est possible et très efficace.

John S. Denker, 1985
(Dans *Les Rêves de la Raison*, Heinz Pagels, pp.118, InterEditions, 1990)

2.1. Introduction

Les réseaux de neurones artificiels ou réseaux connexionnistes sont fondés sur des modèles qui tentent d'expliquer comment les cellules du cerveau et leurs interconnexions parviennent, d'un point de vue globale, à exécuter des calculs complexes.

Ces systèmes qui stockent et retrouvent l'information de manière "similaire" au cerveau sont particulièrement adaptés aux traitements en parallèle de problèmes complexes comme la reconnaissance automatique de la parole, la reconnaissance de visages ou bien la simulation de fonctions de transfert. Ils offrent donc un nouveau moyen de traitement de l'information utilisé en reconnaissance de formes (vision, image, parole, etc).

Les architectures connexionnistes s'inspirent de l'organisation neuronale du cerveau humain. Dans les réseaux de neurones artificiels de nombreux processeurs appelés cellules ou unités, capables de réaliser des calculs élémentaires, sont structurés en couches successives capables d'échanger des informations au moyen de connexions qui les relient. On dit de ces unités qu'elles miment les neurones biologiques.

Grâce à ce parallélisme massif, on peut espérer pouvoir surmonter les problèmes posés par des temps d'attente importants caractéristiques à la résolution de tâches complexes par des méthodes numériques (tel que de tâches en reconnaissance de vissages, de voix, ...).

Nous donnerons dans ce chapitre les notions de base pour la compréhension des réseaux de neurones.

Ce chapitre est organisé de la façon suivante. Nous présenterons tout d'abord l'élément de base d'un réseau connexionniste : le neurone ou processeur élémentaire.

Ces éléments sont assemblés suivant une certaine architecture, dont nous discuterons le rôle, pour former un réseau. Cette architecture définit une composition de fonctions élémentaires qui peut être utilisée de plusieurs façons lors du fonctionnement du réseau, c'est ce qu'on appelle la dynamique du réseau.

Enfin, et c'est certainement un des points les plus importants, nous traitons de l'apprentissage dans ces réseaux, c'est à dire de la façon dont on fixe les paramètres des différents composants du réseau afin qu'il accomplisse une tâche donnée.

2.2. Les neurones

Un réseau connexionniste est constitué d'éléments extrêmement simples qui interagissent pour donner au réseau son comportement global. Dans les modèles connexionnistes, ces éléments sont des processeurs élémentaires dont la définition est faite en analogie avec les cellules nerveuses, les neurones.

Ces unités de base reçoivent des signaux provenant de l'extérieur ou d'autres neurones du réseau. Ils calculent une fonction, simple en général, de ces signaux et envoient à leur tour des signaux vers un ou plusieurs autres neurones ou vers l'extérieur. La Figure 2.1 montre un schéma comportant les organes principaux d'un neurone artificiel.

Nous caractérisons un neurone par trois concepts : son état, ses connexions avec d'autres neurones et sa fonction de transition. Les sections

suivantes décrivent ces concepts. Nous détaillerons ensuite la propagation des états des neurones à l'intérieur d'un réseau.

2.2.1. L'état des neurones

Un neurone artificiel est un élément qui possède un ***état*** interne. Il reçoit des signaux qui lui permettent éventuellement, de changer d'état.

Nous noterons **S** l'ensemble des états possibles d'un neurone. **S** pourra être par exemple {0, 1} où 0 sera interprété comme l'état inactif et 1 l'état actif. **S** pourra également prendre un nombre plus grand de valeurs {0, 1, …, *P*} pour une image en *P* + 1 niveaux de gris ou même, par extension, un continuum de valeurs [-1, 1] ou R tout entier. Dans une application en télédétection les valeurs correspondant aux signaux écho-radar peuvent être représentées à l'aide de neurones à valeurs continues. L'état d'un neurone peut alors être définit dans l'intervalle **S** = [-1, 1], où -1 représente la valeur minimum du signal, et 1 le maximum.

Un neurone possède une fonction qui lui permet de changer d'état en fonction des signaux qu'il reçoit : c'est sa fonction de transition.

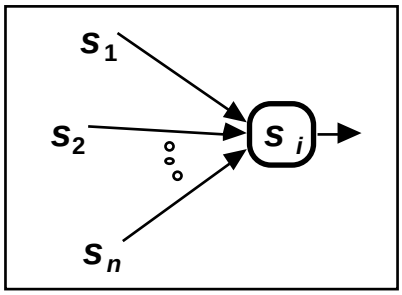


Figure 2.1 : *représentation d'un neurone. L'état s_i du neurone est fonction des entrées s_1, …, s_n. Le neurone produit une sortie qui sera transmise aux neurones reliés.*

L'état d'un neurone est fonction des états des neurones auxquels il est relié. Pour calculer l'état d'un neurone il faut donc considérer les ***connexions*** entre ce neurone et d'autres neurones. Nous définirons par la suite les connexions entre neurones et leur poids. Puis, nous parlerons du calcul de l'état d'un neurone.

2.2.2. Les connexions entre neurones

En 1943, Warren S. McCulloch et Walter Pitts [McCulloch et Pitts 43] ont introduit la notion de réseaux de neurones artificiels. Leur but était de représenter l'activité électrique des cellules nerveuses du cerveau. Les réseaux qu'ils ont proposés, appelés ***réseaux neuro-logiques***, étaient composés par l'interconnexion des petites unités élémentaires : les ***neurones formels***. Leur modèle était inspiré en partie des nouvelles théories mathématiques des automates à états finis de l'époque. Dans ce modèle, les neurones étaient arrangés d'une telle façon que le tout formait une machine capable, en particulier, de "reconnaître" des formes[1]. L'organisation des neurones dans les réseaux, autrement-dit l'architecture des réseaux, a été ici un facteur déterminant pour l'obtention de résultats intéressants.

Architecture est le terme le plus général pour désigner la façon dont sont disposés et connectés les différents neurones qui composent un réseau. On parle également de ***topologie*** (terme emprunté de la théorie des graphes). Au niveau des neurones on parle plutôt de ***voisinage***. Ce terme fait allusion à la façon dont un neurone est connecté à d'autres neurones. Il est donc en rapport direct avec l'architecture du réseau. Voyons de plus près la signification du mot voisinage dans une architecture de réseaux de neurones.

Le voisinage

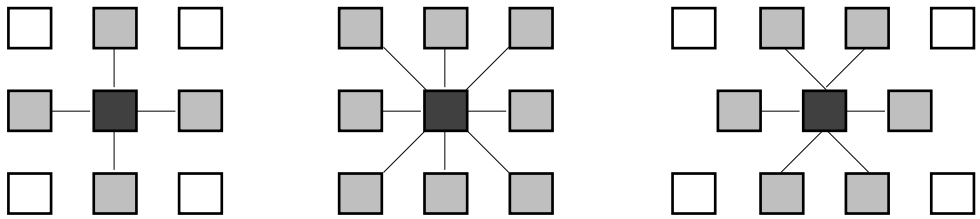
Le voisinage d'un neurone est l'ensemble des neurones connectés a ce neurone. On parle de ***voisinage d'ordre*** *n* pour un neurone *i*, s'il y a *n* neurones connectés à ce neurone. Les connexions entre neurones ont souvent un sens.

Dans la Figure 2.2 nous présentons des neurones avec des voisinages d'ordre 4, 8 et 6 respectivement. Les connexions utilisées ici n'ont pas de sens particulier, elles sont bidirectionnelles. Ces types de voisinages sont très utilisés, notamment dans les modèles à base d'automates cellulaires [Codd 68], [Fogelman-Soulié 85]. D'autres types de voisinage plus complexes

---

[1] Dans les réseaux utilisés par McCulloch et Pitts "reconnaître" une forme signifiait avoir une réponse "vraie" dans le neurone booléen de sortie. Ils pouvaient reconnaître des formes sur lesquelles on avait appliqué des transformations géométriques [McCulloch et Pitts 47].

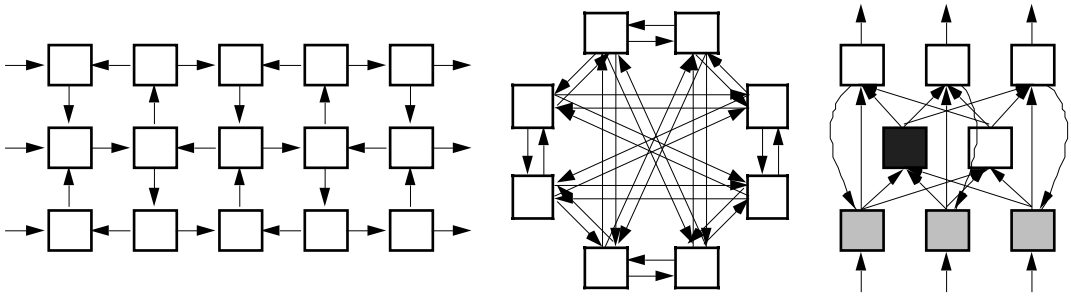
sont possibles (voir Figure 2.3). On peut par exemple envisager des connexions complètes entre neurones, on aura alors un voisinage d'ordre N, N étant le nombre total de neurones du réseau (Figure 2.3b).



(a) (b) (c)

Figure 2.2 : *plusieurs types de voisinage entre neurones. Les neurones en gris représentent le "voisinage" de celui du milieu.*

Différents types de voisinage permettent de définir des architectures de réseaux de neurones différentes. Par exemple, la Figure 2.3a montre un réseau où les neurones sont arrangés dans une grille. Ici les connexions relient les neurones ont un sens précis. Les voisins les plus proches dans le sens horizontal ou vertical sont les seuls reliés au neurone du centre. La Figure 2.3c présente des neurones organisés par couches. Les connexions sont exclusivement entre un ou plusieurs neurones appartenant à une couche du réseau et un neurone d'une couche différente. Il n'y a pas de connexions entre des unités appartenant à la même couche de neurones.



(a) (b) (c)

Figure 2.3 : *organisation et voisinage dans un réseau de neurones. (a) les connexions du réseau sont partielles, les neurones sont arrangés dans une grille et ils ont des connexions exclusivement avec les voisins proches; (b) voisinage complet, chaque neurone dans le réseau est connecté avec la totalité des neurones du réseau; (c) les neurones sont organisés par couches, il y a des connexions seulement entre les neurones de couches différentes. En gris est signalé le voisinage du neurone en noir.*

Les connexions

Une connexion est un lien établi explicitement entre deux neurones. Les connexions sont aussi appelées ***synapses***, en analogie avec le nom des connecteurs des neurones réels[2].

On note dans la Figure 2.3 que les liens entre neurones ont un sens, indiqué par une flèche (pour spécifier le sens bidirectionnel il faut définir deux connexions). Ce sens implique un flux d'information donc, une dépendance. En effet, l'état d'un neurone *i* est fonction des états des neurones *j* connectés à *i* (flèches allant de *j* à *i*). De même, les états des neurones *l* auxquels *i* est connecté (flèches allant de *i* à *l*) sont influencés par l'état de *i*. Ceci spécifie encore plus notre notion de voisinage qui peut être défini finalement de la façon suivante :

$$i,j \quad \mathbf{N}, si \quad LIEN(j,i) \quad j \quad VOISINAGE(i) \qquad (2.1)$$

Où **N** est l'ensemble des neurones du réseau; *LIEN(j,i)* est une fonction booléenne qui est "vraie" si et seulement si le lien entre les neurones *j* et *i*, dans les sens *j-i*, existe; et *VOISINAGE(i)* représente l'ensemble des neurones connectés à *i*. Notez cependant que les neurones *l* auxquels *i* est connecté ne sont pas compris dans le voisinage de *i* à moins qu'il existe de façon réciproque un lien *l-i*.

Une connexion entre deux neurones a une valeur numérique associée appelé ***poids de connexion***.

Les poids des connexions

Le poids de connexion w_{ij} entre deux neurones *j* et *i* peut prendre des valeurs discrètes dans Z ou bien continues dans R. L'information qui traverse la connexion sera affectée par la valeur du poids correspondent. Une connexion avec un poids $w_{ij} = 0$ est équivalente à l'absence de connexion.

---

[2] La partie principale d'une cellule nerveuse, ou neurone, est appelé ***soma***. Elle contient les composantes courants aux cellules. La membrane cellulaire qui la recouvre forme des ramifications appelés ***dendrites***. Pour communiquer, un neurone envoi des signaux vers d'autres neurones à travers une fibre appelée l'***axone***. Enfin, c'est à travers des formations spéciales au bout de l'axone, les ***synapses***, que le signal provenant du soma d'un neurone arrive aux dendrites, ou au soma d'autres neurones.

On définit une matrice des poids de connexions **W** où les lignes et les colonnes correspondent aux neurones et chaque valeur w_{ij} représente le poids de la connexion entre la cellule *j* et la cellule *i* du réseau.

2.2.3. La fonction de transition

Nous nous intéressons ici aux neurones qui calculent leur état à partir de l'information qu'ils recoivent. Nous utiliserons par la suite la notation suivante :

S : l'ensemble d'états possibles des neurones.

x_i : l'état d'un neurone *i*, où x_i **S**.

A_i : l'activité du neurone *i*.

w_{ij} : le poids de la connexion entre les neurones *j* et *i*.

L'***activité*** d'un neurone est calculée en fonction des états des neurones de son voisinage et des poids de leurs connexions, selon la formule suivante :

$$A_i = \sum_j w_{ij}\, x_j \qquad (2.2)$$

Comme il est illustré dans la Figure 2.4 l'état x_i du neurone *i* est une fonction son activité A_i :

$$x_i = f(A_i) \qquad (2.3)$$

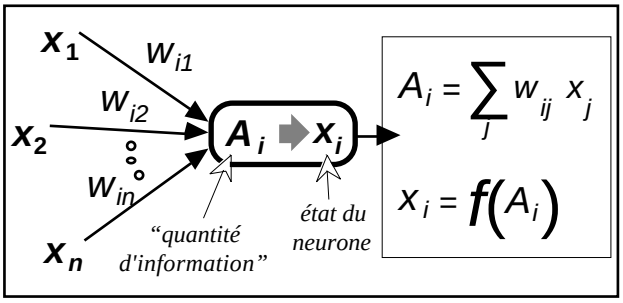


Figure 2.4 : *calcul de l'état d'un neurone. L'état* x_i *d'un neurone* i *est une fonction des états des neurones* j*, de son voisinage, et des poids des connexions* w_{ij}.

La fonction f, appelée ***fonction de transition*** peut avoir plusieurs formes différentes. L'ensemble des états possibles dépend, bien entendu, de la fonction de transition utilisée.

Nous décrivons par la suite les fonctions de transition les plus utilisées actuellement dans le cadre des réseaux neuronaux : la fonction identité, la fonction à seuil et la fonction sigmoïde.

La fonction identité

Les neurones dont la fonction de transition est la fonction identité (f^I) sont appelés ***automates linéaires***. Pour un tel automate, l'état est calculé à l'aide de l'équation suivante :

$$x_i = f^I(A_i) = A_i = \sum_j w_{ij}\, x_j \qquad (2.4)$$

La mise en œuvre de modèles de simulation fondés sur des automates linéaires est facilitée par la simplicité de leur fonction de transition (i.e. fonction identité). En effet, leur comportement peut être décrit à l'aide d'outils mathématiques empruntés à l'algèbre linéaire. Nous remarquons que cette fonction admet des valeurs non bornées pour les états, ce qui peut entraîner des débordements lors des simulations. Les automates linéaires sont employés, entre autres, par T. Kohonen pour construire son modèle de mémoires associatives [Kohonen 84].

La fonction à seuil

Si la fonction de transition est égale à une fonction à seuil (f^S) on parle d'***automates à seuil***. Ces automates ont été utilisés par McCulloch et Pitts dans leur modèle d'automate formel [McCulloch et Pitts 43].

Pour ces automates les états x_i sont binaires. Les ensembles de valeurs possibles les plus couramment utilisées sont **S** = {-1, 1} et **S** = {0, 1}.

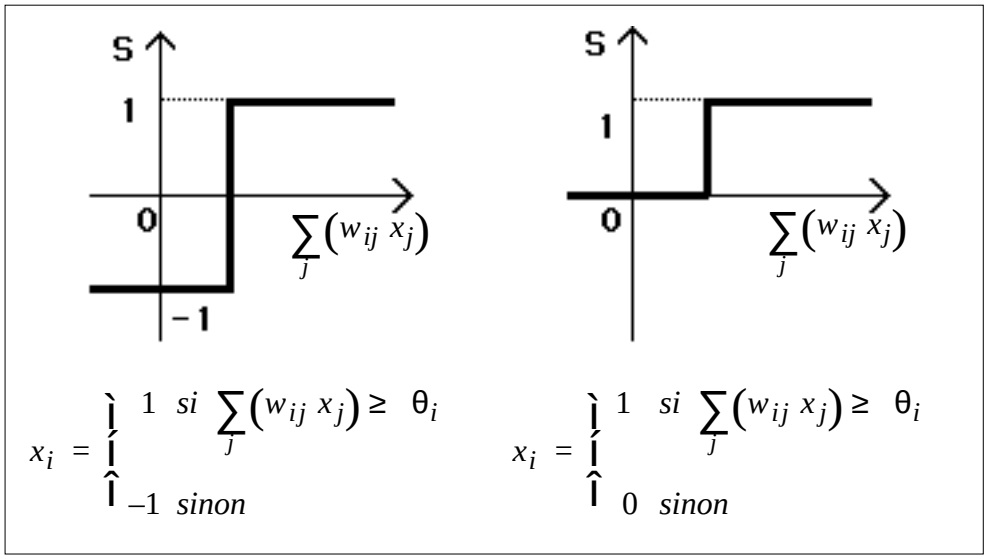
Dans un automate à seuil l'équation qui définit l'état du neurone (2.4) est :

$$x_i = f^S(A_i - \theta_i) = f^S\left(\sum_j w_{ij}\, x_j - \theta_i\right) \qquad (2.5)$$

où θ_i est ***le seuil***. Ainsi, pour l'ensemble **S** = {-1, 1} :

$$x_i = \begin{cases} 1 & si\ A_i \geq \theta_i \\ -1 & autrement \end{cases} \qquad (2.6)$$

Ce qui veut dire que l'état du neurone *i* est égal à -1 tant que l'activité A_i du neurone ne dépasse pas le seuil θ_i (voir la Figure 2.5).



(a) (b)

Figure 2.5 : *fonction à seuil. Sa valeur est 1 si* $\sum_i (w_i\, x_i) \geq \theta$*, sinon -1 (ou 0 dans le cas où* ***S*** *={0,1}).*

Il est également possible de définir des fonctions de transition multi-seuil. De telles fonctions permettent d'introduire des non-linéarités au niveau du fonctionnement du réseau ce qui peut être nécessaire pour résoudre des problèmes complexes. Leur défaut est d'introduire un grand nombre de discontinuités qui amènent un comportement instable pour le réseau et de ne pas permettre l'utilisation de valeurs continues pour les états. Une manière de dépasser ce handicap est d'utiliser une ***fonction sigmoïde*** qui est une fonction continue, différentiable et bornée.

La fonction sigmoïde

La fonction sigmoïde est inspirée directement de l'examen du comportement des cellules nerveuses face aux signaux qui leur arrivent. Une fonction sigmoïde a la forme suivante :

$$f(x) = a\frac{e^{Kx} - 1}{e^{Kx} + 1} \qquad (2.7)$$

Cette fonction, contrairement à la fonction identité, est une fonction bornée. En effet, elle tend vers a quand $x \rightarrow +\infty$ et tend vers $-a$ quand $x \rightarrow -\infty$. Le paramètre a régule alors la valeur de saturation; le paramètre K sert à réguler la pente de la courbe en tout point hors saturation (voir Figure 2.6). En particulier $\frac{aK}{2}$ spécifie la pente à l'origine.

Si l'on choisit les paramètres appropriés on peut simuler le comportement d'un automate à seuil. La Figure 2.6 illustre le cas où une valeur élevée de K a été choisie (voir courbe en pointillé). Dans cette figure nous montrons une famille de fonctions sigmoïde caractérisées par différentes valeurs de K. Notons que pour les valeurs des paramètres que nous avons utilisée pendant nos recherches, a = 1.7159 et K = 1.3333, on obtient $f(-1) = -1$ et $f(1) = 1$. Ces valeurs induisent un comportement proche du linéaire dans l'intervalle [-1, 1].

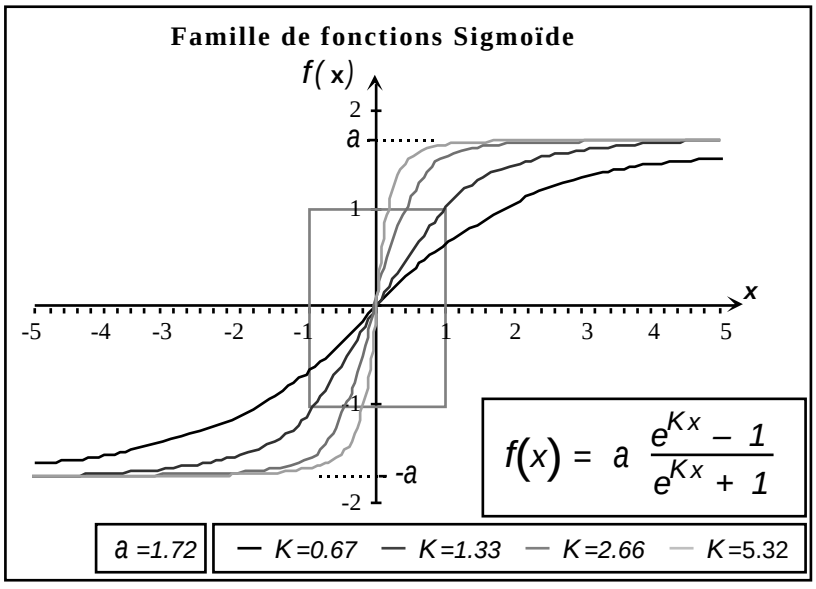


Figure 2.6 : *exemple d'une famille de fonctions sigmoïde. Pour a = 1.72 et K = 1.33 la fonction sigmoïde a un comportement proche d'une fonction linéaire dans l'intervalle [-1, 1].*

A la différence de la fonction à seuil la fonction sigmoïde est continue et différentiable. Elle est également non-décroissante. Nous l'appelerons fonction ***quasi-linéaire*** car elle est presque linéaire dans l'intervalle d'intérêt. Comme le montre la Figure 2.7, la fonction sigmoïde $f(\mathbf{x})$ et sa dérivée première $f'(\mathbf{x})$ sont toutes les deux des fonctions continues dans R. Ceci est fort utile lors des manipulations mathématiques de ces fonctions.

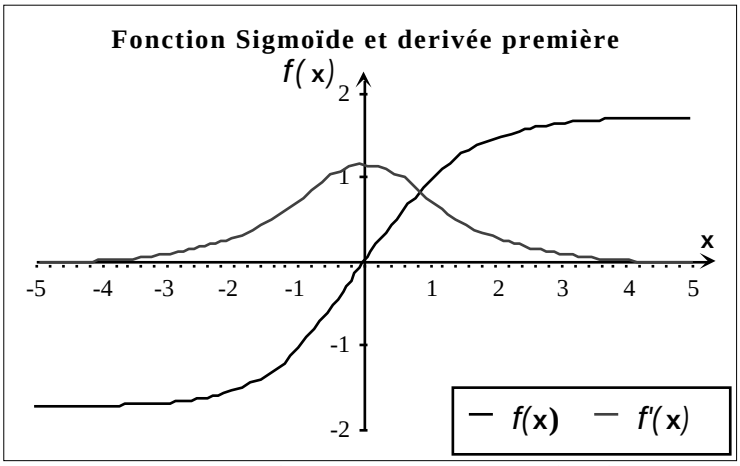


Figure 2.7 : *fonction sigmoïde f et dérivée première f'. f est une fonction quasi-linéaire : différentiable et non-décroissante. La dérivée f' est une fonction continue.*

Nous appelons les réseaux multicouches qui utilisent ce type de fonctions ***Réseaux Multicouches Quasi-linéaires*** (M.Q.L.). Dans nos modèles nous utilisons des réseaux M.Q.L. dont la fonction de transition est la fonction sigmoïde que nous venons de décrire.

2.3. Architectures simples

Dans la section précédente nous avons vu l'intérêt d'organiser les neurones dans des architectures particulières (cf. §2.2.2). Aussi, nous avons vu quelques exemples élémentaires d'architectures utilisées dans les modèles connexionnistes. Dans cette section nous décrivons un premier exemple d'architecture à couches souvent utilisée pour l'apprentissage supervisé (cf. chapitre 3) : les ***mémoires associatives***. Il s'agit d'une architecture simple qui a une dynamique de fonctionnement peu complexe.

2.3.1. Les mémoires associatives

Une mémoire associative est un système qui représente des associations entre deux séries de vecteurs : une série de vecteurs d'entrée $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$, définis dans $\mathbb{R}^n$, et une série de vecteurs de sortie $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m$, définis dans $\mathbb{R}^p$. Ces systèmes sont souvent implémentés à l'aide de réseaux neuronaux.

Le modèle présenté ici est celui de "Mémoire Associative Linéaire" proposé par Teuvo Kohonen [Kohonen 1984] (voir Figure 2.8). Il s'agit d'un réseau de neurones à deux couches totalement connectées et composées d'automates linéaires. Les états des automates de la première couche sont initialisés avec les éléments d'un des vecteurs de la série d'entrée. Ces états sont propagés vers les automates de la deuxième couche, qui représentent la réponse du réseau.

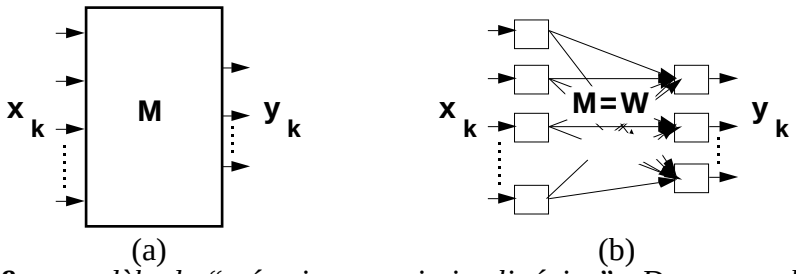


Figure 2.8 : *modèle de "mémoire associative linéaire". Dans un tel système on associe les vecteurs* $\mathbf{x}_k$ *et* $\mathbf{y}_k$*. Cette association est faite avec la matrice* **M***, selon la formule* $\mathbf{y}_k = \mathbf{M} \cdot \mathbf{x}_k$*. La figure (b) montre la représentation du modèle mémoire associative linéaire par un réseau de neurones où les automates en entrée sont complètement connectés aux automates linéaires en sortie, la matrice des poids des connexions* **W** *étant la même matrice d'association* **M***.*

Il est possible de modéliser le comportement du réseau à l'aide de l'algèbre linéaire. En effet, si $\mathbf{x}_k$ est le vecteur des signaux d'entrée défini dans $\mathbb{R}^n$, $\mathbf{y}_k$ le vecteur des signaux de sortie défini dans $\mathbb{R}^p$ et **M** est une matrice dans $\mathbb{R}^{p \times n}$, alors l'équation qui associe le couple de vecteurs $(\mathbf{x}_k, \mathbf{y}_k)$ est :

$$\mathbf{y}_k = \mathbf{M} \cdot \mathbf{x}_k \qquad (2.8)$$

Dans le cas d'un réseau d'automates linéaires à deux couches, la matrice **M** est la matrice de poids des connexions **W**. En réécrivant (2.8) en notation scalaire, on obtient pour un vecteur *k* l'équation suivante :

$$i \quad \mathbf{N}_{SORTIE}, \qquad y_i^k = \sum_{j \quad \mathbf{N}_{ENTRÉE}} w_{ij} \, x_j^k \qquad (2.9)$$

où $\mathbf{N}_{ENTRÉE}$ est l'ensemble des neurones de la couche d'entrée et $\mathbf{N}_{SORTIE}$ l'ensemble des neurones de sortie. Remarquons que cette équation correspond à la fonction de transition identité (2.4) que nous avons définie auparavant.

En conclusion, les mémoires associatives sont implémentées à l'aide de réseaux neuronaux ayant des architectures simples : ils n'ont que deux couches de cellules (la couche d'entrée et la couche de sortie). Ceci représente une limitation lors du traitement de problèmes complexes. Nous allons présenter maintenant des architectures à plus de deux couches. Comme nous le montrerons par la suite, ces architectures offrent une plus grande capacité à résoudre les problèmes complexes que nous sommes amenés à résoudre dans le cadre de nos applications.

2.4. Architectures multicouches

Dans la section précédente nous avons présenté une architecture de réseaux comportant deux couches de neurones. Bien que le terme ***multicouches*** puisse s'appliquer à une telle architecture, il est plutôt utilisé pour désigner des architectures ayant trois ou plus couches de neurones. De telles architectures sont généralement désignées dans la littérature sous le nom : "Multi-Layer Perceptron" (MLP).

Comme nous l'avons vu, le voisinage est un paramètre important dans l'architecture des réseaux neuronaux. Nous allons définir dans la première partie de cette section différents types de voisinage et de connexions. Nous illustrerons ces divers types de connexions à l'aide d'exemples d'architectures utilisées pour résoudre des problèmes issus de domaines tels que la vision et la parole. Dans la deuxième partie nous parlerons de la dynamique de propagation des états dans le cadre de ces architectures multicouches.

2.4.1. Les Connexions

Dans une architecture multicouches le réseau est arrangé en couches de neurones avec les caractéristiques suivantes :

- Il peut y avoir une connexion entre le neurone j_q d'une couche q et le neurone i_p d'une couche p, si et seulement si $p \neq q$. Autrement-dit, il

n'existe pas de connexion entre les neurones d'une même couche (voir Figure 2.9).

- La couche *p* doit se trouver "en aval" de la couche *q*, c'est-à-dire $p > q$, sauf pour des architectures très particulières[3] .

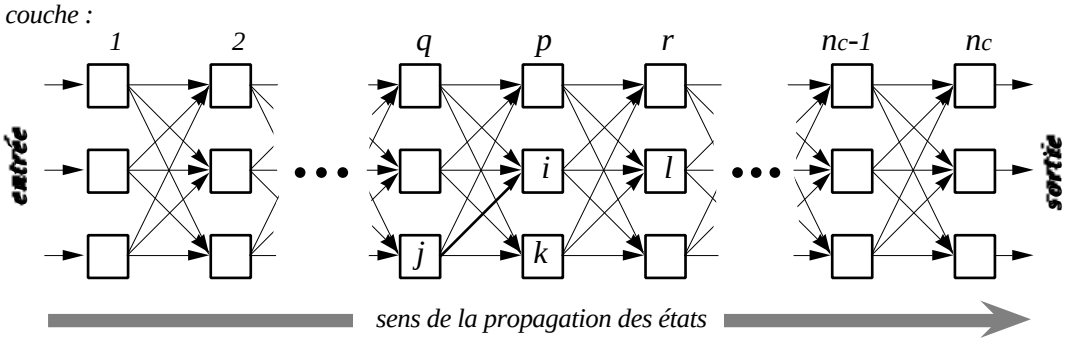


Figure 2.9 : *réseau de neurones avec architecture multicouches. Il y a des connexions seulement entre cellules i et j de couches différentes, i se trouvant dans une couche "en aval" de celle où se trouve j (p > q).*

Ces caractéristiques imposent des contraintes dans l'architecture des réseaux, il reste cependant un nombre important des choix à faire concernant les connexions entre les neurones de couches différentes. Dans les sections suivantes nous allons présenter 4 types de connexions classiques : connexions globales, connexions locales, connexions à masque et connexions à délai.

2.4.1.1. Connexions globales

L'architecture des réseaux de neurones la plus simple est celle où les connexions sont ***globales***, c'est-à-dire, tous les neurones de deux couches successives sont connectés entre eux (voir Figure 2.10). Par conséquent, l'état de chaque neurone de la couche *p* est calculé en fonction des états de

---

[3] Dans certains cas [Pineda 87] ou [Nerrand et al. 91] on trouve des architectures de réseaux ayant des connexions allant d'une couche *q* vers une couche *p* où $q > p$. Souvent *q* est la couche de sortie et *p* celle d'entrée. Il s'agit là de réseaux récurrents et ces connexions agissent comme une sorte de "rétro-alimentation" : les sorties sont projetées inchangées vers l'entrée. Les sorties produites à l'instant *k* sont fonction des entrées à l'instant *k* mais également des sorties à l'instant précédent *k*-1.

tous les neurones dans *q*. Les mémoires associatives, que nous avons décrites précédemment, sont un exemple de réseaux utilisant ce type d'architecture.

Les connexions globales sont simples à réaliser car aucun choix ne doit être fait en ce qui concerne les neurones à connecter. Cependant, pour un réseau où le nombre de neurones est important, le choix d'une architecture à connexions globales, entraîne un nombre très important de paramètres libres.

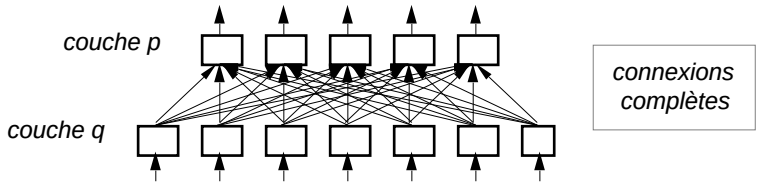


Figure 2.10 : *connexions globales. Les neurones de la couche p sont connectés à tous les neurones de la couche q.*

Prenons par exemple, un réseau de reconnaissance de visages permettant de reconnaître 5 personnes à partir de 24 images par personne, chaque image étant définie avec une résolution de 40×50 pixels. Ce réseau peut être défini par une grille d'entrée de 40×50 neurones et une couche de sortie de 5 neurones. Ce problème comporte une certaine complexité et il faut que le réseau ait au moins une couche intermédiaire et que la taille de cette couche soit de l'***ordre de grandeur***[4] du problème. Si nous prenons le choix fait par E. Viennet [Viennet 92] on aura deux couches intermédiaires de 1000 et 210 neurones respectivement. Avec des connexions globales on atteint un nombre de paramètres à ajuster (les poids des connexions) de l'ordre de $\approx 10^6$. Pour

---

[4] L'évaluation de l'ordre de grandeur d'un problème n'est pas simple. Disons que le nombre de cellules cachées doit être plus au moins en rapport au nombre de cellules des couches en avant et en aval de la couche en question. Plus important que le nombre de cellules cachées est, peut-être, le nombre de paramètres (c-à-d, le nombre de poids de connexions) par rapport au nombre d'exemples utilisés dans l'entraînement – c'est ce que nous pourons appeler "ordre de grandeur" du problème –. Une plus riche base d'apprentissage garantiras des performances plus fiables au niveau de la généralisation sur des nouveaux exemples. Statistiquement il est conseillé d'avoir à-peut-près 6 fois plus d'exemples que de paramètres à ajuster. Les réseaux multicouches peuvent fournir des bons résultats même avec moins d'exemples que de paramètres, mais il faut surveiller lors du processus d'apprentissage les performances sur la base de test pour ne pas tomber dans l'apprentissage "par cœur" de la base d'apprentissage.

représenter et manipuler un tel réseau il est nécessaire de disposer d'une machine ayant des grandes capacités de stockage et de calcul. Les micro-ordinateurs et stations de travail actuels arrivent tout-de-même à gérer de tels réseaux, mais les temps de réponse peuvent être assez mauvais.

Le problème du nombre important de paramètres n'est pas exclusivement un problème de moyens de calcul ou de place de mémoire. En effet, plus il y a de paramètres libres à ajuster, plus on a besoin d'information (d'exemples) pour calculer la fonction globale qui relie l'entrée à la sortie du système.

C'est pourquoi il faut penser à limiter le nombre de connexions du réseau pour ainsi diminuer le nombre de paramètres à estimer. Dans les sections suivantes nous présentons des types de connexions conçus pour tenir compte de ce besoin de réduction du nombre de connexions du réseau.

2.4.1.2. Connexions locales

On parle de ***connexions locales*** dans un réseau, si pour chaque neurone *i* d'une couche *p*, seulement un groupe réduit de neurones de la couche précédente *q*, est connecté à *i*. La Figure 2.11 illustre les connexions locales entre deux couches de neurones. Outre le concept de connexion locale, on voit aussi la notion de ***recouvrement*** des poids. En effet, parmi les trois cellules dans *q* connectées à une cellule *i* dans *p* deux sont connectés aussi à la cellule *i*+1. On parle alors d'un recouvrement de 2 unités, ou de façon complémentaire, d'un déplacement de 1.

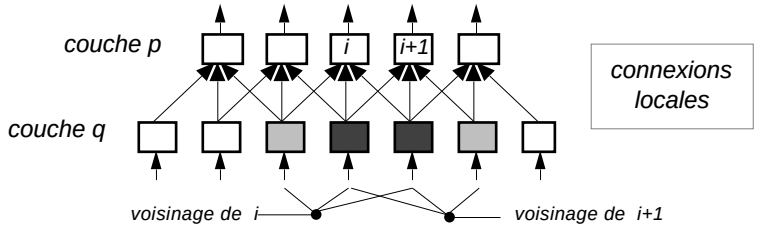


Figure 2.11 : *connexions locales. Les neurones de la couche p sont connectés à quelques neurones de la couche q. L'état de chaque cellule de la couche p est calculé en fonction de l'information provenant de seulement 3 cellules dans la couche q.*

L'intérêt d'utiliser des connexions locales n'est pas exclusivement la réduction du nombre de paramètres d'un réseau quelconque. En effet, l'état d'un neurone est fonction des états des neurones avec lesquels il est connecté.

Or, pour un neurone ayant des connexions locales provenant d'un groupe réduit de neurones dans une autre couche, son état sera fonction exclusivement des états des neurones de ce groupe. Ceci entraîne de multiples possibilités : regroupement d'information, traitement localisé de paramètres du vecteur d'entrée, extraction des caractéristiques des signaux, prise en compte du contexte, compression de l'information, ... Ce qui veut dire ***grosso modo*** que l'on peut spécifier, dans l'architecture du réseau, des caractéristiques du problème à résoudre. Evidemment, il n'y a pas de règle précise pour ceci, chaque problème a ses propres caractéristiques qui requièrent un examen attentif dans le contexte du problème. On ne peut pas affirmer ***a priori***, dans tous les cas, les qualités d'une architecture quelconque. Sa dépend de la tâche et seuls des test réussis peuvent confirmer sa validité.

2.4.1.3. Connexions à masque

Si les connexions locales permettent de réaliser des traitements localisés sur des groupes de cellules, avec les ***connexions à masque***, ou ***connexions à poids partagés***, l'information arrivant à un groupe de neurones peut être traitée de façon identique car ils partagent les mêmes poids de connexion. L'idée de poids partagés est illustrée dans la Figure 2.12 : plusieurs couples de neurones (j_q, i_p), dans les couches q et p respectivement, sont joints par des connexions ayant une même valeur du poids de connexion.

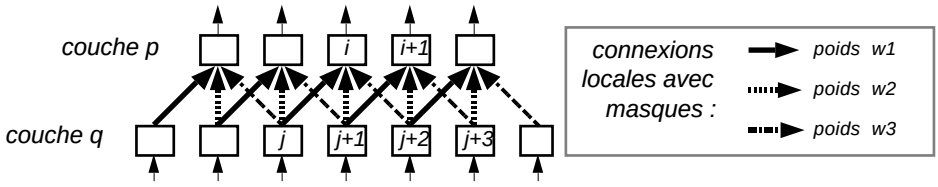
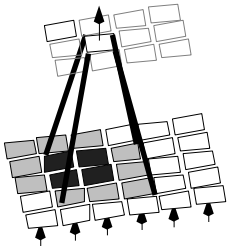


Figure 2.12 : *connexions locales à masque ou à poids partagés. Le voisinage est local, les poids des connexions arrivant sur les neurones de la couche p sont appelés "masque" car chacun des neurones dans la couche p "voit" avec les mêmes valeurs de poids les neurones dans la couche q. Ainsi dans la figure, un même type de pointillé dans les flèches représente une même valeur du poids de connexion.*

Dans la Figure 2.12, les neurones $(j, j+1, j+2)$ de la couche q sont connectés au neurone i de p, respectivement, avec les poids de connexions $(w1, w2, w3)$. De même, les neurones $(j+1, j+2, j+3)$ de q sont connecté à $i+1$ de p avec exactement les mêmes poids $(w1, w2, w3)$. Ainsi de suite pour tous les

neurones des couches *q* et *p* du réseau. L'information des neurones de la couche *q* arrivant à chaque neurone *i* de *p* est donc affectée de la même façon par les poids des connexions qui le connectent. C'est la notion de "masque d'opérateurs" utilisée dans le domaine d'analyse d'images : c'est un masque à travers lequel les neurones de la couche *p* "voient" les neurones de la couche *q*.

Le masque illustré dans la Figure 2.12 est unidimensionnel : les neurones de *q* sont arrangés dans une formation à une dimension et le masque "parcourt" cette couche en longueur. Dans la Figure 2.13 l'arrangement de *q* est bidimensionnel. Le masque est déplacé dans les deux sens, en largeur et en profondeur, c'est la notion de masque bidimensionnel. Le masque montré a un déplacement dans les deux sens de 1 neurone.



(a)

0	–1	0
–1	+4	–1
0	–1	0

Laplacien

–1	0	+1
–2	0	+2
–1	0	+1

Sobel

–1	0	+1
–1	0	+1
–1	0	+1

Gradient

(b)

Figure 2.13 : *masques de connexions. (a) Un masque dans un réseau de neurones où un voisinage de 3×3 neurones est vu par chaque neurone de la couche supérieure. (b) Différents masques utilisés couramment pour des tâches d'extraction des contours dans une application de traitement des images par de méthodes classiques. Les valeurs pourraient correspondre aux poids de connexions dans une approche connexionniste.*

Les applications en traitement d'image sont un exemple important où l'utilisation des réseaux multicouches avec des poids partagés s'avère indispensable. La Figure 2.13b montre trois masques typiques utilisés par des méthodes classiques de traitement d'images pour effectuer des traitements de bas niveau sur l'information. Prenons par exemple le masque ***gradient***. Si nous prenons une image binaire, où **S** = [0, 1] (en ***noir***=1 et ***blanc***=0), dans une grille à deux dimensions, où chaque cellule de la grille représente l'état d'un pixel de l'image, la convolution du masque[5] avec l'image

[5] Par convolution nous voulons dire l'application systématique du masque en parcourant l'image originale complète.

donnera comme résultat l'extraction des changements d'état de l'image[6]. La Figure 2.14 montre un exemple d'utilisation d'un tel masque.

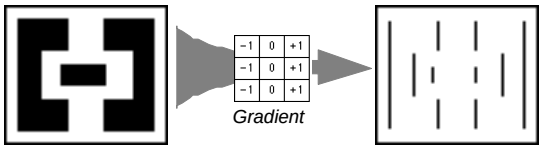


Figure 2.14 : *exemple d'utilisation du masque du gradient, Un masque dédié à l'obtention des changements de tons dans le sens horizontal est appliqué.*

Pour utiliser des masques comme ceux de la Figure 2.13b dans les réseaux de neurones, il suffit de donner les bonnes valeurs aux poids de connexions. Néanmoins, la capacité d'apprentissage des méthodes connexionnistes est telle que le réseau lui même peut trouver le masque nécessaire afin de réaliser l'opération désirée comme c'est le cas dans une application au traitement des images présenté dans [Loncelle 91].

Nous avons vu dans cette section comment nous pouvons utiliser les connexions locales à poids partagés pour effectuer des opérations intéressantes sur les données. Il est donc clair que la réduction du nombre de paramètres n'est pas la seule raison d'utiliser ce type de connexions.

Exemple d'une architecture pour la reconnaissance de visages

Reprenons l'exemple de reconnaissance de visages introduit précédemment (cf. §2.4.1.1. Connexions globales) pour mettre en évidence l'utilisation des architectures de réseaux à poids partagés [Viennet 92]. Nous avons montré comment, en utilisant des connexions globales, on atteint pour cette architecture un nombre de poids à ajuster de l'ordre de $\approx 10^6$. Définissons maintenant une nouvelle architecture en utilisant les connexions locales à poids partagés.

L'architecture que nous avons défini pour l'exemple cité consiste en une grille d'entrée de 40×50 neurones, deux couches intermédiaire de 1000 et 210 neurones respectivement, et une couche de sortie de 5 neurones.

---

6 Vu le masque, les gradients repérés seront dans l'axe horizontal. Pour extraire des gradients dans le sens vertical il faudrait utiliser la transposée du masque.

Divisons les 1000 neurones qui formaient précédemment la première couche intermédiaire, en deux couches chacune à 500 unités arrangées dans une géométrie bidimensionnelle de 20×25. On a donc le même nombre de neurones, la différence vient de la façon dont ils sont connectés aux neurones de la couche d'entrée : chacun des neurones de ces deux couches intermédiaires est connecté à la couche d'entrée avec un masque de connexions de taille 3×3. Un déplacement de 2 unités est spécifié dans les deux sens (voir Figures 2.13 et 2.15). Dans chacune des couches intermédiaires le masque de connexions à des valeurs différentes. Par conséquent, entre la couche d'entrée et chacune de ces couches il n'y a que 9 poids de connexions différents, donc 18 au total. Il faut noter cependant que bien que le nombre de poids de connexions soit extrêmement réduit, le nombre de connexions n'est pas réduit dans la même proportion. En effet, d'un nombre de connexions de l'ordre de $\approx 10^6$ dans l'exemple à connexions globales, on est passé à un nombre de connexions locales de l'ordre de $\approx 10^4$, où il n'y a que 18 valeurs de poids différentes.

L'effet d'une telle architecture pour ces couches intermédiaires est d'obtenir une version compressée, et affectée par un opérateur –le masque–, de l'image originale. D'une seule image formée dans la couche d'entrée de 40×50 pixels on est passé à deux d'un quart de la taille originale, de 20×25 pixels chacune. Les deux couches ayant des masques de connexions différents obtiennent deux versions différentes de l'image compressée. Les caractéristiques extraites par ces deux masques, bien que différentes, apportent des informations complémentaires qui seront exploitées par les couches successives du même réseau.

Dans notre exemple original, la deuxième couche intermédiaire avait 210 neurones connectés globalement à la couche précédente. Substituons-la maintenant par 5 couches identiques avec au total le même nombre de neurones. Chacune des couches ayant une géométrie de 6×7 neurones. Pour chacune des couches, chaque neurone est connecté par deux masques de 5×7 à chacune des couches intermédiaires du premier niveau. Chaque masque ayant des paramètres différents. Cette nouvelle couche sert à l'extraction de caractéristiques importantes sur les images compressés.

Enfin, les neurones de la couche de sortie –ou neurones de décision– sont connectés de façon globale par tous les neurones des couches du second niveau intermédiaire. La couche de sortie est formée par 5 cellules de

décision, une pour chacun des individus à reconnaître. La reconnaissance est mesurée par le niveau d'activation, l'état, de chacun des neurones de sortie. Chaque neurone étant attribué à un individu, le neurone ayant l'état le plus actif, i.e. supérieur numériquement aux autres neurones de sortie, indiquera, suite à la présentation d'une image à l'entrée, l'individu "reconnu" par le réseau.

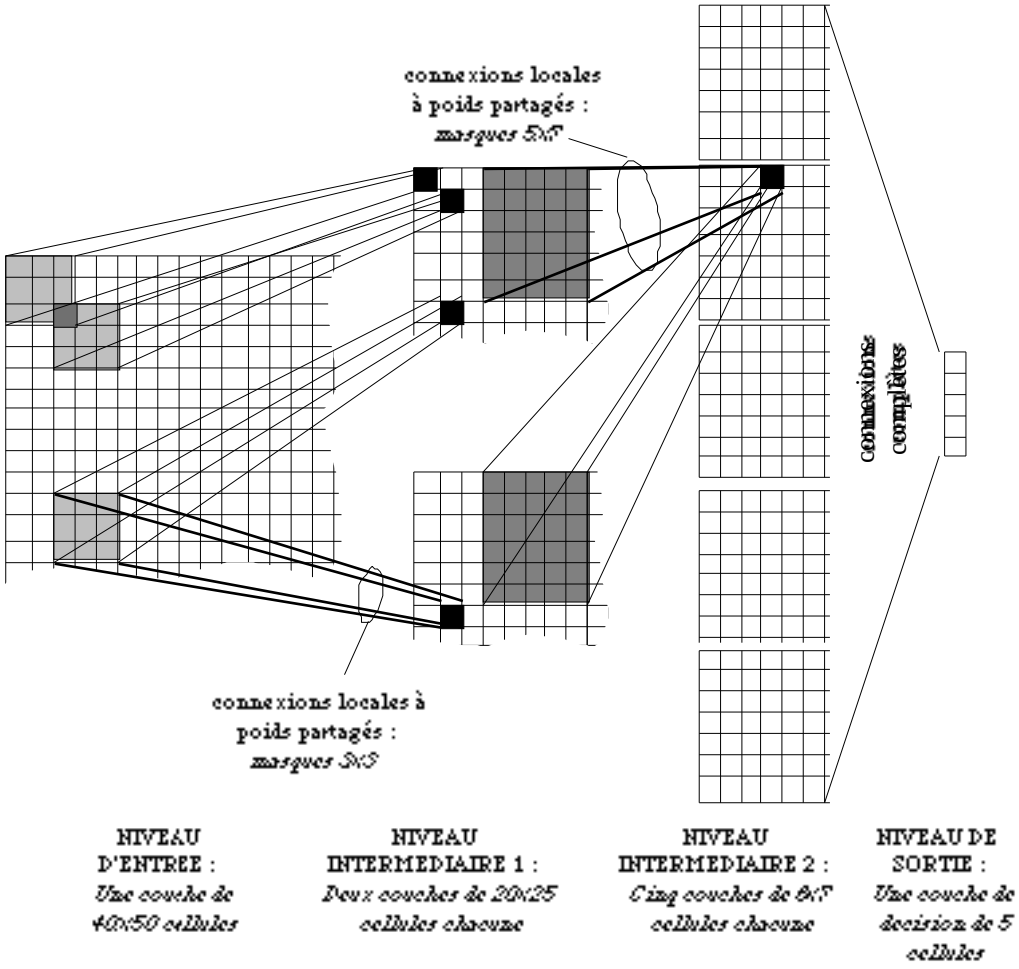


Figure 2.15 : *exemple d'une architecture multicouches pour la reconnaissance de visages. L'architecture utilisée contient plusieurs couches à poids partagés [Viennet 92].*

En utilisant les connexions locales à poids partagés le nombre total de poids différents est maintenant de 403, contre $\approx 10^6$ dans le cas des connexions complètes (le nombre total de paramètres à ajuster est cependant de 1618, 403 poids plus 1215 valeurs de seuil).

Les deux exemples montrés, celui utilisant des connexions globales et celui utilisant des connexions locales à poids partagés exécutent la même tâche : la reconnaissance, ou plutôt la classification de 5 individus en fonction de leur visage. Le choix des connexions locales est ici assez justifié, car mis à part le fait que nous voulions réduire la taille du réseau, il s'agit d'une application où les données en entrée ont un sens géométrique. Des points proches dans la grille d'entrée contiennent des informations représentant des traits proches dans le visage examiné. Il est naturel de les prendre en compte ensemble. En revanche, des points éloignés peuvent représenter des endroits dans le visage sans la moindre corrélation. Les connexions locales tireraient profit des points physiquement proches, donc très probablement correlés.

2.4.1.4. Connexions permettant invariance dans le temps

Dans plusieurs applications le temps est une donnée indissociable des signaux d'entrée. La reconnaissance de la parole est un des domaines le plus étudié dans le cadre des réseaux neuronaux où le temps joue un rôle très important. Ce genre de problème nécessite un modèle capable de représenter des relations entre les différents signaux dans le temps.

Les réseaux de neurones, les plus adaptés pour de telles applications sont ceux qui utilisent les connexions ***à délai*** (en anglais : "Time-Delay Neural Networks" TDNN) introduites par Alexander Waibel et Geoffrey Hinton [Waibel et Hinton 87][7]. Il s'agit, en fait, d'une utilisation particulière de connexions à poids partagés afin de propager l'information du temps à travers le réseau. Les applications liés au problème de la reconnaissance de la parole sont par excellence des cas où la prise en compte du paramètre ***temps*** dans l'information est capitale afin d'espérer avoir des modèles capables de bonnes performances dans des tâches comme la classification de ces signaux. Pour illustrer ceci nous allons présenter une architecture de réseaux de neurones conçue à cette effet.

---

7 Leur publication de 1987, révisée mais essentiellement la même a été publié à nouveau récemment dans ***Readings in Speech Recognition*** [Waibel ***et al.*** 90].

Exemple d'une architecture pour l'identification du locuteur

L'identification vocale du locuteur est une application où le temps est une donnée fondamentale, cette information doit donc être prise en compte par le modèle. Le temps n'est pas cependant une donnée explicite que l'on peut présenter au réseau par le biais des états des cellules d'entrée tel qu'on le fait couramment avec d'autres types d'information. Nous pouvons donner aux réseaux neuronaux des informations sur le problème à résoudre par le biais d'architectures particulières. Il faut donc construire des architectures appropriées afin d'extraire, ou plus précisément, de conserver l'effet du temps sur l'information du signal à traiter. L'idée étant de propager l'apport du temps jusqu'aux dernières couches du réseau y compris la couche de sortie. Pour réaliser ceci les connexions à délai sont utilisées.

Pour illustrer ce type de connexions nous présentons une architecture conçue par Younès Bennani dans le cadre de sa thèse : ***Approches Connexionnistes pour la Reconnaissance Automatique du Locuteur*** [Bennani 92]. Ce réseau utilise une architecture à TDNN (voir Figure 2.16).

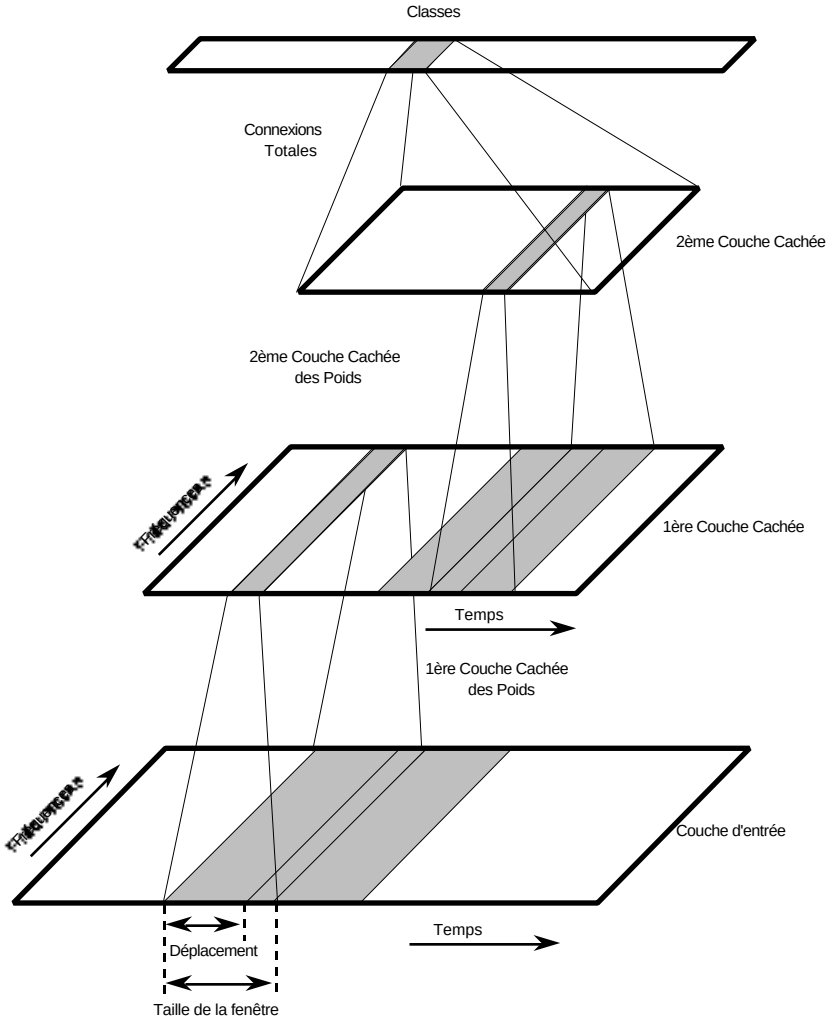


Figure 2.16 : *exemple d'un réseau de neurones TDNN. Architecture utilisée pour l'identification vocale du locuteur [Bennani 92].*

Explication du codage des données et couche d'entrée

On présente en entrée du réseau un signal de parole correspondant à une durée approximative de 0,25 sec. Cette durée du signal est modélisée par une suite de 25 valeurs (trames) de dimension 16 (une analyse LPC d'ordre 16 à été effectué sur le signal échantillonné à 16 KHz). Chaque vecteur représente une durée du signal de 10 ms.

La couche d'entrée est dimensionnée pour coder les 25 trames dans lesquels est échantillonné le signal, et les 16 fréquences dans lesquelles sont décomposées les trames. La couche d'entrée est par conséquent de 25×16 unités. Chaque unité prend une valeur réelle dans $\mathbf{S} = [a, b]$, a et b étant respectivement un minimum et un maximum arbitraires.

Premier extracteur de caractéristiques

La première couche intermédiaire est un ***extracteur*** de caractéristiques dans le temps sur 5 trames consécutives. Chaque cellule comporte un masque sur la couche d'entrée et "voit" 5 trames consécutives du signal parole sur l'ensemble des 16 fréquences. Son état sera donc fonction des trames "observées". 12 cellules comme celle décrite ci-dessus, sont mises côte-à-côte, avec la même couverture mais des paramètres différents. Cet ensemble de 12 masques parallèles est déplacé sur le signal d'entrée, de trame en trame. Chaque déplacement signifie un nouvel ensemble de cellules qui "observent", comme les précédentes, 5 trames du signal d'entrée dont les 4 premières étaient aussi observées par les 12 cellules précédantes. Il y a donc un déplacement de 1 et un recouvrement de 4. Afin d'avoir une couverture totale du signal d'entrée 21 déplacements sont nécessaires pour couvrir les 25 trames de l'entrée. Cette couche intermédiaire requiert donc 21×12 cellules, où chaque colonne de 21 cellules partage un des 12 masques de 5×16 sur la couche d'entrée.

Le signal de parole en entrée est codé : sur chaque neurone de cette couche intermédiaire sont codés 5 trames consécutives de l'entrée. Cela représente approximativement 50 ms du signal. Sur une colonne de 21 neurones nous avons le codage complet du signal d'entrée par tranches de ≈50 ms. Les 12 masques nous permettent d'avoir 12 codages différents sur les mêmes intervalles de temps. Nous pourrions comprendre cette couche comme un extracteur de caractéristiques ayant une durée de l'ordre de 50 ms dans le signal d'entrée.

La relation temporelle des caractéristiques du signal est gardée : les masques au début des colonnes de neurones codent les caractéristiques des premières trames du signal parole donnée à l'entrée, c'est-à-dire du début du signal, tandis que les derniers masques codent les caractéristiques présentes dans les dernières trames, la fin du signal.

Deuxième extracteur de caractéristiques

Une deuxième couche intermédiaire utilisant le même principe de masques, ou d'extracteurs, utilisé dans la première couche intermédiaire est connectée à cette dernière. La deuxième couche intermédiaire se compose de 10 extracteurs, ou masques, connectés à 7 rangées consécutives de 12 neurones de la première couche intermédiaire. Le déplacement est aussi d'une unité, et il faut se déplacer 15 fois pour avoir une couverture complète

des 21 rangées de la couche précédente. La deuxième couche intermédiaire a donc une taille de 15×10 cellules, avec des masques de connexions de 7×12 sur la première couche cachée.

Ici, sur chaque cellule sera codée l'information de 7 colonnes de la première couche intermédiaire, et, indirectement, grâce au codage réalisé par la première, sur 11 trames consécutives du signal parole en entrée. La première rangée d'extracteurs sur la deuxième couche, couvre les 7 premières de la première, c'est-à-dire les 11 (6+5) premières trames du signal d'entrée. Cela représente approximativement 110 ms sur le signal de parole.

Couche de décision

Enfin, la couche de sortie, avec 10 unités de décision, une pour chacun des locuteurs à identifier, est connectée de façon globale avec la deuxième couche intermédiaire du réseau.

Ici la relation temporelle à été gardée jusqu'à la fin grâce à l'utilisation des connexions à délai. Seuls les cellules de la couche de sortie ont dans le temps une couverture globale du signal parole. C'est à ce niveau que la décision est prise, elle est faite en fonction des états des cellules la couche de sortie. La forme du signal parole dans le temps joue donc un rôle essentiel dans la décision.

2.4.2. La dynamique de propagation des états

Dans les applications présentées dans ce mémoire c'est cette architecture multicouches qui a été retenue. Comme nous le montrerons dans les sections suivantes ce type d'architecture permet un traitement parallèle de l'information pour les cellules dans une couche, et séquentiel couche par couche.

La propagation des états

Nous avons montré (cf. §2.2.1) qu'un neurone est défini par son état, sa fonction de transition, ses connexions, etc. et que cet état, grâce aux connexions entre neurones et à la fonction de transition, peut être modifié. Essayons de comprendre maintenant la façon dont les états des neurones se propagent dans un réseaux.

La ***dynamique du réseau*** établit le moment où les états des différents neurones doivent être mis à jour. Le type de dynamique est caractérisé par les deux concepts suivants :

- Le mode de changement des états dans le temps : on parle alors de dynamique en ***temps discret*** et dynamique en ***temps continu***.
- L'ordre dans lequel sont effectués les changements des états des neurones dans le réseau : on parle alors d'itérations ***parallèles***, ***itérations séquentielles*** ou encore ***itérations bloc-séquentielles***.

Nous nous intéressons ici principalement aux modes d'itération et à la dynamique utilisées par les réseaux multicouches MLP (pour une description détaillé des différents modes de changement et d'itération voir [Thiria 89]).

La dynamique de propagation

Dans les réseaux MLP nous avons une dynamique de changement d'état à temps discret associée à un mode d'itération bloc-séquentiel. Nous allons maintenant préciser ces notions.

Une ***dynamique à temps discret*** est une dynamique où la propagation des états des neurones est faite à des instants discrets de temps. Cette dynamique définit l'instant de changement des états mais elle ne définit pas les neurones qui doivent changer d'état à un instant donné. En effet, les neurones dans une même couche peuvent changer d'état simultanément, comme c'est le cas dans les itérations parallèles. D'autre part, les neurones d'une couche doivent attendre le changement d'état des neurones des couches précédentes afin de pouvoir à leur tour changer d'état. C'est le cas des itérations séquentielles.

Le mode d'***itération bloc-séquentiel*** est une combinaison des modes parallèle et séquentiel fondé sur la définition des partitions ou blocs de neurones qui changent d'état simultanément. L'idée générale est que pendant une itération, tous les neurones d'une même partition I_q calculent leurs états en fonction d'une part, des nouveaux états $s(t+1)$ des neurones appartenant aux partitions inférieures à q, et d'autre part, des anciens états $s(t)$ des neurones des partitions supérieures à q. Tous les neurones

appartenant à une même partition changent leurs états simultanément. Les différentes partitions –ou blocs I_q– opèrent de façon séquentielle. Par la suite, nous définirons formellement ces idées.

Soit F la fonction de transition globale d'un réseau composé de N neurones :

$$F : S^N \rightarrow S^N$$

avec $F = (f_1, f_2, f_3, \ldots, f_N)$ l'ensemble des fonctions de transition des N neurones du réseau.

Soit $(I_q)_{q = 1, 2, \ldots, c}$ une partition ordonnée de l'ensemble des neurones $\{1, 2, \ldots, N\}$ du réseau telle que, pour les neurones i et j [Fogelman-Soulié 85] :

$$i \in I_q, \quad j \in I_r, \quad q < r \quad \Rightarrow \quad i < j \qquad (2.10)$$

Une itération bloc-séquentielle sur F associée à la partition ordonnée $(I_q)_q$ est définie de la manière suivante :

Itérations bloc-séquentielles

$$q \in (1, 2, \ldots, c), \qquad (2.11)$$

$$i \in I_q, \quad s_i(t+1) = f_i\left(y^q(t)\right)$$

où

$$y^1(t) = s(t)$$

$$q \in (2, \ldots, c), \quad y_j^q(t) = \begin{cases} s_j(t+1) & \text{si } j \in \left(I_1 \cup I_2 \cup \ldots \cup I_{q-1}\right) \\ s_j(t) & \text{sinon} \end{cases}$$

Pour les réseaux multicouches, chaque partition I_p représente une couche de neurones, la partition I_1 étant constituée des neurones de la couche d'entrée et la partition I_c des neurones de la couche de sortie. Les partitions

inférieures à la partition I_q correspondent aux couches en amont de de cette partition.

Dans les réseaux multicouches qui font l'objet de nos travaux, les neurones d'une même partition I_q calculent leurs états uniquement en fonction des nouveaux états $s(t+1)$ des neurones appartenant aux partitions inférieures à q.

3. Apprentissage supervisé

3.1. Introduction

Dans le chapitre précédent nous avons introduit les notions de base concernant les composants des réseaux de neurones et la façon dont ils sont interconnectés dans les architectures. Nous avons également mentionné les notions d'apprentissage et de reconnaissance qui permettent respectivement d'estimer les paramètres de ces modèles lors de la réalisation d'une tâche et de mettre en œuvre le modèle lors de sa stimulation par un nouveau signal. Les réseaux de neurones permettent ainsi de réaliser des associations entre des signaux d'entrée qui sont les stimuli du réseau et des signaux de sortie qui sont le résultat du calcul de ce réseau.

Nous allons par la suite définir de façon plus précise ces différentes notions en insistant sur celle d'apprentissage supervisé qui est au centre de cette thèse.

Nous considérerons par la suite un réseau comme un associateur défini par son architecture et un ensemble de paramètres à ajuster. Pour simplifier la présentation, nous considérerons que les seuls paramètres à ajuster sont les poids des connexions.

Ces poids sont déterminés par un algorithme dit d'***apprentissage***. Les algorithmes utilisés en connexionnisme fonctionnent à partir de règles mathématiques ou heuristiques qui modifient les poids itérativement quand on présente des exemples au réseau. On parle alors d'***apprentissage à partir d'exemples***. L'ensemble des données ayant servi à mettre au point les poids du réseau est appelé ***ensemble d'apprentissage***. Plusieurs quantités peuvent servir à mesurer la qualité des performances du réseau. Elles diffèrent selon la nature de la tâche que l'on veut faire réaliser au réseau. Ces mesures peuvent être calculées sur l'ensemble d'apprentissage, toutefois, on est la plupart du temps intéressé non pas par ce type de performances mais par celles que le réseau obtiendra en phase opérationnelle sur des formes qui ne

lui ont jamais été présentées. Pour estimer ces performances on utilise la plupart du temps un ensemble de signaux dont on connait les caractéristiques et qui n'ont pas servi à l'apprentissage. Cet ensemble est appelé ***ensemble test***. D'autres ensembles de signaux sont quelquefois employés pour réaliser l'apprentissage, notamment pour faire de la validation croisée.

Pour une même architecture de réseau on peut avoir différents algorithmes d'apprentissage qui diffèrent par la règle de mise à jour des poids. On distingue deux grandes catégories de techniques d'apprentissage qui sont les algorithmes ***supervisés*** et ceux qui sont ***non supervisés***. Pour les premières, on connait à la fois le signal d'entrée présenté au réseau et un signal désiré que l'on essaie de lui faire reproduire. Le but de l'apprentissage est d'apprendre à réaliser cette association pour l'ensemble des couples de formes constituant l'ensemble d'apprentissage. Dans le second cas, on ne connait que les formes d'entrée et l'on désire que le réseau extraie de ces formes une certaine connaissance structurelle que l'on puisse par la suite interpréter ou utiliser dans d'autres systèmes. Par exemple, on peut imaginer des signaux inconnus sur lesquels le système apprendra à détecter la présence répétée de formes de base ou de similarités. Ce dernier type d'apprentissage inclus les algorithmes dits de regroupement ou "clustering" dans la littérature de reconnaissance des formes.

Nous avons employé dans cette thèse uniquement des algorithmes supervisés dont nous détaillons le principe.

Ces algorithmes visent à apprendre des associations entre des couples de vecteurs (donnée d'entrée, sortie désirée). Nous conservons les notations du chapitre précédent pour décrire ces ensembles de vecteurs. De très nombreux modèles de réseaux et algorithmes ont été proposés dans la littérature connexionniste pour réaliser de telles associations. Toutefois, seul un certain nombre d'entre eux possèdent les qualités requises pour les tâches qui nous intéressent. Comme nous l'avons annoncé dans l'introduction, ces qualités sont principalement la puissance d'approximation pour des fonctions complexes entre espaces qui peuvent être de grande dimension et la possibilité de mise en œuvre sur un nombre très important de données de grande taille. Le modèle que nous avons utilisé et qui est également le modèle le plus populaire depuis 1988 pour les applications de grande taille est celui

des ***perceptrons multicouches*** (MLP). L'algorithme d'apprentissage est celui de la ***rétro-propagation du gradient*** [Rumelhart et al. 86b].

Dans ce chapitre, nous allons présenter le problème général de l'apprentissage supervisé, puis nous décrirons l'algorithme de l'***Adaline*** avant de passer à celui de rétro-propagation. Nous décrirons ensuite à titre d'illustration un problème sur lequel nous avons travaillé au début de cette thèse qui est celui de la structure secondaire des protéines globulaires.

3.2. Associer des données

L'association entre données peut être de différent type selon les caractéristiques du problème à traiter. On distingue classiquement deux grand types qui sont l'auto-association et l'hétéro-association.

Auto-association

L'***auto-association*** consiste à associer un vecteur $\mathbf{x}_k$ à lui même. Quand on veut utiliser un réseau de neurone pour cela, on se retrouve devant le problème d'apprendre à associer, avec le réseau que l'on utilise, un ensemble de $\{(\mathbf{x}_k, \mathbf{x}_k)\}$.

Bien que cette notion puisse paraître un peu surprenante au premier abord, il y a de nombreuses applications qui nécessitent d'associer un signal à lui même. Nous avons introduit dans le chapitre précédent la notion de mémoire associative et celle plus générale d'architecture d'un réseau multicouche. Ces réseaux peuvent tous deux être utilisés pour réaliser des associations quelconques et en particulier pour faire de l'auto-association. Les deux applications les plus courantes de l'auto-association sont :

- ***la réalisation de mémoires associatives résistantes au bruit :*** apprendre ce type d'association permettra par exemple de retrouver une des formes apprises quand on présentera au réseau une version bruitée d'une des formes qui lui a été présentée pendant l'apprentissage. Ce bruit peut résulter des caractéristiques du capteur de signal, d'une erreur de manipulation, d'une troncature de l'information, ...

- ***la compression d'information :*** la Figure 3.1 montre un réseau multicouches où les couches intermédiaires ont moins d'unités que celles

d'entrée. Supposons que l'on utilise ce type de réseau pour réaliser une auto-association. L'information d'entrée sera tout d'abord projetée sur un espace de dimension plus faible que celle de l'espace d'entrée, elle sera donc sous forme compressée. Cette projection servira ensuite à reproduire l'information de sortie qui sera soit identique soit proche de l'information d'entrée. Ce type de réseau peut donc servir à réaliser de la compression d'information.

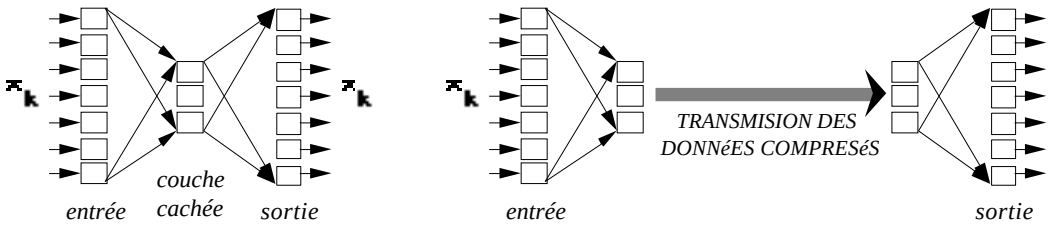


Figure 3.1 : *architecture pour une application en auto-association. Pour une entrée* $\mathbf{x}_k$ *donnée au réseau, les états des neurones des couches cachées, avec moins d'unités que la couche d'entrée, représentent une version compressé de* $\mathbf{x}_k$*. Ce fait peut être exploité pour compresser des signaux afin par exemple de les transmettre sous forme réduite. Ceci avec les premières couches du réseau. Lors de la réception, le signal sera décompressé avec le réseau complémentaire.*

Nombre de travaux utilisent ce genre de réseaux, notamment en traitement des images satellites [Cottrell 82, Mougeot et al. 90, Loncelle 90].

Hétéro-association

Le cas plus général est celui de l'***hétéro-association*** où l'on entraîne le réseau à associer des vecteurs différents entre eux. Ces vecteurs peuvent appartenir à des espaces réels de taille quelconque. On distinguera un cas particulier très fréquemment rencontré qui est celui de la ***classification***.

3.3. Classification

Pour ce type d'application, on veut étiqueter des signaux par une classe choisie parmi un ensemble fini. Les réponses désirées, où vecteurs $\mathbf{y}_k$, sont les mêmes pour tous les exemples appartenant à une même ***classe*** –les vecteurs $\mathbf{x}_{1k}, \mathbf{x}_{1k}$, ... associés à $\mathbf{y}_k$–. La valeur numérique du vecteur $\mathbf{y}_k$ que l'on attribue à la classe k n'est pas significative en elle-même. Cette valeur

permet uniquement de retrouver les classes : différentes valeurs pour $\mathbf{y}_k$ peuvent amener à résoudre la même classification.

Ainsi par exemple, pour une application de reconnaissance automatique de caractères, indépendamment de la police utilisée, on fait correspondre à chacune des lettres de l'alphabet la même classe quelle que soit la police utilisée (voir Figure 3.2).

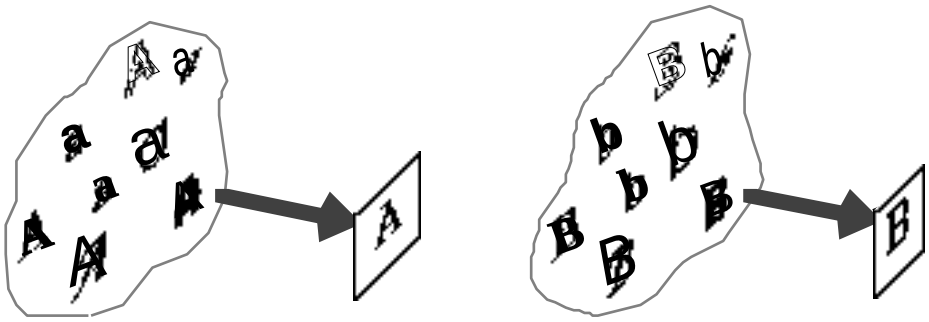


Figure 3.2 : *exemple de l'hétéro-association. Un lettre sous plusieurs types de polices de caractères est associé au caractère représentant le lettre dans une représentation plus universelle. Chaque lettre de l'alphabet correspond à une classe.*

La Figure 3.3 montre une représentation des classes pour l'exemple de la reconnaissance automatique des textes écrits. Cette représentation qui est la plus communément utilisée consiste à décrire les classes par des vecteurs dans l'espace $\{0, 1\}^p$ –ou bien $\{-1, 1\}^p$– où p est le nombre total de classes. La classe numéro i sera identifiée par un vecteur de sortie désirée composé uniquement de 0 (ou de -1) excepté à l'emplacement i où l'on aura la valeur 1.

A : **1**000000000000000000000000000
B : 0**1**000000000000000000000000000
C : 00**1**000000000000000000000000000
⋮
Z : 000000000000000000000000000**1**

Figure 3.3 : *représentation des classes. Une représentation plus universelle d'une classe peut être celle montrée dans la figure. Il faut autant de chiffres que de classes. Les différents lettres "A" composant la première classe, ils sont associés au vecteur "1000…0". Pour la deuxième classe, lettre "B", le vecteur "0100…0" lui est associé; et ainsi de suite jusqu'à la classe des "Z" où c'est le vecteur "000…01" qui est associé.*

Pour l'exemple de la Figure 3.2, les différentes lettres "A" composant la première classe, sont associés au vecteur [1000…0]. Pour la deuxième classe, lettre "B", le vecteur [0100…0] lui est associé; et ainsi de suite jusqu'à la classe des "Z" où ce vecteur est [000…01].

3.4. L'Adaline et l'algorithme LMSE

Au début des années 60 B. Widrow et M.E. Hoff ont proposé un système ***adaptatif***[8] qu'ils ont appelé ***Adaline*** (de l'anglais "adaptive linear element") [Widrow et Hoff 60, Widrow 62]. L'***Adaline*** permet d'apprendre des associations de R^n dans R. Nous détaillons cet algorithme car il peut être considéré comme l'ancêtre de très nombreux algorithmes adaptatifs utilisés actuellement en connexionnisme; l'algorithme de rétro-propagation que nous présenterons en §3.5 n'est qu'une version du même type d'algorithme appliqué à une architecture différente.

Adaline est un système adaptatif qui, comme son nom l'indique, calcule sa sortie par une combinaison linéaire de ses entrées. La règle de modification des poids connue également sous le nom de règle de Widrow-Hoff est une version adaptative du gradient de la plus grande pente. Nous décrirons tout d'abord l'architecture du réseau puis l'algorithme lui même.

La mise en œuvre physique

La première mise en œuvre de l'***Adaline*** était un circuit électrique constitué d'un ensemble de potentiomètres –résistances variables– connectés à un circuit capable d'additionner les intensités de courant électrique produites par les signaux de voltage en entrée. Pour l'utilisation en classification, cet additionneur était suivi d'un élément à seuil dont la sortie était +1 ou –1, selon la polarité de la somme obtenue. La Figure 3.4 montre le circuit correspondant.

---

8 Le terme de ***adaptatif*** s'applique aux systèmes de transformation de signaux, qui sont capables de s'adapter graduellement, ou d'adapter leurs paramètres, en fonction de certaines caractéristiques des signaux.

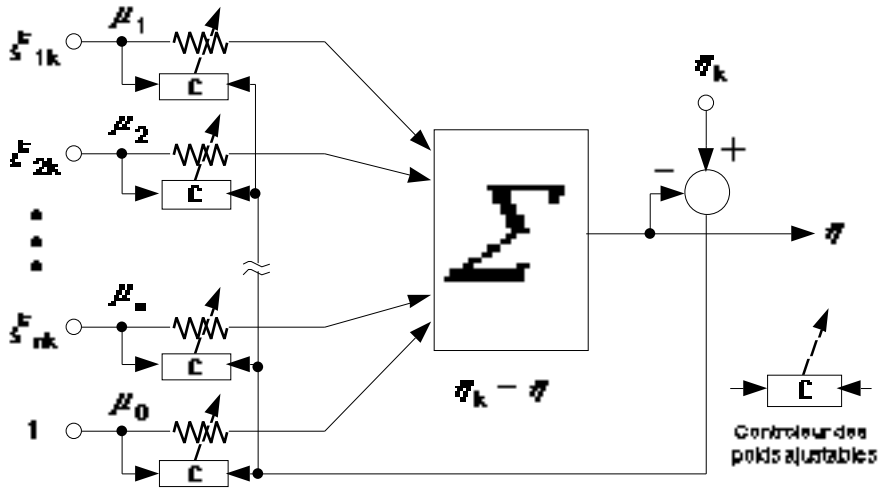


Figure 3.4 : *Adaline. Représentation physique du système adaptatif linéaire.*

Si on note les conductances –l'inverse des résistances– par μ_j, $j = 0, 1, \ldots, n$, ξ_j les signaux d'entrée et η le sortie, alors la sortie du bloc additionneur sera donnée par :

$$\eta = \sum_{j=1}^{n} \mu_j \ \xi_j - \mu_0 \qquad (3.2)$$

On peut remarquer que la mise en parallèle de *p* ***Adalines*** permet de construire un système linéaire ou à seuil suivant le cas qui associe des vecteurs de R^n à des vecteurs de R^p. Ces systèmes dupliquent tout simplement la cellule de base présentée sur la Figure 3.4. Les mémoires associatives que nous avons décrites au chapitre précédant peuvent parfaitement être mises en oeuvre sur ce type de machine. Par la suite, nous présenterons l'algorithme en considérant une architecture avec plusieurs adalines qui fonctionnent en parallèle sur le même signal d'entrée. Cela permet d'être plus général au prix d'une faible complexification des notations.

L'adaptation

L'équation (3.2) n'est autre que la fonction de transition définie pour les automates linéaires, biaisée d'un paramètre μ_0 qui permet de régler l'endroit où se fera le seuillage. Pour cette raison, on l'appelle tout simplement le ***seuil*** du neurone.

Bien que le système physique de l'***Adaline*** soit un système continu dans le temps, nous allons décrire ici l'algorithme adaptatif avec une dynamique adaptative à temps discret. Ceci permet de simplifier la présentation et de décrire simultanément l'algorithme que l'on met en œuvre.

On considère désormais des intervalles de temps t_k où $k = 1, 2, \ldots$.

L'équation (3.2) pour plusieurs unités de sortie d'indice i, où $i = 1, 2, \ldots, p$, peut être réécrite de la façon suivante :

$$s_i^k = \sum_{j=1}^{n} w_{ij}^k \, x_j^k - \theta_0 \qquad (3.3)$$

Le problème d'adaptation consiste en la détermination des coefficients w_{ij} de façon à ce que la réponse du système pour une entrée donnée soit correcte ($s_i = y_i$). Si, à cause de la nature du problème, une réponse exacte n'est pas possible pour tous les couples entrée-sortie, on tentera de minimiser l'erreur moyenne entre les réponses désirées et calculées.

L'équation générale d'un algorithme adaptatif pour la modification des coefficients w_{ij} est donnée par :

$$w_{ij}^{k+1} = w_{ij}^k + \Delta w_{ij}^k \qquad (3.4)$$

où Δw_{ij}^k est la variation du coefficient w_{ij} au temps t_k. Ce coefficient peut être obtenu de différentes façons, chacune d'entre elles correspondent à un algorithme d'apprentissage. De nombreuses méthodes proposent des règles heuristiques pour ces modifications de poids. Une autre grande famille de méthodes est basée sur la mise en oeuvre de techniques dites de gradient adaptatif. Ce sont elles que nous allons développer.

Fonction de coût et méthode de gradient

Les techniques de gradient permettent l'estimation des paramètres w par l'optimisation d'une fonction de coût qui caractérise les performances du réseau. Cette fonction de coût sera calculée entre les vecteurs désirés et les vecteurs calculés par le réseau.

Soit l'ensemble des vecteurs d'entrée $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$, définis dans $\mathbb{R}^n$, les vecteurs réponses $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m$, définis dans $\mathbb{R}^p$ et $\mathbf{W}$ la matrice de poids de connexions définie dans $\mathbb{R}^{p \times n}$. Alors, on a :

$$\mathbf{x}_k = \left[x_1^k,\ x_2^k,\ \ldots,\ x_n^k\right]^{\mathrm{T}},\ \mathbf{y}_k = \left[y_1^k,\ y_2^k,\ \ldots,\ y_p^k\right]^{\mathrm{T}} \text{ et} \qquad (3.6)$$

$$\mathbf{W} = \left\{\mathbf{w}_1 \mid \mathbf{w}_2 \mid \ldots \mid \mathbf{w}_p\right\}^{\mathrm{T}} = \left\{w_{ij}\right\}_{p\times n}$$

où $\mathbf{w}_i = \left[w_{1i}, w_{2i}, \ldots, w_{ni}\right]^{\mathrm{T}}$ est le vecteur colonne composé par les poids des connexions entre les n cellules d'entrée et l'automate i en sortie. Définissons la fonction de coût suivante :

$$C(\mathbf{W}) = \frac{1}{2}\sum_{k=1}^{m}\left|\left|\mathbf{W}^{\mathrm{T}}\cdot\mathbf{x}_k - \mathbf{y}_k\right|\right|^2 \qquad (3.7)$$

où $\|\cdot\|^2$ est la norme cuadratique. Le problème est alors de trouver un $\mathbf{W}^*$ pour $\mathbf{W}$ tel qu'il minimise $C(\mathbf{W})$. Une condition nécessaire est la suivante :

$$C(\mathbf{W})\Big|_{\mathbf{W}=\mathbf{W}^*} = \left[\frac{\partial C(\mathbf{W})}{\partial w_{ij}}^{\mathrm{T}}\right]_{p\times n}\Bigg|_{\mathbf{W}=\mathbf{W}^*} = \mathbf{0} \qquad (3.8)$$

Les techniques de gradient sont des méthodes d'optimisation itératives qui permettent de trouver des solutions vérifiant la condition (3.8). La version la plus simple de ces techniques est celle dite du ***gradient de la plus grande pente*** :

Définir la configuration initiale $\mathbf{W} = \mathbf{W}_0$ (3.9)

$t_k,\ k = 1, 2, \ldots$

$$\mathbf{W}_{k+1}^{\mathrm{T}} = \mathbf{W}_k^{\mathrm{T}} - \varepsilon\ C\left(\mathbf{W}_k\right)$$

où ε est un scalaire positif, souvent dépendant de k –auquel cas on ecrit ε^k– et qui est appelé le ***pas d'itération***. Ce coefficient est incorporé dans la terme Δw_{ij}^k de l'équation (3.4).

Dans le cas quadratique, en remplaçant (3.7) dans (3.9) on obtient l'expression :

$$\mathbf{W}_{k+1}^{\mathrm{T}} = \mathbf{W}_k^{\mathrm{T}} - \varepsilon^k\sum_{k=1}^{m}\left(\mathbf{W}^{\mathrm{T}}\cdot\mathbf{x}_k - \mathbf{y}_k\right)\cdot\mathbf{x}_k^{\mathrm{T}} \qquad (3.10)$$

Dans (3.10) les changements de chaque w_{ij} $\mathbf{W}$ sont fonction des toutes les données $(\mathbf{x}_k, \mathbf{y}_k)$, $k = 1, 2, \ldots, m$, m étant le nombre d'exemples.

Le gradient stochastique : la règle de Widrow-Hoff

Ces techniques de gradient ne sont pas adaptées au cas des réseaux connexionnistes où l'on désire mettre en oeuvre des systèmes adaptatifs capables de modifier leurs paramètres en continu en fonction d'un environnement qui peut changer. Widrow et Hoff ont proposé en 1960 la règle adaptative qui porte leur nom et qui est également connue sous le nom de règle LMSE [Widrow et Hoff 60]. Il s'agit tout simplement d'une règle de gradient suivant la plus grande pente où les paramètres w sont modifiés non pas en tenant compte à chaque itération de l'erreur globale réalisée sur un ensemble de formes, mais où après chaque présentation de forme on modifie le vecteur de paramètres. La formulation mathématique de la règle est la suivante :

$$\mathbf{W}_{k+1}^{\mathrm{T}} = \mathbf{W}_k^{\mathrm{T}} - \varepsilon^k\left(\mathbf{W}^{\mathrm{T}}\cdot\mathbf{x}_k - \mathbf{y}_k\right)\cdot\mathbf{x}_k^{\mathrm{T}} \qquad (3.11)$$

où $(\mathbf{x}_k, \mathbf{y}_k)$ est le couple présenté à l'instant de temps t_k. Il a été démontré que, dans le cas d'une erreur quadratique, (3.11) converge avec probabilité 1 vers $\mathbf{W}^*$, la solution de $\mathbf{W}$ que minimise la fonction de coût $C(\mathbf{W})$.

L'algorithme de l'***Adaline*** est très général puisqu'il permet d'apprendre des associations quelconques. Quelque soit la nature du problème étudié, cet algorithme fournira une solution qui est la meilleure que l'on puisse avoir avec cette architecture au sens de l'erreur aux moindres carrés. Toutefois, l'architecture utilisée étant extrêmement simple, il est évident que la qualité de l'approximation trouvée ne sera pas toujours excellente. La Figure 3.5 illustre sur des exemples simples de classification les hyperplans séparateur trouvés par une ***Adaline***. L'hyperplan trouvé sera celui minimisant l'erreur quadratique. Notons qu'il ne correspond pas toujours à la séparation optimale et qu'il peut exister des problèmes linéairement séparables pour lesquels l'***Adaline*** ne trouvera pas de solution [Figure 3.5b].

L'algorithme de l'***Adaline*** est extrêmement employé, c'est notamment l'algorithme de base en traitement adaptatif du signal. Toutefois, à cause des limitations évoquées plus haut, on a cherché des architectures et des algorithmes permettant de réaliser des tâches plus complexes ou avec une meilleure approximation que celle fournie par une simple ***Adaline***. Ces

recherches ont donné lieu à de très nombreuses publications dans les années 60-70. Toutefois, aucun algorithme général n'a été formulé permettant d'entraîner des réseaux plus complexes et d'usage général. Widrow par exemple a lui même proposé la notion de ***Madaline*** [Widrow et Winter 88] où plusieurs adalines sont utilisées simultanément ou séquentiellement pour résoudre des problèmes non solubles de façon exacte par une seule ***Adaline***. Les Madalines, comme tous les algorithmes présentés à la même époque, nécessitent une importante connaissance ***a priori*** du problème qui permette de définir l'architecture à employer. Ce type de connaissance n'est bien sûr quasiment jamais disponible quand on s'attaque à un problème réel. Il ne s'agit donc pas d'une machine d'apprentissage général.

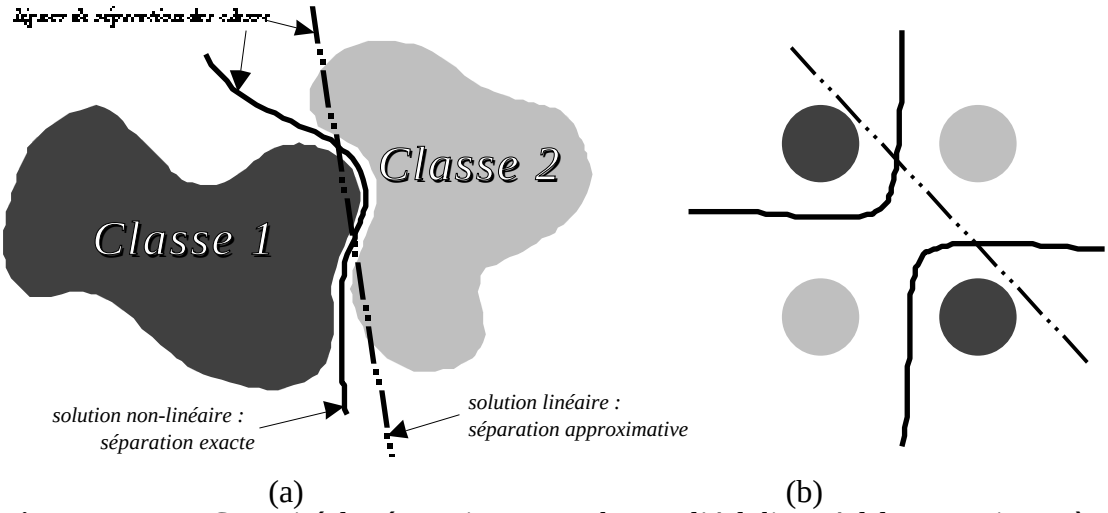


Figure 3.5 : *Capacité de séparation entre classes d'Adaline.* Adaline *convient très bien pour des problèmes ayant un hyperplan séparateur entre les différentes classes. Quand ce n'est pas le cas, (a) Adaline trouve un Hyperplan séparateur optimal. Dans un problème à plusieurs entrées : (b) le problème XOR, une solution linéaire peut être très contraignante par rapport à une solution non-linéaire.*

Il faut attendre le milieu des années 80 pour voir apparaître des machines beaucoup plus puissantes que celle des années 60 avec des procédures d'apprentissage générales et efficaces. Le modèle le plus célèbre qui est également celui que nous avons employé pour nos applications est celui des perceptrons multicouches entraînés par l'algorithme de ***rétro-propagation du gradient***. Toutefois, cet algorithme n'est qu'une généralisation à un type d'architecture plus complexe de celui de l'***Adaline***.

3.5. La rétro-propagation du gradient

Les limitations des modèles neuronaux des années 60 comme l'***Adaline*** ou ***le Perceptron*** [Rosenblatt 57], ont conduit les chercheurs à abandonner progressivement cette ligne de méthodes d'apprentissage au profit des approches symboliques de ce que l'on appelle aujourd'hui l'Intelligence Artificielle classique.

Dans les années 80, grâce aux travaux de Teuvo Kohonen [Kohonen 84] et de J. Hopfield [Hopfield 82], cette voie à été remise au goût du jour et a suscité un intérêt croissant de la part de nombreux chercheurs issus de différentes disciplines.

C'est ainsi que fut proposé simultanément par plusieurs équipes travaillant indépendamment ***l'algorithme de la rétro-propagation du gradient***. David E. Rumelhart, Geoffrey E. Hinton et Ronald J. Williams en 1986 [Rumelhart ***et al.*** 86], ont présenté la méthode, qu'ils ont appelé ***la Règle Delta Généralisée*** [Rumelhart ***et al.*** 86b]. Presque simultanément, Yan le Cun [le Cun 85] et David B. Parker [Parker 85] présentaient chacun séparément des algorithmes d'apprentissage similaires.

Des dérivations formelles de la méthode de rétro-propagation du gradient peuvent être trouvées dans [Fogelman-Soulié ***et al.*** 87] et [le Cun 87].

3.5.1. Une nouvelle architecture

L'apport fondamental de ce nouveau type de réseau a été de proposer une méthode d'apprentissage efficace pour entraîner des réseaux composés d'unités non linéaires assemblés dans des architectures complexes.

Les unités

Dans les perceptrons multicouches, les unités peuvent éventuellement être linéaires comme dans ***Adaline***; elles peuvent être aussi non linéaires et c'est l'intérêt de ce type d'unités. Pour des raisons à la fois historiques et pratiques, les unités employées dans la plupart des modèles neuronaux sont du type quasi-linéaire au sens où on l'a défini en §2.2.5.

Ainsi, l'état d'une cellule *i* est calculé par :

$$x_i = f\left(\sum_{j=1}^{n} w_{ij}\, x_j - \theta_0\right) \qquad (3.13)$$

où f est une fonction différentiable, la plus communément employée étant celle originellement proposée dans le modèles des MLP, il s'agit de la ***fonction sigmoïde*** (cf. §2.2.5); x_j représente les états des cellules connectés à la cellule *i*; w_{ij} le poids de connexion entre les cellules *j* et *i*; et θ_0 le seuil associé à la cellule *i*.

L'organisation des cellules

L'***Adaline*** possède des unités d'entrée et des unités de sortie. Les deux types de cellules étant accessibles par le monde extérieur sont dites unités externes. Elles constituent respectivement l'interface d'entrée et celle de sortie du système.

Les réseaux MLP possèdent le même type d'interface, mais ont également des cellules qui ne sont ni des unités d'entrée ni des unités de sortie et qui sont appelées ***cellules cachées***. L'état des cellules cachées n'étant pas accessible de l'extérieur, il est modifié en fonction des états des autres cellules du réseau. Une donnée présentée en entrée du réseau est propagée à travers les connexions jusqu'aux cellules cachées. De même, les états de cellules cachées sont propagés à travers, éventuellement d'autres ***couches cachées***, jusqu'à la couche de sortie du réseau. La rétro-propagation du gradient permet de travailler avec un réseau à plusieurs couches de neurones, ou réseau multicouches; on parle souvent de ***Perceptron Multicouches***.

L'utilisation des automates quasi-linéaires dans l'algorithme de rétro-propagation du gradient conjuguée aux architectures multicouches vont permettre à cette méthode de résoudre certain type de problèmes qu'il n'était pas possible de résoudre efficacement par les réseaux plus simples comme l'***Adaline*** et le ***Perceptron***.

3.5.2. L'algorithme de rétro-propagation du gradient

Notons x_i l'état d'un neurone i, $x_i \in \mathbf{S}$ qui est l'ensemble des états possibles. Selon (2.3) l'état x_i du neurone i est calculé par :

$$x_i = f(A_i - \theta_i) \qquad (3.14)$$

où, pour les MLP, f est une fonction sigmoïde, A_i l'activité du neurone i et θ_i le seuil associé au neurone i. L'activité A_i est donnée par :

$$A_i = \sum_j w_{ij} x_j \qquad (3.15)$$

où j est indice des neurones "en amont" du neurone i, et w_{ij} est le poids de la connexion du neurone j au neurone i.

Propagation des états

On considère encore une fois l'ensemble des vecteurs d'entrée $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$, définis dans $\mathbb{R}^n$ et les vecteurs réponses $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m$, définis dans $\mathbb{R}^p$ et $\mathbf{W}$ la matrice de poids de connexions cette fois-ci définie dans $\mathbb{R}^{nc \times nc}$, nc étant le nombre total de cellules. Alors,

$$\mathbf{x}_k = \left[x_1^k, x_2^k, \ldots, x_n^k\right]^T, \ \mathbf{y}_k = \left[y_1^k, y_2^k, \ldots, y_p^k\right]^T \text{ et} \qquad (3.16)$$

$$\mathbf{W} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} \\ \mathbf{w}_1 & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{w}_2 & \ldots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{w}_p & \mathbf{0} \end{pmatrix} = \left\{w_{ij}\right\}_{nc \times nc}, \quad \text{où} \quad \mathbf{w}_i = \left\{w_{kl}\right\}_{nc_{(i-1)} \times nc_{(i)}}$$

chaque sous-matrice $\mathbf{w}_i$ contient les poids de connexions entre la couche i-1 –avec $nc_{(i-1)}$ cellules– et la couche i –avec $nc_{(i)}$ cellules–.

A chaque instant k, un vecteur $\mathbf{x}_k$ est présenté dans la couche d'entrée du réseau. Les états des n cellules en entrée prennent les valeurs $(x_1^k, x_2^k, \ldots, x_n^k)$. Ces états sont propagés selon les équations (3.14) et (3.15), vers des unités se trouvant dans les couches en aval de la couche d'entrée jusqu'à arriver à la couche de sortie.

Appelons $\mathbf{s}_k = (s_1^k, s_2^k, \ldots, s_p^k)$ la réponse du réseau pour une entrée $\mathbf{x}_k$ donnée. $\mathbf{s}_k$ est le vecteur composé par les états de p cellules de la couche de sortie. Leurs états ont été calculés par les équations (3.14) et (3.15) en fonction des cellules j connectées à la couche de sortie.

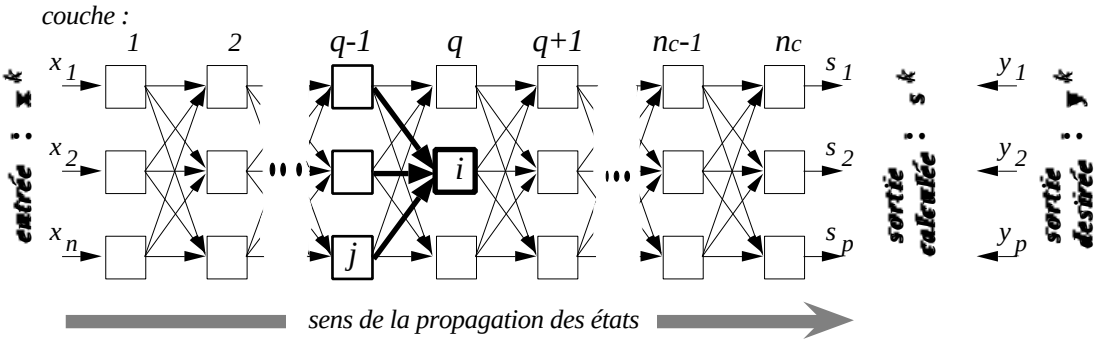


Figure 3.6 : *propagation des états dans un réseau multicouche.La propagation des états se fait "en aval" dès la couche d'entrée vers la couche de sortie. L'état de lacellule i dans la couche q est fonction des états des cellules j des couches précédentes q-1, etc.*

Rétro-propagation de l'erreur

Le but étant d'obtenir pour une entrée $\mathbf{x}_k$ une réponse $\mathbf{s}_k$ la plus proche possible du vecteur $\mathbf{y}_k$ désiré correspondant, on est amené à optimiser notre réseau afin de réduire la différence entre sortie désirée et calculée comme pour l'Adaline.

Définissons la fonction de coût suivante :

$$C^k(\mathbf{W}) = \sum_{i=1}^{p} \left(s_i^k - y_i^k \right)^2 \qquad (3.17)$$

qui est l'erreur ou distance quadratique entre la sortie calculée $\mathbf{s}_k$ et la sortie désirée $\mathbf{y}_k$. La totalité des poids de connexions du réseau doit être ajustée en fonction de cette erreur. Cette fonction de coût dépendra bien sûr de l'état du système et de l' exemple qui lui est présenté.

Pour minimiser la fonction de coût $C^k(\mathbf{W})$ on doit calculer son gradient par rapport à **W**.

$$\frac{\partial C^k(\mathbf{W})}{\partial w_{ij}} = \frac{\partial C^k(\mathbf{W})}{\partial A_i}\frac{\partial A_i}{\partial w_{ij}} = \frac{\partial C^k(\mathbf{W})}{\partial A_i} x_j = \delta_i^k x_j \qquad (3.18)$$

ceci en posant $\delta_i^k = \frac{\partial C^k(\mathbf{W})}{\partial A_i}$

- Pour les cellules *i* de la couche de sortie on peut réécrire δ_i^k de la manière suivante :

$$\delta_i^k = \frac{\partial C^k(\mathbf{W})}{\partial A_i} = \frac{\partial C^k(\mathbf{W})}{\partial s_i} f'(A_i) = 2\left(s_i^k - y_i^k\right) f'(A_i) \qquad (3.19)$$

- Pour les cellules *i* ne faisant pas partie des cellules de la couche de sortie on a :

$$\delta_i^k = \frac{\partial C^k(\mathbf{W})}{\partial A_i} = \frac{\partial C^k(\mathbf{W})}{\partial x_i} f'(A_i)$$

mais

$$\frac{\partial C^k(\mathbf{W})}{\partial x_i} = \sum_l \frac{\partial C^k(\mathbf{W})}{\partial A_l}\frac{\partial A_l}{\partial x_i} = \sum_l \delta_l^k w_{li}$$

où les cellules *l* sont “en aval” de la cellule *i*. On obtient alors :

$$\delta_i^k = f'(A_i) \sum_l \delta_l^k w_{li} \qquad (3.20)$$

pour les neurones *i* dans les couches intermédiaires du réseau.

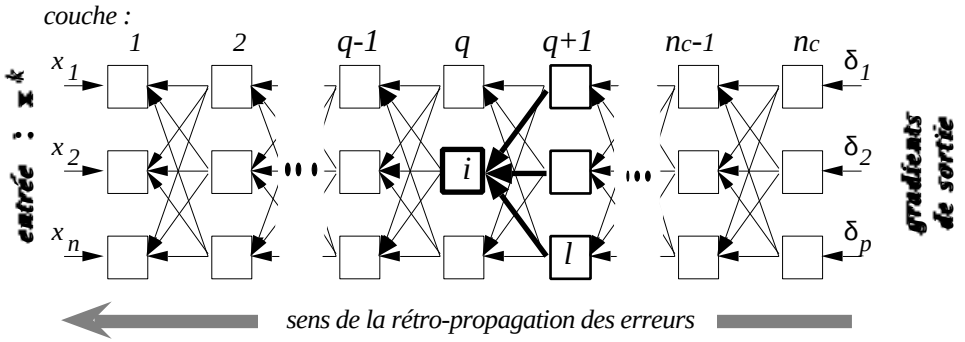


Figure 3.7 : *rétro-propagation des gradients.La rétro-propagation des gradients se fait "en amont" dès la couche de sortie ver la couche d'entrée. Le gradient de la cellule i dans la couche q est fonction des gradients des cellules l des couches supérieures q+1, etc.*

Enfin, selon l'équation générale d'adaptation des coefficients w_{ij} (3.4) :

$$w_{ij}^{k+1} = w_{ij}^{k} + \Delta w_{ij}^{k} \qquad (3.21)$$

dont les Δw_{ij}^{k} sont données par l'expression

$$\Delta w_{ij}^{k} = \varepsilon^{k} \frac{\partial C^{k}(\mathbf{W})}{\partial w_{ij}} = \varepsilon^{k} \delta_{i}^{k} x_{j} \qquad (3.22)$$

La valeur du pas d'apprentissage ε^{k} est typiquement assez petite et décroît dans le temps.

Les gradients δ_{i}^{k} sont donnés par les équations (3.19) et (3.20).

L'algorithme d'apprentissage est résumé dans le tableau ci-dessous. Différents critères peuvent être employés pour arrêter le processus.

Algorithme d'Apprentissage de la rétro-propagation du gradient stochastique

Soit N *le nombre total d'automates d'un réseau,*
c *le nombre de couches ou partitions dans lesquels sont arrangés ces automates,*
n *la taille des vecteurs d'entrée* $\mathbf{x}_k$ *(ou nombre d'automates dans la couche d'entrée), et*
p *la taille des vecteurs de sortie* $\mathbf{y}_k$ *(ou nombre d'automates dans la couche de sortie).*

Soit $(I_q)_{q = 1, 2, \ldots, c}$ *une partition ordonnée de l'ensemble d'automates* $\{1, 2, \ldots, N\}$, *définie telle que pour les automates* j *et* i :

$j \quad I_q, \quad i \quad I_r, \quad q < r \quad j < i$

La première partition $I_1 = (1, 2, \ldots, n)$ *correspond aux indices des n automates représentant la couche d'entrée.*

La dernière partition $I_c = (c-p-1, c-p-2, \ldots, c)$ *correspond aux indices des p automates représentant la couche de sortie.*

Soit $I_0 = (0)$ *une partition contenant seulement un automate, l'automate* 0. *L'état de cet automate est fixé à* –1. *Cet automate est connecté à tous les autres automates du réseau, à l'exception des automates de la couche d'entrée. La valeur* $w_{i,0}$ *du poids de connexion entre cet automate et un automate i quelconque représente le seuil* θ_i *pour ce dernier dans l'équation du calcul d'état (3.14).*

Propagation des états :

1) *Présenter un exemple* $\mathbf{x}_k$
Pour $q = 1$ ..... ≪ *la couche d'entrée* ≫
$i \quad I_q, \quad s_i = x_i^k$

2) *Propager les entrées*
$q \quad (2, \ldots, c), \quad$ ..... ≪ *les couches intermédiaires et la couche de sortie* ≫
$i \quad I_q, \quad A_i = \sum_{j \quad (I_0 \cup I_1 \cup I_2 \cup \ldots \cup I_{q-1})} w_{ij}^k \, s_j$
$s_i = f(A_i)$

3) *Calculer les gradients à partir de la sortie calculée* $\mathbf{s}_k$ *et la sortie désirée* $\mathbf{y}_k$.
Pour $q = c$ ..... ≪ *la couche de sortie* ≫
$i \quad I_q, \quad \delta_i^k = f'(A_i) \left[s_i - y_i^k \right]$

4) *Rétro-propager les gradients*
$q \quad (c-1, c-2, \ldots, 2),$ ..... ≪ *les couches intermédiaires en ordre descendant* ≫
$i \quad I_q, \quad \delta_i^k = f'(A_i) \left[\sum_{l \quad (I_{q+1} \cup I_{q+2} \cup \ldots \cup I_c)} \delta_l^k \, w_{li}^k \right]$

5) *Correction des poids de connexions*
$q \quad (2, \ldots, c), \quad$ ..... ≪ *les couches intermédiaires et la couche de sortie* ≫
$i \quad I_q,$
$j \quad (I_0 \cup I_1 \cup I_2 \cup \ldots \cup I_{q-1})$
$w_{ij}^{k+1} = w_{ij}^k + \varepsilon^k \, \delta_i^k \, x_j$

6) *Répéter à partir de (1) avec un nouvel exemple* $\mathbf{x}_k$.

Gradient stochastique ou gradient déterministe

La fonction de coût définie dans la ***méthode du gradient***, telle qu'on l'a vue lors de la présentation d'***Adaline*** (définie dans l'équation (3.7)), est la moyenne du coût donnée par l'équation (3.17) pour les *m* exemples constituant la base d'apprentissage i.e. :

$$C(\mathbf{W}) = \frac{1}{2}\sum_{k=1}^{m}\left(\mathbf{F}(\mathbf{x}_k,\mathbf{W}) - \mathbf{y}_k\right)^2$$

où $\mathbf{F}(\mathbf{x}_k,\mathbf{W})$ correspond à la sortie du réseau après présentation de $\mathbf{x}_k$, et $\mathbf{y}_k$ est la sortie désirée associée. Dans ce cas, les poids des connexions sont ajustés seulement après avoir présenté tous les exemples et calculé le coût global donné pour l'équation précédente. Ceci correspond au ***gradient total*** calculé lors d'une minimisation classique par ***descente de gradient***.

Il a été démontré que la méthode utilisée dans nos travaux, où les paramètres du réseau sont modifiés selon le gradient calculé pour chaque exemple (fonction de coût donnée par (3.17)), appelée ***gradient stochastique***, converge avec probabilité 1 vers un minimum local tout en étant moins prédisposée que la vraie méthode de la descente de gradient à rester bloquée dans des minima inintéressants [Bottou 91].

Cet algorithme présente de nombreux avantages, outre son adaptativité; il est totalement général et peut être adapté à tout type d'architecture. Nous détaillerons par la suite quelques cas intéressants.

Algorithme avec poids partagés

Dans le cas de poids partagés la minimisation de la fonction de coût $C^k(\mathbf{W})$ par rapport à $\mathbf{W}$ varie légèrement par rapport à (3.18). En effet, l'algorithme doit assurer une mise à jour des poids afin de donner la même valeur aux poids partagés.

Soit w_{ij} le poids de connexion entre les cellules j et i du réseau. Ce poids étant partagé par d'autres connexions entre cellules, appelons N_l l'ensemble des couples de cellules qui partagent la connexion w_{ij}, et d'une façon plus générale ω_l le poids w_{ij} qui qui les relient. On a donc : $\mathrm{N}_l = \{(i_1,j_1), (i_2,j_2), \ldots\}$. Le gradient de la fonction de coût $C^k(\mathbf{W})$ par rapport à ω_l est donné par :

$$\frac{\partial C^k(\mathbf{W})}{\partial \omega_l} = \sum_{(i,j)\ \mathrm{N}_l} \frac{\partial C^k(\mathbf{W})}{\partial A_i} \frac{\partial A_i}{\partial w_{ij}} = \sum_{(i,j)\ \mathrm{N}_l} \delta_i^k x_j \qquad (3.23)$$

De cette façon, on peut modifier le pas n° 5 dans l'algorithme de la rétro-propagation du gradient stochastique et donner une forme plus générale de la mise au point des poids de connexions :

5)	*Correction des poids de connexions*
	q $(2, \ldots, P)$, ... ≪ *les couches intermédiaires et la couche de sortie* ≫
	ω_l {*ensemble de poids* w_{ij} *arrivant aux cellules i de la partition* I_q}
	$\omega_l^{k+1} = \omega_l^k + \varepsilon^k \sum_{(i,j)\ \mathrm{N}_l} \delta_i^k x_j$
	≪ *où* N_l *est l'ensemble de couples de cellules* (i,j) *partageant le poids* ω_l, ≫
	≪ *pour* i I_q *et* j $\left(I_0 \cup I_1 \cup I_2 \cup \ldots \cup I_{q-1}\right)$ ≫

Notons cependant que cette façon de modifier les poids englobe le cas où il n'y a pas de poids partagés et où tous les poids sont différents. Dans ce cas, les différents ensembles N_l ne contiennent qu'un seul couple (i, j) indiquant les deux cellules correspondant à la connexion qui a comme poids $\omega_l = w_{ij}$.

Le pas d'apprentissage

L'apprentissage avec la rétro-propagation du gradient dépend fortement de la valeur donnée au scalaire ε^k dans l'expression (3.21). Pour la mise en oeuvre la plus simple de cet algorithme, ε est un scalaire unique pour tout le réseau. Cette valeur est modifiée –diminuée– manuellement au fur et à mesure que le réseau apprend. Il n'est pas simple de faire manuellement une modification optimale de cette valeur : une valeur trop importante va très vite faire diverger le réseau de la solution recherchée –les poids tombent dans la zone de saturation des fonctions sigmoïdes utilisées comme fonctions de transition–. Une trop petite valeur ralentira énormément le processus d'apprentissage. Avec un peu d'habitude, on peut bien sûr devenir expert dans les modifications des valeurs de ε pendant l'apprentissage et arriver ainsi à obtenir de très bonnes courbes de convergences. Toutefois, ce procédé n'est pas totalement satisfaisant et l'utilisateur novice aimerait bien disposer de procédures automatiques de variation de ε.

La valeur optimale de ε dépend à la fois du problème à résoudre et de la nature du réseau utilisé. Intuitivement, on peut très bien comprendre que dans une région relativement plate de la fonction de coût, on ait intérêt à aller beaucoup plus vite que dans le cas d'une région ayant des pentes raides. Une valeur importante d'ε accélère la convergence pour le premier cas tandis que pour le deuxième elle peut faire diverger la procédure.

La connaissance d'une information locale sur la forme de l'espace des erreurs pourrait nous permettre de choisir le bon ε, itération après itération et cellule par cellule. Cette connaissance est possible en exploitant l'information des dérivées de la fonction de coût d'ordre supérieur à 1.

Dérivées de second ordre : méthode de Newton

Pour trouver une solution $\mathbf{W}^*$ qui minimise $C(\mathbf{W})$ la méthode précédente utilise uniquement l'information donnée par la dérivée de premier ordre. Il est connu en optimisation classique que l'utilisation des informations données par les dérivées d'ordre supérieur peut accélérer le processus de recherche d'une solution optimale. De nombreux travaux ont tenté d'étendre cette approche aux techniques adaptatives dans le cadre des réseaux de neurones.

On peut ainsi citer [Waltrous 87, Becker & le Cun 89, Prina ***et al.*** 88]. Tous partent de la méthode de Newton, et se différencient selon les approximations qu'ils proposent pour formuler des algorithmes du second ordre adaptatifs.

Nous présentons les notions de base communes aux méthodes du second ordre et qui sont nécessaires pour comprendre ces différents algorithmes. Nous renvoyons aux différents papiers cités pour une description des algorithmes eux mêmes.

Le développement de séries de Taylor pour la fonction de coût $C^k(\mathbf{W})$ autour du point $\mathbf{W}^\circ$ donne :

$$C\left(\mathbf{W}^\circ + \Delta\mathbf{W}\right) \equiv C(\mathbf{W}^\circ) + C(\mathbf{W})\Big|_{\mathbf{W}^\circ}\cdot\Delta\mathbf{W} + \frac{1}{2}\Delta\mathbf{W}^{\mathrm{T}}\cdot H|_{\mathbf{W}^\circ}\cdot\Delta\mathbf{W} + \ldots \qquad (3.24)$$

où H –la matrice Hessienne– représente le terme de deuxième ordre, et les points indiquent les termes de troisième ordre et d'ordres supérieurs. Ces termes sont en pratique presque nuls et ne sont pas pris en compte.

La solution qui minimise (3.24) est de la forme :

$$\mathbf{W}^{k+1} = \mathbf{W}^k - \varepsilon^k H^{-1}\big|_{\mathbf{W}^k} \cdot C(\mathbf{W})\big|_{\mathbf{W}^k} \qquad (3.25)$$

Pour $\varepsilon^k = 1$ cette équation représente la ***méthode de Newton*** pour résoudre des systèmes d'équations non-linéaires.

La solution de H^{-1}, dans le cas général, présente des difficultés d'ordre divers. L'une d'entre elles est une question de moyens physiques : pour une application de grand taille où l'espace des exemples est d'une taille considérable, le matrice **W** peut atteindre des dimensions qui font de la gestion du hessien H –stockage et inversion– une tâche presque impossible. En effet, si n est le nombre de poids, l'opération d'inversion du hessien est d'une complexité très élevée ($\approx O(n^3)$, i.e. le nombre d'opérations nécessaires pour inverser H est de l'ordre de $n^3!$). Une telle quantité d'opérations exclurait de fait l'utilisation pratique de cette méthode dans des cas réels de grand taille.

Des méthodes numériques s'avèrent nécessaires pour résoudre cette relation de façon plus efficace en temps de calcul et en gestion de mémoire. C'est dans la méthode utilisée et les simplifications faites pour résoudre H^{-1} dans l'expression (3.25) que se différencient entre eux les différents travaux cités ci-dessus ainsi que la méthode utilisée dans nos travaux. Dans les apprentissages effectués pour nos travaux, on a utilisé d'abord une approximation diagonale de H^{-1} à chaque présentation – chaque instant k – et après la ***méthode de Levenberg-Marquardt*** pour résoudre cette approximation (voir [Press ***et al.*** 88] pour les détails de calcul).

3.6. L'approche modulaire

Dans les sections précédentes, nous avons introduit les algorithmes que nous avons utilisé pour nos différentes applications ainsi que différentes variantes qui en constituent des adaptations pour des architectures données

ou des améliorations. Ce type de réseau de neurones permet de réaliser des tâches extrêmement complexes et permet de construire des machines dont la puissance a de nos jours été largement démontrée tant au niveau pratique que théorique. Toutefois, différents problèmes liés à la qualité des estimateurs obtenus, à la quantité de calculs nécessaire pour réaliser l'apprentissage et donc au temps de mise en oeuvre ou de test de ces machines, rendend leur utilisation brute inefficace pour des problèmes complexes ou de très grande taille. Nous avons, au sein de l'équipe, rapidement été confronté à ce problème dès que nous avons essayé de résoudre des applications de ce type. La solution que nous avons trouvée, qui s'est révélée la seule viable au travers des différentes expériences que nous avons eus sur des problèmes de taille importante, est de construire des systèmes modulaires. De nombreux travaux ont été réalisés pour à la fois établir une théorie de ces systèmes et permettre leur mise en oeuvre pratique. Citons par exemple les ceux effectués par Léon Bottou [Bottou 91], Michel de Bollivier [Bollivier 92] et Younès Bennani [Bennani 92] au sein de l'équipe.

L'utilisation d'une approche modulaire est une des façons d'introduire de la connaissance dans la résolution des problèmes[9] par des méthodes connexionnistes. Les différents modules et leur assemblage permettent d'intégrer des connaissances élémentaires que l'on possède sur le problème à résoudre et cela de façon extrêmement simple. Les architectures ainsi construites permettent de réduire considérablement la dimension de l'espace de recherche de la procédure d'apprentissage par rapport à l'utilisation d'un réseau monolithique. Elles permettent également de guider la recherche vers des solutions intéressantes et d'éviter des minima locaux dangereux. Contrairement à ce qui avait été fait dans les années 60 pour les ***Madalines*** par exemple, ce type d'approche est parfaitement utilisable pour des problèmes de grande taille. La décomposition d'un problème complexe en plusieurs sous-problèmes traités chacun de façon modulaire par un réseau particulier amène souvent à une meilleure approche du problème.

Dans l'application principale présentée dans ce mémoire, le calcul des caractéristiques du vent sur l'océan en fonction des mesures radar, nous avons effectué une décomposition suivant deux directions :

---

9 L'expression "résolution des problèmes" est assez large, nous nous limitons ici à des problèmes de classification de signaux ainsi qu'à la simulation des fonctions de transfert.

- ***Décomposition verticale*** : la tâche à résoudre est décomposée en plusieurs sous-tâches, chacune pouvant être décomposée à son tour. La solution se fait en résolvant successivement les différentes tâches. L'information issue de la solution d'une tâche peut s'avérer indispensable à la solution d'autres tâches dans des niveaux supérieurs.

- ***Décomposition horizontale*** : dans certains types de problèmes, et dans le nôtre en particulier, la tâche à résoudre doit se faire systématiquement à partir de données issues de différentes conditions de prise d'information connues. Souvent, un simple traitement sur les données peut suffire à l'homogénéisation de l'information. D'autres fois, ce n'est pas si simple, et un réseau –ou un ensemble de réseaux– dédié à la solution du problème s'avère nécessaire pour la solution de la tâche pour chaque source de données. La décomposition horizontale est aussi effectuée quand une tâche peut se décomposer en plusieurs sous tâches indépendantes; dans ce cas aussi, il y aura un réseau par sous tâche.

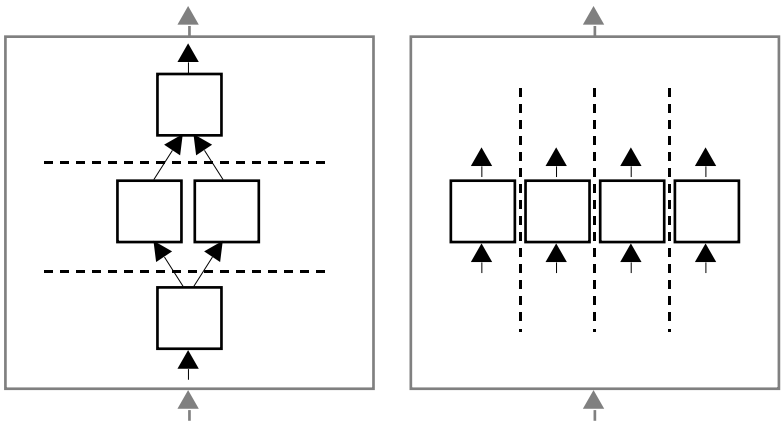


Figure 3.8 : *Décomposition verticale et horizontale d'une tâche. (a) Dans le cas de décomposition verticale, la tâche est divisé en sous-tâches qui coopèrent de façon ascendante à la solution complète : la solution d'une sous-tâche peut être relevante dans la solution d'une autre plus haute, une résolution des tâches en série s'impose. (b) Dans le cas horizontal, la tâche peut se décomposer en plusieurs sous-tâches indépendantes, là nous pouvons intervenir de façon simultanée dans la résolution des sous-tâches.*

La coopération est donc effectuée par les réseaux en alternant les modes parallèle et séquentiel. Les réseaux appartenant à un même niveau sont indépendants entre eux et peuvent donc travailler en parallèle. L'exécution séquentielle est dûe à l'interdépendance des entrées aux différents niveaux du système ainsi que celle des sous-tâches. En effet, la décomposition que

nous avons effectuée est dans le sens vertical et les calculs des réseaux d'un niveau donné s'appuient sur les résultats du niveau précédent. L'ensemble constitue une architecture spécialisée massivement parallèle. Ce type d'architecture nous a permis d'atteindre à moindre coût des performances élevées.

L'utilisation d'une approche modulaire a apporté à nos travaux principalement les caractéristiques suivantes :

- ***l'efficacité*** : la solution du problème traité s'est avérée très difficile à l'aide d'un réseau unique. La décomposition du problème en sous-tâches nous à permis de le traiter efficacement à l'aide des réseaux neuronaux.

- ***la précision*** : la séparation du problème en sous-tâches, la délimitation du problème selon différentes hiérarchies de données et la décomposition selon ces hiérarchies nous a permis d'améliorer les précisions dans les paramètres calculés.

- ***la rapidité des calculs*** : la solution d'une tâche complexe à l'aide des architectures de réseaux modulaires conduit à une réduction de la taille des réseaux, la résolution de la tâche avec un unique réseau nécessitant un réseau d'une taille et une complexité supérieures. Ceci se traduit par une accélération du calcul. La décomposition horizontale, du fait de l'indépendance des calculs, permet d'envisager l'exécution parallèle des modules avec les améliorations évidentes dans le temps de traitement que ceci entraîne.

Grâce à tous ces concepts, nous avons défini une méthodologie qui permet de construire des systèmes de réseaux de neurones pour traiter des problèmes complexes souvent liés aux applications réelles.

4. Propriétés théoriques des MLP

4.1. Approximation des fonctions par réseaux multicouches

4.1.1. Familles des fonctions générées par les réseaux MLP

Considérons des architectures du type réseaux multicouches définies dans la section §2.4. Pour ce type d'architectures, les automates sont répartis par niveaux et les connexions sont toujours dirigées des couches inférieures vers les couches supérieures. Notons par c le nombre de couches du réseau, f la fonction sigmoïde utilisée par chaque automate pour calculer son input et $\mathbf{W}$ la matrice des poids du réseau.

Nous supposons par la suite que la couche d'entrée contient n cellules et celle de sortie contient p cellules. De même, nous supposons m le nombre d'exemples.

Pour tout exemple $\mathbf{x} = (x_1, \ldots, x_n)$, $\mathbf{x} \in \mathbb{R}^n$, présentée à la couche d'entrée, le réseau calcule sur ses p cellules de sortie un vecteur de $\mathbb{R}^p$: $\mathbf{F}(\mathbf{x},\mathbf{W}) = (F_1(\mathbf{x},\mathbf{W}), \ldots, F_p(\mathbf{x},\mathbf{W}))$. Ainsi le réseau définit une fonction de $\mathbb{R}^n$ dans $\mathbb{R}^p$ notée par $\mathbf{F}(\cdot,\mathbf{W}) = (F_1(\cdot,\mathbf{W}), \ldots, F_p(\cdot,\mathbf{W}))$.

Quand nous considérons un réseau avec une architecture A bien déterminée (le nombre de couches, le nombre de cellules par couche, le type de fonction de transition et les connexions entre cellules) nous spécifions une des familles de fonctions $F_A = \{\mathbf{F}(\cdot,\mathbf{W}) \mid \mathbf{W}\}$ continues de $\mathbb{R}^n$ dans $\mathbb{R}^p$. Les différentes architectures définissent différentes classes de fonctions F_A. L'ensemble des classes définit une famille de fonctions continues de $\mathbb{R}^n$ dans $\mathbb{R}^p$ notée par F.

Nous allons maintenant caractériser cette famille de fonction F_A. Pour les architectures considérées, les connexions vont toujours des couches

inférieures vers les couches supérieures ($i < j \;\; w_{ij} = 0$), il est possible de représenter l'ensemble des valeurs numériques des connexions dans une matrice carrée par blocs **W** de dimension (*nb*, *nb*) dans laquelle *nb* représente le nombre total d'automates. Chaque bloc représentant l'ensemble des poids joignant les automates de deux couches adjacentes.

Si l'on appelle $n_0 = n, n_1, .., n_{c-2}, n_{c-1} = p$ le nombre d'automates respectifs des *c* couches, la matrice **W** (par exemple pour $c = 4$) a la forme suivante :

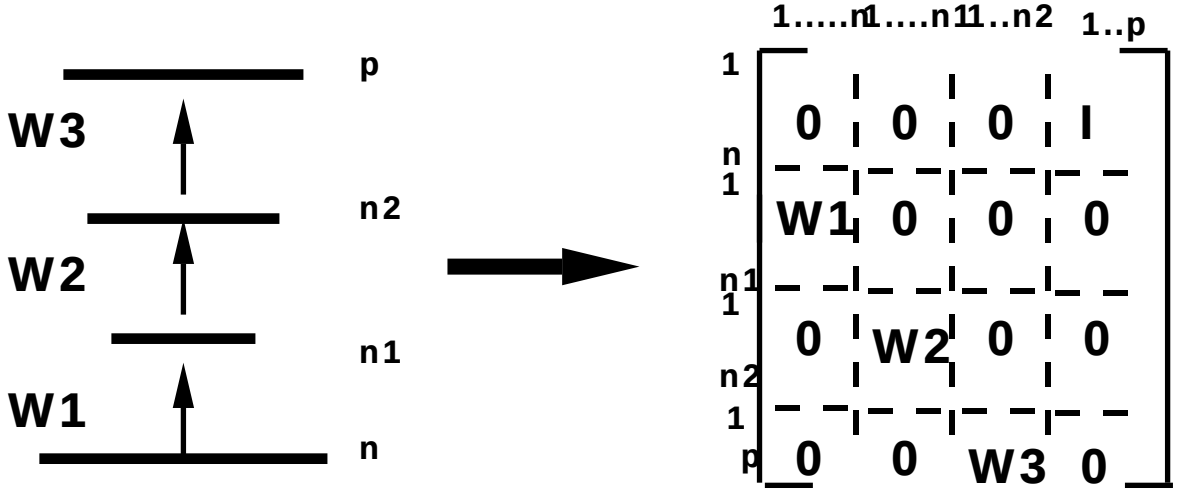


Figure 4.1 : *matrice des poids **W** d'un réseau à c=4 couches de cellules et 3 couches de poids.*

où $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3, \ldots$ représentent les matrices de connexions liant la couche *i*-1 à la couche *i* pour *i* variant de 1 à *c*–1.

Un état du réseau peut donc être représenté par un vecteur de dimension *nb* dont chaque composante est l'état d'un automate du réseau. Nous noterons f_{sig} la fonction vectorielle de dimension *nb* dont chaque élément est une fonction sigmoïde. Une forme $\mathbf{x} = (x_1,..., x_n)$ de dimension *n* présentée en entrée du réseau se représente donc par un vecteur de dimension *nb*, où $nb > n$, $\mathbf{x} = (x_1,..., x_n, 0, \ldots, 0)$. Si l'on applique au réseau une dynamique de changement d'état bloc-séquentielle (cf. §2.4.2), le calcul des valeurs de sortie du MLP s'obtient en *c*–1 itérations de la fonction $(f_{sig} \circ \mathbf{W})$. A chaque itération, le calcul du nouveau vecteur d'état s'obtient à l'aide de la matrice de connexions **W**. Par exemple, la première itération donne :

$$x^{(1)} = (f_{sig} \circ \mathbf{W}) = (0, \ldots, 0, s_1, \ldots, s_{n1}, 0, \ldots, 0)$$

où s_1, ..., s_{n1} représentent les états des automates de la première couche cachée. Au bout de c-1 itérations on obtient le vecteur de sortie étendu :

$\mathbf{s} = (f_{sig} \circ \mathbf{W})^{c-1}\mathbf{x} = (0, \ldots, 0, s_1, \ldots, s_p)$ où $(s_1, \ldots, s_p)$ est le vecteur de sortie classique du réseau. La famille de fonctions F_A représentées par une architecture A donnée est donc $f = (f_{sig} \circ \mathbf{W})^{c-1}$. Chaque matrice $\mathbf{W}$ du type précédent permet d'obtenir un élément de cette famille.

4.1.2. Approximation d'une fonction continue par réseaux MLP

Approximer une fonction $\mathbf{T}$ quelconque de $\mathbb{R}^n$ dans $\mathbb{R}^p$ à l'aide d'un réseau MLP implique le fait de choisir une architecture de réseau dont la première couche possède n automates et la dernière p, ainsi qu'un système de poids associés. Ceci revient à choisir une fonction dans la famille de fonctions F. L'architecture A du réseau, et en particulier le nombre de couches intermédiaires et le nombre d'automates du réseau, est choisie en fonction de la complexité de la fonction à approximer. Ce choix fixe ***a priori*** la famille F_A à l'intérieur de laquelle nous désirons trouver l'approximation. Des résultats expérimentaux et théoriques permettent de montrer que 1 ou 2 couches de cellules cachées suffisent pour approximer un grand nombre de fonctions [Lippmann 87, Cybenko 89, Funahashi 89 et Hornik et al. 89].

Nous rappelons ici un résultat théorique sous sa forme la plus générale montrant que toute fonction continue d'un compact de $\mathbb{R}^n$ dans $\mathbb{R}^p$ peut être approximée par un réseau multicouche dont les automates des couches cachées utilisent des fonctions sigmoïdes et dont les automates de sortie sont linéaires. Nous renvoyons à [Cybenko 89] pour les démonstrations.

> ***Théorème :*** *Soit* $f(\mathbf{x})$ *une fonction non-constante, bornée, monotone croissante et continue. Soit* $\mathbf{K}$ *un sous ensemble compact de* $\mathbb{R}^n$. *Pour toute fonction* $\mathbf{T}$ *continue* $\mathbf{T} : \mathbf{K} \rightarrow \mathbb{R}^p$, *définie par* $\mathbf{x} = (x_1, x_2, \ldots, x_n) \rightarrow (T_1(\mathbf{x}), \ldots, T_p(\mathbf{x}))$, *et pour tout* $\varepsilon > 0$, *il existe un réseau* A *à une couche cachée, dont la fonction de transition est* $f(\mathbf{x})$ *pour chacune des cellules cachées et linéaire pour les cellules de sortie, tel que :*
>
> $$max_{\mathbf{x} \in \mathbf{K}}[d(\mathbf{T}(\mathbf{x}) - \mathbf{F}(\mathbf{x}, \mathbf{W}))] < \varepsilon$$
>
> *où* $\mathbf{W}$ *est la matrice des poids de connexions associée,* $d(\cdot)$ *la métrique usuelle de* $\mathbb{R}^p$ *et* $\mathbf{F}(\cdot, \mathbf{W})$ *la fonction générée par le réseau.*

Le résultat précédent démontre l'existence d'un réseau à une seule couche cachée permettant d'approximer une fonction continue **T** à un ε près dans un sous ensemble compact **K** de R^n. Cependant, ce résultat ne nous donne pas d'indications précises sur le nombre de cellules cachées à considérer. Expérimentalement, il semble qu'augmenter le nombre de couches permette d'atteindre l'approximation cherchée à moindre coût (nombre de connexions et de cellules cachées).

Un travail récent qui va dans ce sens est celui de Blum et Li [Blum et Li 91]. Dans ce travail, les auteurs étudient l'approximation des fonctions à l'aide de réseaux multicouches dans lesquelles les fonctions de transition des unités cachées sont des fonctions à seuil à valeurs dans {1, 0}, alors que celles des unités de sorties sont linéaires. Nous notons par la suite ces réseaux par $\boldsymbol{BL}$. Les fonctions générées par de tels réseaux sont des fonctions constantes par morceaux. Une fonction $g : \mathbf{K} \subset R^n \rightarrow R$ définie sur le compact **K** est dite constante par morceaux s'il existe une partition finie de **K** $\{\boldsymbol{D_i}\}_{\boldsymbol{i=1..n}}$ tel que *g* soit constante dans chaque $\boldsymbol{D_i}$.

Les auteurs démontrent qu'étant donnée une fonction continue $f : \mathbf{K} \subset R^n \rightarrow R$ il existe un réseau $\boldsymbol{BL}$ ayant deux couches cachées qui approxime uniformément sur **K** la fonction *f* à un ε près. La démonstration utilise le fait que *f* peut être approximée uniformément sur **K**, et à un ε près, par une fonction simple *g* dont la partition associée est formée par *m* hypercubes. Ils construisent alors un réseau $\boldsymbol{BL}$ à deux couches cachées qui génère exactement la fonction *g*, la construction de ce réseau nécessite $2 \times n \times m$ unités sur la première couche cachée et *m* unités sur la seconde, le nombre de poids de ce réseau étant égal à $m \times (4n+1)$. Le problème de la détermination du nombre *m*, pour une fonction *f* donnée et pour une précision ε souhaitée, nécessite des informations globales sur *f*. Ces informations peuvent être données par l'une des propriétés suivantes :

- *f* continue sur **K** implique qu'elle est uniformément continue, donc pour tout $\varepsilon > 0$ il existe $\omega(\varepsilon)$ telle que si $\|x - y\| < \omega(\varepsilon)$ alors $\|f(x) - f(y)\| < \varepsilon$. Ainsi *m* est égal au plus petit entier supérieur ou égal à $\frac{1}{\omega(\varepsilon)}$.
- Si *f* est continûment dérivable avec $\|f'\| \leq k$, alors, pour $\|x - y\|$ suffisamment petite, nous avons $\|f(x) - f(y)\| \leq k\|x - y\|$. Nous pouvons prendre alors $m \geq \frac{k}{\varepsilon}$.

- Pareillement si f est Lipschitzienne (i.e. satisfaisant $||f(x) - f(y)|| \leq k||x - y||$) de constante k, alors pour tout ε nous pouvons prendre $m \geq \frac{k}{\varepsilon}$.

Ces trois propriétés donnent une borne supérieure du nombre des cellules cachées. Elles nous précisent aussi que, dans le cas général, ce nombre est inversement proportionnel au ***module de continuité*** $\omega(\varepsilon)$ de la fonction f.

4.2. Propriétés des réseaux pris en tant que classifieurs

Les réseaux multicouches sont très souvent utilisés en tant que classifieurs (voir §3.3). Une des méthodes les plus utilisées en théorie statistique de la décision est la règle de décision de Bayes pour un risque minimum. C'est une méthode probabiliste qui suppose que des informations probabilistes sur les nuages de points constitués par les différentes classes sont accessibles. La détermination de cette fonction nécessite donc la connaissance de valeurs qui ne sont pas directement accessibles à l'expérience.

Dès que le nombre d'éléments de l'ensemble d'apprentissage est trop élevé, il est impossible d'estimer avec précision les valeurs de $p(X)$ et $p(X \mid F^i)$, densité et densité conditionnelle de probabilité de X, sans faire d'hypothèses sur les propriétés que vérifient les exemples. Il existe alors deux sortes de méthodes paramétriques :

- les méthodes paramétriques Bayesiennes permettent d'estimer la densité de probabilité, en recherchant la meilleure densité à l'intérieur d'une famille de densité, pour en déduire selon la théorie Bayesienne les surfaces séparatrices.

- les méthodes paramétriques non Bayesiennes permettent de construire directement les surfaces séparatrices choisies à l'intérieur d'une famille de surfaces, sans passer par les densités de probabilité. On peut ici aussi parler d'approche paramétrique, puisque l'on s'intéresse à une famille de surfaces discriminantes engendrée par des paramètres dont nous cherchons à déterminer les meilleures valeurs possibles.

Les réseaux multicouches comme nous venons de le rappeler appartiennent aux méthodes paramétriques non Bayesiennes. Leur emploi

permet de ne faire aucune hypothèse sur les distributions de probabilités contrairement aux méthodes paramétriques Bayesiennes qui présupposent des conditions très fortes, comme celle de la distribution Gaussienne des exemples dans chaque classe, ce qui est le plus souvent irréaliste.

Le formalisme de modélisation utilisé par les réseaux est le même, mais les réponses $\mathbf{y}_k$ désirées doivent maintenant permettre de classer les différentes formes. Pour un vecteur d'entrée $\mathbf{x}_k$ donné, si l'on choisit un codage en $\mathbf{S} = \{a, b\}$, nous prenons :

$$\mathbf{y}_k = (b, b, \ldots, a, b \ldots, b)$$

avec $y_i^k = a$, seulement si $\mathbf{x}_k$ appartient à la classe ***i***.

Si nous appelons F_A la famille de fonctions générées par l'architecture du réseau choisi (voir la section précédente) nous cherchons à déterminer la fonction $\mathbf{F}(\mathbf{x},\mathbf{W}) = \left(F_1(\mathbf{x},\mathbf{W}), \ldots, F_p(\mathbf{x},\mathbf{W})\right)$ de F_A qui minimise la fonction de coût $C(m,\mathbf{W})$:

$$C(m,\mathbf{W}) = \sum_{k=1}^{m} \left[\sum_{i=1}^{p} \left(F_i(\mathbf{x}_k,\mathbf{W}) - y_i^k \right)^2 \right] \qquad (4.1)$$

m étant le nombre d'exemples.

4.2.1. Réseaux multicouches et règle de décision de Bayes

Notations :

$\mathbf{X}$: ensemble de toutes les formes d'apprentissage.

$\mathbf{x}$: forme quelconque de $\mathbf{X}$.

p : nombre de classes.

Φ_i : ensemble des formes de la classe i, et par extension désigne aussi la classe.

$\Phi_{\bar{i}}$: ensemble des formes qui n'appartiennent pas à la classe i.

m : nombre de formes ou cardinal de $\mathbf{X}$.

m_i : l'effectif de la classe Φ_i.

$m_{\bar{i}}$: l'effectif de $\Phi_{\bar{i}}$; $m_{\bar{i}} = m - m_i$.

Nous supposons connus :

- $p(\mathbf{x})$ la fonction densité de probabilité au point $\mathbf{x}$.
- $P_i = P(\Phi_i)$ la probabilité ***a priori*** de la classe Φ_i.
- $P_{\bar{i}} = P(\Phi_{\bar{i}})$ la probabilité ***a priori*** de $\Phi_{\bar{i}}$; $P_{\bar{i}} = 1 - P_i$.
- $p(\mathbf{x} \mid \Phi_i)$ la densité de probabilité au point $\mathbf{x}$, conditionnée par la classe Φ_i.

Pour une forme $\mathbf{x}$, la probabilité d'appartenance à la classe Φ_i peut être calculée à l'aide de la formule de Bayes.

$$P(\Phi_i \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \Phi_i)\cdot P_i}{p(\mathbf{x})} \qquad (4.2)$$

avec

$$p(\mathbf{x}) = \sum_{i=1}^{p} p(\mathbf{x} \mid \Phi_i)\cdot P_i \qquad (4.3)$$

La règle de décision de Bayes est la suivante :

Décider Φ_i *si* $P(\Phi_i \mid \mathbf{x}) = sup_j\ P(\Phi_j \mid \mathbf{x})$

Ce qui est équivalent à :

Décider Φ_i *si* $p(\mathbf{x} \mid \Phi_i)\cdot P_i = sup_j\ p(\mathbf{x} \mid \Phi_j)\cdot P_j$

Nous définissons la fonction discriminante de Bayes :

$$g(\mathbf{x}) = (g_1, \ldots, g_p) \qquad avec\ \ g_i = P(\Phi_i \mid \mathbf{x}) - \sum_{j \neq i} P(\Phi_j \mid \mathbf{x})$$

La règle de décision de Bayes écrit :

Décider Φ_i *si* $g_i(\mathbf{x}) \geq g_j(\mathbf{x}) \qquad j \neq i$

Nous définissons un critère de décision $\alpha(\cdot)$ comme étant une fonction de $R^n \rightarrow \{1, 2, \ldots, p\}$. Pour un $\mathbf{x}$ donné ce critère permet d'affecter $\mathbf{x}$ à la classe $\Phi_{\alpha(x)}$. Dans ce cas :

$$P(erreur \mid \mathbf{x}) = \sum_{j \neq \alpha(x)} P(\Phi_j \mid \mathbf{x}) = 1 - P(\Phi_{\alpha(x)} \mid \mathbf{x})$$

et la probabilité d'erreur totale :

$$P(erreur) = \int P(erreur \mid \mathbf{x}) \cdot p(\mathbf{x}) dx = \int \left[1 - P(\Phi_{\alpha(x)} \mid \mathbf{x})\right] \cdot p(\mathbf{x}) dx$$

Il n'est pas très difficile de voir que la règle de décision de Bayes énoncée ci-dessus est celle qui minimise la probabilité d'erreur [Duda et Hart 73].

4.2.2. Apprentissage par un réseau des fonctions de décisions de Bayes

Nous définissons :

$$C_k(m, \mathbf{W}) = \sum_{\mathbf{x} \; \Phi_k} \left[F_k(\mathbf{x}, \mathbf{W}) - a\right]^2 + \sum_{\mathbf{x} \; \Phi_k} \left[F_k(\mathbf{x}, \mathbf{W}) - b\right]^2 \qquad (4.4)$$

et

$$C(m, \mathbf{W}) = \sum_{k=1}^{p} C_k(m, \mathbf{W}) \qquad (4.5)$$

Nous pouvons écrire :

$$C(m, \mathbf{W}) = \qquad (4.6)$$

$$m \cdot \sum_{k=1}^{p} \left[\sum_{\mathbf{x} \; \Phi_k} \frac{m_k}{m} \cdot \frac{1}{m_k} \left(F_k(\mathbf{x}, \mathbf{W}) - a\right)^2 + \sum_{\mathbf{x} \; \Phi_{\overline{k}}} \frac{m_{\overline{k}}}{m} \cdot \frac{1}{m_{\overline{k}}} \left(F_k(\mathbf{x}, \mathbf{W}) - b\right)^2 \right]$$

Ainsi, quand $m \rightarrow \infty$ l'expression $\frac{C(m, \mathbf{W})}{m}$ tend vers :

$$\lim_{m \rightarrow \infty} \frac{C(m, \mathbf{W})}{m} = \qquad (4.7)$$

$$\sum_{k=1}^{p} \left[P_k \, E_k \left\{ \left(F_k(\mathbf{x}, \mathbf{W}) - a\right)^2 \right\} + P_{\overline{k}} \, E_{\overline{k}} \left\{ \left(F_k(\mathbf{x}, \mathbf{W}) - b\right)^2 \right\} \right]$$

où

$$E_k\left\{\left(\mathrm{F}_k(\mathbf{x},\mathbf{W}) - a\right)^2\right\} = \int\left(\mathrm{F}_k(\mathbf{x},\mathbf{W}) - a\right)^2 p(\mathbf{x} \mid \Phi_k)\, dx$$

$$E_{\overline{k}}\left\{\left(\mathrm{F}_k(\mathbf{x},\mathbf{W}) - b\right)^2\right\} = \int\left(\mathrm{F}_k(\mathbf{x},\mathbf{W}) - b\right)^2 p(\mathbf{x} \mid \Phi_{\overline{k}})\, dx$$

$\Phi_{\overline{k}}$ étant le complémentaire de Φ_k. Compte tenu de $p(\mathbf{x} \mid \Phi_k)P_k = P(\Phi_k \mid \mathbf{x})p(\mathbf{x})$ l'expression (4.7) devient alors :

$$\lim_{m\to\infty} \frac{C(m,\mathbf{W})}{m} =$$

$$\sum_{k=1}^{p}\left[\int\left(\mathrm{F}_k(\mathbf{x},\mathbf{W}) - a\right)^2 P(\Phi_k \mid \mathbf{x})p(\mathbf{x})dx + \int\left(\mathrm{F}_k(\mathbf{x},\mathbf{W}) - b\right)^2 P(\Phi_{\overline{k}} \mid \mathbf{x})p(\mathbf{x})dx\right]$$

en développant le carré et en groupant les termes nous obtenons :

$$\lim_{m\to\infty} \frac{C(m,\mathbf{W})}{m} = \sum_{k=1}^{p}\left[\int \mathrm{F}_k(\mathbf{x},\mathbf{W})^2\left[P(\Phi_k \mid \mathbf{x}) + P(\Phi_{\overline{k}} \mid \mathbf{x})\right]p(\mathbf{x})dx - 2\int \mathrm{F}_k(\mathbf{x},\mathbf{W})\left[aP(\Phi_k \mid \mathbf{x}) + bP(\Phi_{\overline{k}} \mid \mathbf{x})\right]p(\mathbf{x})dx + \int\left[a^2P(\Phi_k \mid \mathbf{x}) + b^2P(\Phi_{\overline{k}} \mid \mathbf{x})\right]p(\mathbf{x})dx\right]$$

L'expression $P(\Phi_k \mid \mathbf{x}) + P(\Phi_{\overline{k}} \mid \mathbf{x})$ est égale à 1. Si nous remplaçons alors $aP(\Phi_k \mid \mathbf{x}) + bP(\Phi_{\overline{k}} \mid \mathbf{x})$ par $h_k(\mathbf{x})$ et si nous complétons le carré par rapport à $h_k(\mathbf{x})$, nous obtenons enfin :

$$\lim_{m\to\infty} \frac{C(m,\mathbf{W})}{m} = \sum_{k=1}^{p}\left[\int\left[\mathrm{F}_k(\mathbf{x},\mathbf{W}) - h_k(\mathbf{x})\right]^2 p(\mathbf{x})dx - \int h_k(\mathbf{x})^2 p(\mathbf{x})dx + \int\left[a^2P(\Phi_k \mid \mathbf{x}) + b^2P(\Phi_{\overline{k}} \mid \mathbf{x})\right]p(\mathbf{x})dx\right] \qquad (4.8)$$

Le deuxième et le troisième terme sont indépendants de $\mathbf{W}$. Ainsi, pour un nombre d'exemples m suffisamment grand, minimiser $C(m,\mathbf{W})$ revient à minimiser $e^2(\mathbf{F}) = \sum_{k=1}^{p}\int[\mathrm{F}_k(\mathbf{x},\mathbf{W}) - h_k(\mathbf{x})]^2 p(\mathbf{x})dx$ qui est l'erreur quadratique moyenne entre les sorties du réseau et $h_k(\mathbf{x})$.

Examinons le terme $h_k(\mathbf{x}) = aP(\Phi_k \mid \mathbf{x}) + bP(\Phi_{\overline{k}} \mid \mathbf{x})$ défini précédemment pour un codage général $\mathbf{S} = \{a, b\}$. Dans le cas particulier d'un codage $\mathbf{S} = \{1, -1\}$, $h_k(\mathbf{x}) = P(\Phi_k \mid \mathbf{x}) - P(\Phi_{\overline{k}} \mid \mathbf{x})$ représente la fonction discriminante de Bayes $g_k(\mathbf{x})$.

De même, dans le cas où le codage général est $\mathbf{S} = \{1, 0\}$, $h_k(\mathbf{x}) = P(\Phi_k \mid \mathbf{x})$ représente la probabilité d'appartenance à la classe Φ_k pour une forme donnée $\mathbf{x}$. En conséquence, avec un codage $\mathbf{S} = \{1, -1\}$, lors de la minimisation de C(m,**W**) les réseaux multicouches approximent la fonction discriminante de Bayes $g_k(\mathbf{x})$.

4.3. Utilisation d'un MLP en tant que classifieur afin de modéliser les fonctions de transfert complexes

Souvent, nous disposons des données représentant des mesures d'un phénomène complexe dont nous ne connaissons pas d'une façon précise la loi. De même, nous espérons disposer d'une méthode permettant de modéliser le phénomène à partir de ces données. Or, nous avons vu que les MLP forment un outil puissant pour modéliser des fonctions univoques ($\mathbf{x} \rightarrow T(\mathbf{x})$).

Nous présentons dans ce paragraphe une méthode utilisant les MLP qui permet de déduire des informations supplémentaires sur la complexité de la fonction de transfert étudiée. L'interprétation des sorties du réseau permet de proposer plusieurs valeurs à T(**x**) avec des coefficients de vraisemblances associés, les valeurs les plus probables ayant les plus grands coefficients. Ceci est important quand la fonction de transfert étudiée est multivoque, les mesures associées fortement bruitées et admettant des ambiguïtés intrinsèques.

Supposons que **T** prenne ses valeurs dans l'intervalle $[a, b]$, nous discrétisons cet intervalle en p intervalles égaux : $I_i = \left[a + \frac{b-a}{p}(i-1), a + \frac{b-a}{p} i\right]$, $1 \le i \le p$; **x** sera dite de classe i si $T(\mathbf{x})$ I_i. Posé de cette manière, nous nous ramenons à un problème de classification. Nous identifions par la suite chaque intervalle I_i par son "milieu" $m_i = a + \frac{b-a}{p}\left(i - \frac{1}{2}\right)$. Nous notons par $p(I_i \mid \mathbf{x})$ la probabilité qu'étant donné **x**, alors $T(\mathbf{x})$ I_i, et par $g_i(\mathbf{x}) = p(I_i \mid \mathbf{x}) - \sum_{j \neq i} p(I_i \mid \mathbf{x})$ les fonctions discriminantes de Bayes définies au paragraphe précédent. Nous utilisons un MLP pouvant recevoir les données **x** en entrée et comportant p cellules de sortie, la i-ème cellule de sortie représentant l'intervalle I_i. Pour un vecteur d'entrée **x** la réponse désirée $\mathbf{y} = (y_1, y_2, \ldots, y_p)$ est définie par $y_i = +1$ si $T(\mathbf{x})$ appartient à I_i. et $y_i = -1$ sinon.

Pour une matrice de poids **W** donnée, le MLP génère la fonction $\mathbf{F}(\cdot,\mathbf{W}) : \mathbb{R}^n \rightarrow \mathbb{R}^p$. Pour une entrée **x**, le réseau calcule en sortie un vecteur de dimension p, $\left(F_1(\mathbf{x},\mathbf{W}), \ldots, F_p(\mathbf{x},\mathbf{W})\right)$. Le processus d'apprentissage consiste à minimiser la fonction coût :

$$C(m,\mathbf{W}) = \sum_{k=1}^{m} \|\mathbf{F}(\mathbf{x}_k,\mathbf{W}) - \mathbf{y}\|^2 = \sum_{k=1}^{m} \sum_{i=1}^{p} \|F_i(\mathbf{x}_k,\mathbf{W}) - y_i\|^2$$

Après apprentissage, et à chaque donnée **x** présentée, le réseau calcule p états de sorties $F_i(\mathbf{x},\mathbf{W}^*)$, $(1 \leq i \leq n)$, le i-ème état étant une approximation, à l'intérieur de la famille déterminée par l'architecture, de la fonction discriminante de Bayes $g_i(\mathbf{x})$ (§4.2). Le réseau propose pour chaque donnée **x**, p valeurs possibles m_i (milieu des I_i) avec les états $F_i(\mathbf{x},\mathbf{W}^*)$ comme coefficients de vraisemblances attachés à chacun d'eux.

Nous donnons dans le chapitre 10 une application utilisant cette méthode. Cette application nous permet de mettre en évidence le fait que la première fonction traitée est univoque alors que la seconde est multivoque.

A chaque exemple **x** de l'ensemble d'apprentissage, nous pouvons associer une courbe des états des cellules de sorties représentant, en abscisses i $(1 \leq i \leq n)$ et en ordonnée $F_i(\mathbf{x},\mathbf{W}^*)$.

Un pic de la courbe des états de sorties, peut être caractérisé par le maximum local i et les deux états $(i - 1)$ et $(i + 1)$ qui l'entourent. L'intensité d'un pic étant définie par $F_i(\mathbf{x},\mathbf{W}^*)$, nous pouvons lui associer comme valeur numérique la moyenne pondérée :

$$y_i = \frac{\sum_{k=i-1}^{i+1} F_k(\mathbf{x},\mathbf{W}^*)\ m_k}{\sum_{k=i-1}^{i+1} F_k(\mathbf{x},\mathbf{W}^*)} \qquad (4.9)$$

Ainsi, en fonction du nombre des pics et de leur intensité, le réseau propose, pour chaque **x**, des valeurs y_{i_k} $(k = 1, \ldots, np)$ correspondant aux np premiers pics choisis. Ces pics sont ordonnés par ordre décroissant des probabilités ***a priori***, $\left(p(I_{i_1} \mid \mathbf{x}) \geq \ldots \geq p(I_{i_{np}} \mid \mathbf{x})\right)$. La valeur y_i proposée par la formule (4.9) pour le pic de centre i est une valeur numérique de l'intervalle I_i, plus précise que m_i, car elle tient aussi compte des probabilités ***a priori*** des

classes voisines. **T**, qui est une fonction réelle, est donc approximée par des valeur réelles.

La visualisation des différentes courbes des états des cellules de sorties, donne une idée sur la complexité de la fonction **T**. Ainsi, une courbe des états avec un seul pic indique une fonction de nature univoque alors qu'une courbe des états de sorties présentant au moins deux pics indique une fonction multivoque et complexe (Figure 4.2).

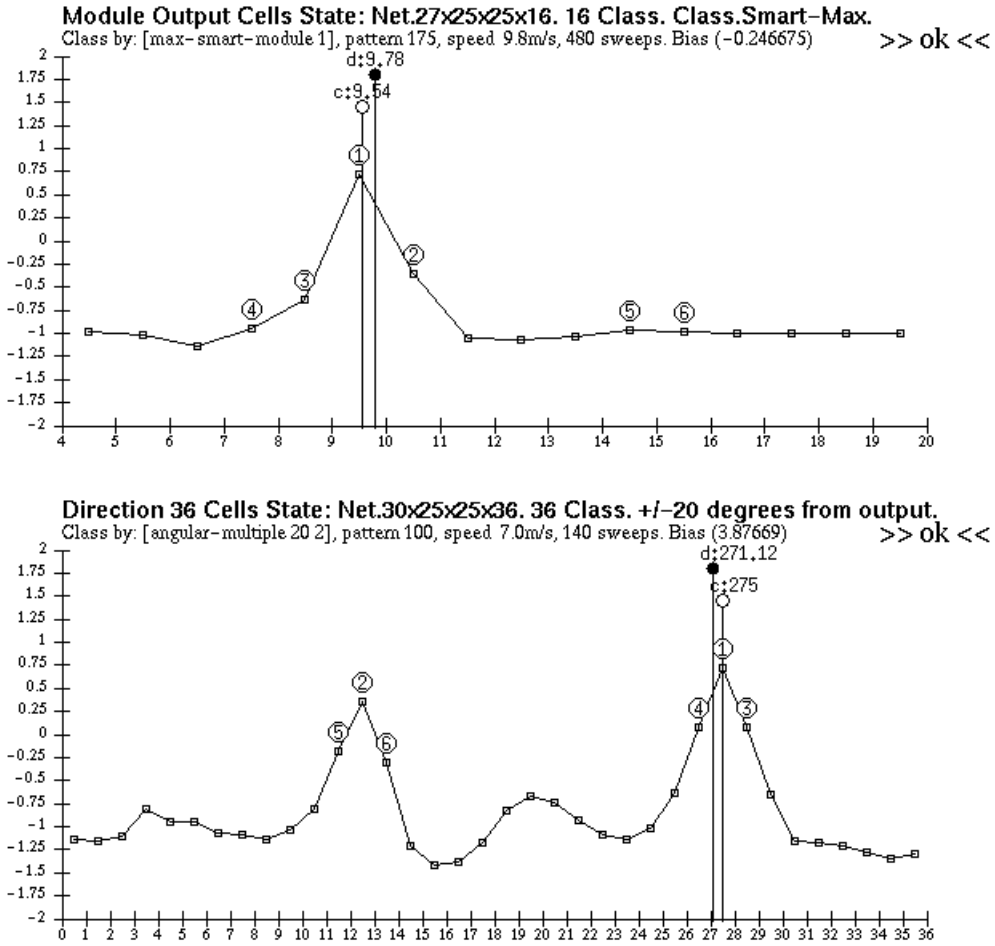


Figure 4.2 : *représentation de l'activation des sorties du réseau dans le cas de deux problèmes différents. Pour le premier problème, le calcul de la vitesse du vent, un seul pic est présent dans la solution proposée par le réseau. En revanche, dans le cas du calcul de la direction, deux pics (et parfois plus) son proposés comme solution par le réseau. Dans la figure c indique "réponse calculée" et d "réponse désirée".*

Par la suite, cette méthode d'approximation d'une fonction de transfert, sera appelée méthode d'approximation par classification. Une telle méthode amène à utiliser des classifieurs qui utilisent un nombre *p* très grand de

classes : *p* représentant le nombre d'intervalles considérés, il est donc lié à la précision de l'approximation cherchée.

4.4. Pondération de la métrique

Nous avons mis au point une amélioration de la fonction de coût

$$C(m,\mathbf{W}) = \sum_{k=1}^{m} \|\mathbf{F}(\mathbf{x}_k,\mathbf{W}) - \mathbf{y}\|^2$$

facilitant la mise au point de tels classifieurs. La présente section illustre les fondements théoriques d'une telle amélioration.

Les réseaux permettent, sous certaines conditions, de calculer les probabilités ***a priori*** $P(\Phi_j \mid \mathbf{x})$. Ces probabilités permettent alors de déterminer les fonctions discriminantes de Bayes $g_i(\mathbf{x}) = P(\Phi_i \mid \mathbf{x}) - \sum_{j\neq i} P(\Phi_j \mid \mathbf{x})$. Or, d'une manière générale, le problème fondamental est de déterminer des fonctions discriminantes $h_i(\mathbf{x})$ vérifiant la propriété :

$$\mathbf{x}, \quad h_i(\mathbf{x}) = \mathrm{SUP}_j h_j(\mathbf{x}) \quad \Leftrightarrow \quad g_i(\mathbf{x}) = \mathrm{SUP}_j g_j(\mathbf{x})$$

Ces fonctions définissent alors les mêmes surfaces séparatrices que celles définies par les fonctions discriminantes de Bayes ($g_i(\mathbf{x}) = g_j(\mathbf{x})$).

Souvent, nous chercherons des fonctions $h_i(\mathbf{x})$ ayant des valeurs numériques plus discriminantes que les $g_i(\mathbf{x})$ à l'intérieur de leur classes respectives, et suffisamment proche des $g_i(\mathbf{x})$ autour des frontières séparatrices.

Si nous choisissons un codage en $\mathbf{S} = \{1, -1\}$, nous prenons :

$\mathbf{y}_k = (-1, -1, \ldots, 1, -1 \ldots, -1)$ *avec* $y_i^k = 1$ *si* $\mathbf{x}_k$ *appartient à la classe i.*

Nous considérons alors la fonction coût :

$$Q_j(m,\mathbf{W}) = \alpha \sum_{\mathbf{x}\,\Phi_j} \left[\mathrm{F}_j(\mathbf{x},\mathbf{W}) - 1\right]^2 + \sum_{\mathbf{x}\,\Phi_j} \left[\mathrm{F}_j(\mathbf{x},\mathbf{W}) + 1\right]^2 \qquad (4.10)$$

où α est un coefficient de pondération ($\alpha > 1$), et

$$Q(m,\mathbf{W}) = \sum_{j=1}^{p} Q_j(m,\mathbf{W}) \qquad (4.11)$$

Nous avons alors,

$$Q(m,\mathbf{W}) = C(m,\mathbf{W}) + (\alpha - 1) \sum_{j=1}^{p} \left[\sum_{\mathbf{x} \in \Phi_j} \left[F_j(\mathbf{x},\mathbf{W}) - 1 \right]^2 \right] \qquad (4.12)$$

Quand $m \rightarrow \infty$ l'expression $\frac{Q(m,\mathbf{W})}{m}$ tend vers :

$$\lim_{m \rightarrow \infty} \frac{Q(m,\mathbf{W})}{m} = \sum_{j=1}^{p} \int \left[F_j(\mathbf{x},\mathbf{W}) - g_j(\mathbf{x}) \right]^2 p(\mathbf{x}) dx + \textit{constante} +$$

$$(\alpha - 1) \sum_{j=1}^{p} \int \left[F_j(\mathbf{x},\mathbf{W}) - 1 \right]^2 P(\Phi_j \mid \mathbf{x}) p(\mathbf{x}) dx \qquad (4.13)$$

En analysant de près le dernier terme nous remarquons que :

- l'importance de ce terme croît avec la valeur de α.
- si une partie de la classe j est représentée par une région de centroïde c, alors plus $\mathbf{x}$ est proche de c, plus la probabilité $P(\Phi_j \mid \mathbf{x})$ est grande et plus le troisième terme a tendance à "tirer" $F_j(\mathbf{x},\mathbf{W})$ vers 1. D'autre part, plus $\mathbf{x}$ s'éloigne de c, plus la probabilité $P(\Phi_j \mid \mathbf{x})$ est petite et plus le premier terme a tendance à tirer $F_j(\mathbf{x},\mathbf{W})$ vers $g_j(\mathbf{x})$.

Ainsi, pour un nombre suffisamment grand d'exemples (m) et pour un α convenablement choisi, la fonction de coût précédente permet de générer des fonctions $F_j(\mathbf{x},\mathbf{W})$ qui ont la propriété suivante :

- chaque $F_j(\mathbf{x},\mathbf{W})$ prend des valeurs numériques, à l'intérieur de sa classe j, supérieures à celles prises par la fonction discriminante de Bayes et prend des valeurs proches de celle-ci autour de la surface séparatrice et à l'extérieur de la classe j.

Nous donnons par la suite les performances obtenues sur un même problème avec une même architecture en utilisant d'une part le critère de coût classique $C(m,\mathbf{W})$ et d'autre part le critère modifié $Q(m,\mathbf{W})$. Il s'agit d'un

classifieur avec 36 classes, donc 36 cellules dans la couche de sortie, deux couches cachées, chacune de 25 cellules et 30 cellules en entrée – ce que fait en tout 2361 paramètres à ajuster (2275 poids de connexions + 86 seuils) –. L'entrée est globalement connectée à la première couche cachée. De même entre les deux couches cachées ainsi qu'entre la deuxième couche cachée et la sortie. L'ensemble de données utilisés dans l'apprentissage est un ensemble équilibré composé de 172 patterns par classe – soit un total de 6192 patterns– .

La Table 4.1 illustre les avantages d'une modification de la pondération de la métrique de sortie dans le cas où le nombre de classes est important. Ces essais ainsi que d'autres que nous avons effectué pour un nombre différent de classes (16 ou 11) nous laissent conclure que l'utilisation de la métrique modifié $Q(m,\mathbf{W})$ augmente la vitesse de convergence. Nous avons remarqué expérimentalement qu'en laissant entraîner longuement un réseau avec la fonction de coût classique $C(m,\mathbf{W})$ nous arrivons, dans la plupart de cas, à des performances très comparables à celles avec la fonction modifié $Q(m,\mathbf{W})$, mais jamais significativement supérieures. Dans ce dernier cas, cependant, nous avons trouvé profitable de continuer à utiliser $Q(m,\mathbf{W})$ d'abord parce qu'elle accélère le processus d'apprentissage, et ensuite parce que les sorties du réseau sont beaucoup plus discriminantes que dans le cas classique. Si nous sommes intéressés aux sorties du réseaux en tant que coefficients de vraisemblance, la fonction de coût $Q(m,\mathbf{W})$ utilisé pendant l'apprentissage nous assure une beaucoup plus importante activité dans les cellules de sortie qui indiquent la classe la plus probable.

Comparaison des performances : apprentissage avec un critère de coût classique $C(m,\mathbf{W})$ contre critère modifié $Q(m,\mathbf{W})$			
36	**classes**	**critère classique $C(m,\mathbf{W})$**	**critère modifié $Q(m,\mathbf{W})$**
1 pic	**app.**	67,4 %	75,3 %
	test	67,8 %	76,0 %
2 pics	**app.**	98,5 %	99,4 %
	test	98,8 %	99,4 %

Table 4.1 : *Pondération de la métrique. Comparaison des résultats obtenus en entraînant un réseau sans pondération de la métrique de sortie et un autre avec pondération. Les performances sont supérieures dans le cas du réseau entraîné avec pondération.*

La Figure 4.3 nous montre les états de la couche cachée pour deux réseaux différents, lors de la présentation du même pattern en entrée. Il s'agit a nouveau du classifieur avec 36 classes mentionné ci-dessus. Un réseau a été entraîné en utilisant la fonction de coût classique $C(m,\mathbf{W})$, figure supérieure, et l'autre en utilisant la fonction modifiée $Q(m,\mathbf{W})$), figure inférieure.

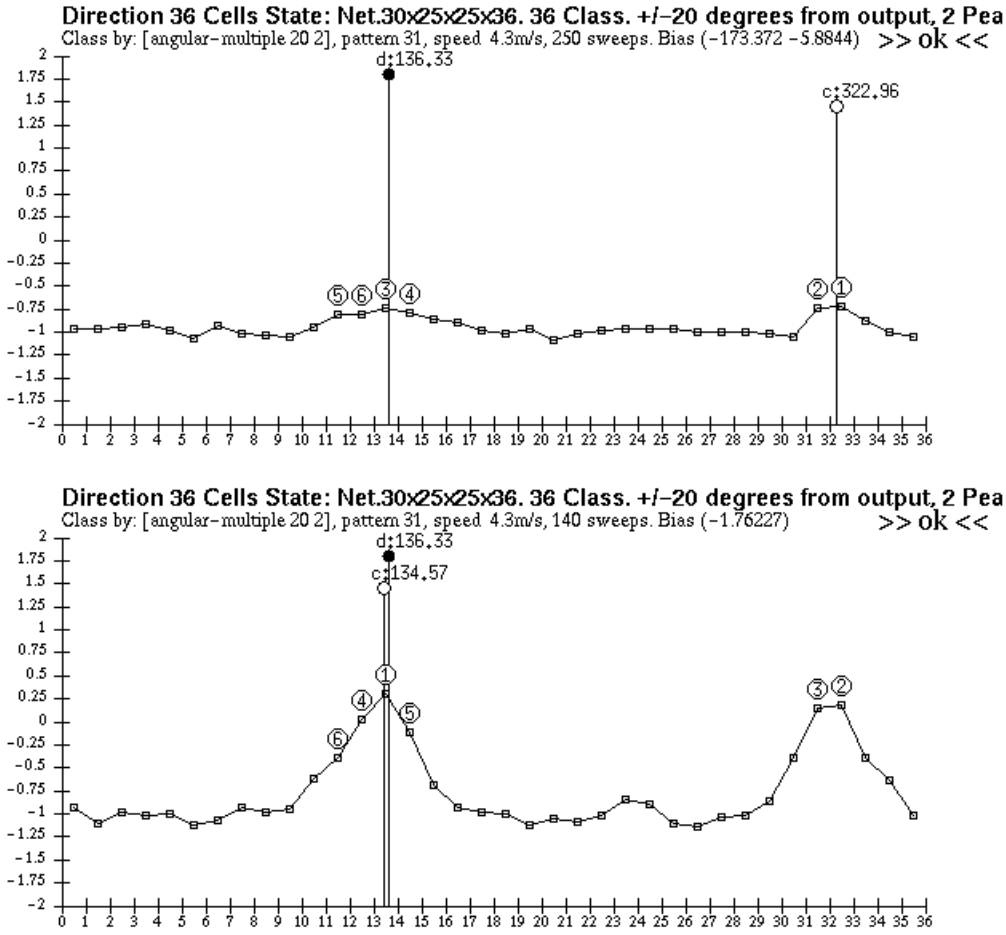


Figure 4.3 : *états des cellules de sortie lors des apprentissages sans et avec modification de la métrique. Le pouvoir séparateur du coefficient de vraisemblance $F_j(\boldsymbol{x},\boldsymbol{W})$ est bien meilleur dans le cas où l'on utilise la nouvelle métrique (figure inférieure).*

L'amélioration apportée par l'utilisation de cette métrique est très clairement illustrée par la Figure 4.3 sur laquelle nous voyons les réponses apportées par les deux réseaux à une même forme. Les états des cellules 14 (direction du vent entre 130° et 140°) et 33 (direction entre 320° et 330°) qui sont les plus probables, sont largement augmentés. Le pouvoir séparateur du coefficient de vraisemblance $F_j(\mathbf{x},\mathbf{W})$ est bien meilleur.

5. Simulateurs de réseaux neuronaux

5.1. La simulation des modèles connexionnistes

Bien qu'il s'agisse d'un domaine de recherche très développé, les modèles connexionnistes sont encore loin d'être implémentés sur des machines possédant de véritables architectures neuronales. La grande variété de modèles existants et leurs multiples différences conceptuelles, d'implémentation et de dynamique font de la construction physique de réseaux neuronaux tout un sujet de recherche de grande complexité.

La simulation de modèles connexionnistes sur des ordinateurs conventionnels reste de loin l'outil le plus répandu dans la recherche et l'exploitation des réseaux de neurones. Les ***simulateurs*** étaient initialement des programmes permettant l'exploitation d'un unique formalisme. Des nouvelles tendances, notamment le développement de systèmes modulaires qui utilisent plusieurs modèles neuronaux différents, posent des problèmes aux usagers des simulateurs.

Actuellement, des systèmes hybrides mettant en concours diverses techniques sont de plus en plus utilisés dans le domaine. La génération actuelle de simulateurs doit par conséquent être suffisamment souple afin de s'adapter aux besoins des chercheurs sans les obliger à se plonger dans des détails de programmation.

Deux approches de simulation sont présentées par la suite, les deux étant utilisées pour développer nos recherches. Tour d'abord, le simulateur ***SN2*** : un puissant outil permettant une programmation aisée en langage de haut niveau. En suite, la bibliothèque de fonctions ***GALATEA*** qui fournit des environnements de simulation adaptés aux différents modèles et une panoplie de fonctions en langage C permettant, avec une relative aisance, la programmation de machines hybrides complexes (multi-réseaux et multi-formalisme).

5.2. Le simulateur SN

Le simulateur ***SN*** [Bottou et le Cun 88] est un puissant outil conçu pour des réseaux du type perceptron multicouches, entraînés par des algorithmes de la famille de la rétro-propagation du gradient –GBP. Le noyau du simulateur est programmé en langage C. L'interface utilisateur est un interpréteur Lisp, et les fonctions de haut niveau sont écrites dans ce langage, faisant appel à celles de bas niveau en langage C. Cette caractéristique permet aux usagers d'utiliser les fonctions proposées par défaut ou bien de les modifier selon les besoins de l'application à traiter. Ses capacités graphiques permettent de surveiller visuellement les performances, les taux d'erreurs ainsi que d'examiner les états des neurones et les poids des connexions.

Il existe une version qui intègre des techniques de second ordre pour la minimisation de l'erreur du gradient dans GBP –Méthode de Newton associé à l'algorithme de Levemberg-Marquardt–. Cette version permet un "pilotage automatique" lors de l'entraînement ou apprentissage. En effet, l'usager est libéré de la tâche de surveillance des performances et de diminution progressive du pas d'apprentissage, ceci étant fait de façon automatique par l'algorithme au fur et à mesure que l'entraînement du réseau progresse.

La possibilité de modifier les procédures Lisp fournies par défaut sur SN permet aussi de programmer sur le simulateur d'autres algorithmes, différents de la rétro-propagation du gradient, en utilisant tout de même des fonctions ainsi que des structures de données utilisées par ce dernier.

Pendant le déroulement de ma thèse j'ai suivi l'évolution des différentes versions de ***SN***. La dernière version, ***SN2.5***, introduit une conception orientée objet des éléments constitutifs du simulateur. De plus, elle intègre une interface graphique, absente dans les versions précédentes, permettant la spécification de l'architecture du réseau. Elle permet également de spécifier le type de connexions entre couches –global, local, poids partagés– et, d'une façon générale, les différents paramètres intervenant dans une simulation.

5.3. La bibliothèque Galatea

La bibliothèque ***Galatea*** [Mejía et al. 90] a été développée dans le cadre du projet ESPRIT-Pygmalion. L'objectif recherché pour cette bibliothèque est de

fournir aux utilisateurs des bibliothèques de fonctions en langage C pouvant être appelées par un programme C implémentant des algorithmes de réseaux de neurones. Au cours du développement de la bibliothèque une description unifiée des algorithmes a été retenue afin de faciliter la conception des modules présentés et les développement ultérieurs.

Un réseau décrit dans ***Galatea*** est une boîte noire, ayant un vecteur d'entrée et un vecteur de sortie. Un certain nombre de fonctions permettent d'avoir accès aux variables internes du réseau qui sont organisées en vecteurs. L'utilisateur peut définir un réseau sous forme d'une variable qu'il peut initialiser avec une topologie spécifique. Il peut avoir accès au vecteur d'entrée, de sortie, aux états et aux poids internes, et poser et consulter les différents paramètres. Il peut aussi faire activer et entraîner le réseau pour un vecteur d'entrée donné.

Le problème de base dans la conception de ***Galatea*** était la définition des structures des données. En effet, des structures de données complexes peuvent être très générales dans certains cas et restrictives dans d'autres (surtout pour les développements futurs). Ce qui est gagné, en niveau d'abstraction, dans un cas, est perdu par manque de souplesse pour un autre. Ainsi, le choix a été d'utiliser des structures de données de bas niveau : les vecteurs de nombres réels. Ce choix a un inconvénient : le calcul sur les réseaux de neurones est souvent mal adapté aux structures de vecteurs, sauf dans des cas particuliers (par exemple : les réseaux multicouches entièrement connectés). Mais l'inconvénient précédent est compensé par le fait que dans beaucoup d'applications les données se présentent comme des suites de vecteurs.

Des procédures adaptées pour le calcul sur les réseaux des neurones sont fournies. Ces procédures sont basées sur l'idée de "règles de calcul" (***recomputation rules***). Une règle de calcul est formée principalement par une fonction et une suite de pointeurs sur des vecteurs de réels. Quand une règle est exécutée, sa liste de pointeurs est passée à sa fonction qui exécute un calcul bien déterminé.

Les régles de calculs peuvent être créées, initialisées, exécutées et libérées Elles décrivent principalement le calcul qui doit être exécuté sur un réseau donné et non la structure du réseau.

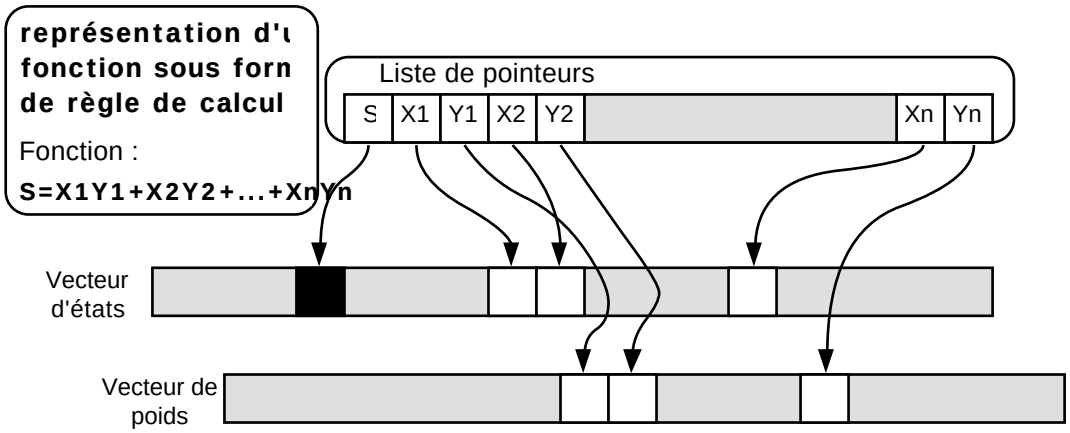


Figure 5.1 : *Exemple d'une règle de calcul. Cette règle réalise le produit scalaire entre les vecteurs* $\{x_i\}$*–les états– et* $\{y_i\}$*–les poids– où* $i=1, \ldots, n$*, et stocke le résultat dans* s.

Organisation de Galatea

Le noyau de base de ***Galatea*** comporte plusieurs modules de bibliothèques de fonctions : deux module d'outils et plusieurs modules dédiés à des algorithmes spécifiques. En plus de ce noyau, ***Galatea*** comporte un environnement qui permet de construire des programmes interactifs pouvant exécuter des commandes par lignes ou en chargeant des fichiers de commandes. Enfin, ***Galatea*** comporte des programmes d'évaluation des modules spécifiques aux algorithmes (voir Figure 5.2).

1) Les deux ***Modules d'outils***. Ils contiennent :

- Les définitions des structures de données des matrices de dimension deux et de dimension variable, et des règles de calcul.
- Les fonctions qui gèrent les structures de données précédentes, initialisation, destruction, sauvegarde et lecture d'un fichier et toutes les opérations de base susceptibles d'être utilisées dans les algorithmes des réseaux de neurones.
- Les structures de données et les fonctions dédiées à l'interpolation des fonctions non linéaires par des "splines" cubiques.
- Les fonctions de traitement des erreurs et de gestion de la mémoire.

- Des structures de données de bases permettant de définir les connexions dans des réseaux à couches et ayant différents clusters par couche.

- Des fonctions standard de calcul de distances, de classification et de performance.

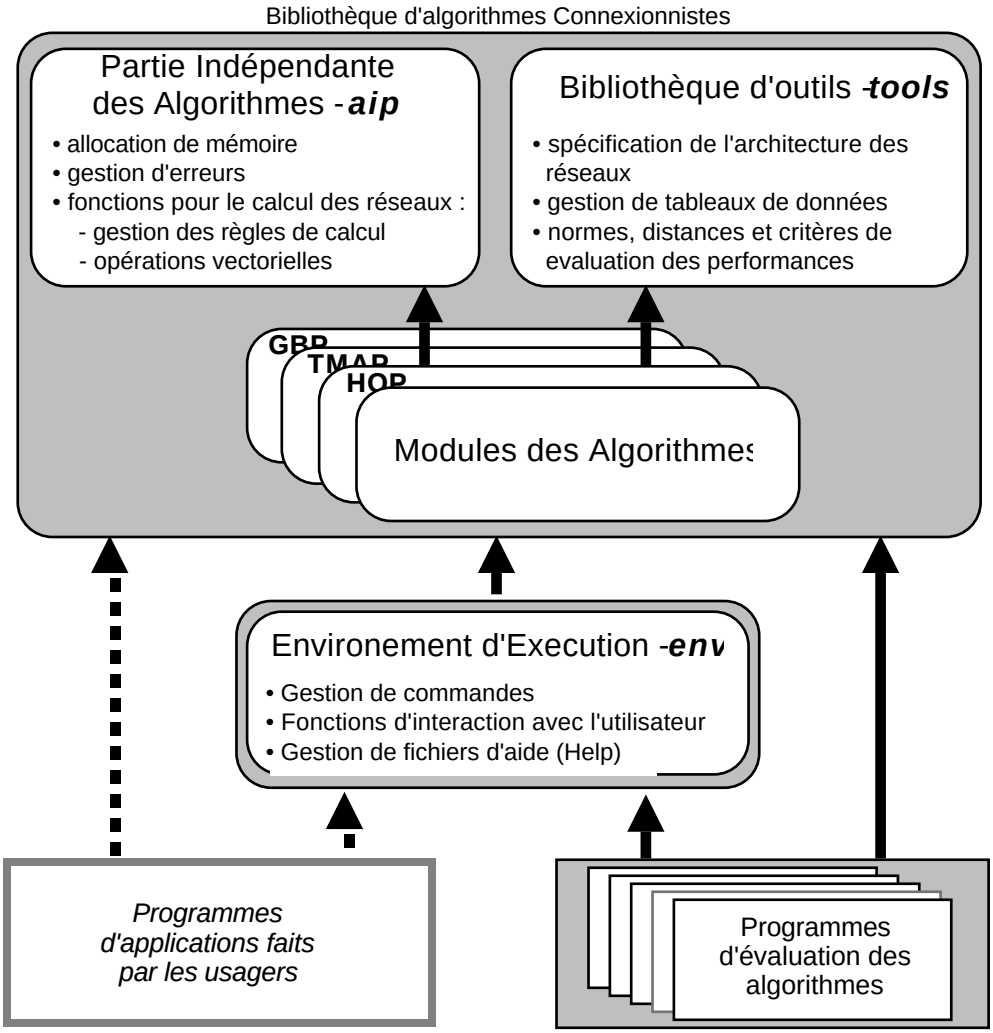


Figure 5.2 : *organisation des composants de GALATEA. Les composants de GALATEA sont cinq dont les principaux sont la partie indépendante des algorithmes –AIP–, la bibliothèque des outils–TOOLS– et les modules des algorithmes. Ce dernier regroupe touts les algorithmes connexionnistes programmés (voir Table 5.1). La bibliothèque de l'environnement standard –ENV– fournit aux usagers des programmes d'évaluation un moyen simple mais efficace d'accéder aux différents fonctionnalités des algorithmes programmés.*

2) ***Les modules des algorithmes*** : Ils regroupent, à l'heure actuelle, tous les algorithmes connexionnistes énumérés dans la Table 5.1.

Chaque module est dédié à un modèle et un algorithme particulier des réseaux de neurones. Il contient la structure de données permettant de définir le type de réseau donné, et des groupes de fonctions associées. Le réseau est défini par une variable, un groupe de fonctions permettent d'avoir accès à tous les états, les poids et les paramètres internes, et implémentent les opérations d'initialisation et de destruction du réseau. D'autres groupes de fonctions représentent les opérations de base et permettent d'implémenter toutes les variantes connues de cet algorithme. Ainsi, le module dédié à l'algorithme de la rétro-propagation du gradient dispose de procédures de bas niveau pouvant être utilisées pour implémenter différentes variantes tels les masques à poids partagés.

La Table 5.1 représente la liste des modules disponibles actuellement :

algorithme	abréviation
Hopfield Nets	Hop
Gradient Back Propagation	GBP
Gradient Back Propagation with Feedback	GBPF
Kanerva associative memory	Kan
Linear Associative Memory (Kohonen)	LAM
Adaptive Resonance Theory 1 (Grossberg)	ART1
Boltzmann Machines	BM
Topological Maps (Kohonen)	TMap
Linear Vector Quantization (Kohonen)	LVQ
Bidirectional Associative Memory	BAM
Competitive Learning	CMPL

Table 5.1 : *algorithmes connexionnistes dans GALATEA.*

3) ***Le module de l'environnement standard*** : Il s'agit d'un programme assez général comportant un environnement du type interactif opérant par ligne de commandes. Des fonctionnalités pour gérer des fichiers de commandes, des fichiers d'aide (***help***), l'exécution des commandes du système d'exploitation, ainsi qu'une fonction de ***statut*** sont comprises de façon standard. L'addition de fonctionnalités est une tâche facile : il suffit d'incorporer les fonctions désirées et de définir les noms des commandes associées à ces fonctionnalités. Cet environnement fournit aussi des fonctions permettant la saisie interactive de données de types différents, ce qui permet la capture des paramètres nécessaires lors de l'appel de certaines fonctions.

4) ***Les programmes d'évaluation des algorithmes*** : Les programmes d'évaluation des algorithmes ont été écrits en utilisant le module précédent. Ces programmes sont écrits afin de tester et valider le fonctionnement des différentes fonctions des modules des algorithmes de la Table 5.1. Ils servent aussi comme exemples de l'utilisation et la fonctionnalité des différentes fonctions de ces modules. Ces programmes peuvent être utilisés pour traiter des applications réelles.

Limitations

La bibliothèque ***Galatea*** n'a pas été à l'origine conçue pour devenir un produit complet. Bien qu'on puisse développer des applications avec les environnements de test fournis avec la bibliothèque, l'exploitation véritable des algorithmes et des caractéristiques modulaires de ***Galatea*** ne peut se faire qu'avec un peu de programmation de la part de l'utilisateur.

Une limitation de conception est liée à l'utilisation de vecteurs comme structure de donnée. Les modèles nécessitant l'allocation dynamique de mémoire (addition de nouveaux neurones, nouvelles connexions) peuvent se voir limités par ce choix. C'est une limitation acceptable, car dans la plupart des cas on peut estimer à l'avance la taille maximale prévue lors du fonctionnement du programme. En revanche, les arrangements vectoriels des données accélèrent considérablement les tâches parfois si lourdes en calcul telles l'entraînement des réseaux et les tests sur de bases de données de grande taille.

Conclusion

Les travaux de conception et d'implémentation de la bibliothèque ***Galatea*** nous ont permis de mieux comprendre le mécanisme d'apprentissage de plusieurs algorithmes peu connus.

Le travail a été un véritable apport d'équipe. Initialement, lors de la conception j'ai eu l'occasion de travailler de près avec Leon Bottou, que je remercie de l'aide qu'il m'a apporté ainsi que de ses innombrables bonnes idées qui sont maintenant au cœur de ***Galatea***.

Quelques modules ont été programmés par d'autres partenaires du projet, mais la plupart ont été écrits par différents membres de notre laboratoire. Le travail final est un produit de près de 40 000 lignes écrites en langage C. Mon

travail a été de superviser le développement de ***Galatea***, d'intégrer de façon homogène tous les modules écrits et de produire la documentation finale [Mejía 89]. ***Galatea*** a eu initialement un statut de libre circulation et elle a été distribuée à de nombreux laboratoires situés un peu partout dans le monde.