IDIAP RESEARCH REPORT

IDIAP

Martigny - Valais - Suisse

Reconnaissance de caractères manuscrits à l'aide de réseaux neuromimétiques

Jean-Luc Beuchat

IDIAP–RR 97-18

February 97

Dalle Molle Institute for Perceptive Artificial Intelligence • P.O.Box 592 • Martigny • Valais • Switzerland

phone +41 – 27 – 721 77 11
fax +41 – 27 – 721 77 12
e-mail secretariat@idiap.ch
internet http://www.idiap.ch

IDIAP Research Report 97-18

Reconnaissance de caractères manuscrits à l'aide de réseaux neuromimétiques

Jean-Luc Beuchat

February 97

"En icelui fut ladite généalogie trouvée, écrite tout au long de lettres cancelleresques, non sur papier, non sur parchemin, non sur cire, mais sur escorse d'ulmeau, tant toutefois usée par vétusté qu'à peine en pouvait-on reconnaître trois de ranc.
Je (combien qu'indigne) y fus appelé, et, à grand renfort de besicles, pratiquant l'art dont on peut lire lettres non apparentes, comme enseigne Aristote, la translatai (...)."
Rabelais, "Gargantua"

Chapitre 1

Prolégomènes

> "Mais alors, excellent Glaucon, quelle sera cette étude ? Car les arts nous sont tous apparus comme mécaniques ...
> Sans doute. Mais quelle autre étude reste-t-il si nous écartons la musique, la gymnastique et les arts ?
> Eh bien ! répondis-je, si nous ne trouvons rien à prendre hors de là, prenons quelqu'une de ces études qui s'étendent à tout.
> Laquelle ?
> Par exemple cette étude commune, qui sert à tous les arts, à toutes les opérations de l'esprit et à toutes les sciences, et qui est une des premières auxquelles tout homme doit s'appliquer.
> Laquelle ? demanda-t-il.
> Cette étude vulgaire qui apprend à distinguer un, deux et trois (...)."
> Platon, "La République"

1.1 Objectifs du projet

Nous communiquons actuellement avec nos ordinateurs à l'aide de claviers et de souris. Le progrès est considérable depuis l'époque, heureusement révolue, des cartes perforées. Cependant, la parole et l'écriture manuscrite constituent nos moyens naturels de communication. La réalisation d'interfaces vocales ou traitant l'écriture manuscrite constitue un formidable défi monopolisant des centaines de chercheurs. Les concepteurs de tels systèmes (appelés *online systems* dans la littérature anglaise) disposent d'une information temporelle exploitable lors de la reconnaissance.

Etudions brièvement l'exemple de l'interface reconnaissant l'écriture. Elle est généralement composée d'une tablette et d'un stylo. Afin de repérer les mots ou caractères, le système détecte les instants durant lesquels le crayon n'est plus en contact avec la tablette.

La reconnaissance d'écriture connaît cependant d'autres applications. Certaines institutions disposent d'une importante quantité de documents qu'elles souhaitent transcrire sous forme électronique, afin de constituer des bases de données aisément consultables. Les systèmes effectuant ce travail sont appelés *offline* dans la littérature anglaise. Ne possédant plus d'information temporelle, une des grandes difficultés consiste à segmenter le texte en mots ou caractères isolés qu'il s'agit ensuite d'identifier. La complexité de ces différentes tâches est intimement liée au type d'écriture analysée. Un texte dactylographié possède ainsi des propriétés habituellement étrangères à un document manuscrit : existence de fontes standard, régularité, ...

Nous ne nous intéresserons ici qu'aux systèmes *offline* de reconnaissance d'écriture manuscrite. Supposons que nous souhaitions transcrire un document ancien sous forme électronique (figure 1.1). En le numérisant au moyen d'un scanner, nous obtenons une image représentant le texte, ainsi que les éventuelles enluminures et miniatures. Un algorithme de segmentation permet d'isoler le texte, puis de le découper en caractères qui seront reconnus indépendamment les uns des autres.

Divers algorithmes de prétraitement restreignent les multiples variations de l'écriture manuscrite, facilitant ainsi sa reconnaissance. L'étape cruciale consiste alors à déterminer un ensemble de caractéristiques permettant une identification aisée des motifs présentés au système. Il est cependant possible que certaines des mesures effectuées soient corrélées ou ne permettent pas la distinction des divers caractères.

Les informations ainsi recueillies sont exploitées par un système de classification. La parfaite reconnaissance s'avérant impossible, diverses méthodes permettent finalement la correction des erreurs.

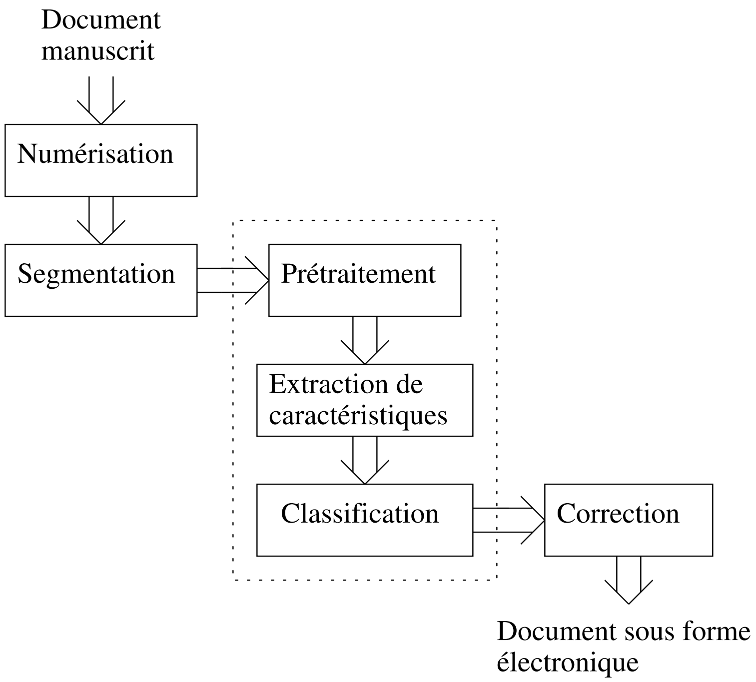


Figure 1.1: Un système de reconnaissance d'écriture *offline* (*Optical Character Recognition* ou *OCR*). Nous ne concevrons que les trois éléments apparaissant dans le cadre en pointillé.

La segmentation et la correction d'erreurs proposent des problèmes ardus auxquels il serait indispensable de consacrer des projets particuliers. Nous ne réaliserons par conséquent que trois éléments du dispositif illustré par la figure 1.1 : les modules de prétraitement, d'extraction de caractéristiques et de classification. Notre système ne traitera ainsi que des caractères préalablement segmentés.

La base de données *NIST Special Database 3* [29] propose de tels caractères, représentant de multiples styles d'écriture manuscrite. Utilisée par plusieurs chercheurs, elle permet la comparaison des différents systèmes de reconnaissance d'écriture. Nous disposons des résultats obtenus par T. M. Breuel et G. Maître [31] lors de la conception d'un logiciel de reconnaissance de chiffres manuscrits et souhaitons tester notre système à l'aide des mêmes données. Nous nous ne travaillerons par conséquent qu'avec des chiffres.

Les réseaux de neurones proposent une solution intéressante aux problèmes de classification. Le concepteur constitue une base d'apprentissage décrivant diverses situations auxquelles sera confronté le système. Au terme d'un processus d'apprentissage utilisant uniquement ces exemples, le réseau se révèle capable de traiter un grand nombre de situations relatives au problème considéré. Cette propriété, appelée généralisation, explique l'engouement pour de tels systèmes. Nous souhaitons réaliser le module de classification à l'aide d'un système neuromimétique. L'apprentissage et la reconnaissance s'effectueront évidemment à partir des informations recueillies par l'algorithme d'extraction de caractéristiques.

Chaque application nécessite cependant une topologie particulière. Diverses expériences ont en effet prouvé que la capacité de généralisation est liée à cette dernière. Plusieurs chercheurs proposent ainsi des méthodes permettant l'obtention du réseau optimal pour un problème donné. Les algorithmes d'élagage suggèrent de débuter l'apprentissage avec un réseau de taille quelconque, capable d'apprendre la tâche considérée, puis d'éliminer les éléments (neurones ou connexions) superflus. Cette approche nous paraît particulièrement attrayante. Il est en effet plausible qu'un tel algorithme élague quelques entrées du réseau, sélectionnant ainsi les caractéristiques pertinentes.

Lors de ce projet, nous travaillerons avec deux types de réseaux neuronaux (*perceptrons* multicouches et *high order perceptrons*) et appliquerons divers algorithmes d'élagage.

1.2 Choix des simulateurs de réseaux neuronaux

Nous avons complété la gamme d'algorithmes d'élagage proposée par *SNNS*[1] (*Stuttgart Neural Network Simulator*) [23] lors d'un précédent projet [4]. Ce logiciel ne permet cependant pas de travailler avec des *high order perceptrons*.

[1] *SNNS* est disponible en *ftp* anonyme : *ftp.informatik.uni-stuttgart.de*.

Lors de la réalisation de sa thèse consacrée aux *high order perceptrons*, G. Thimm [47] a implanté plusieurs algorithmes d'élagage dans *Sesame*. La version actuelle du logiciel n'est malheureusement pas documentée. Le lecteur trouvera néanmoins une brève description dans [19]. Nous utiliserons *Sesame* lors des simulations faisant intervenir des *high order perceptrons*.

1.3 Structure du document

Ce document comporte quatre parties. Au cours de la première, le lecteur découvrira les différents modèles neuromimétiques, règles d'apprentissage et algorithmes d'élagage intervenant dans ce projet. Nous consacrerons la seconde partie au prétraitement et à l'extraction de caractéristiques de chiffres segmentés. Nous présenterons ensuite les diverses expériences réalisées (troisième partie), proposerons une synthèse des résultats et suggérerons diverses extensions du système (quatrième partie).

1.4 Conventions et notations

La terminologie liée aux réseaux neuronaux est essentiellement anglaise. La traduction de certaines expressions s'avérant parfois hasardeuse, nous préférons les écrire dans leur langue d'origine. Les termes concernés figurent en italique dans le texte.

Nous utilisons le format du "Traité d'Electricité" de L'Ecole Polytechnique Fédérale de Lausanne (EPFL) pour les références bibliographiques. Ces dernières sont représentées par un chiffre arabe entre crochets. Lorsque nous mentionnons des algorithmes ou formules célèbres, abondamment décrits dans la littérature, nous ne proposons une référence qu'à un seul ouvrage présentant le sujet.

Nous écrivons les vecteurs en caractères gras. Par exemple, $\boldsymbol{\xi}^\rho$ désigne un vecteur d'entrée d'un réseau de neurones. Notons que nous utiliserons indifféremment les expressions "motif" ou "vecteur d'entrée" afin de désigner $\boldsymbol{\xi}^\rho$. Comme nous travaillons toujours avec une collection de tels vecteurs, nous les différencions à l'aide de l'indice ρ.

C_p^n dénote une combinaison de p éléments parmi n :

$$C_p^n = \binom{n}{p} = \frac{n!}{(n-p)! \cdot p!} \tag{1.1}$$

Remarquons finalement que nous utilisons indifféremment *NIST Special Database 3* ou *NIST* afin de désigner la base de données de caractères segmentés.

Partie I

Systèmes d'apprentissage neuromimétiques

Chapitre 2

Introduction à la première partie

L'objectif de cette partie consiste à présenter au lecteur les modèles de réseaux neuromimétiques utilisés dans le cadre de ce projet.

Après avoir présenté quelques éléments de neurophysiologie[1], nous décrirons le neurone formel proposé par McCulloch et Pitts, constituant la base de plusieurs modèles (chapitre 3). Nous exposerons ensuite les fondements de l'apprentissage supervisé (chapitre 4) et décrirons le *Perceptron* et l'*Adaline*, deux architectures neuromimétiques élémentaires (chapitre 5). L'étude de leurs limitations nous permettra de présenter les *high order perceptrons* et *perceptrons* multicouches, deux architectures neuromimétiques utilisées dans ce projet (chapitre 6) et définies formellement au chapitre 7. Nous décrirons alors deux algorithmes d'apprentissage dédiés à ces réseaux (chapitre 8). Nous discuterons finalement l'importance de l'architecture de nos systèmes neuromimétiques (chapitre 9) et présenterons divers algorithmes déterminant la topologie optimale d'un réseau (chapitre 10).

[1] Ce paragraphe s'inspire fortement de [11] [28] [24] [30].

Chapitre 3

Du neurone biologique au modèle formel

3.1 Eléments de neurophysiologie

Les neurones constituent les éléments de base du système nerveux. Il en existe une grande variété remplissant diverses fonctions (olfaction, audition, ...). Nous ne présentons néanmoins que l'archétype d'une cellule nerveuse. Cette dernière se compose de trois éléments essentiels (figure 3.1) :

- le corps cellulaire, ou **soma**, constitué du cytoplasme et d'un noyau détenteur des gènes;
- l'**arbre dendritique**, collectant les signaux provenant d'autres cellules nerveuses (ou de l'extérieur, s'il s'agit d'un neurone afférent);
- l'**axone**, diffusant le signal et se divisant parfois à son extrémité afin d'entrer en contact avec un grand nombre de cellules (nerveuses, musculaires, ...).

Comme toutes les cellules, le neurone crée une différence de charges ioniques entre l'intérieur et l'extérieur de sa membrane[1]. Chaque neurone possède ainsi un potentiel de repos d'environ -60 millivolts.

Un processus électrochimique complexe permet la transmission de signaux dans le système nerveux. Des capteurs de différentes natures (pression, température, ...) provoquent une dépolarisation de la membrane des neurones afférents (ou sensoriels). Cette stimulation se propage le long de leur membrane et s'atténue en fonction de la distance parcourue. Si elle s'avère suffisamment importante lorsqu'elle atteint le segment initial de l'axone, elle produit un potentiel d'action (codage en fréquence et durée). Ce dernier se propage alors le long de l'axone.

Arrivé à l'extrémité de l'axone, le potentiel d'action est transformé en un neurotransmetteur diffusé dans la fente synaptique (figure 3.2). Nous assistons ici à un phénomène de nature chimique. Lorsque le neurotransmetteur atteint le neurone suivant, il génère un potentiel synaptique se propageant passivement à la surface de la membrane. Ce dernier provoque une excitation (dépolarisation) ou une inhibition (hyperpolarisation) du neurone.

[1] La charge est négative à l'intérieur de la membrane et positive à l'extérieur. La référence à zéro est à l'extérieur de la membrane.

Le neurone reçoit de tels potentiels synaptiques de toutes les cellules nerveuses avec lesquelles il est en liaison. Il effectue une sommation spatio-temporelle de ces informations. Si le résultat obtenu est supérieur à un seuil, le soma génère un potentiel d'action (ou *spike*), d'amplitude et de durée fixes. Chaque neurone prend ainsi une décision locale.

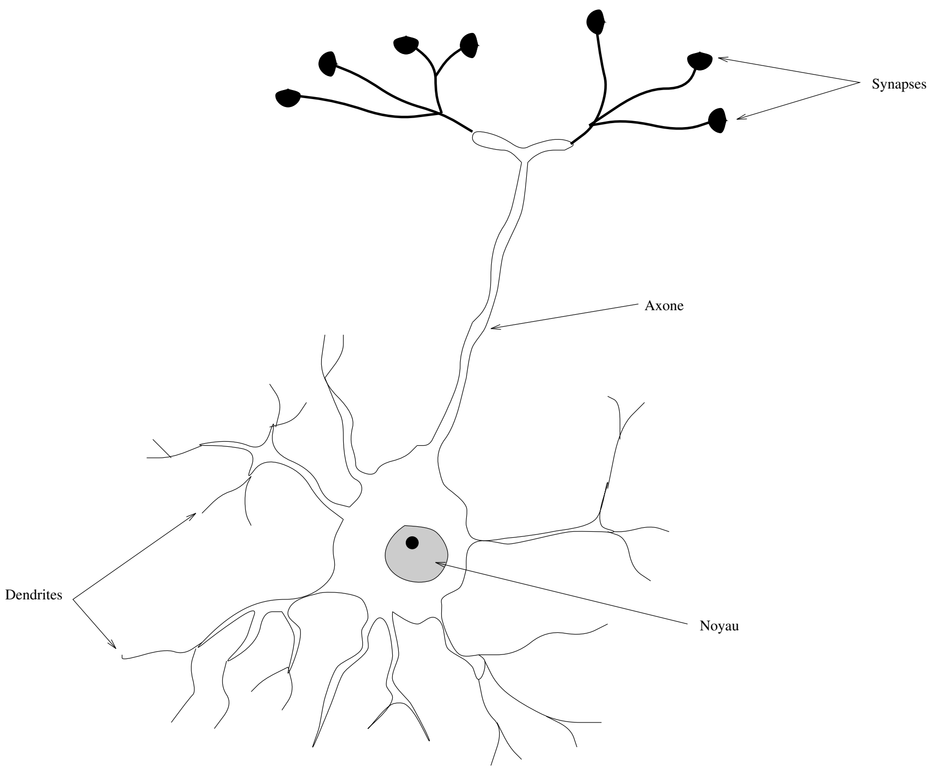


Figure 3.1: Schéma d'un neurone biologique

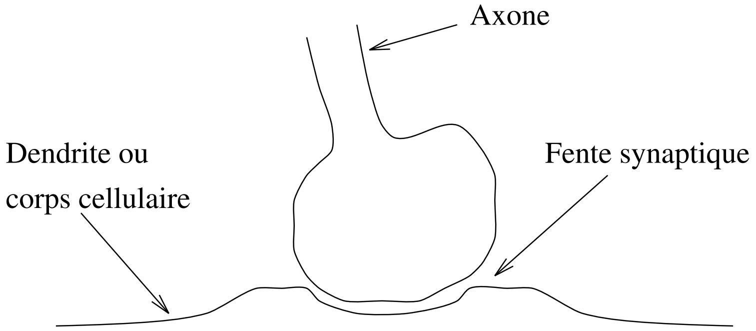


Figure 3.2: La fente synaptique

3.2 Le modèle de McCulloch et Pitts

C'est en 1943 que W. McCulloch et W. Pitts, désireux de comprendre le fonctionnement du système nerveux, proposèrent un modèle de neurone formel. Leur travail décrit les fonctions remplies par le neurone et la synapse et propose une approche binaire du traitement de l'information [49] :

> *"Because of the all-or-none character of the nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes"*

Le neurone formel de McCulloch et Pitts (figure 3.3 (a)) reçoit des stimulations x_i qu'il pondère par des coefficients (ou poids) synaptiques w_i. Lorsque ces derniers sont positifs, nous parlons de connexions excitatrices; les connexions sont dites inhibitrices lorsque les poids synaptiques sont négatifs. Le neurone calcule ensuite le potentiel $p = \sum_i (w_i \cdot x_i)$ du neurone, le compare à un seuil θ et prend une décision :

- si $p > \theta$, la sortie du dispositif est égale à $+1$;
- sinon, elle est égale à -1.

Ce calcul revient à vérifier si la différence $p-\theta$ est supérieure à zéro, c'est-à-dire à remplacer le seuil par une entrée fixe, de valeur -1, à laquelle nous associons un poids variable, appelé biais (*bias*) du neurone (figure 3.3 (a)). Nous modélisons alors la prise de décision du neurone à l'aide de la fonction d'activation $\varphi(p-\theta) = \text{Signe}(p-\theta)$ illustrée par la figure 3.3 (b).

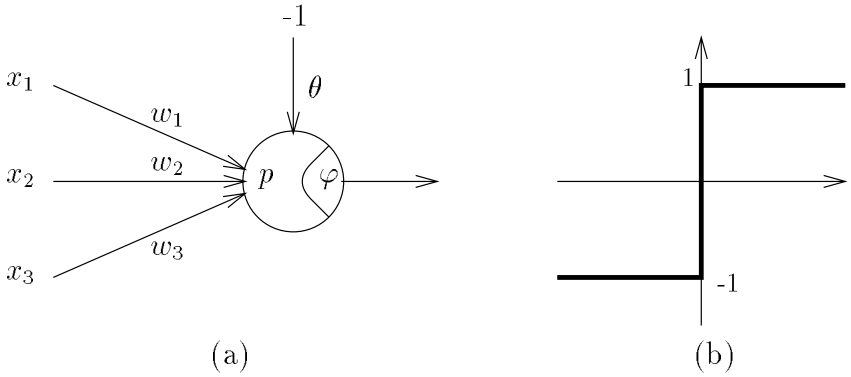


Figure 3.3: Le neurone de McCulloch et Pitts. (a) Schéma du neurone. (b) Représentation de la fonction "Signe".

Considérons un neurone ne comportant que deux entrées. Si

$$w_1 \cdot x_1 + w_2 \cdot x_2 > \theta, \tag{3.1}$$

la sortie du dispositif est égale à $+1$. Cette dernière vaut -1 lorsque

$$w_1 \cdot x_1 + w_2 \cdot x_2 \leq \theta. \tag{3.2}$$

Géométriquement, les équations (3.1) et (3.2) signifient que nous avons séparé le plan par une droite d'équation (figure 3.4)

$$x_2 = -\frac{w_1}{w_2} \cdot x_1 + \frac{\theta}{w_2} \tag{3.3}$$

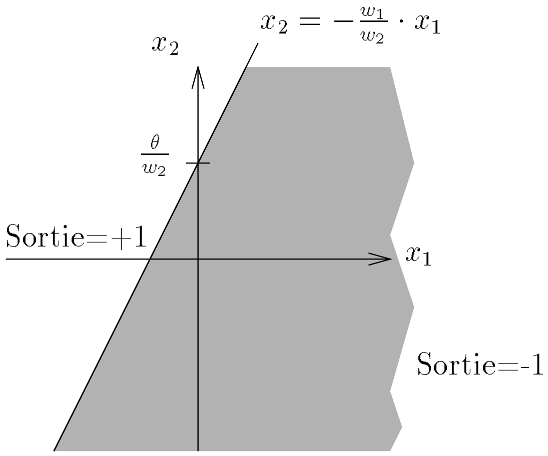


Figure 3.4: Interprétation géométrique du fonctionnement du neurone de McCulloch et Pitts. Cette interprétation se généralise évidemment à des dimensions supérieures à deux.

Ce modèle, bien que très simple, constitue encore aujourd'hui un élément de base des réseaux de neurones artificiels.

Chapitre 4

Apprentissage supervisé

Nous présentons ici les principes de l'apprentissage supervisé à l'aide du problème de reconnaissance d'écriture manuscrite que nous étudions. La figure 4.1 illustre l'architecture générale du réseau. Ce dernier possède une sortie associée à chacun des caractères considérés.

Diverses bases de données, comme *U.S. Postal Service OAT Handwritten Zip Code Database* ou *NIST Special Database 3* [29], proposent des collections de chiffres manuscrits écrits par des centaines de personnes. La première étape du travail consiste à sélectionner et prétraiter (nous consacrerons le chapitre 12 à ce sujet) divers caractères afin de constituer une base d'entraînement dont le tableau 4.1 décrit le format général. Nous associons à chaque motif une information concernant la classe à laquelle il appartient. Lorsque nous présentons le chiffre "5", nous souhaitons que seule la sortie correspondante soit active. Nous obtenons ainsi les vecteurs de sortie du tableau 4.1.

Entrée du réseau	Sorties désirées
Description du chiffre 1	0 1 0 0 0 0 0 0 0 0
Description du chiffre 0	1 0 0 0 0 0 0 0 0 0
Description du chiffre 5	0 0 0 0 0 1 0 0 0 0

Tableau 4.1: Format de la base d'apprentissage. Les niveaux de gris des pixels constituent une représentation possible d'un chiffre. Nous proposerons quelques autres méthodes permettant de décrire un caractère dans les chapitres 13 et 14. Remarquons que la première sortie du réseau de la figure 4.1 est associée au chiffre "0". Ainsi, lorsque nous présentons le chiffre "5", nous souhaitons que la sixième sortie soit active.

L'apprentissage supervisé comporte les étapes suivantes :

- **Présentation d'un chiffre au réseau.**
  Nous présentons un motif au réseau et observons la réponse proposée.

- **Comparaison et adaptation.**
  Nous avons associé la réponse désirée à chaque motif de la base d'apprentissage. Il est ainsi facile de déterminer si le réseau classifie correctement le chiffre présenté. Lorsque

survient une erreur (figure 4.1), le superviseur (algorithme d'apprentissage) modifie les poids synaptiques.

La présentation de tous les motifs d'entraînement (généralement dans un ordre aléatoire) et l'adaptation des coefficients synaptiques constituent un cycle d'apprentissage. Idéalement, le réseau classifie correctement toutes les données d'entraînement après quelques cycles (figure 4.2). Leur nombre dépend essentiellement de la difficulté du problème ainsi que du choix de divers paramètres. Il est toutefois probable que le système n'apprenne pas certains caractères fortement bruités ou très particuliers. Au terme de l'apprentissage, le réseau se révèle habituellement capable de classifier un nombre important de données relatives au problème et ne figurant pas parmi les motifs d'entraînement.

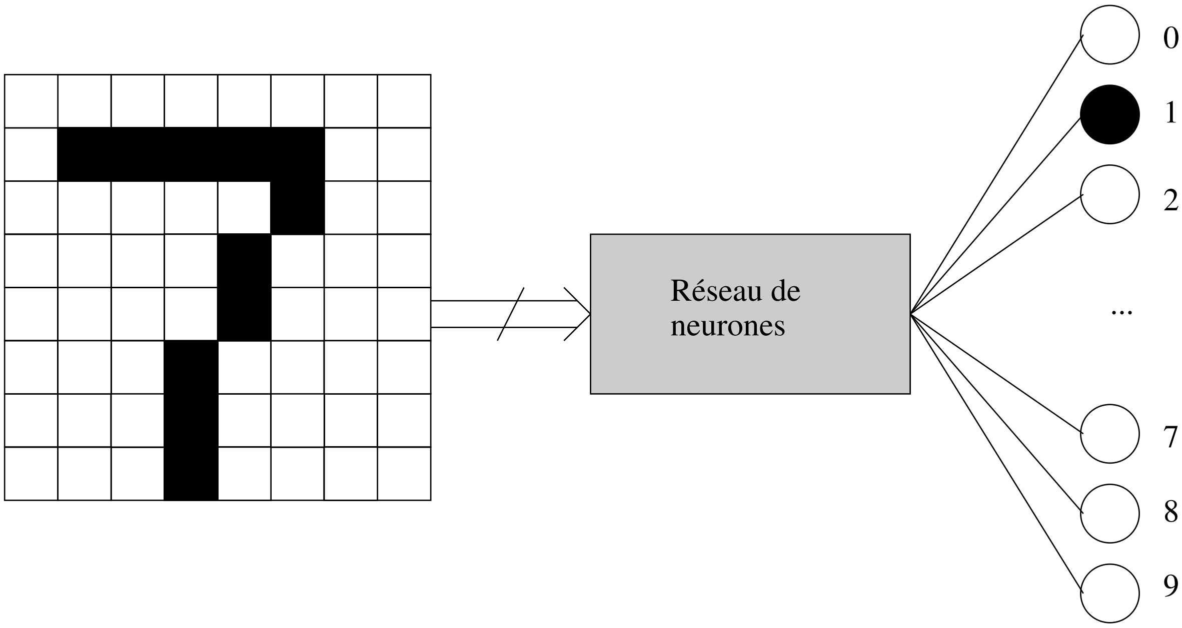


Figure 4.1: Exemple d'apprentissage supervisé (a). Durant l'apprentissage, nous présentons le chiffre “7” au réseau. Il le confond avec le chiffre “1”.

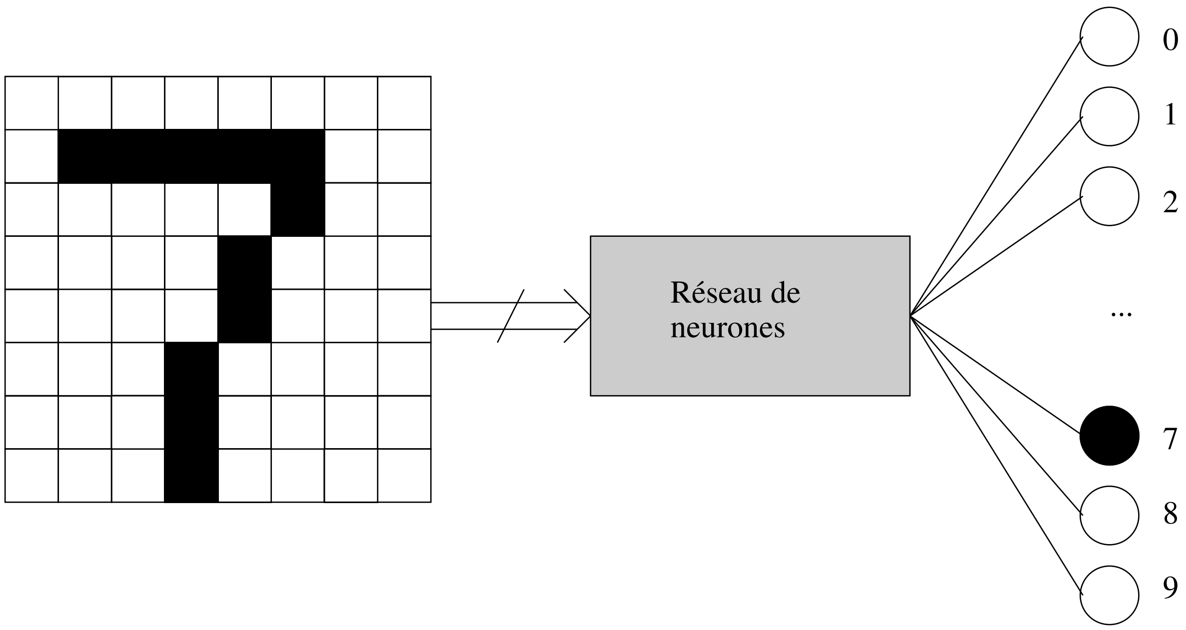


Figure 4.2: Exemple d'apprentissage supervisé (b). Idéalement, une fois l'apprentissage terminé, le réseau fournit la réponse correcte lors de la présentation du chiffre "7".

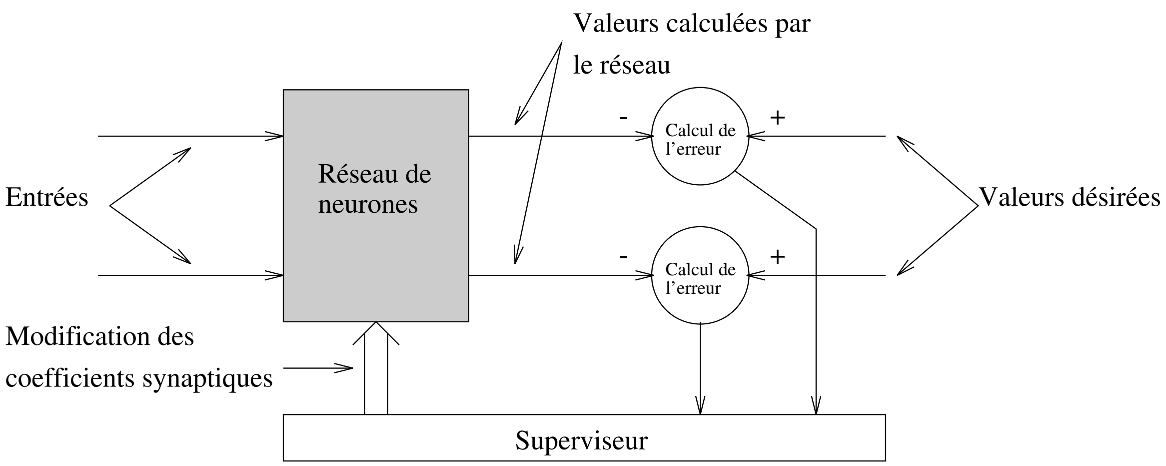


Figure 4.3: Apprentissage supervisé

Chapitre 5

Les réseaux monocouches : l'*Adaline* et la règle du *Perceptron*

Nous présentons dans ce chapitre deux algorithmes d'apprentissage supervisé destinés aux réseaux monocouches. Ces derniers constituent une classe de systèmes neuromimétiques élémentaires : leurs entrées sont reliées aux sorties par une couche de poids synaptiques (figure 5.1). Le fonctionnement des neurones est analogue à celui du modèle formel de McCulloch et Pitts.

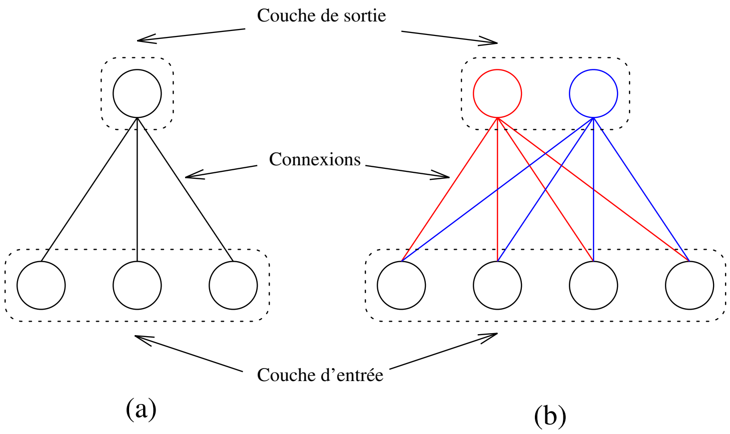


Figure 5.1: Deux réseaux monocouches. (a) Le réseau de gauche, adapté à l'apprentissage de fonctions booléennes, ne possède qu'un neurone de sortie. (b) Nous utiliserions par contre le réseau de droite pour un problème de classification. Remarquons qu'il se décompose en deux réseaux ne comportant qu'une sortie.

Bien que très simples, ces systèmes fournissent de bons résultats en reconnaissance de caractères manuscrits [46]. Nous espérons de plus que leur étude permettra au lecteur de se familiariser avec les mathématiques des réseaux neuronaux et de mieux appréhender les chapitres suivants.

Nous souhaitons apprendre un ensemble de vecteurs $\boldsymbol{\xi}^\rho = [\xi_1^\rho, \dots, \xi_N^\rho]^T$, où N désigne le nombre d'entrées du système, au réseau de la figure 5.1 (a)[1]. Chacun d'eux appartient à une classe $t^\rho \in \{-1, 1\}$. Le vecteur $\boldsymbol{w} = [w_1, \ldots, w_N]^T$ contient les coefficients synaptiques du réseau.

A quelles conditions un problème doit-il satisfaire afin d'être résolu par un réseau monocouche ? Soient A et B les deux classes auxquelles appartiennent les vecteurs $\boldsymbol{\xi}^\rho$. Nous avons constaté que le neurone détermine un hyperplan de dimension $N-1$ dans un espace de dimension N (paragraphe 3.2). S'il existe un hyperplan séparant les motifs de la classe A de ceux de la classe B, nous dirons que le problème est linéairement séparable[2]. Les figures 5.2 et 5.3 illustrent des problèmes linéairement et non linéairement séparables. Un réseau monocouche n'est capable de résoudre que des problèmes linéairement séparables.

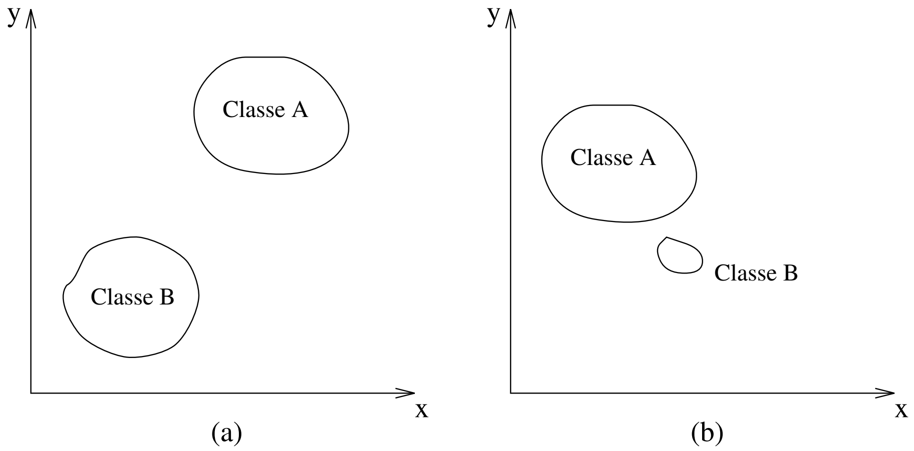


Figure 5.2: Deux problèmes linéairement séparables

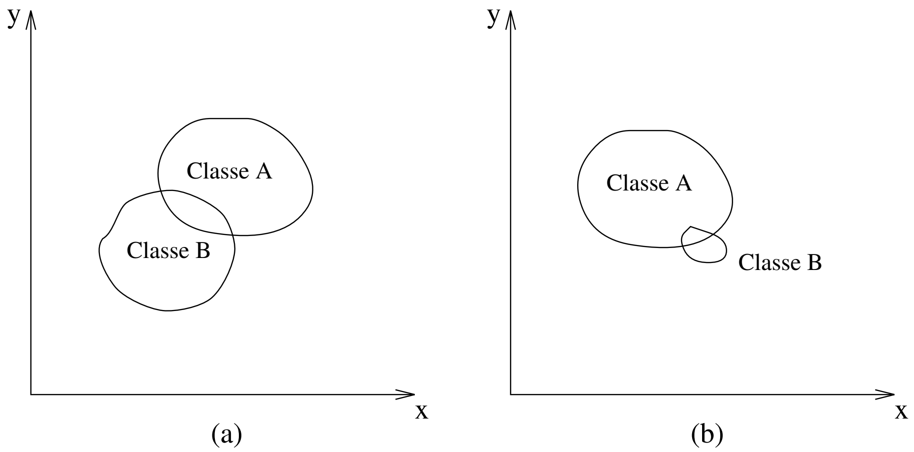


Figure 5.3: Deux problèmes non linéairement séparables

[1] Apprendre une tâche au réseau de la figure 5.1 (b) revient à entraîner séparément deux réseaux ne comportant qu'une sortie.

[2] Plus formellement, k classes $C_1, C_2, \ldots, C_k$ sont linéairement séparables si et seulement si leurs enveloppes convexes sont disjointes.

5.1 La règle du *Perceptron*

La règle du *Perceptron*, proposée par Rosenblatt [41] [12], décrit un algorithme modifiant les poids $\boldsymbol{w} = [w_1, \ldots, w_N]^T$ afin de réaliser la classification. Notons que cette méthode ne s'applique qu'à des problèmes linéairement séparables.

Après avoir initialisé aléatoirement les poids, nous présentons le motif $\boldsymbol{\xi}^\rho$ au réseau et calculons la sortie du neurone :

$$\begin{aligned} a^\rho &= \varphi\left(\boldsymbol{w}^T \boldsymbol{\xi}^\rho\right) \\ &= \varphi\left(\sum_{i=1}^{N} w_i \cdot \xi_i^\rho\right) \end{aligned} \tag{5.1}$$

où $\varphi(x)$ est la fonction “Signe” illustrée par la figure 3.3 (b). Nous comparons ensuite la sortie du réseau à la valeur désirée t^ρ et ne modifions les poids que si la classification s'avère incorrecte[3]. La règle d'adaptation de Rosenblatt est décrite par :

$$\boldsymbol{w} = \boldsymbol{w} + \eta \cdot (t^\rho - a^\rho) \cdot \boldsymbol{\xi}^\rho \tag{5.2}$$

où η désigne le coefficient d'apprentissage. Généralement, la valeur de η est comprise entre 0 et 1. Tant que le réseau ne classifie pas correctement l'ensemble des données d'apprentissage, nous réitérons ce processus. Une question primordiale subsiste néanmoins : cet algorithme converge-t-il ?

Théorème 1
Si le problème considéré est linéairement séparable, l'algorithme du Perceptron *converge en un nombre fini d'itérations. Le lecteur trouvera une démonstration dans [11] ou [25].*

Ainsi, lorsque le problème n'est pas linéairement séparable, l'algorithme modifiera indéfiniment les coefficients synaptiques. Remarquons finalement que la règle du *Perceptron* (dont la figure 5.5 illustre le fonctionnement) minimise une distance dans l'espace des poids [11].

[3] C'est-à-dire si $t^\rho - a^\rho \neq 0$.

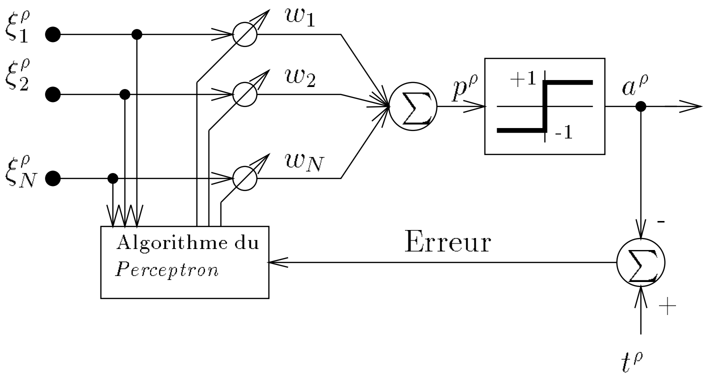


Figure 5.4: Le modèle du *Perceptron*

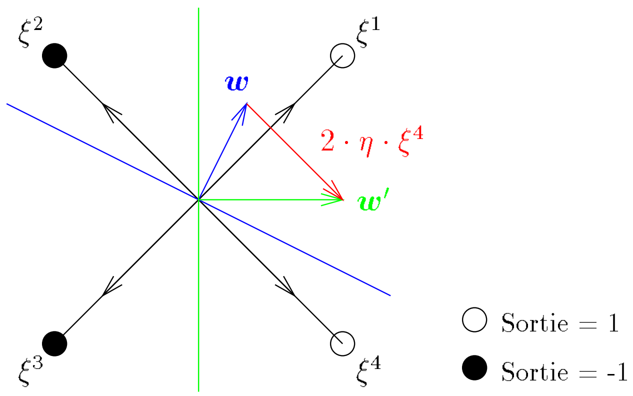


Figure 5.5: Illustration du fonctionnement de la règle du *Perceptron*. Le vecteur $\boldsymbol{w}$ est initialisé aléatoirement. Nous présentons alors au réseau les motifs $\boldsymbol{\xi}^1$, $\boldsymbol{\xi}^4$, $\boldsymbol{\xi}^3$ et finalement $\boldsymbol{\xi}^2$. Le vecteur $\boldsymbol{\xi}^1$ étant correctement classifié ($\varphi\left(\boldsymbol{w}^T\boldsymbol{\xi}^1\right) = t^1$), nous ne modifions pas les coefficients synaptiques. Le réseau commet par contre une erreur lors de la présentation de $\boldsymbol{\xi}^2$. Nous corrigeons alors $\boldsymbol{w}$ selon la règle (5.2) et obtenons $\boldsymbol{w'} = \boldsymbol{w} + 2 \cdot \eta \cdot \boldsymbol{\xi}^4$. Nous constatons que le réseau classifie correctement les quatre vecteurs $\boldsymbol{\xi}^\rho$ et achevons le processus d'apprentissage.

5.2 L'*Adaline*

Ce modèle, proposé par Widrow et Hoff [3], est calqué sur celui de McCulloch et Pitts. Notons cependant une différence subtile : la fonction d'activation "Identité" est substituée à la fonction "Signe"[4]. La sortie du neurone, lors de la présentation du vecteur $\boldsymbol{\xi}^\rho$, est ainsi définie par :

$$\begin{aligned} a^\rho &= \boldsymbol{w}^T \boldsymbol{\xi}^\rho \\ &= \sum_{i=1}^{N} w_i \cdot \xi_i^\rho \end{aligned} \tag{5.3}$$

Nous définissons ensuite le terme d'erreur par

$$E^\rho = \frac{1}{2} \left(t^\rho - a^\rho\right)^2 \tag{5.4}$$

Remarquons que E^ρ est une fonction différentiable des poids dont une méthode de descente de gradient permet de trouver le minimum global. Lors de la présentation d'un motif, les poids subissent une modification $\boldsymbol{\Delta w}$, avec

$$\Delta w_i = -\eta \cdot \frac{\partial E^\rho}{\partial w_i} = \eta \cdot \left(t^\rho - \sum_{i=1}^{N} w_i \cdot \xi_i^\rho\right) \cdot \xi_i^\rho \tag{5.5}$$

où η est le coefficient d'apprentissage. La figure 5.7 (a) illustre le fonctionnement de l'algorithme. Désignons par $\boldsymbol{\xi}^*$ le motif de la classe A se trouvant dans l'angle inférieur droit du graphe. Un tel motif résulte par exemple d'une erreur de mesure lors de la conception de la base d'apprentissage. Il est alors étranger au problème. Lorsque nous entraînons le réseau sans tenir compte de $\boldsymbol{\xi}^*$, nous obtenons la séparatrice rouge; si $\boldsymbol{\xi}^*$ figure dans les données d'apprentissage, l'algorithme fournit la séparatrice bleue (figure 5.7 (a)). Une modification du modèle permet d'atténuer l'effet du vecteur d'entrée $\boldsymbol{\xi}^*$. Remplaçons la fonction d'activation "Identité" par une fonction φ de type sigmoïde, continue et différentiable en tout point. La sortie du neurone est alors définie par :

$$a^\rho = \varphi\left(\boldsymbol{w}^T \boldsymbol{\xi}^\rho\right) = \varphi\left(\sum_{i=1}^{N} w_i \cdot \xi_i^\rho\right) \tag{5.6}$$

Nous obtenons ainsi une nouvelle règle de modification des poids, appelée *Least Mean Square Algorithm* ou *LMS Algorithm* dans la littérature anglaise :

$$\begin{aligned} \Delta w_i &= -\eta \cdot \frac{\partial E^\rho}{\partial w_i} \\ &= \eta \cdot \left(t^\rho - \varphi\left(\sum_{i=1}^{N} w_i \cdot \xi_i^\rho\right)\right) \cdot \xi_i^\rho \cdot \varphi'\left(\sum_{i=1}^{N} w_i \cdot \xi_i^\rho\right) \end{aligned} \tag{5.7}$$

Effectuons la même expérience que précédemment. Nous obtenons la séparatrice rouge en écartant $\boldsymbol{\xi}^*$ de la base d'apprentissage (figure 5.7 (b)). Lorsque $\boldsymbol{\xi}^*$ apparaît lors de l'entraînement, l'algorithme *LMS* détermine la séparatrice bleue. Nous constatons que la non-linéarité réduit passablement l'importance de $\boldsymbol{\xi}^*$.

Généralement, nous combinons deux critères d'arrêt afin de décider quand achever l'apprentissage :

[4] La valeur délivrée à la sortie du système est égale au potentiel.

- Lorsque l'erreur $E = \sum_\rho E^\rho$ calculée sur l'ensemble des données d'apprentissage est inférieure à un seuil θ fixé, ou
- si l'erreur E est supérieure à θ après un nombre donné de cycles d'apprentissage[5], nous terminons l'entraînement.

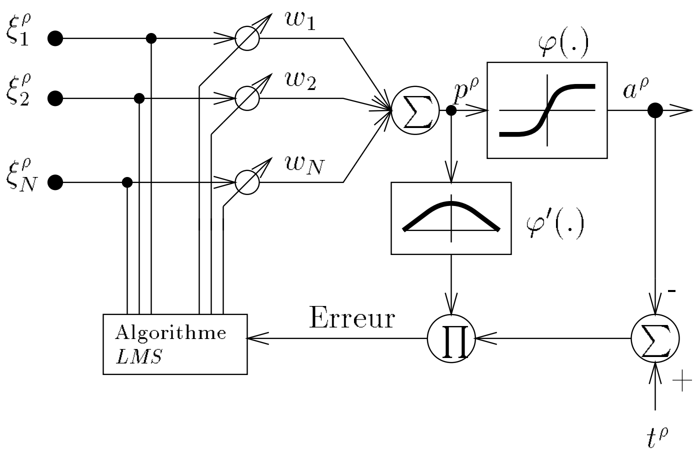


Figure 5.6: Le modèle de l'*Adaline* non linéaire

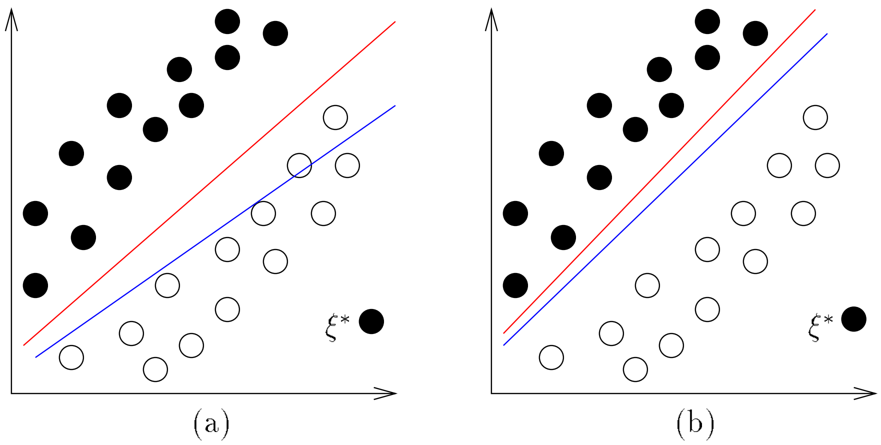


Figure 5.7: Effets de la non-linéarité. Les points noirs (respectivement blancs) appartiennent à la classe A (respectivement B).

5.3 Comparaison du *Perceptron* et de l'*Adaline*

Etudions les solutions proposées par le *Perceptron* et l'*Adaline* aux problèmes des figures 5.2 et 5.3. Le théorème du *Perceptron* garantit que l'algorithme trouvera une solution si la tâche

[5] Ce critère évite d'entraîner indéfiniment le réseau si le seuil θ est mal choisi. Par exemple, si θ est égal à zéro alors que le problème n'est pas linéairement séparable, la relation $E = \sum_\rho E^\rho \leq \theta$ ne sera jamais satisfaite : il existe au moins un motif ρ pour lequel $E^\rho \neq 0$.

est linéairement séparable. Remarquons que cette méthode appliquée plusieurs fois au même problème fournit des solutions différentes. Examinons la figure 5.5. Nous constatons qu'une valeur différente du coefficient η aurait conduit à une autre solution. L'ordre dans lequel sont présentés les motifs influe aussi la solution.

Seul l'*Adaline* déterminera une solution lorsque le problème n'est pas linéairement séparable (figure 5.10). Cet algorithme présente cependant un inconvénient majeur. Il minimise simultanément deux critères des moindres carrés [7] (les lignes bleues et rouges des figures 5.9 et 5.10 dénotent respectivement les droites des moindres carrés des classes A et B). Nous constatons (figure 5.9 (b)) que l'algorithme ne détermine pas toujours la solution d'un problème linéairement séparable.

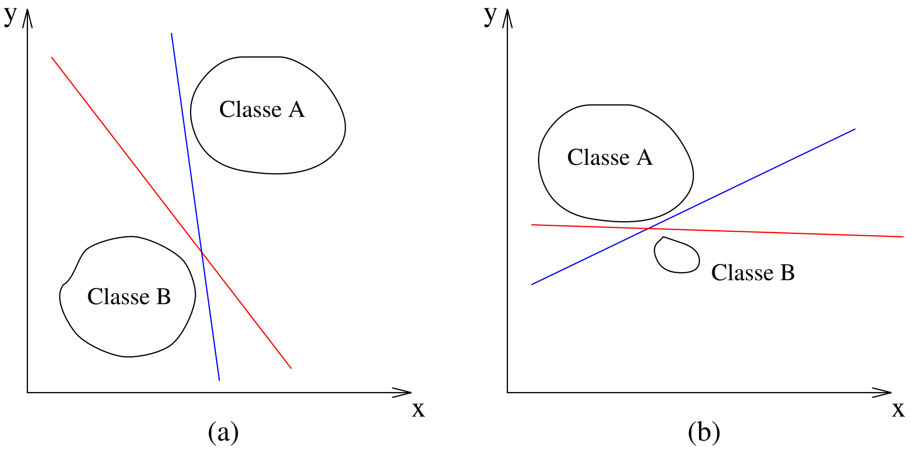


Figure 5.8: Solutions obtenues avec la règle du *Perceptron* dans les situations illustrées par la figure 5.2. Les lignes de différentes couleurs indiquent quelques-unes des solutions du problème.

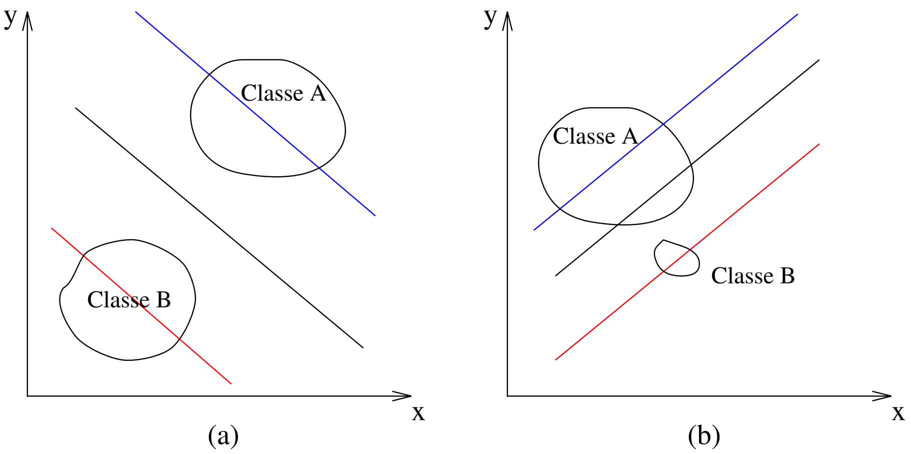


Figure 5.9: Solutions obtenues avec l'*Adaline* dans les situations illustrées par la figure 5.2. La ligne bleue (respectivement rouge) désigne la droite des moindres carrés de la classe A (respectivement B).

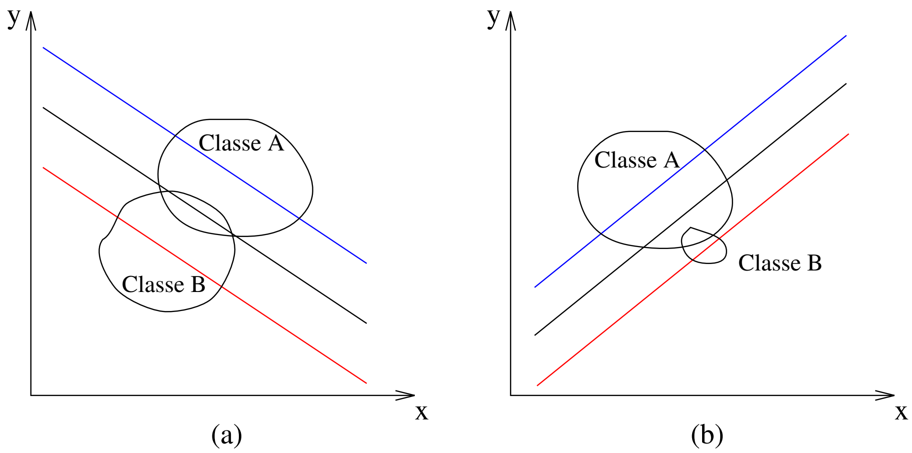


Figure 5.10: Solutions obtenues avec l'*Adaline* dans les situations illustrées par la figure 5.3. La ligne bleue (respectivement rouge) désigne la droite des moindres carrés de la classe A (respectivement B). Contrairement au *Perceptron*, l'*Adaline* détermine une solution lorsque le problème n'est pas linéairement séparable. Cette dernière minimisant le critère d'erreur quadratique, nous la considérons optimale.

Chapitre 6

Limitations des modèles monocouches : vers les *perceptrons* multicouches et les *high order perceptrons*

Afin d'illustrer les limitations des réseaux monocouches, nous étudierons le problème du "ou exclusif" de deux variables booléennes x et y (figure 6.1). Ce problème, pourtant trivial, n'est pas linéairement séparable. Il est par conséquent impossible de le résoudre à l'aide d'un réseau monocouche. Lors de nos réflexions, nous utiliserons la fonction "Signe" de la figure 6.1 (b).

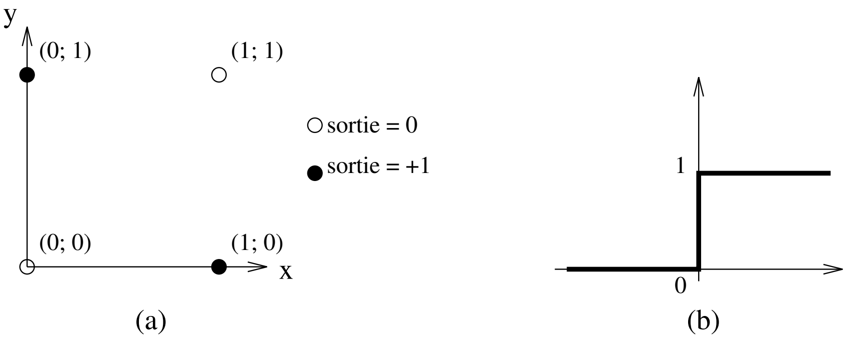


Figure 6.1: (a) Le problème du "ou exclusif" de deux variables. (b) Fonction d'activation des neurones.

6.1 Le problème du "ou exclusif" résolu à l'aide d'un réseau multicouche

Une solution consiste à tracer deux droites séparatrices dans le plan (figure 6.2) à l'aide de deux *Perceptrons*. Les lignes bleue et rouge isolent respectivement les points $(0; 0)$ et $(1; 1)$. Un troisième *Perceptron* combine alors ces deux lignes afin de réaliser la séparation désirée. Le réseau ainsi obtenu est un *perceptron* multicouche : outre ses neurones d'entrée et de sortie, il possède une couche intermédiaire (appelée couche cachée ou *hidden layer* dans la littérature anglaise), constituée ici de deux neurones (figure 6.3).

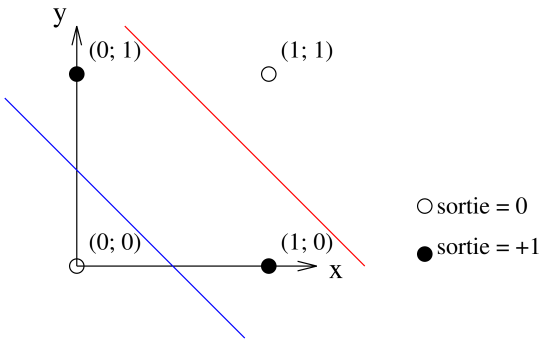


Figure 6.2: Séparation réalisée par un *perceptron* multicouche

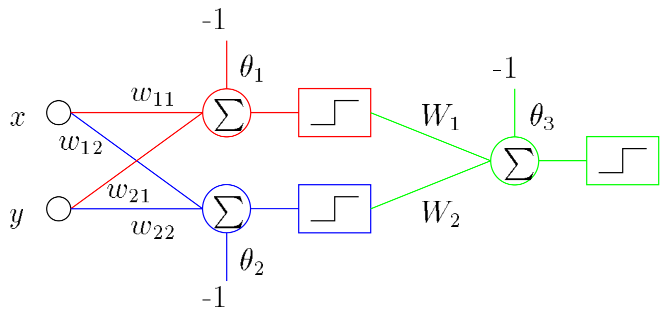


Figure 6.3: *Perceptron* multicouche réalisant la séparation de la figure 6.2. Le *perceptron* dessiné en bleu (respectivement en rouge) trace la séparatrice bleue (respectivement rouge) de la figure 6.2. Le neurone dessiné en vert combine alors ces deux lignes afin de réaliser la séparation désirée. Des valeurs possibles pour les connexions et les biais sont : $w_{11} = w_{12} = w21 = w22 = 1$, $\theta_1 = 1.5$, $\theta_2 = 0.5$, $W_1 = -2$, $W_2 = 1$ et $\theta_3 = 0.5$.

Une couche de neurones cachés permet de réaliser une séparation de deux classes en régions convexes. Deux couches cachées s'avèrent indispensables afin de déterminer des régions non convexes [24] [16]. Il est possible de démontrer qu'en substituant une fonction d'activation de type sigmoïde à la fonction "Signe", une couche cachée se révèle suffisante afin de réaliser une séparation en régions non convexes. Remarquons enfin que ces propriétés sont extensibles aux réseaux possédant plusieurs sorties. Nous pouvons en effet les décomposer en réseaux élémentaires à une sortie.

6.2 Le problème du "ou exclusif" résolu à l'aide d'un *high order perceptron*

Développons l'équation logique de $x \oplus y$:

$$\begin{aligned} x \oplus y &= \overline{x}y + x\overline{y} \\ &= (1-x)\cdot y + x\cdot(1-y) \\ &= x + y - 2\cdot x\cdot y \end{aligned} \tag{6.1}$$

Le terme $-2\cdot x\cdot y$ constitue une combinaison non linéaire des entrées du réseau. Considérons une connexion particulière (appelée connexion d'ordre supérieur ou *high order connection* dans la littérature anglaise) multipliant les sorties de deux neurones, puis pondérant le résultat par un coefficient synaptique. Nous pouvons ainsi résoudre le problème du "ou exclusif" à l'aide du réseau de la figure 6.4, appelé *high order perceptron*.

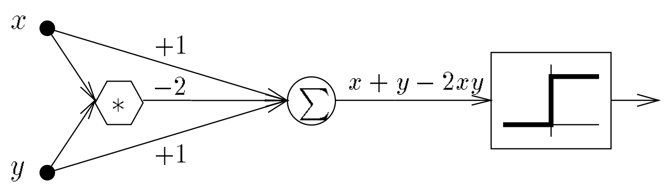


Figure 6.4: *High order perceptron* résolvant le problème du "ou exclusif"

L'utilisation de connexions d'ordre supérieur permet la réalisation de séparations quelconques à l'aide de réseaux monocouches. Considérons le *high order perceptron* de la figure 6.5 (a). Un choix judicieux des coefficients synaptiques conduit à une séparation résolvant le problème du "ou exclusif" (figure 6.5 (b)).

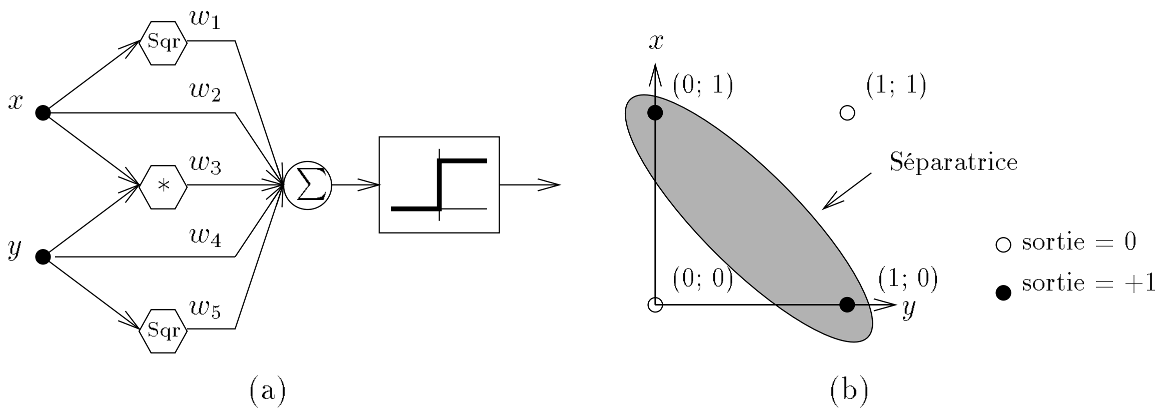


Figure 6.5: *High order perceptron* résolvant le problème du "ou exclusif". "Sqr" dénote l'élévation au carré. Cet exemple est inspiré de [2].

Chapitre 7

Formalisation des *perceptrons* multicouches et *high order perceptrons*

Nous présentons dans ce chapitre la définition formelle d'un *perceptron* multicouche (*multilayer perceptron*) et d'un *high order perceptron*. Nous nous inspirons des notations proposées dans [13] et [47].

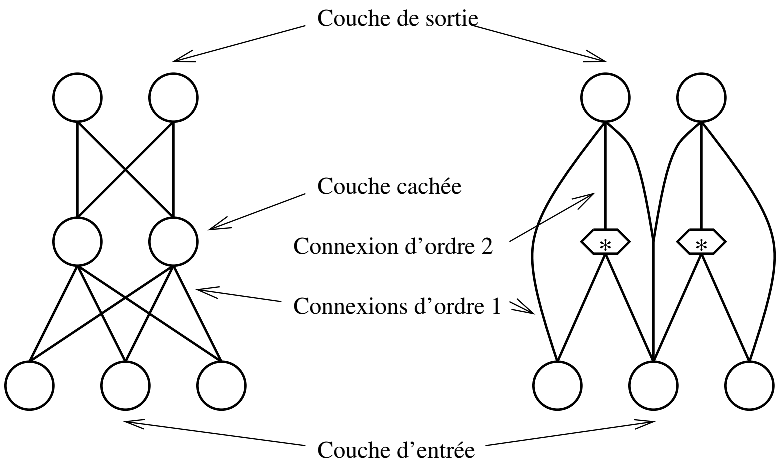


Figure 7.1: Architecture du *perceptron* multicouche et du *high order perceptron*

Nous avons rencontré deux types de connexions lors du précédent chapitre. Les connexions de premier ordre relient simplement deux neurones. Les connexions d'ordre supérieur combinent les sorties de différents neurones à l'aide d'une multiplication[1]. Le nombre de signaux intervenant dans la multiplication détermine l'ordre Ω de la connexion.

Définition 7.1 (*Perceptron* multicouche (*multilayer perceptron*))
Un perceptron *multicouche (figure 7.1) est un réseau de neurones comportant, outre les*

[1]D'autres fonctions sont évidemment envisageables. Nous nous restreindrons cependant à la multiplication dans le cadre de ce projet.

*couches d'entrée et de sortie, une ou plusieurs couches cachées (*hidden layers*). Un n-uplet $N_1, N_2, \cdots, N_L$, où L est le nombre total de couches et N_i le nombre de neurones de la couche i $(1 \leq i \leq L)$, permet de définir la charpente du réseau. Les connexions s'effectuent entre deux couches adjacentes N_i et N_{i+1}. Nous dirons qu'un réseau est totalement connecté si toutes les connexions de premier ordre possibles entre deux couches N_i et N_{i+1} sont effectuées. Un* perceptron *multicouche admet des entrées et des sorties continues, discrètes, booléennes, ou un mélange de ces différents types.*

DÉFINITION 7.2 (*Perceptron* D'ORDRE SUPÉRIEUR (*High order perceptron*))
Un perceptron *d'ordre supérieur est un réseau monocouche possédant des connexions de premier ordre et d'ordre supérieur (figure 7.1). Nous dirons qu'un réseau d'ordre Ω est totalement connecté si toutes les combinaisons des entrées $C_1^{N_1}, C_2^{N_1}, \ldots, C_\Omega^{N_1}$ sont effectuées (N_1 désigne le nombre de neurones de la couche d'entrée). Le* perceptron *d'ordre supérieur admet les mêmes types d'entrées et de sorties que le* perceptron *multicouche.*

Définissons encore les notations utilisées dans la suite de ce rapport.

- ξ_i^ρ désigne la $i^{\text{ème}}$ composante du vecteur d'entrée $\boldsymbol{\xi}^\rho$.
- L est le nombre de couches (*layers*) du réseau. Les couches 1 et L sont respectivement appelées couches d'entrée et de sortie.
- N_l dénote le nombre de neurones que comporte la $l^{\text{ème}}$ couche du réseau.
- C_l dénote le nombre de connexions (respectivement de connexions d'ordre supérieur) de la $l^{\text{ème}}$ couche d'un *perceptron* multicouche (respectivement d'un *perceptron* d'ordre supérieur). Pour le *perceptron* multicouche, C_l est évidemment égal à N_{l-1} $(2 \leq l \leq L)$.
- $c^\rho_{l,i,\{(l_1,n_1),(l_2,n_2),\ldots\}}$ est la valeur de sortie de la $i^{\text{ème}}$ connexion (d'ordre supérieur) de la couche l du réseau. Chaque paire (l, n) de l'ensemble de neurones $\{(l_1, n_1), (l_2, n_2), \ldots\}$ détermine une entrée de cette connexion : n est l'indice d'un neurone de la couche l.
- $w_{l,i,j}$ désigne le facteur de pondération (ou poids synaptique) par lequel est multipliée la sortie de la connexion i de la couche l. Le signal ainsi obtenu est acheminé vers le neurone j de la couche l.
- $\varphi_{l,j}$ dénote la fonction d'activation du neurone j de la couche l. Parmi les fonctions couramment utilisées, citons
  - la tangente hyperbolique
  - les fonctions de type sigmoïde

$$f_\beta(x) = \frac{1}{1 + e^{-\beta_{l,j} \cdot x}}$$

  où $\beta_{l,j}$ dénote la pente de $f_\beta(x)$ à l'origine
- t_j^ρ représente la sortie désirée du neurone j de la couche de sortie lorsque nous présentons le motif ρ en entrée;

- $h^{\rho}_{l,i}$ désigne le potentiel du neurone i de la couche l obtenu lors de la présentation du motif ρ.
- $a^{\rho}_{l,i}$ dénote la sortie du neurone i de la couche l calculée lorsque nous présentons le motif ρ en entrée.
- η est le coefficient d'apprentissage.

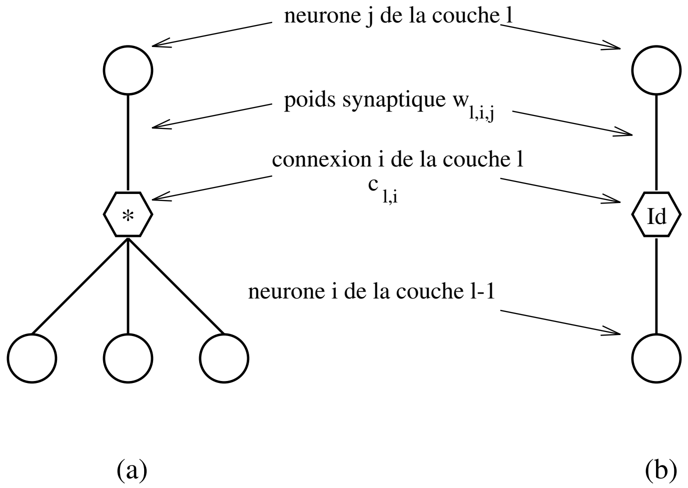


Figure 7.2: Connexions des *high order perceptrons* et *perceptrons* multicouches. (a) Connexion de troisième ordre (*high order perceptrons*). (b) A chaque neurone d'un *perceptron* multicouche, nous adjoignons une connexion de premier ordre réalisant la fonction identité. Nous pouvons ainsi utiliser les mêmes notations pour les *perceptrons* multicouches et les *high order perceptrons*. Nous associons toujours la connexion $c_{l,i}$ au neurone i de la couche $l-1$. Nous obtenons ainsi la relation $c^{\rho}_{l,i} = a^{\rho}_{l-1,i}$, $2 \leq l \leq L$.

Chapitre 8

Algorithmes d'apprentissage pour *multilayer* et *high order perceptrons*

8.1 L'algorithme de *backpropagation*

L'algorithme de *backpropagation* [2] (ou algorithme de rétropropagation du gradient) est la méthode d'apprentissage supervisé la plus utilisée pour les *multilayer perceptrons* et les *high order perceptrons*. Dans la version présentée dans ce chapitre[1], chaque neurone possède sa propre fonction d'activation[2] $\varphi_{l,j}(x)$. L'algorithme se compose des étapes suivantes :

1. **Initialisation.**
   Les poids $w_{l,i,j}$ et les biais sont initialisés aléatoirement. Nous discuterons ce problème plus en détail dans les chapitres dédiés aux simulations.

2. **Présentation d'un motif et propagation des signaux.**
   Nous présentons alors un motif au système :

$$a_{1,i}^{\rho} = \xi_i^{\rho} \qquad \forall i \in [1 \ldots N_1] \tag{8.1}$$

   et propageons le signal à travers le réseau. Le potentiel des divers neurones se calcule selon la formule :

$$h_{l,j}^{\rho} = \left(\sum_{i=0}^{C_l} w_{l,i,j} \cdot c_{l,i,\{(l_1,n_1),(l_2,n_2),\ldots\}}^{\rho} \right) \qquad 2 \leq l \leq L \tag{8.2}$$

   L'activation d'un neurone est alors donnée par :

$$a_{l,j}^{\rho} = \varphi_{l,j}\left(h_{l,j}^{\rho}\right) \tag{8.3}$$

   où $\varphi_{l,j}$ est supposée continue et dérivable en tout point et

$$c_{l,i,\{(l_1,n_1),(l_2,n_2),\ldots\}}^{\rho} = \prod_{(l',n') \in \{(l_1,n_1),(l_2,n_2),\ldots\}} a_{l',n'}^{\rho} \tag{8.4}$$

[1] Le lecteur intéressé trouvera le calcul détaillé des formules dans l'annexe A.

[2] Dans la majorité des applications, la fonction d'activation est la même pour chaque neurone.

3. **Calcul et rétropropagation de l'erreur.**
Nous comparons ensuite les valeurs obtenues en sortie à celles désirées et obtenons l'erreur pour le motif $\boldsymbol{\xi}^\rho$:

$$E^\rho = \frac{1}{2}\sum_{j=1}^{N_L}\left(t_j^\rho - O_j^\rho\right)^2 \tag{8.5}$$

L'erreur totale sur la base d'apprentissage s'écrit par conséquent :

$$\begin{aligned} E &= \sum_\rho E^\rho \\ &= \frac{1}{2}\sum_\rho \sum_{j=1}^{N_L}\left(t_j^\rho - O_j^\rho\right)^2 \end{aligned} \tag{8.6}$$

L'algorithme de *backpropagation*, une variante de descente du gradient, tente de minimiser l'erreur en modifiant les poids du réseau[3]. Lors de la rétropropagation de l'erreur à travers le réseau, chacun des poids subit une modification :

$$w_{l,i,j} = w_{l,i,j} + \Delta w_{l,i,j} \tag{8.7}$$

Il existe diverses variantes de *backpropagation*. L'algorithme *online*[4] suggère la mise à jour des poids après la présentation de chacun des motifs. $\Delta w_{l,i,j}$ est alors défini par :

$$\begin{aligned} \Delta w_{l,i,j} &= -\eta \cdot \frac{\partial E}{\partial w_{l,i,j}} \\ &= \eta \cdot \delta_{l,j}^\rho \cdot c_{l,i}^\rho \end{aligned} \tag{8.8}$$

L'algorithme *offline* (ou *batch*) préconise par contre de ne modifier les poids qu'après avoir présenté au réseau la totalité des motifs d'apprentissage. $\Delta w_{l,i,j}$ se calcule alors comme suit :

$$\begin{aligned} \Delta w_{l,i,j}(n) &= -\eta \sum_\rho \frac{\partial E^\rho}{\partial w_{l,i,j}} \\ &= \eta \cdot \sum_\rho \left(\delta_{l,j}^\rho \cdot c_{l,i}^\rho\right) \end{aligned} \tag{8.9}$$

Les coefficients $\delta_{l,j}^\rho$ sont définis par :

$$\delta_{l,j}^\rho = \begin{cases} \varphi'_{l,j}\left(h_{l,j}^\rho\right) \cdot \left(t_j^\rho - O_j^\rho\right) & \text{si } l = L \\ \varphi'_{l,j}\left(h_{l,j}^\rho\right) \cdot \sum_{k=1}^{N_{l+1}} \delta_{l+1,k}^\rho w_{l+1,j,k} & \text{si } 2 \le l < L \end{cases} \tag{8.10}$$

[3] L'erreur est une fonction différentiable des poids.
[4] ou algorithme de descente stochastique

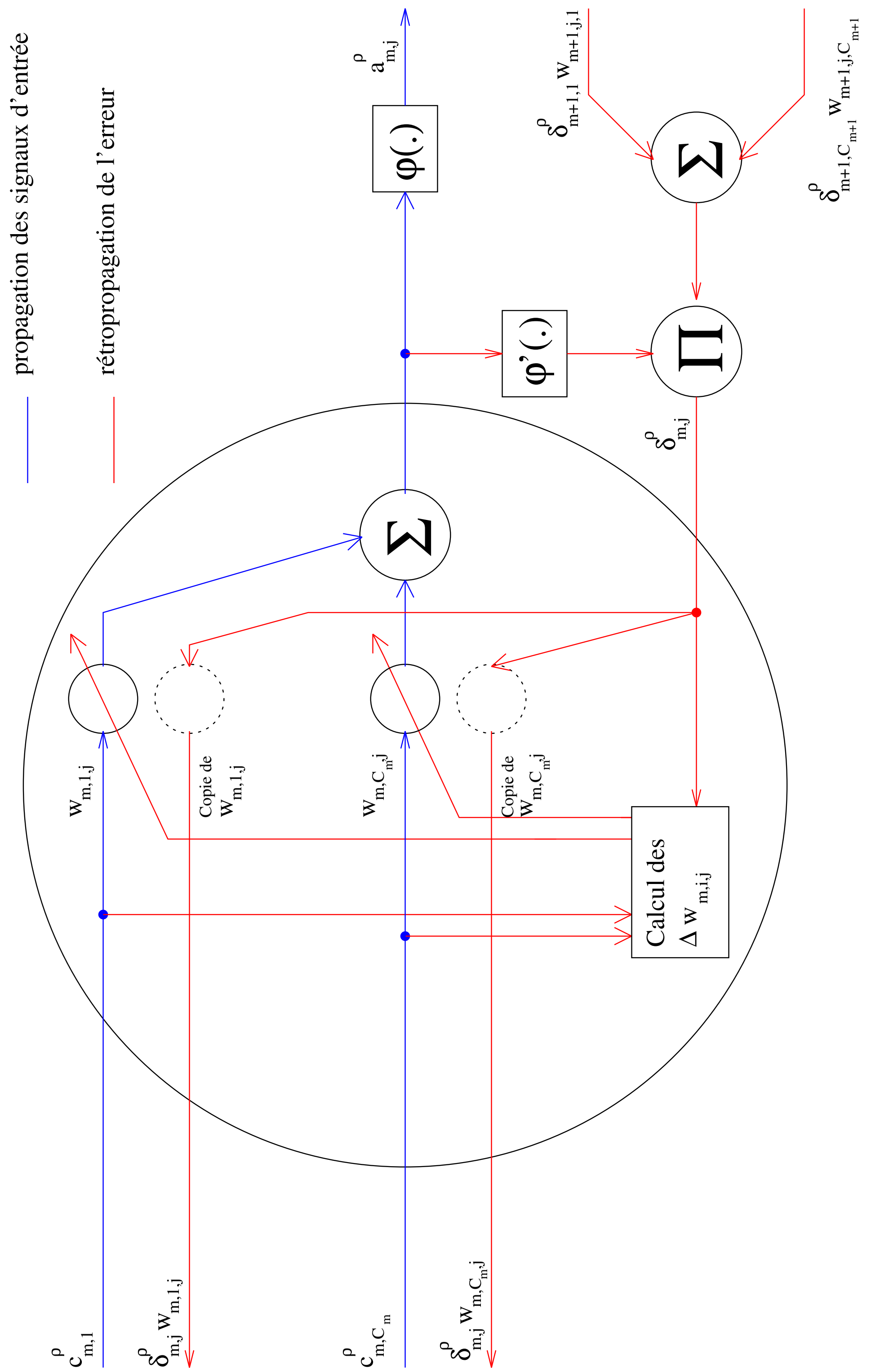


Figure 8.1: Détail de la propagation du signal d'entrée et de la rétropropagation de l'erreur dans un neurone j de la couche cachée m d'un *multilayer perceptron*.

8.2 L'algorithme de rétropropagation non linéaire

L'algorithme de *backpropagation*, présenté dans le paragraphe précédent, nécessite le calcul de la dérivée de la fonction d'activation. Triviale lors de simulations, cette opération complique les implantations *on chip* de l'algorithme.

L'algorithme de rétropropagation non linéaire (*non-linear backpropagation*), proposé par John Hertz [25], utilise la même fonction d'activation lors de la propagation des signaux d'entrée et de la rétropropagation des erreurs. L'algorithme de Hertz se révèle spécialement intéressant lors de réalisations matérielles de systèmes neuromimétiques. D'autre part, lors d'un précédent projet [5], nous avons constaté que les performances de la rétropropagation non linéaire se révélaient parfois nettement supérieures à celles de l'algorithme de *backpropagation* standard. Il nous semble par conséquent approprié d'appliquer ce procédé d'apprentissage (dédié aux *perceptrons* multicouches) à notre problème de reconnaissance d'écriture. Les étapes suivantes constituent l'algorithme :

1. **Initialisation.**
   Les poids et les biais sont initialisés aléatoirement.

2. **Présentation d'un motif et propagation des signaux.**
   La propagation des signaux s'effectue comme dans l'algorithme de *backpropagation*.

3. **Calcul et rétropropagation de l'erreur.**
   L'erreur $j^{\text{ème}}$ neurone de la couche de sortie est déterminée par

$$\epsilon_j^\rho = t_j^\rho - O_j^\rho \tag{8.11}$$

   Nous définissons le terme de *backward activation* du neurone j de la couche m, lors de la présentation du motif ρ, par :

$$\delta_{m,j}^\rho = \begin{cases} \varphi\left(\sum_{i=1}^{N_{m-1}} w_{m,i,j} \cdot a_{m-1,i}^\rho + \frac{\gamma_j}{\alpha_j} \cdot \epsilon_j^\rho \right) & \text{si } m = L \\ \varphi\left(\sum_{i=1}^{N_{m-1}} w_{m,i,j} \cdot a_{m-1,i}^\rho + \right. & \\ \left. \qquad \frac{\gamma_j}{\alpha_j} \cdot \sum_{k=1}^{N_{m+1}} \frac{\alpha_k}{\gamma_k} \left(\delta_{m+1,k}^\rho - a_{m+1,k}^\rho \right) \cdot w_{m+1,j,k} \right) & \text{si } 2 \leq m < L \end{cases} \tag{8.12}$$

   où α_j et γ_j remplacent le coefficient η de l'algorithme standard[5]. Il est cependant nécessaire que le terme γ_j / α_j soit petit. Les poids sont modifiés selon l'équation (8.7), avec

$$\Delta w_{m,i,j} = \alpha_j \cdot \left(\delta_{m,j}^\rho - a_{m,j}^\rho \right) \cdot a_{m-1,i}^\rho \tag{8.13}$$

   pour l'algorithme *online* et

$$\Delta w_{m,i,j} = \alpha_j \cdot \sum_\rho \left(\delta_{m,j}^\rho - a_{m,j}^\rho \right) \cdot a_{m-1,i}^\rho \tag{8.14}$$

   pour l'algorithme *offline*.

[5] Dans ses calculs, John Hertz utilise des α_j et γ_j distincts pour chaque couche du réseau.

Lorsque $\alpha_j = \gamma_j$, nous obtenons un algorithme particulier :

$$\delta^{\rho}_{m,j} = \begin{cases} \varphi\left(\sum_{i=1}^{N_{m-1}} w_{m,i,j} \cdot a^{\rho}_{m-1,i} + \epsilon^{\rho}_{j}\right) & \text{si } m = L \\ \varphi\left(\sum_{i=1}^{N_{m-1}} w_{m,i,j} \cdot a^{\rho}_{m-1,i} + \right. \\ \qquad \left. \sum_{k=1}^{N_{m+1}} \left(\delta^{\rho}_{m+1,k} - a^{\rho}_{m+1,k}\right) \cdot w_{m+1,j,k}\right) & \text{si } 2 \leq m < L \end{cases} \tag{8.15}$$

Les équations (8.13) et (8.14) s'écrivent alors

$$\Delta w_{m,i,j} = \gamma_j \cdot \left(\delta^{\rho}_{m,j} - a^{\rho}_{m,j}\right) \cdot a^{\rho}_{m-1,i} \tag{8.16}$$

et

$$\Delta w_{m,i,j} = \gamma_j \cdot \sum_{\rho} \left(\delta^{\rho}_{m,j} - a^{\rho}_{m,j}\right) \cdot a^{\rho}_{m-1,i} \tag{8.17}$$

Nous avons implanté cette version de l'algorithme dans *SNNS* [23].

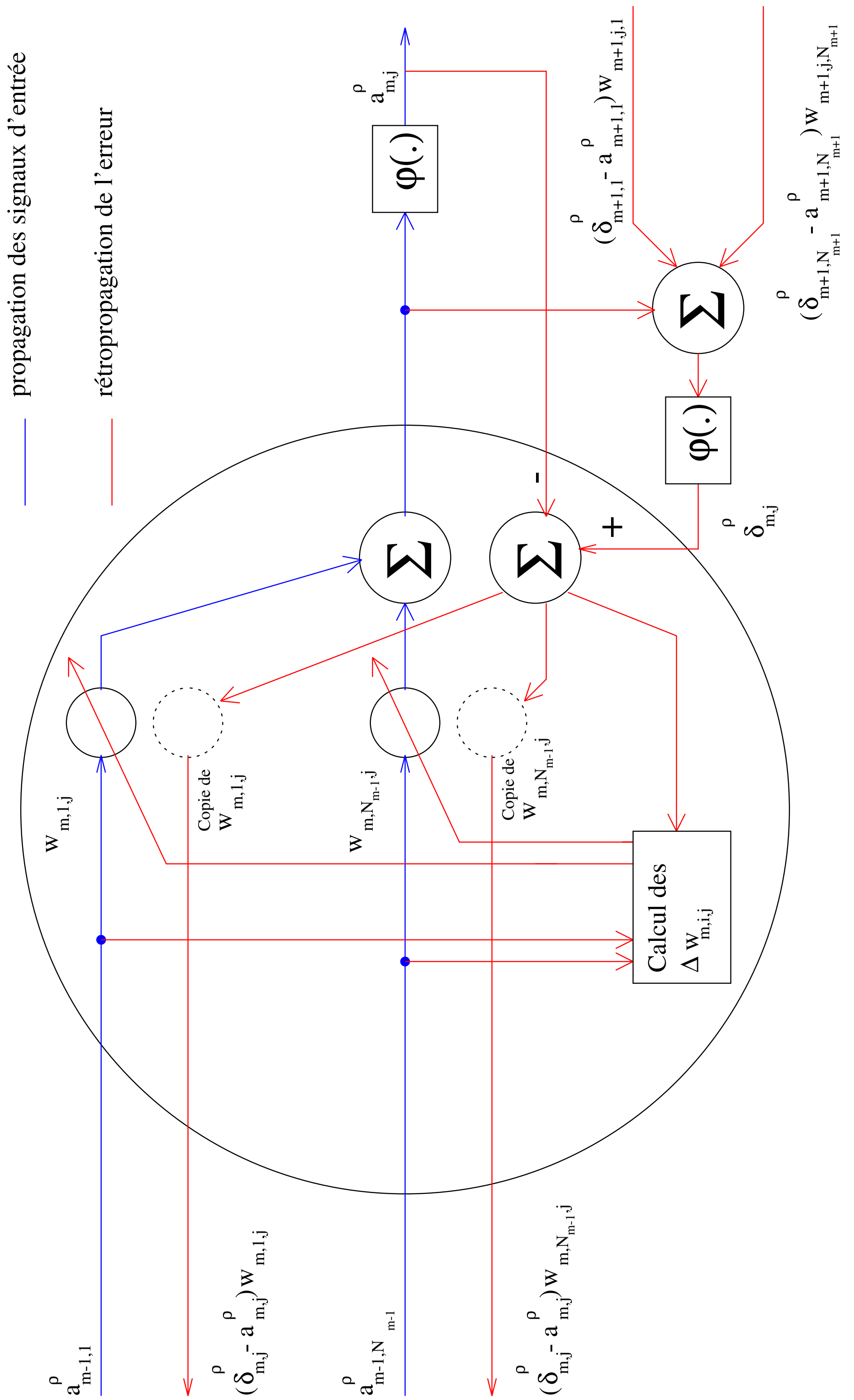


Figure 8.2: Détail de la propagation du signal d'entrée et de la rétropropagation de l'erreur dans un neurone j de la couche cachée m d'un *multilayer perceptron* (algorithme de rétropropagation non linéaire)

Chapitre 9

Eléments de la théorie de la généralisation

L'application des algorithmes d'apprentissage proposés lors du précédent chapitre comporte deux difficultés :

- **Réglage des différents paramètres intervenant lors de l'entraînement**. Généralement, quelques simulations permettent de déterminer des paramètres adéquats.

- **Choix de la topologie du réseau**. Les algorithmes de *backpropagation* et *non-linear backpropagation* ne fournissent aucune indication sur le nombre de neurones nécessaires sur la couche cachée d'un *perceptron* multicouche ou sur l'ordre des connexions d'un *high order perceptron*.

Montrons à l'aide d'un exemple que le choix de la topologie du réseau s'avère crucial.

Considérons le problème de classification illustré par la figure 9.1 (a). Nous constituons deux bases de données composées de motifs des classes A et B linéairement séparables :

- Les données d'apprentissage sont utilisées par l'algorithme de modification des poids.

- Les données de validation (distinctes de celles d'apprentissage) permettent l'appréciation des performances du réseau. Ce dernier se révèle en effet capable de traiter des motifs n'intervenant pas lors de l'entraînement. Cette importante propriété, appelée généralisation, explique l'engouement pour les systèmes neuromimétiques.

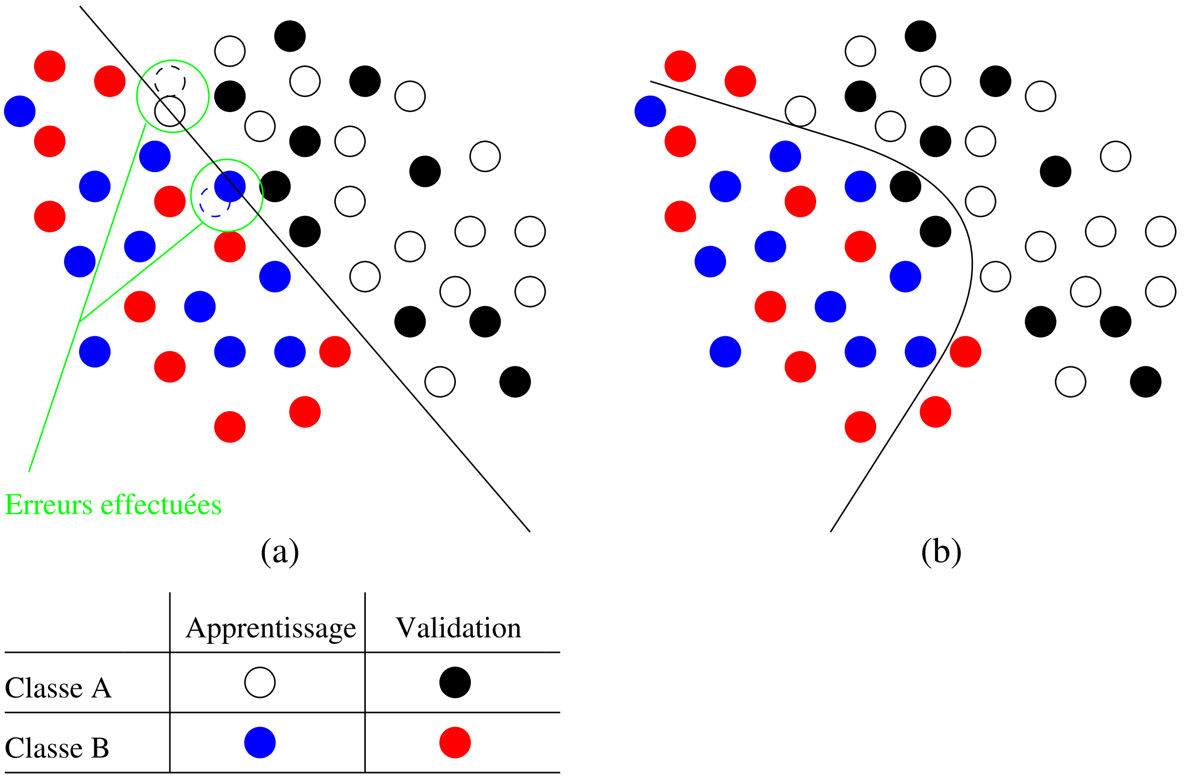


Figure 9.1: Un problème de classification. Nous commettons deux erreurs, repérées par les cercles verts, lors de la conception de la base d'apprentissage. Les cercles en pointillé indiquent les positions exactes de ces deux motifs.
(a) Séparatrice déterminée par l'*Adaline*. (b) Séparatrice obtenue en entraînant un *perceptron* multicouche.

Nous commettons cependant deux légères erreurs lors de nos mesures[1], transformant ainsi la tâche en un problème non linéairement séparable. Résolvons ce dernier à l'aide d'un *Adaline*[2] et d'un *perceptron* multicouche. A la fin de chaque cycle d'entraînement, nous calculons les sommes des erreurs au carré des données d'apprentissage et de validation et les reportons sur un graphe (figures 9.2 (a) et (b)).

L'*Adaline* est incapable d'apprendre parfaitement les données d'entraînement. Néanmoins, ses capacités de généralisation se révèlent excellentes. Il classifie correctement tous les motifs de validation (figure 9.1 (a)).

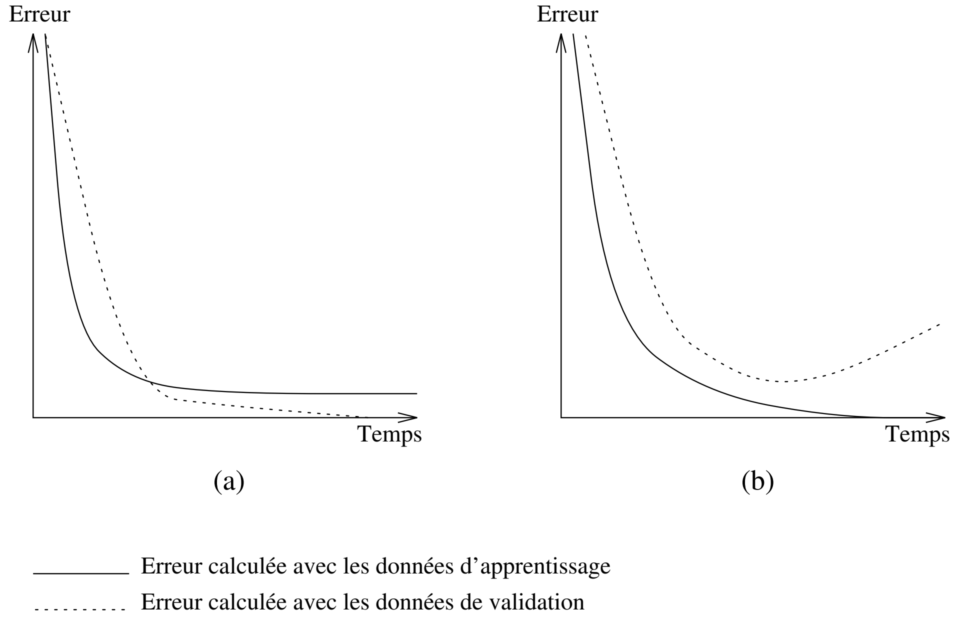


Figure 9.2: Surentraînement

Le *perceptron* multicouche reconnaît la totalité des données d'apprentissage. Observons cependant l'évolution de l'erreur de validation (figure 9.2 (b)). Elle diminue lors des premiers cycles, puis augmente soudainement. Ce phénomène, appelé surentraînement (*overtraining*), est lié à l'architecture du réseau. Si ce dernier possède un trop grand nombre de degrés de liberté[3], nous remarquons qu'il les utilise afin de s'adapter au bruit des données (figure 9.1 (b)). Certains chercheurs ont constaté qu'afin d'obtenir une bonne généralisation, il fallait utiliser le plus petit système capable de mémoriser les données d'apprentissage.

Il est cependant très difficile de déterminer la topologie optimale d'un réseau. Trois catégories d'algorithmes permettent de modifier l'architecture du réseau durant le processus d'apprentissage :

1. Les algorithmes génétiques fournissent d'excellents résultats dans des problèmes d'optimisation; la principale difficulté consiste à coder le réseau de neurones afin de pouvoir

[1] Il est généralement impossible d'effectuer des mesures exactes lors d'applications réelles. Supposons par exemple que nous fassions analyser deux bouteilles d'un vin donné. Les proportions des différents constituants ne seront pas exactement identiques lors des deux études.

[2] Le problème n'étant pas linéairement séparable, la règle du *Perceptron* modifierait indéfiniment les poids (paragraphe 5.1).

[3] C'est-à-dire un nombre trop important de neurones et de connexions par rapport au problème considéré.

aisément appliquer les opérateurs génétiques (une solution est proposée dans [20]). Notons cependant que les algorithmes génétiques nécessitent une mémoire et un temps de calcul considérables et ne trouvent ainsi d'application que sur des systèmes parallèles.

2. Les méthodes constructives [25] proposent des règles de croissance pour un réseau. Le principe consiste à débuter le processus avec un réseau de petite taille et à lui ajouter des éléments jusqu'à ce qu'il résolve le problème considéré.

3. Les algorithmes d'élagage (dont [39] propose une bonne introduction) préconisent de débuter l'apprentissage avec un réseau de taille quelconque capable d'apprendre la tâche désirée, puis d'enlever les neurones et connexions inutiles afin d'obtenir le réseau de taille optimale. Abondamment utilisés dans le cadre de ce projet, le chapitre 10 leur sera consacré.

Chapitre 10

Algorithmes d'élagage

10.1 Principes des algorithmes d'élagage

Supposons que nous connaissions la configuration optimale d'un réseau destiné à résoudre un problème donné. Il semble logique qu'en lui ajoutant des connexions et des neurones, il soit encore capable d'apprendre sa tâche. Le principe des algorithmes d'élagage se fonde sur cette réflexion. Nous débutons l'apprentissage avec un réseau de taille quelconque, capable de classifier les diverses données d'entraînement. Un algorithme d'élagage propose alors des critères afin de déterminer quels éléments (connexions ou neurones) sont inutiles.

Théoriquement, les poids synaptiques superflus devraient tendre vers zéro au cours de l'apprentissage. Nous n'aurions alors plus qu'à les élaguer afin d'obtenir le réseau optimal.

Néanmoins, ceci ne se réalise pas en pratique. Si le réseau comporte un trop grand nombre de degrés de liberté, le phénomène de surentraînement présenté au chapitre 9 risque de survenir[1]. Le procédé d'élagage suggéré lors de nos réflexions théoriques ne fonctionne évidemment pas. Les chercheurs ont proposé diverses catégories d'algorithmes d'élagage. Dans ce projet, nous n'utilisons que la classe d'algorithmes d'estimation de la sensibilité. Leur principe consiste à mesurer l'augmentation de l'erreur lorsque nous enlevons un élément (connexion ou neurone).

Il est cependant inconcevable de calculer l'erreur lorsqu'un élément donné est présent dans le réseau, puis lorsqu'il est élagué. Ce procédé suppose, pour chaque neurone ou connexion, deux présentations des exemples d'apprentissage et nécessite un temps de calcul considérable. Il s'agit par conséquent de déterminer d'autres algorithmes mesurant la sensibilité de l'erreur à l'élagage d'un élément. Outre cette difficulté, nous sommes confronté à divers problèmes :

- **Combien d'éléments enlever lors d'une phase d'élagage ?** La solution la plus

[1] Le surentraînement n'apparaît cependant pas forcément; lors de précédents projets [4] [5], nous avons travaillé avec la base de données *Wine* [1]. Il s'agit d'un problème linéairement séparable. Nous avons cependant entraîné un *multilayer perceptron* comportant dix neurones sur la couche cachée. Après vingt cycles d'apprentissage, les poids ne subissent plus de modifications. Environ 94% des données sont apprises correctement avec l'algorithme de *backpropagation*. Il est possible d'apprendre correctement la totalité des données avec l'algorithme *non-linear backpropagation* [27]. L'erreur calculée sur la base de validation décroît lors des premiers cycles, puis reste constante, les poids n'étant plus modifiés.

simple consiste à enlever un certain pourcentage d'éléments. Lors des premières phases d'élagage, il semble raisonnable d'éliminer un nombre important de connexions ou de neurones. Cependant, afin de déterminer le réseau optimal, il paraît judicieux de restreindre le nombre d'éléments élagués lors des phases ultérieures. Certains algorithmes proposent toutefois des critères plus élaborés [34].

- **Quand effectuer l'élagage ?** Enlever des neurones ou des connexions engendre fréquemment une augmentation de l'erreur. Il est donc indispensable d'effectuer quelques cycles d'apprentissage entre deux élagages successifs. Une solution consiste dès lors à ne permettre un élagage que lorsque l'erreur est inférieure à un seuil donné. Nous déterminons ce dernier de la manière suivante : nous effectuons un apprentissage avec l'algorithme de *backpropagation* jusqu'à ce que le pourcentage de motifs correctement appris nous satisfasse. Nous mémorisons alors l'erreur du réseau et obtenons ainsi notre seuil. Il s'agit en fait d'une première approximation; diverses expériences sont ensuite nécessaires afin d'obtenir de bons résultats.

  Des chercheurs ont proposé des critères plus sophistiqués. L'élagage s'avère indispensable lorsqu'apparaît le phénomène de surentraînement. Prechelt [34] suggère par conséquent l'utilisation de deux jeux de données lors de l'entraînement. Le premier (jeu d'apprentissage) est utilisé par l'algorithme d'adaptation des poids. Le second (jeu de validation) permet de détecter et de quantifier le surentraînement. Soit $E_{va(t)}$ l'erreur calculée avec le jeu de validation. Soit encore

  $$E_{opt(t)} = \min_{1 \leq t' \leq t} E_{va(t')} \tag{10.1}$$

  Prechelt définit alors la perte de généralisation comme l'augmentation relative de l'erreur de validation par rapport à l'erreur optimale :

  $$\mathrm{GL(t)} = 100 \cdot \left(\frac{E_{va(t)}}{E_{opt(t)}} - 1 \right) \tag{10.2}$$

  L'algorithme *Lambda-Prune* proposé par Prechelt élimine un nombre de connexions proportionnel à la perte de généralisation dès que survient le surentraînement. L'utilisation de *Lambda-Prune* nécessite cependant le réglage d'un nombre important de paramètres, dont de petites variations provoquent des performances très différentes [4].

- **Quand terminer le processus ?** Nous proposons deux réponses à cette question :

  - La solution la plus simple consiste à entraîner et élaguer le réseau durant un nombre de cycles donné.
  - L'élagage d'une connexion ou d'un neurone provoque une augmentation de l'erreur. Si l'élément détruit ne s'avère pas primordial, il est possible de compenser l'accroissement de l'erreur en quelques cycles d'apprentissage. Sinon, il faut évidemment terminer le processus[2].

L'élagage d'un réseau est un problème complexe demandant plusieurs simulations afin de régler les divers paramètres. Nous présentons encore dans ce chapitre les algorithmes que nous utiliserons lors de nos simulations. Tous sont implantés dans *SNNS* [23] ou *Sesame*.

[2]Nous conseillons de sauvegarder le réseau avant chaque étape d'élagage. Il est ainsi possible de recréer l'élément dont la destruction a engendré l'irrémédiable augmentation de l'erreur.

10.2 *Smallest weights*

Une des heuristiques les plus simples consiste à enlever les connexions dont les poids sont les plus petits en valeur absolue.

Supposons que nous élaguions la connexion $w_{l,i,j}$. Afin de minimiser l'accroissement de l'erreur, Sietsma [26] suggère d'ajouter la contribution moyenne

$$\frac{1}{T}\sum_{t=1}^{T} w_{l,i,j} \cdot c_{l,i}^{t} \tag{10.3}$$

au biais du neurone j de la couche l (T désigne le nombre d'itérations de l'algorithme de *backpropagation online* effectuées au moment de l'élagage).

10.3 *Smallest variance*

G. Thimm s'est inspiré de l'article de Sietsma et Dow [26] afin de concevoir cet algorithme. Nous calculons la variance $\sigma_{l,i,j}$ de la contribution de chaque connexion :

$$\sigma_{l,i,j} = \mathrm{Var}(c_{l,i}^{\rho} \cdot w_{l,i,j}) \tag{10.4}$$

Nous élaguons ensuite les poids synaptiques auxquels sont associées les plus petites valeurs $\sigma_{l,i,j}$.

10.4 *Autoprune*

Le principe de cet algorithme, proposé par Finnoff [50], consiste à calculer un test statistique indiquant l'importance de chaque connexion. Ce test peut s'effectuer à tout moment lors de l'apprentissage.

DÉFINITION 10.1 (TEST DE FINNOFF)
L'importance d'une connexion est définie par le test statistique T mesurant la probabilité que la valeur du poids tende vers zéro lors de l'apprentissage :

$$T(w_{l,i,j}) = log \frac{\left| \sum_{\rho=1}^{n} \left(w_{l,i,j} - \eta \cdot \frac{\partial E^{\rho}}{\partial w_{l,i,j}} \right) \right|}{\eta \cdot \sqrt{\sum_{\rho=1}^{n} \left(\frac{\partial E^{\rho}}{\partial w_{l,i,j}} - \overline{\left(\frac{\partial E}{\partial w_{l,i,j}} \right)} \right)^{2}}} \tag{10.5}$$

où $\overline{(\cdot)}$ dénote la moyenne sur tous les exemples d'apprentissage.

Comme nous utilisons l'algorithme de rétropropagation du gradient, nous pouvons effectuer quelques simplifications [4] :

$$\Delta w_{l,i,j} = -\eta \frac{\partial E^{\rho}}{\partial w_{l,i,j}} \tag{10.6}$$

d'où

$$-\frac{\partial E^\rho}{\partial w_{l,i,j}} = \frac{\Delta w_{l,i,j}}{\eta} = \delta^\rho_{l,j} \cdot c^\rho_{l,i} \tag{10.7}$$

Nous obtenons ainsi :

$$T(w_{l,i,j}) = \log \frac{\left| \sum_{\rho=1}^{n} \left(w_{l,i,j} + \eta \cdot \delta^\rho_{l,j} \cdot c^\rho_{l,i} \right) \right|}{\eta \sqrt{\sum_{\rho=1}^{n} \left(\overline{\delta_{l,j} \cdot c_{l,i}} - \delta^\rho_{l,j} \cdot c^\rho_{l,i} \right)^2}} \tag{10.8}$$

Cette formule demande cependant quelques modifications en vue d'une implantation efficace. Nous les présentons dans l'annexe B.

Finnoff conseille d'enlever 35% des connexions lors du premier élagage et 10% lors des étapes suivantes. Cette méthode ne spécifie malheureusement ni quand effectuer le premier élagage, ni combien de cycles d'apprentissage sont nécessaires entre deux phases d'élagage.

Diverses observations de la distribution des $T(w_{l,i,j})$ ont suggéré l'utilisation de sa moyenne afin de déterminer combien de connexions enlever [34]. Ces constatations suggèrent la règle suivante :

> A chaque phase d'élagage, enlever toutes les connexions satisfaisant $T(w_{l,i,j}) < \lambda \cdot \mu_T$, où $0 \leq \lambda \leq 1$

avec

$$\mu_T = \frac{\sum_k T(w_{l,i,j})}{k} \tag{10.9}$$

où k est le nombre de connexions du réseau.

Nous avons programmé cet algorithme dans SNNS [23] lors d'un précédent projet [4] et invitons le lecteur intéressé aux détails de l'implantation à s'y rapporter.

10.5 *Skeletonization*

Mozer et Smolensky [32] ont proposé une méthode permettant de déterminer les neurones significatifs d'un réseau. Le principe de l'algorithme consiste à entraîner le système jusqu'à la satisfaction d'un critère de performance, puis à calculer l'importance de chacun des neurones.

Soient $E_{\text{avec neurone l,j}}$ l'erreur calculée sur la base d'apprentissage lorsque le neurone j de la couche l est présent et $E_{\text{sans neurone l,j}}$ l'erreur obtenue lorsque ce dernier est enlevé. $\rho_{l,j}$ mesure l'importance du neurone j de la couche l :

$$\rho_{l,j} = E_{\text{sans neurone l,j}} - E_{\text{avec neurone l,j}} \tag{10.10}$$

Le calcul de $\rho_{l,j}$ nécessite cependant la présentation de la totalité des motifs d'apprentissage. Afin de remédier à cet inconvénient, Mozer et Smolensky proposent une alternative pour

$\rho_{l,j}$. A chaque neurone est associé un coefficient $\alpha_{l,j}$, appelé *attentional strength*, contrôlant son activité (figure 10.1) :

$$a_{l,j}^{\rho} = \alpha_{l,j} \cdot \varphi_{l,j} \left(\beta_{l,j} \cdot h_{l,j}^{\rho} \right) \tag{10.11}$$

Lorsque $\alpha_{l,j} = 1$, le neurone se comporte de manière conventionnelle. Cependant, lorsque $\alpha_{l,j} = 0$, l'activation $a_{l,j}^{\rho}$ est égale à zéro. Nous pouvons maintenant calculer l'importance d'un neurone en fonction de $\alpha_{l,j}$:

$$\begin{aligned} \rho_{l,j} &= E_{\alpha_{i,j}=0} - E_{\alpha_{i,j}=1} \\ &= -\int_0^1 \frac{\partial E}{\partial \alpha_{l,j}} d\alpha_{l,j} \end{aligned} \tag{10.12}$$

Mozer et Smolensky proposent d'utiliser le critère alternatif

$$\hat{\rho}_{l,j} = -\left.\frac{\partial E}{\partial \alpha_{l,j}}\right|_{\alpha_{l,j}=1 \quad \forall l \in [1..L] \quad \forall j \in [1..N_l]} \tag{10.13}$$

afin de déterminer l'importance de chacun des neurones [32].

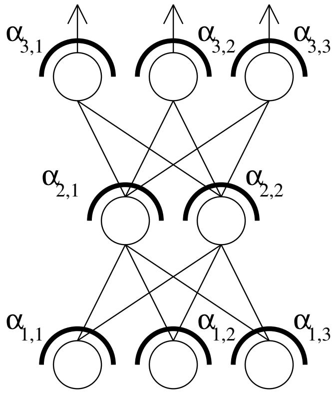


Figure 10.1: *Perceptron* multicouche avec les coefficients $\alpha_{l,j}$

10.6 *Optimal Brain Damage (OBD)*

Soit $\boldsymbol{w}$ le vecteur obtenu par concaténation de tous les poids du réseau. Supposons que certains coefficients synaptiques subissent une modification δw_i. Nous souhaitons calculer la perturbation de l'erreur, notée δE, ainsi engendrée ($\boldsymbol{\delta w}$ dénote le vecteur contenant les perturbations δw_i associées à chacun des poids w_i). Le Cun [44] propose d'effectuer une approximation de l'erreur à l'aide d'un développement en série de Taylor :

$$\begin{aligned} \delta E = &\sum_i \frac{\partial E}{\partial w_i} \delta w_i + \frac{1}{2} \sum_i \frac{\partial^2 E}{\partial w_i^2} (\delta w_i)^2 \\ &+ \frac{1}{2} \sum_{i \neq j} \frac{\partial^2 E}{\partial w_i w_j} (\delta w_i)^2 + O(\|\boldsymbol{w}\|^2) \end{aligned} \tag{10.14}$$

Il suggère ensuite diverses simplifications de l'équation 10.14. En supposant que nous ayons entraîné le système durant quelques cycles et atteint un minimum local, nous pouvons éliminer la première somme de l'équation. En effet, dans une telle situation,

$$\sum_i \frac{\partial E}{\partial w_i} \delta w_i = 0 \tag{10.15}$$

Le Cun considère de plus que tous les éléments $h_{ij,i\neq j}$ de la matrice Hessienne $H = (h_{ij}) = \frac{\partial^2 E}{\partial w_i w_j}$ sont négligeables[3]. La dernière somme de l'équation 10.14 s'annule ainsi. La perturbation de l'erreur s'écrit

$$\delta E = \frac{1}{2} \sum_i \frac{\partial^2 E}{\partial w_i^2} (\delta w_i)^2 \tag{10.16}$$

Finalement, en ne considérant que l'annulation d'un seul poids ($\boldsymbol{\delta w} = [0, \ldots, 0, -w_j, 0, \ldots, 0]^T$), nous obtenons

$$s_j = \frac{\partial^2 E}{\partial w_j^2} \cdot w_j^2 \tag{10.17}$$

L'équation 10.17 estime l'augmentation de l'erreur lorsque nous élaguons la connexion w_j.

[3] Hassibi et Stork suggèrent de conserver toute l'information contenue dans la matrice Hessienne [10]. *Optimal Brain Surgeon (OBS)*, leur algorithme d'élagage, nécessite cependant le calcul de H^{-1}. Nous considérons cette opération trop coûteuse et n'utiliserons par conséquent pas *OBS* lors de nos expériences.

L'algorithme *Optimal Brain Damage* [44] comporte les étapes suivantes :

1. Déterminer un réseau capable de résoudre le problème considéré.
2. Entraîner le réseau jusqu'à l'obtention d'une solution satisfaisante.
3. Calculer le coefficient s_j associé à chaque connexion w_j.
4. Elaguer les poids synaptiques dont les s_j sont les plus faibles.
5. Retourner au point 2 ou achever le processus.

Partie II

Prétraitement et extraction de caractéristiques

Chapitre 11

Introduction à la seconde partie

La figure 11.1 illustre l'architecture générale d'un système classifiant des chiffres segmentés. Divers algorithmes de prétraitement (chapitre 12) restreignent les infinies variations de l'écriture manuscrite, facilitant ainsi sa reconnaissance. Le problème crucial consiste alors à extraire des caractéristiques permettant la distinction des chiffres présentés au système. Les performances de ce dernier sont intimement liées à la qualité des informations ainsi recueillies [37].

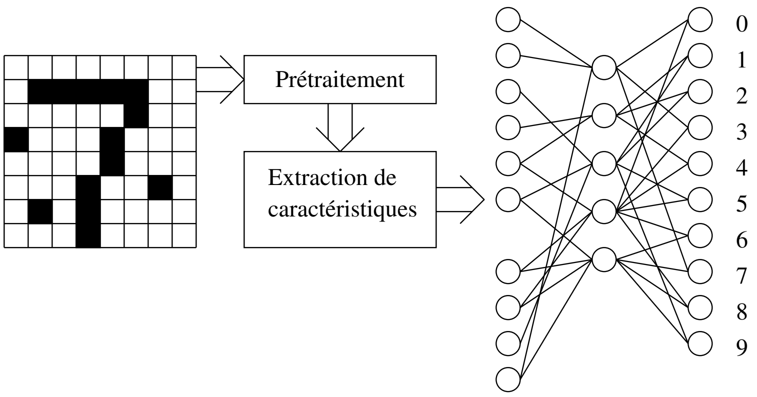


Figure 11.1: Schéma général du système de reconnaissance de chiffres manuscrits

Les algorithmes d'extraction de caractéristiques se répartissent en plusieurs catégories :

- *Template matching.*
  Chaque image est caractérisée par l'état de chacun de ses pixels (noir ou blanc). Une mesure de similarité entre des motifs de référence mémorisés dans le système et le chiffre inconnu permet la classification [42].

- **Développements en séries.**
  Plusieurs transformées, dont celle de Fourier, possèdent des applications en reconnaissance d'écriture [37] [42].

- **Distribution des points.**

Cette catégorie d'algorithmes englobe les méthodes extrayant des caractéristiques de la distribution statistique des pixels [8] [37]. Citons quelques techniques :

- *Zoning*. L'image est décomposée en de multiples régions se recouvrant quelquefois partiellement. La densité de pixels noirs dans chacune de ces régions constitue le vecteur de caractéristiques [36].
- Moments. Les moments des pixels noirs, calculés par rapport au centre de gravité de l'image, déterminent un ensemble de valeurs utilisées lors de la classification.
- *Crossing*. Cette méthode consiste à déterminer le nombre d'intersections entre le contour du chiffre et des droites de référence [43] [37].
- Projections et étude du contour. Nous présenterons ces deux méthodes aux paragraphes 13.1 et 13.3.

• **Analyse structurelle.**
Ces méthodes, tolérantes aux variations et distorsions de l'écriture, utilisent les propriétés géométriques et topologiques d'un caractère (position des boucles, intersections de segments, ...). L'extraction de telles caractéristiques se révèle cependant difficile et constitue un sujet de recherche [42].

• **Méthodes statistiques.**
L'analyse discriminante linéaire ou l'analyse en composantes principales connaissent des applications en reconnaissance d'écriture. Le chapitre 14 est consacré à cette dernière.

Les informations recueillies par l'algorithme d'extraction de caractéristiques sont finalement fournies au système neuromimétique réalisant la classification. L'apprentissage et la reconnaissance s'effectuent à partir des caractéristiques de chiffres appartenant à diverses écritures.

Chapitre 12

Prétraitement et normalisation

La base de données *NIST* propose des images binaires représentant des caractères segmentés. Nous associons une valeur booléenne à chaque pixel et décrivons une image binaire de N pixels sur N par une matrice carrée $I = (i_{mn})$:

$$i_{mn} = \begin{cases} 1 & \text{si le pixel correspondant est noir} \\ 0 & \text{sinon} \end{cases} \tag{12.1}$$

Divers procédés permettent alors de restreindre les multiples variations de l'écriture, simplifiant ainsi l'extraction de caractéristiques. Une bibliothèque d'algorithmes de prétraitement, dont nous ne présenterons pas les détails d'implantation, est disponible à l'IDIAP. Nous proposerons des références à des articles consacrés aux divers prétraitements ou offrant une intéressante bibliographie. Afin d'illustrer les effets des divers prétraitements, nous les appliquerons aux quelques chiffres de la figure 12.1.

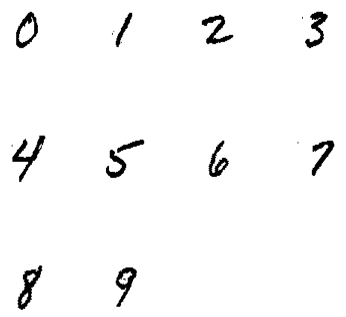
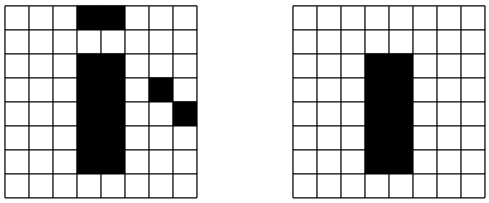


Figure 12.1: Quelques chiffres non prétraités

12.1 Imperfections liées à la qualité du document et au procédé de numérisation

Le traitement de documents anciens, de fax ou de chèques constitue l'une des multiples applications des systèmes *OCR*. La qualité du papier influe sur le processus de numérisation. Un palimpseste portera éventuellement les traces des multiples textes auxquels il a servi de support. Diverses salissures ou froissures entacheront peut-être le chèque ou le fax. Un *scanner* de piètre qualité accentuera encore ces défauts.

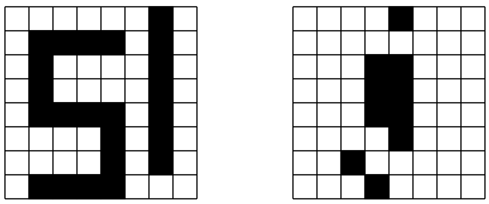
Toutes ces imperfections transparaîtront malheureusement sur le texte numérisé. Divers algorithmes permettent de les atténuer partiellement. Nous présentons brièvement les principes de la méthode proposée dans la bibliothèque de fonctions de l'IDIAP. Définissons une composante connexe d'une image binaire comme un groupe de pixels de même intensité (l'image représentée par la figure 12.2 (a) possède ainsi trois composantes connexes). Si une composante connexe comporte moins de θ pixels, nous l'effaçons. En appliquant cette méthode à l'image de la figure 12.2 (a) avec $\theta = 3$, nous obtenons ainsi l'image 12.2 (b).



(a) (b)

Figure 12.2: Elimination du bruit. (a) L'image numérisée est entachée de bruit. (b) Les composantes connexes comportant moins de trois pixels ont été éliminées.

L'utilisation de cet algorithme se révèle cependant délicate. Comment distinguer les pixels constituant le point d'un "i", un signe de ponctuation ou un accent, de ceux engendrés par les imperfections de la numérisation ? De plus, le réglage de θ s'avère crucial[1]. Considérons l'image 12.3 (a) représentant un "5". En fixant la valeur de θ à 7, nous éliminons la tache[2] à droite du caractère (figure 12.4 (a)). L'application de l'algorithme au caractère 12.3 (b) s'avère par contre catastrophique : la totalité de l'image est effacée (figure 12.4 (b)) !

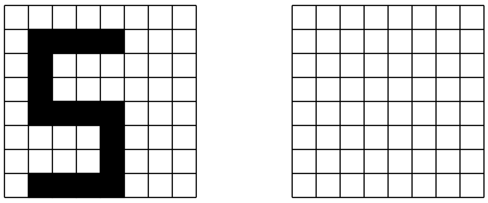


(a) (b)

Figure 12.3: Deux caractères bruités

[1] Il est évident que nous appliquons un prétraitement identique à chacun des caractères. Chaque image est ainsi traitée avec le même paramètre θ.

[2] Ce groupe de pixels ne constitue pas forcément une tache. Nous pouvons interpréter l'image 12.3 (a) comme une représentation du nombre 51.



(a) (b)

Figure 12.4: Destruction d'une image

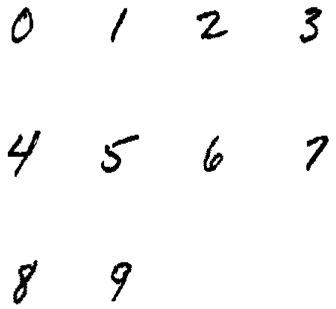


Figure 12.5: Effets de l'algorithme d'élimination de bruit. La tache à l'intérieur de la boucle du "9" a disparu.

12.2 Variabilité des représentations d'un caractère

Il existe une multitude de représentations d'un même caractère. Par exemple, nous écrivons tous les termes anglais en italique dans ce document. Uderzo et Goscinny utilisent une fonte distincte afin de transcrire les dialogues des personnages grecs, goths ou égyptiens des aventures d'Astérix. Nous pourrions encore citer énormément d'exemples : chaque écriture possède ses particularités.

La diversité des fontes offre un problème bien plus délicat. Il est en effet impossible de réaliser un prétraitement transcrivant tout caractère dans la fonte *Helvetica*[3]. Seule la diversité des écritures intervenant lors de l'apprentissage permet de résoudre cette difficulté.

Diverses méthodes permettent de corriger l'inclinaison des caractères. L'algorithme proposé dans la bibliothèque de fonctions de l'IDIAP annule les moments de second ordre. [42] et [43] abordent ce sujet et proposent des références à plusieurs articles.

12.3 Normalisation de la taille

La taille des caractères constituant le corps du texte, les titres ou les notes de bas de page d'un document varie énormément. Un algorithme de normalisation modifie chacun des caractères afin de standardiser sa hauteur [45] [42] [8].

[3]Une telle opération nécessiterait la reconnaissance du caractère !

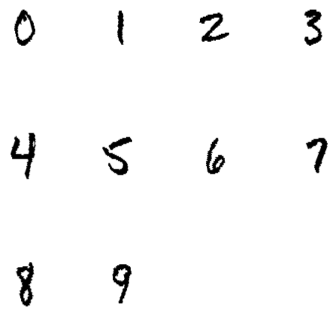


Figure 12.6: Effets de l'algorithme de correction de l'inclinaison

Les caractères de *NIST* sont contenus dans des images de 128 pixels sur 128. L'algorithme de normalisation de la bibliothèque de l'IDIAP ne modifie que leur taille, les inscrivant ainsi dans une zone de k pixels sur k de l'image, où k est le paramètre définissant la hauteur standard. L'algorithme fournit les coordonnées de l'angle inférieur gauche de cette zone.

12.4 Centrage du caractère

Certaines représentations d'un caractère varient lors d'une translation de l'image. Il s'avère par conséquent indispensable de centrer chacun des caractères[4].

DÉFINITION 12.1 (POINT CENTRAL D'UN CARACTÈRE [21])
Soit un caractère C représenté par une image binaire $I = (i_{mn})$. Son point central est dénoté par $I_c(x, y)$, avec

$$x = \frac{\sum_{k=1}^{N} \sum_{j=1}^{N} i_{kj} \cdot j}{\sum_{k=1}^{N} \sum_{j=1}^{N} i_{kj}} \tag{12.2}$$

$$y = \frac{\sum_{k=1}^{N} \sum_{m=1}^{N} i_{mk} \cdot m}{\sum_{k=1}^{N} \sum_{j=1}^{N} i_{kj}} \tag{12.3}$$

Soit $\Delta_x = x - N/2$ et $\Delta_y = y - N/2$. Afin de centrer le caractère, il suffit d'appliquer les transformations suivantes (les distances de translations sont respectivement de $|\Delta_x|$ et $|\Delta_y|$ pixels) :

$$\text{Si } \Delta_x \begin{cases} > 0 & \text{l'image subit une translation vers la gauche} \\ < 0 & \text{l'image subit une translation vers la droite} \\ = 0 & \text{l'image ne subit aucune translation} \end{cases} \tag{12.4}$$

$$\text{Si } \Delta_y \begin{cases} > 0 & \text{l'image subit une translation vers le haut} \\ < 0 & \text{l'image subit une translation vers le bas} \\ = 0 & \text{l'image ne subit aucune translation} \end{cases} \tag{12.5}$$

12.5 Lissage

Le lissage constitue l'ultime étape de prétraitement. Réalisé à l'aide d'un masque gaussien, il permet la transformation d'une image binaire en une image en niveaux de gris, adoucissant ainsi les contours et atténuant le bruit résiduel [15] (figure 12.8).

[4]Nous avons implanté cet algorithme. Il n'était pas proposé par la bibliothèque de fonctions de l'IDIAP.

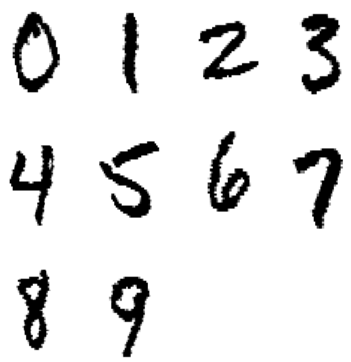


Figure 12.7: Effets des algorithmes de normalisation et de centrage

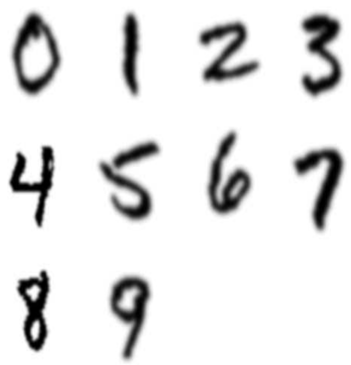


Figure 12.8: Effets de l'algorithme de lissage

Chapitre 13

Extraction de caractéristiques

13.1 L'approche *VH2D*

L'approche *VH2D* [21] (*Vertical-Horizontal-2Diagonal*), proposée par Cheng et Xia, consiste à projeter chaque caractère sur l'abscisse, l'ordonnée, ainsi que sur les diagonales 45° et 135°. Les projections s'effectuent en calculant la somme des valeurs des pixels i_{mn} selon une direction donnée. Cet algorithme, aisément implantable dans un système parallèle, a permis à Cheng et Xia de réaliser un processeur dédié à la reconnaissance d'écriture et fournissant de bons résultats pour l'identification de caractères chinois. Dans leur article, Cheng et Xia proposent des définitions formelles des diverses projections :

Définition 13.1 (Projection verticale)
La projection verticale d'une image $I = (i_{mn})$ (de dimensions $N \times N$) représentant un caractère C est dénotée par $\boldsymbol{P}^v = [p_1^v, p_2^v, \ldots, p_N^v]$, où

$$p_n^v = \sum_{m=1}^{N} i_{mn} \tag{13.1}$$

Définition 13.2 (Projection horizontale)
La projection horizontale d'une image $I = (i_{mn})$ (de dimensions $N \times N$) représentant un caractère C est dénotée par $\boldsymbol{P}^h = [p_1^h, p_2^h, \ldots, p_N^h]$, où

$$p_m^h = \sum_{n=1}^{N} i_{mn} \tag{13.2}$$

DÉFINITION 13.3 (PROJECTION SUR LA DIAGONALE 45°)
La projection sur la diagonale 45° d'une image $I = (i_{mn})$ (de dimensions $N \times N$) représentant un caractère C est dénotée par $\boldsymbol{P}^{d1} = [p_1^{d1}, p_2^{d1}, \ldots, p_{2N-1}^{d1}]$, où

$$p_m^{d1} = \begin{cases} \displaystyle\sum_{l=N-m+1}^{N} \sum_{k=1}^{m} i_{lk} & 1 \leq m \leq N \text{ et } l = k + N - 1 \\ \displaystyle\sum_{l=1}^{2N-m} \sum_{k=m-N+1}^{N} i_{lk} & N+1 \leq m \leq 2N-1 \text{ et } l = k + N - 1 \end{cases} \tag{13.3}$$

DÉFINITION 13.4 (PROJECTION SUR LA DIAGONALE 135°)
La projection sur la diagonale 135° d'une image $I = (i_{mn})$ (de dimensions $N \times N$) représentant un caractère C est dénotée par $\boldsymbol{P}^{d2} = [p_1^{d2}, p_2^{d2}, \ldots, p_{2N-1}^{d2}]$, où

$$p_m^{d2} = \begin{cases} \displaystyle\sum_{l=1}^{m} \sum_{k=1}^{m} i_{lk} & 1 \leq m \leq N \text{ et } k = m - l + 1 \\ \displaystyle\sum_{l=m-N+1}^{N} \sum_{k=m-N+1}^{N} i_{lk} & N+1 \leq m \leq 2N-1 \text{ et } k = m - l + 1 \end{cases} \tag{13.4}$$

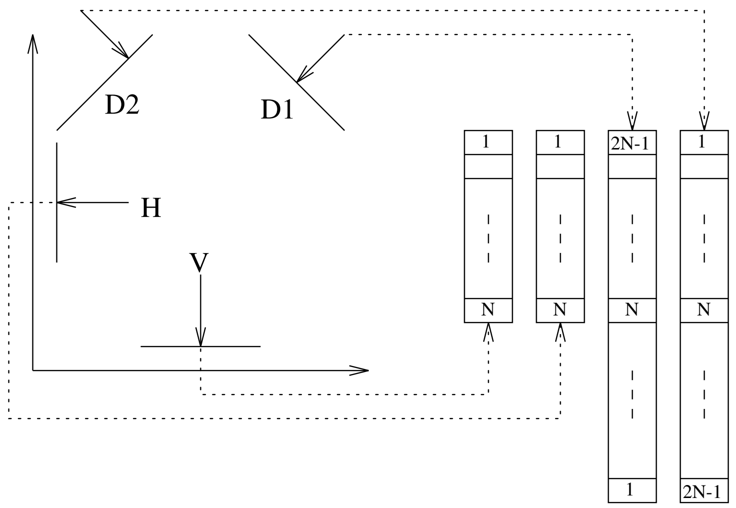


Figure 13.1: Vecteurs obtenus en effectuant les projections *VH2D*

Il est indispensable de centrer le caractère avant d'effectuer ces opérations[1]. L'algorithme d'extraction de caractéristiques comporte les étapes suivantes :

1. Déterminer le point central $I_c(x, y)$ de l'image.

2. Transformer l'image selon la méthode présentée au paragraphe 12.4.

[1]Une légère translation d'un caractère modifie considérablement le résultat des projections.

3. Calculer les quatre projections et combiner les résultats dans un vecteur de $6 \cdot N - 2$ composantes[2].

Cheng et Xia décrivent un algorithme massivement parallèle, dédié aux réalisations matérielles, permettant d'ajuster les projections après les avoir calculées. La seconde étape de l'algorithme d'extraction de caractéristiques est ainsi supprimée[3].

Théorème 2
Les projections VH2D d'une image centrée I'_{ij} s'obtiennent en ajustant les projections VH2D d'une image quelconque I_{ij} par rapport à son point central $I_c(x, y)$ [21].

Nous invitons le lecteur intéressé par la démonstration et l'algorithme proposé à consulter l'article de Cheng et Xia.

Remarquons encore que les projections $\boldsymbol{P}^{d1}$ et $\boldsymbol{P}^{d2}$ s'avèrent indispensables afin d'éviter des erreurs de classification. Considérons les écritures stylisées des figures 13.2 et 13.3. Seules $\boldsymbol{P}^{d1}$ et $\boldsymbol{P}^{d2}$ permettent de distinguer le "2" du "5".

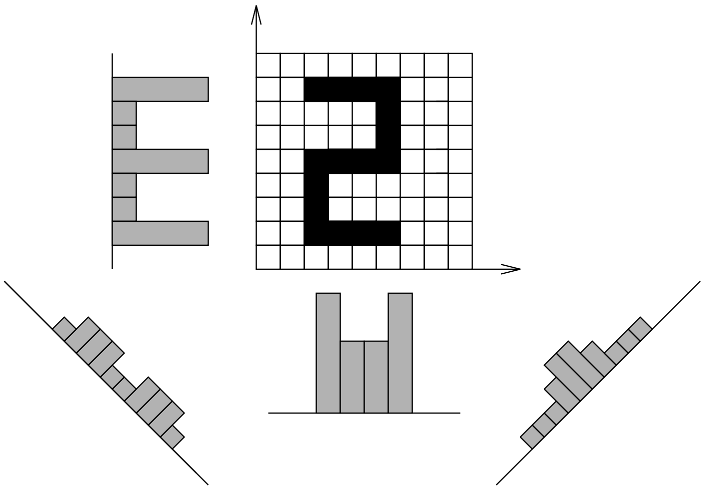


Figure 13.2: Projections du chiffre 2 stylisé

13.2 Réduction de la dimension des vecteurs obtenus avec l'approche *VH2D*

Nous souhaitons constituer les bases d'apprentissage et de validation de nos réseaux neuronaux à l'aide des vecteurs obtenus par la méthode *VH2D*. Supposons que nous travaillions avec des images de seize pixels sur seize. Chaque motif d'entraînement est par conséquent constitué de $16 \cdot 6 - 2 = 94$ éléments. Un *high order perceptron* de second ordre destiné à la classification de telles données comportera :

- 94 neurones sur la couche d'entrée;

[2] Les projections verticales et horizontales fournissent chacune N valeurs. $2 \cdot N - 1$ valeurs supplémentaires résultent de chaque projection diagonale.

[3] La complexité de l'algorithme parallèle d'extraction de caractéristiques est $O(N)$.

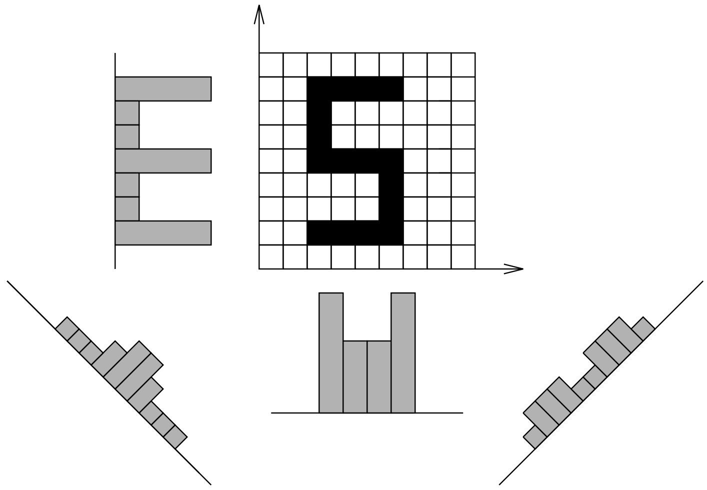


Figure 13.3: Projections du chiffre 5 stylisé

- 10 neurones sur la couche de sortie (nous ne travaillons qu'avec des chiffres dans ce projet; si nous souhaitions aussi traiter les lettres minuscules et majuscules, 62 neurones s'avéreraient indispensables sur la couche de sortie);
- $C_2^{94} \cdot 10 + 94 \cdot 10 = 44650$ connexions (et $C_2^{94} \cdot 62 + 94 \cdot 62 = 276830$ dans le cas général).

Il est aujourd'hui inconcevable d'entraîner des réseaux d'une telle taille avec *Sesame*[4]. Nous souhaitons par conséquent restreindre le nombre d'entrées de notre réseau.

13.2.1 Approximation de la projection par une somme de gaussiennes

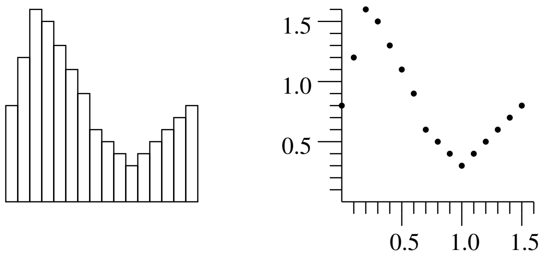
Nous utiliserons la projection verticale lors de nos réflexions. Les résultats présentés sont évidemment valables pour les trois autres projections. Montrons que nous pouvons représenter une projection comme une fonction discrète. Il suffit d'associer à chaque coefficient du vecteur $\boldsymbol{P}^v$ une abscisse (figure 13.4). Nous obtenons ainsi une liste de N points $P_1(x_1, p_1^v), P_2(x_2, p_2^v), \ldots, P_N(x_N, p_N^v)$. Nous attribuons la valeur 0 à x_1 et définissons récursivement la valeur de x_i $(0 < i \leq N)$:

$$x_i = x_{i-1} + 0.1 \tag{13.5}$$

Nous souhaitons effectuer une approximation de la fonction discrète ainsi obtenue à l'aide de la somme de m noyaux gaussiens $\phi_j(x)$:

$$\begin{aligned} f(x) &= \sum_{j=1}^{m} \phi_j(x) \\ &= \sum_{j=1}^{m} h_j \cdot e^{-\left(\frac{x-c_j}{\sigma_j}\right)^2} \end{aligned} \tag{13.6}$$

[4]Le temps de calcul ainsi que la mémoire nécessaires se révéleraient considérables.



(a) (b)

Figure 13.4: Représentation d'une projection par une fonction discrète.
(a) Le graphe de la projection. (b) La fonction discrète obtenue selon la méthode proposée.

où h_j, c_j et σ_j désignent respectivement la hauteur, la position relative à l'origine et la largeur de la gaussienne. Nous obtiendrions ainsi $3 \cdot m$ coefficients définissant une projection. Nous les utiliserions afin de constituer notre base d'apprentissage.

L'algorithme de Levenberg-Marquardt [48], implanté dans le logiciel *Gnuplot*, ajuste toute fonction paramétrique f d'une variable réelle à une liste de points. Une approximation initiale des divers paramètres de f se révèle toutefois indispensable à son fonctionnement. Considérons la fonction discrète de la figure 13.5. Nous souhaitons effectuer une approximation à l'aide de trois noyaux gaussiens. Le tableau 13.2 présente les coefficients déterminés à partir des estimations initiales (tableau 13.1). Les gaussiennes modélisent très bien les données. Etudions maintenant les conséquences d'une très légère perturbation de notre fonction discrète (figure 13.6). Exécuté avec les paramètres du tableau 13.1, l'algorithme de Levenberg-Marquardt se révèle incapable de calculer les coefficients (une erreur survient lors de l'exécution). Nous obtenons la solution de la figure 13.6 en plaçant initialement un noyau gaussien dans le voisinage des données bruitées (tableau 13.3). Le résultat s'avère très médiocre. Une faible perturbation des données engendre ainsi deux solutions très différentes. Considérons finalement la fonction discrète de la figure 13.7. Le choix du nombre de noyaux est critique : pour $m = 1$, nous obtenons la gaussienne représentée par la figure 13.7. Cependant, l'algorithme ne fonctionne plus si $m > 1$.

Paramètre	h_1	c_1	σ_1	h_2	c_2	σ_2	h_3	c_3	σ_3
Valeur	10	1	0.3	9	1.8	0.2	7	2.8	2

Tableau 13.1: Approximation initiale des paramètres des noyaux gaussiens pour la fonction discrète de la figure 13.5

Paramètre	h_1	c_1	σ_1	h_2	c_2	σ_2	h_3	c_3	σ_3
Valeur	13.07	1.02	0.13	11.09	1.76	0.16	7.48	2.69	0.41

Tableau 13.2: Paramètres des noyaux gaussiens déterminés par l'algorithme de Levenberg-Marquardt pour la fonction discrète de la figure 13.5

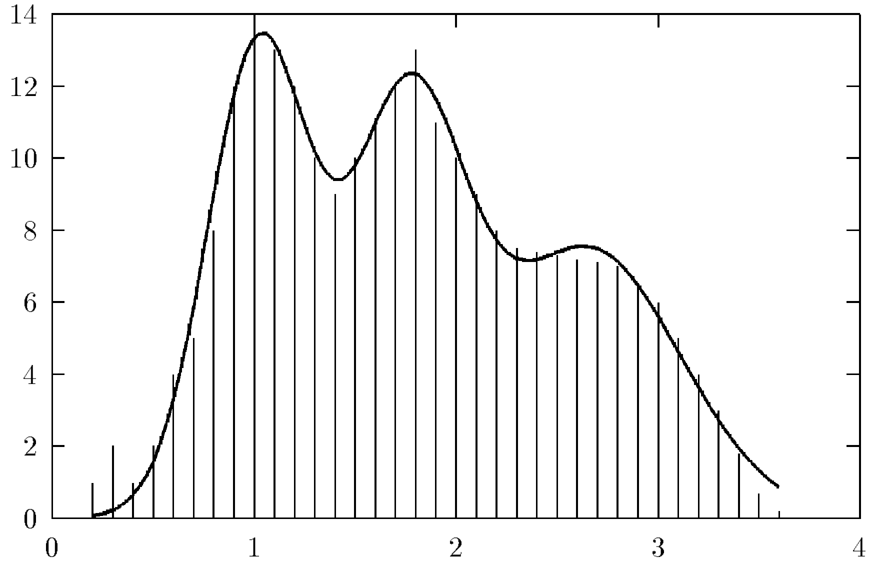


Figure 13.5: Une projection et son approximation par une somme de trois noyaux gaussiens

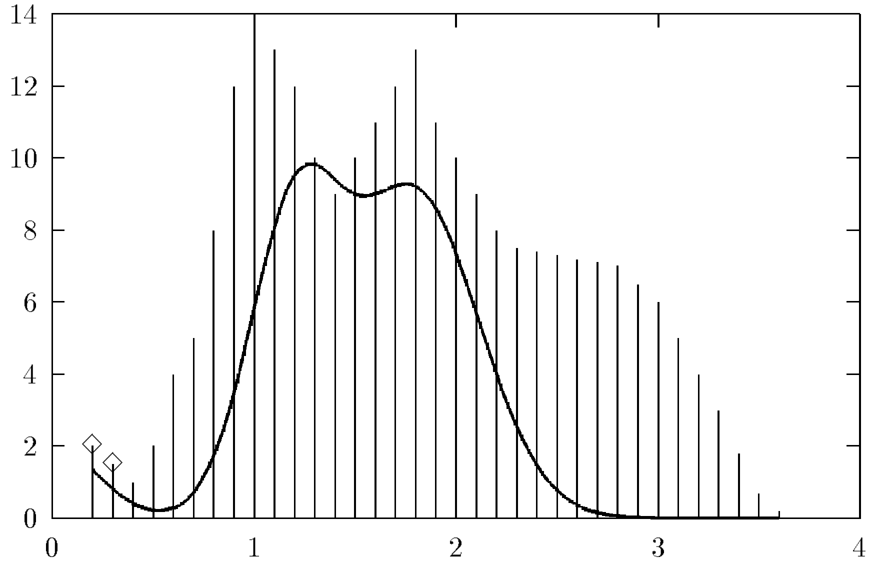


Figure 13.6: Où les paramètres des gaussiennes ne représentent plus la projection. Les losanges indiquent les valeurs modifiées par rapport à la fonction de la figure 13.5.

Paramètre	h_1	c_1	σ_1	h_2	c_2	σ_2	h_3	c_3	σ_3
Valeur	2	0	0.1	8	1.2	0.1	9	1.8	0.2

Tableau 13.3: Approximation initiale des paramètres des noyaux gaussiens pour la fonction discrète de la figure 13.6

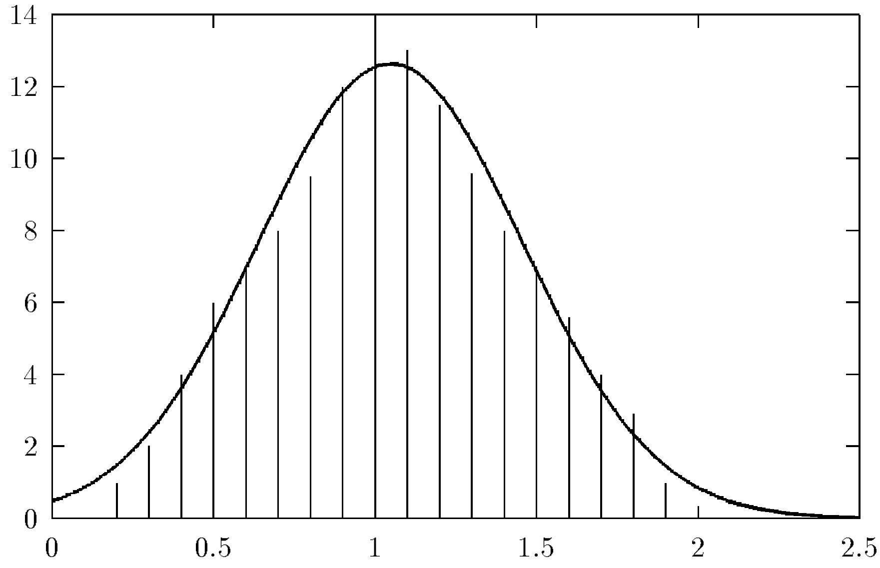


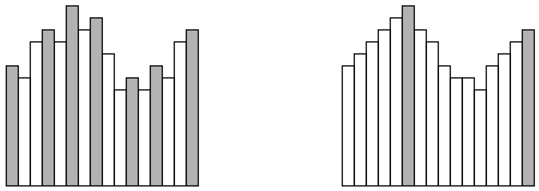
Figure 13.7: Approximation d'une fonction discrète par une gaussienne

Afin d'intégrer l'algorithme de Levenberg-Marquardt dans notre système de reconnaissance d'écriture, il est primordial d'automatiser l'approximation des coefficients des gaussiennes. L'écriture d'un programme réalisant cette tâche se révèle ardue. Nous souhaitons fournir les coordonnées des maxima de la fonction discrète comme paramètres de f. Malheureusement, la plupart des projections ressemblent à celle de la figure 13.8 (a). Dans cette situation, il est évident que deux gaussiennes fournissent un bon modèle de la fonction discrète. Nous disposons cependant de sept maxima. Lors des quelques expériences présentées dans ce paragraphe, nous avons insisté sur l'importance du choix de m (nombre de noyaux gaussiens impliqués dans l'approximation). Notre programme devra ainsi déterminer une valeur judicieuse pour m. Ce problème se révèle insoluble. Nous avons implanté diverses heuristiques et effectué plusieurs expériences. Certaines méthodes fournissaient des résultats encourageants. Toutefois, nous trouvions toujours un caractère de *NIST* pour lequel nos heuristiques s'avéraient catastrophiques.

Nous suggérons de modéliser la fonction discrète définie par la projection d'une image lissée[5] (figure 13.8 (b)). Nous avons constaté **expérimentalement** qu'il s'avérait nettement plus facile d'estimer les paramètres des gaussiennes dans cette situation. Le lissage se répercute en effet sur la projection, éliminant ainsi les petites irrégularités de la fonction discrète. En règle générale, le nombre de noyaux gaussiens est alors égal au nombre de maxima de la

[5] Les projections d'une image lissée se calculent selon les formules (13.1), (13.2), (13.3) et (13.4). Il suffit de substituer la valeur codant le niveau de gris à la valeur binaire i_{mn}.

fonction discrète (figure 13.8 (b)). Nous avons implanté un algorithme tirant parti de cette constatation[6].



(a) (b)

Figure 13.8: Projections d'une image binaire et d'une image lissée. (a) Projection typique de la représentation binaire d'un motif. (b) Projection de l'image lissée du motif. Les maxima des projections apparaissent en gris.

[6]Nous ne présentons pas les détails d'implantation. Le système obtenu n'est pas totalement satisfaisant.

L'approximation des projections par des gaussiennes souffre de plusieurs inconvénients majeurs :

- Comme nous l'avons constaté lors des exemples présentés au début de ce paragraphe, l'algorithme de Levenberg-Marquardt est très sensible à une infime modification de la fonction discrète. Les coefficients des gaussiennes ne sont donc pas toujours représentatifs. Lors de certaines expériences, nous avons même observé des valeurs tendant vers $-\infty$!
- L'estimation automatique des coefficients de quelques projections est impossible.

13.2.2 Extraction des maxima d'une projection

Les paramètres de noyaux gaussiens fournissant une approximation médiocre d'une projection, nous souhaitons étudier une autre méthode. Notre solution consiste simplement à sélectionner les coordonnées d'au plus m maxima de la fonction discrète obtenue par projection de l'image lissée. L'algorithme se compose des étapes suivantes :

- Nous sélectionnons une valeur de m. De multiples observations des projections suggèrent de fixer m à quatre.
- Nous estimons ensuite la dérivée de la fonction discrète en tout point.
- Nous considérons alors les portions de fonctions contenues entre deux minima consécutifs (figure 13.9). Soient $P_1^k(x_1^k, y_1^k), P_2(x_2^k, y_2^k), \ldots, P_{N_k}^k(x_{N_k}^k, y_{N_k}^k)$ les N_k points constituant la $k^{\text{ème}}$ portion. Nous calculons la variance de la distribution selon l'ordonnée :

$$\sigma_k = \frac{\sum_{i=1}^{N_k} \left(y_i^k - \overline{y}^k\right)^2}{N_k - 1} \tag{13.7}$$

où

$$\overline{y}^k = \frac{\sum_{i=1}^{N_k} y_i^k}{N_k} \tag{13.8}$$

Considérons la fonction discrète de la figure 13.9. Les maxima des portions A, B et D la modélisent relativement bien. Le maximum de C, caractérisé par une faible valeur de σ_k, ne fournit par contre que peu d'information[7]. Nous avons déterminé **expérimentalement** un seuil β et ne prenons en considération que les maxima des portions k satisfaisant

$$\sigma_k > \beta \tag{13.9}$$

- Les abscisses et ordonnées des maxima ainsi déterminés constituent le modèle de la fonction discrète. Si nous disposons de plus de m maxima, nous sélectionnons ceux auxquels sont associées les plus grandes valeurs de σ_k. Nous classons les maxima par ordre croissant des abscisses.

[7] Nous avons constaté lors de plusieurs expériences qu'un tel maximum résulte de taches de l'image.

- L'ultime étape consiste à organiser l'information récoltée dans un vecteur de $2 \cdot m$ composantes. Il est nécessaire d'envisager deux cas.
  - Si nous disposons de m maxima, il suffit de placer leurs coordonnées dans le vecteur (figure 13.10 (a)).
  - Si nous ne disposons que de $n < m$ maxima, la situation se révèle plus délicate. Diverses expériences nous ont suggéré d'adopter l'heuristique suivante : les coordonnées du premier (respectivement du $n^{\text{ème}}$) maximum sont placées dans les deux premières (respectivement dans les deux dernières) composantes du vecteur; nous disposons ensuite les coordonnées des autres maxima à partir de la troisième composante du vecteur. Nous fixons finalement à zéro la valeur des composantes inutilisées. La figure 13.10 (b) illustre le résultat de notre algorithme.

En appliquant cet algorithme à chacune des projections, nous obtenons quatre vecteurs constituant un motif d'entrée du système neuromimétique. Remarquons finalement que la méthode proposée repose sur un nombre important d'expériences et d'observations. Nous ne pouvons fournir aucune justification mathématique de son bien-fondé.

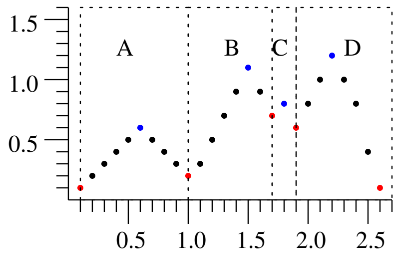


Figure 13.9: Recherche des maxima d'une fonction discrète. Les points bleus et rouges désignent respectivement les maxima et les minima.

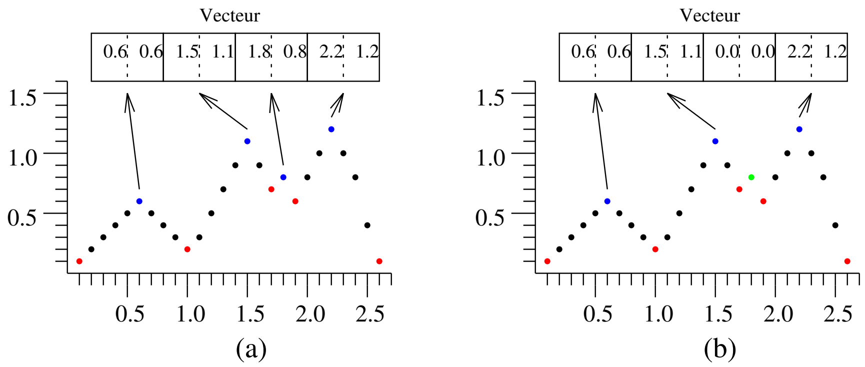


Figure 13.10: Organisation des maxima dans un vecteur. Nous souhaitons modéliser la fonction discrète avec au plus quatre maxima. Les points bleus, rouges et verts désignent respectivement les maxima, les minima et les maxima éliminés par notre algorithme. (a) Pour une certaine valeur de β, nous obtenons quatre maxima que nous plaçons dans le vecteur. (b) Suite à une modification du seuil β, un maximum est éliminé. La figure illustre la disposition des maxima déterminée par notre heuristique.

13.2.3 Où la méthode des projections se révèle inefficace

La base de données *NIST* contient divers caractères de piètre qualité. Ainsi, les boucles des "0", "6", "8" ou "9" sont parfois obturées[8]. Considérons le chiffre "0" illustré par la figure 13.11. Ses projections diffèrent totalement de celles du "0" de la figure 13.12 et s'apparentent à celles du "6" de la figure 13.13. L'approche *VH2D* ne permet pas la distinction entre des "0", "6", "8" ou "9" dont les boucles sont obturées.

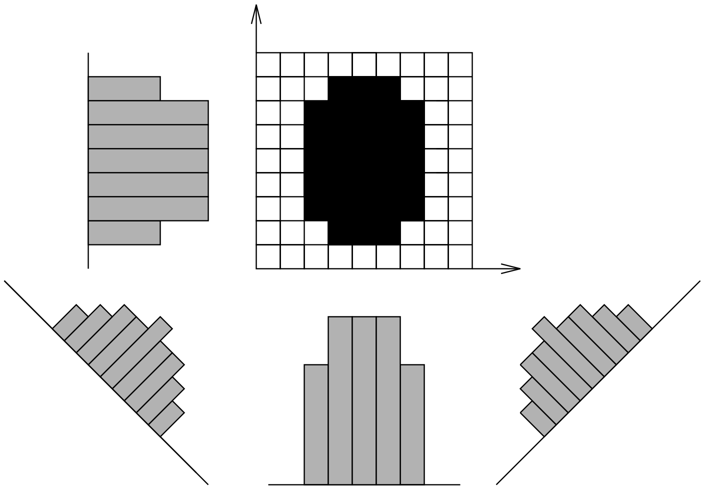


Figure 13.11: Projections du chiffre 0 écrit avec un feutre épais

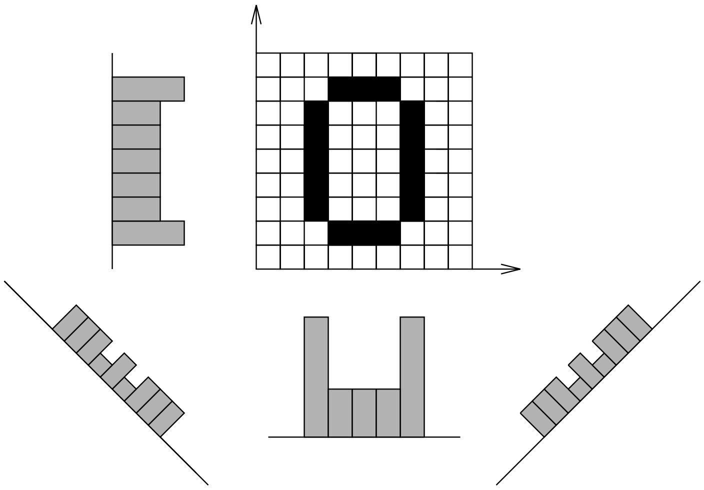


Figure 13.12: Projections du chiffre 0

[8]Diverses possibilités expliquent ce phénomène : utilisation d'un feutre épais, taches du papier, ...

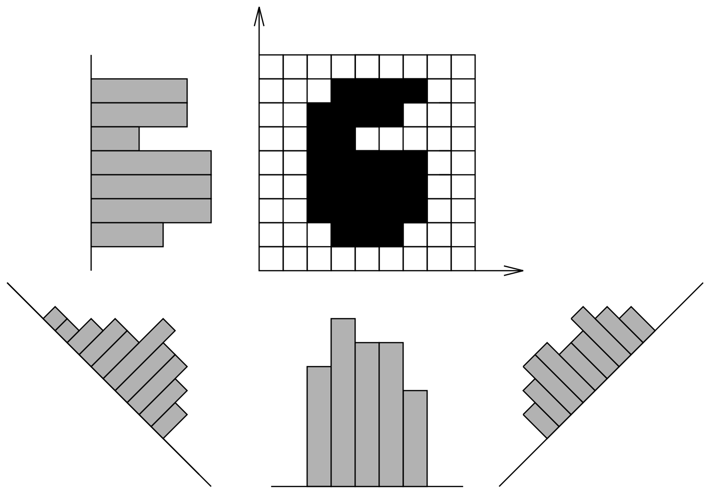


Figure 13.13: Projections du chiffre 6 écrit avec un feutre épais

13.3 Analyse du contour

13.3.1 Extraction du contour

La bibliothèque de traitement d'images de l'IDIAP propose divers algorithmes d'extraction du contour. Une solution (appliquée dans [17]) consiste à calculer le Laplacien et le gradient de l'image préalablement lissée. Les points pour lesquels le Laplacien s'annule constituent le contour de l'image et le gradient détermine leur orientation.

Nous invitons le lecteur à consulter [38] afin de découvrir d'autres algorithmes d'extraction de contour.

13.3.2 Classification des points du contour en fonction de leur orientation

G. Maître propose une classification des points du contour selon leur orientation [17]. Le principe consiste à définir $2n$ domaines d'orientation centrés sur les axes de coordonnées de l'image (figure 13.14).

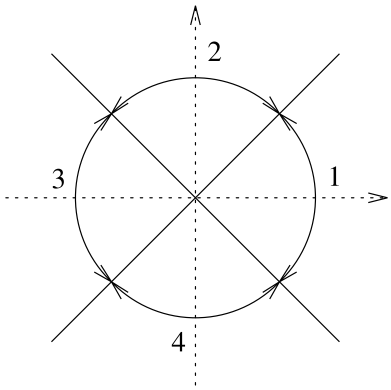


Figure 13.14: Quatre domaines d'orientation

L'information sur le contour connaît diverses exploitations. Il est par exemple possible de constituer n images contenant les pixels appartenant aux orientations j et $j+n, 1 \leq j \leq n$. Ainsi, en travaillant avec quatre orientations, nous obtiendrions deux images comportant les segments horizontaux et verticaux. S. Knerr adopta une solution similaire lors de la conception d'un système de reconnaissance d'écriture [46].

Nous proposons de calculer la proportion de pixels de chacune des $2n$ orientations. Soit N_i le nombre de pixels appartenant au domaine d'orientation i. Le nombre total de pixels N vérifie évidemment l'égalité :

$$N = \sum_{i=1}^{2n} N_i \tag{13.10}$$

Nous obtenons ainsi un vecteur $\boldsymbol{o} = [\frac{N_1}{N}, \ldots, \frac{N_{2n}}{N}]^T$ que nous présenterons au réseau de neurones.

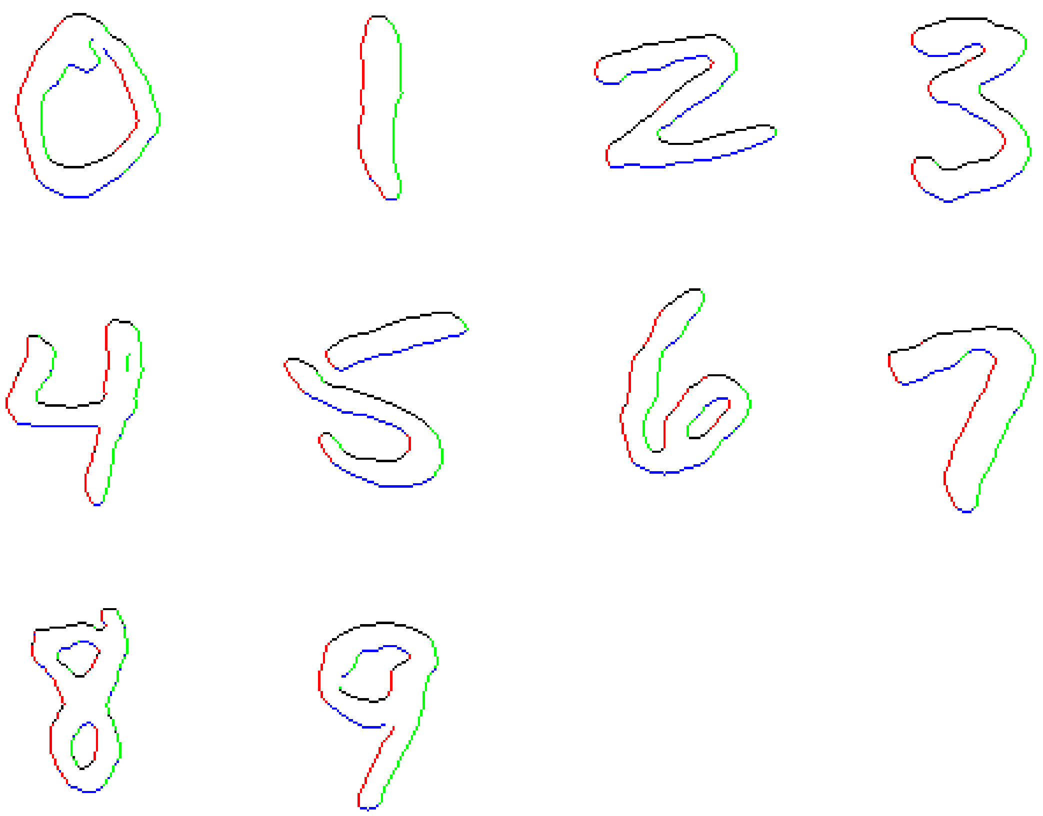


Figure 13.15: Classification selon quatre domaines d'orientation des points du contour de quelques chiffres

Chapitre 14

Une méthode statistique : l'analyse en composantes principales

Une fois un caractère prétraité, normalisé et lissé, nous disposons de $n = N \cdot N$ valeurs codant les niveaux de gris des pixels (figure 14.1). Nous pourrions simplement présenter au réseau neuromimétique les vecteurs $\boldsymbol{\xi}^i = [\xi_1^i, \ldots, \xi_n^i]^T$ contenant ces valeurs. Toutefois, lorsque la corrélation entre les composantes des vecteurs $\boldsymbol{\xi}^i$ est différente de zéro, $m \ll n$ variables indépendantes permettent la description des données. m constitue la dimension intrinsèque des données. En tronquant $\boldsymbol{\xi}^i$, nous provoquons une erreur moyenne au carré égale à la somme des variances des éléments éliminés [22]. L'analyse en composantes principales [28] détermine une transformation linéaire minimisant cette erreur. Son principe consiste à calculer les vecteurs propres $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_m$ associés aux m plus grandes valeurs propres de la matrice de covariance de la distribution (les axes déterminés par ces vecteurs sont appelés axes principaux). La projection des données dans le sous-espace défini par les vecteurs $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_m$ constitue la transformation optimale.

Observons la distribution de la figure 14.2. Nous souhaitons la représenter dans un espace unidimensionnel. La matrice de covariance d'une telle distribution possède évidemment deux vecteurs propres $\boldsymbol{e}_1$ et $\boldsymbol{e}_2$ associés aux valeurs propres λ_1 et λ_2 ($\lambda_1 > \lambda_2$). Nous remarquons que la variance de la projection sur l'axe principal (défini par $\boldsymbol{e}_1$) est supérieure à celle de toute autre projection. Nous minimisons ainsi la perte d'information lors de la transformation. Le lecteur intéressé découvrira les fondements théoriques de l'analyse en composantes principales dans le prochain paragraphe. Sa lecture n'est cependant pas indispensable à la compréhension de ce rapport.

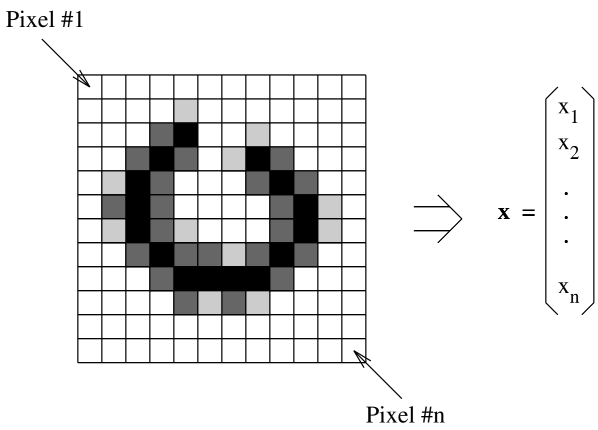


Figure 14.1: Représentation d'une image par un vecteur

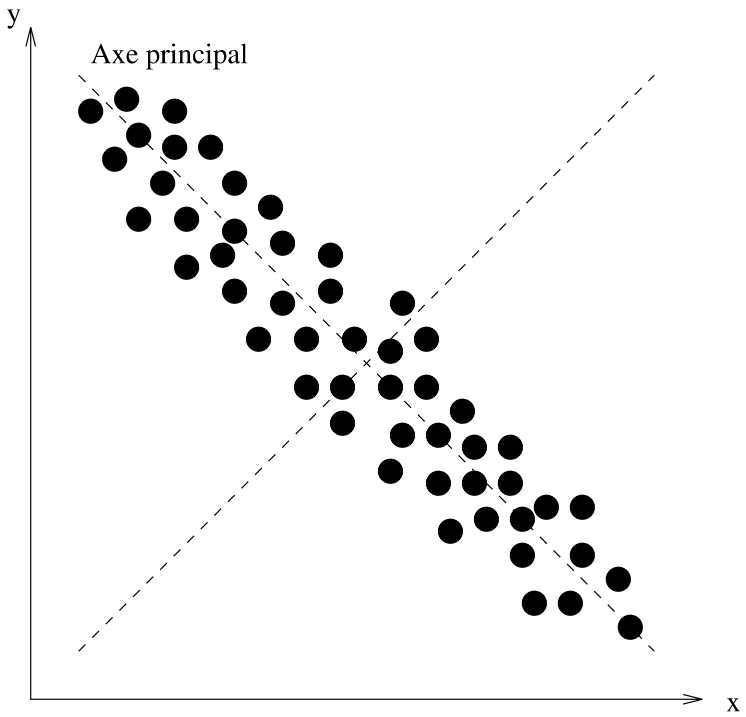


Figure 14.2: Analyse en composantes principales

14.1 Fondements théoriques

Soient un vecteur $\boldsymbol{x} = [x_1, \ldots, x_n]^T$ d'espérance $E\{\boldsymbol{x}\}$ nulle et $R_x = \boldsymbol{x}\boldsymbol{x}^T$ sa matrice de covariance. Le vecteur $\boldsymbol{y} = [y_1, \ldots, y_m]^T$, résultant de l'analyse en composantes principales, est une transformation orthogonale et linéaire des données :

$$\boldsymbol{y} = W\boldsymbol{x} \tag{14.1}$$

Les éléments $y_1, \ldots, y_m$ sont les composantes principales du vecteur $\boldsymbol{x}$. Les lignes de W constituent une base orthonormée d'un sous-espace $\mathfrak{L}$. La transformation inverse, appelée reconstruction, est définie par

$$\begin{aligned} \widehat{\boldsymbol{x}} &= W^T\boldsymbol{y} \qquad \text{car } WW^T = I \\ &= W^TW\boldsymbol{x} \end{aligned} \tag{14.2}$$

L'objectif de l'analyse en composantes principales consiste à minimiser l'espérance de l'erreur de reconstruction J_e (le lecteur trouvera les détails des calculs dans [28]) :

$$\begin{aligned} J_e &= E\{\|\boldsymbol{x} - \widehat{\boldsymbol{x}}\|\} \\ &= tr(R_x) - tr(WR_xW^T) \end{aligned} \tag{14.3}$$

Remarquons que le second terme de l'équation 14.3 est égal à la variance J_v de $\boldsymbol{y}$:

$$J_v = tr(WR_xW^T) = E\{tr(\boldsymbol{y}\boldsymbol{y}^T)\} = \sum_{i=1}^{m} y_i^2 \tag{14.4}$$

$$= tr(E\{W^TW\boldsymbol{x}\boldsymbol{x}^TW^TW\}) = E\{tr(\widehat{\boldsymbol{x}}\widehat{\boldsymbol{x}}^T)\} = \sum_{i=1}^{n} \widehat{x}_i^2 \tag{14.5}$$

Nous avons ainsi établi qu'en minimisant l'erreur de reconstruction, nous maximisons la variance de la projection.

Théorème 3
Soient $\lambda_1, \lambda_2, \ldots, \lambda_n$ les valeurs propres de R_x classées par ordre décroissant. Soient encore $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n$ les vecteurs propres normalisés associés aux valeurs propres. La matrice de projection W minimisant l'erreur de reconstruction, sous la contrainte $WW^T = I$, est déterminée par

$$W_{\text{opt}} = T[\pm\boldsymbol{e}_1, \pm\boldsymbol{e}_2, \ldots, \pm\boldsymbol{e}_m]^T, \quad 1 \leq m < n \tag{14.6}$$

où T est une matrice carrée orthogonale quelconque. L'erreur minimale de reconstruction $minJ_e$ vérifie alors l'égalité :

$$minJ_e = \sum_{i=m+1}^{n} \lambda_i \tag{14.7}$$

La variance maximale est définie par :

$$maxJ_v = \sum_{i=1}^{m} \lambda_i \tag{14.8}$$

14.2 *APEX*, un modèle neuromimétique pour l'extraction des composantes principales

Divers modèles neuromimétiques, basés sur la règle de Hebb[1], permettent l'extraction des composantes principales. Nous présentons les principes du modèle *APEX (Adaptative Principal Component EXtraction)* [28]. Connaissant les j premières composantes principales, l'algorithme calcule itérativement la $j+1^{\text{ème}}$. Le réseau comporte deux types de connexions (figure 14.3) :

- Des connexions excitatrices reliant les entrées à chacun des neurones. Nous noterons $W = (w_{ij}) = [\boldsymbol{w_1}, \ldots, \boldsymbol{w_m}]^T$ la matrice les définissant.
- Des connexions latérales inhibitrices asymétriques. Les neurones sont organisés hiérarchiquement : le neurone i est connecté à tous les neurones $j > i$. Ces coefficients synaptiques sont mémorisés dans la matrice triangulaire inférieure $C = (c_{ij})$

$$C = \begin{pmatrix} 0 & \ldots & \ldots & \ldots & 0 \\ c_{21} & 0 & \ldots & \ldots & 0 \\ \vdots & & \ddots & & \vdots \\ c_{m-1,1} & \ldots & c_{m-1,m-2} & 0 & 0 \\ c_{m,1} & \ldots & \ldots & c_{m,m-1} & 0 \end{pmatrix} \tag{14.9}$$

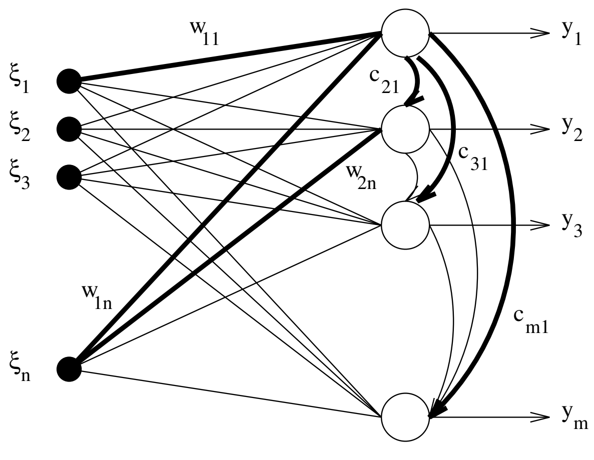


Figure 14.3: Réseau à connexions latérales asymétriques. Les coefficients synaptiques apparaissant sur la figure sont associés aux connexions dessinées en gras.

Supposons que les valeurs propres (triées par ordre décroissant) de la matrice de covariance des données soient positives et satisfassent la relation :

$$\lambda_1 > \lambda_2 > \ldots > \lambda_m \geq \lambda_{m+1} \geq \ldots \geq \lambda_n > 0 \tag{14.10}$$

[1]En 1949, D. Hebb a proposé une règle simple permettant de modifier les poids d'un système neuromimétique en fonction de l'activité des éléments qu'ils relient [9] : *"When an axon of cell A is near enough to excite a cell B and repeatedly and persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."* Le principe consiste à accroître les coefficients synaptiques entre neurones dont l'activité est synchronisée.

$\boldsymbol{e_1}, \boldsymbol{e_2}, \ldots, \boldsymbol{e_n}$ dénotent les vecteurs propres associés. L'algorithme *APEX* se compose des étapes suivantes :

1. **Initialisation.**
   Les coefficients synaptiques sont initialisés à de petites valeurs aléatoires. Il faut encore assigner une petite valeur positive au coefficient d'apprentissage β (par exemple $\beta = 0.02$).

2. **Calcul du vecteur propre associé à λ_1.**
   Soit $i = 1$. Pour $t = 0, 1, \ldots$, calculer

$$y_1(t) = \boldsymbol{w}_1^T \cdot \boldsymbol{\xi}(t) = \sum_{j=1}^{n} w_{1j}\xi_j(t)$$

$$w_{1j}(t+1) = w_{1j}(t) + \beta(t) \cdot \big(y_1(t)\xi_j(t) - y_1(t)^2 w_{1j}(t)\big)$$

   Après quelques itérations, nous obtenons :

$$\boldsymbol{w_1}(t) \rightarrow \boldsymbol{e_1} \tag{14.11}$$

3. **Calcul du vecteur propre associé à λ_i, $2 \leq i \leq m$.**
   Connaissant les $i-1$ premières composantes principales, *APEX* détermine la $i^{\text{ème}}$. Pour $t = 0, 1, \ldots$, calculer

$$\boldsymbol{y}_{i-1}(t) = [y_0(t), y_1(t), \ldots, y_{i-1}(t)]^T$$

$$y_i(t) = \sum_{j=1}^{n} w_{ij}(t)\xi_j(t) - \sum_{k=1}^{i-1} c_{ik}(t)y_k t$$

$$w_{ij}(t+1) = w_{ij}(t) + \beta(t) \cdot \big(y_i(t)\xi_j(t) - y_i(t)^2 w_{ij}(t)\big)$$

$$c_{ij}(t+1) = c_{ij}(t) + \beta(t) \cdot \big(y_i(t)y_j(t) - y_i(t)^2 c_{ij}(t)\big) \quad i > j$$

   Afin de calculer les m vecteurs propres souhaités, il suffit de réitérer cette étape avec $i = 2, 3, \ldots, m$. Lorsque t est suffisamment grand,

$$\boldsymbol{w_i}(t) \rightarrow \boldsymbol{e}_i \tag{14.12}$$

$$c_{ij}(t) \rightarrow 0 \tag{14.13}$$

   Ainsi, lorsque l'algorithme a convergé, tous les coefficients c_{ij} sont nuls et la matrice W contient les vecteurs propres associés aux m plus grandes valeurs propres.

Généralement, un coefficient β distinct est associé à chaque neurone. Afin que l'algorithme converge, il est impératif que

$$\beta(t) \rightarrow 0 \quad \text{lorsque } t \rightarrow \infty \quad \text{et } \sum_{t=0}^{\infty} \beta(t) = \infty \tag{14.14}$$

L'équation ci-dessous propose une règle optimale d'adaptation de ce paramètre (une valeur typique de γ est 0.99) :

$$\beta(t+1) = \frac{\beta(t)}{\gamma + y(t)^2\beta(t)} \tag{14.15}$$

14.3 Talon d'Achille de l'analyse en composantes principales

Considérons le problème de classification illustré par la figure 14.4. L'analyse en composantes principales suggère de projeter les données sur l'axe déterminé par $\boldsymbol{e}_1$. Cette projection ne permet malheureusement plus la distinction des deux classes. Lors de nos expériences, nous utiliserons diverses valeurs pour m. Il est probable que nous obtenions des résultats similaires pour deux valeurs distinctes m_1 et m_2. Ce phénomène indiquerait que les projections sur certains axes ne permettent plus de distinguer les classes.

L'analyse en composantes principales non linéaire[2] et l'analyse discriminante linéaire [14] proposent une solution à ce problème. La projection sur l'axe associé à $\boldsymbol{e}_2$ (figure 14.4) permet de discerner parfaitement les deux classes. Lorsque nous disposons de plusieurs classes, il n'est donc pas toujours judicieux d'effectuer les projections sur les axes de plus grande variance[3].

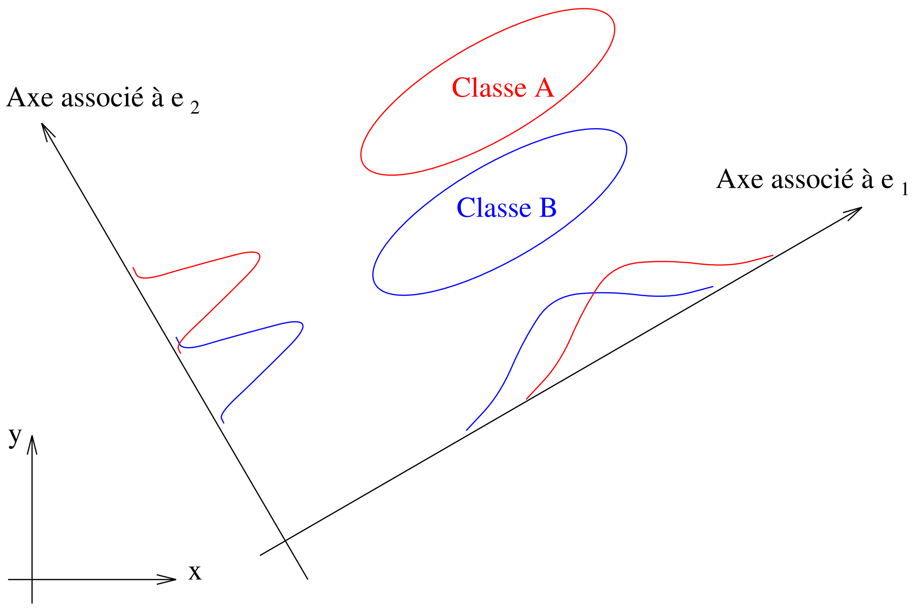


Figure 14.4: Limitations de l'analyse en composantes principales. La projection des données sur l'axe principal ne permet plus la distinction des deux classes.

[2]Nous conseillons au lecteur de consulter les articles de Kambhatla et Leen [33] et de Fyfe et Baddeley [6].

[3]Nous n'avons malheureusement pas eu le temps d'achever notre programme d'analyse discriminante linéaire. Nous supposons que cette méthode permet d'obtenir de bons résultats pour un petit nombre m de projections (par exemple, $m = 20$ pour des images de seize pixels sur seize).

Partie III

Expériences

Chapitre 15

Considérations générales

Nous consacrons cette troisième partie aux descriptions et résultats des diverses expériences effectuées. La figure 15.1 illustre l'architecture générale des systèmes que nous utilisons. Dans un premier temps, le chiffre est prétraité, normalisé et lissé. L'algorithme d'extraction de caractéristiques détermine ensuite le vecteur d'entrée du système neuromimétique. Tous les réseaux utilisés lors des simulations comportent dix sorties modélisant chacune un chiffre. Rappelons que lorsque nous présentons un motif au réseau, nous souhaitons que la sortie correspondante soit active.

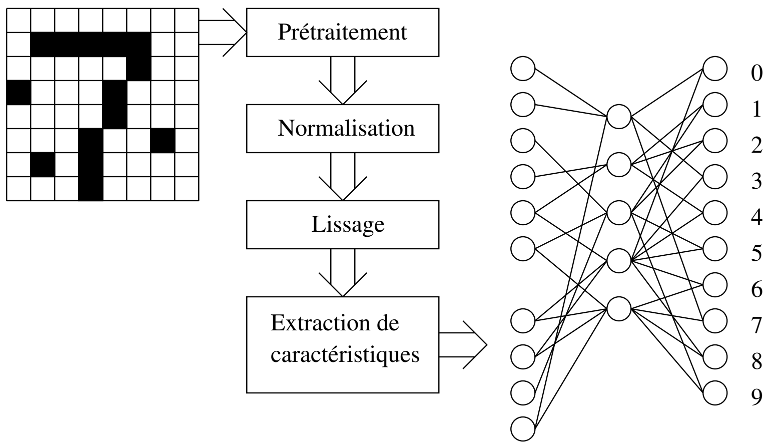


Figure 15.1: Schéma général du système de reconnaissance d'écriture

Lors des diverses expériences, nous construisons la base d'apprentissage à partir de 15025 chiffres écrits par 140 personnes. Le tableau 15.1 présente sa composition. 4975 chiffres, écrits par 48 personnes (étrangères à la conception de la base d'apprentissage), constituent les données de validation.

Chiffres	0	1	2	3	4	5	6	7	8	9
Exemplaires	1557	1698	1461	1580	1431	1296	1460	1569	1484	1489

Tableau 15.1: Nombre de caractères de chaque classe dans la base d'apprentissage

15.1 Méthodes d'évaluation

Les différentes applications d'un système de reconnaissance d'écritures manuscrites possèdent leurs propres contraintes. Etudions deux exemples.

- **Numérisation et transcription de documents**
  Supposons qu'une bibliothèque souhaite numériser et transcrire, sous forme électronique, un document manuscrit[1] de plusieurs centaines de pages. Le travail de copiste se révélant laborieux, un système automatique nous semble adéquat. Une fois le texte numérisé et traité, nous disposons, par exemple, d'outils de correction orthographique. Ainsi, lorsqu'un mot ne figure pas dans le dictionnaire[2], le système signale une erreur[3]. Dans cette situation, il est intéressant de maximiser le nombre de caractères reconnus. Nous restreignons ainsi le nombre d'interventions d'un opérateur chargé de corriger les erreurs détectées.

- **Traitement automatique du courrier**
  Un tel système classe le courrier en fonction du numéro postal figurant sur l'enveloppe. Chaque erreur s'avère coûteuse, la missive étant acheminée vers une mauvaise destination. Il est donc primordial de minimiser le nombre d'erreurs. Un mécanisme de rejet permet d'écarter certains envois. Un employé se charge alors de déterminer leur destination.

Nous utilisons deux critères de performance adaptés aux problèmes évoqués.

- **Calcul du pourcentage de motifs correctement classés**
  Présentons un motif $\boldsymbol{\xi}^{\rho}$ de la classe i au réseau. Soit j l'indice du neurone de sortie dont l'activation est la plus élevée. Le système classifie correctement $\boldsymbol{\xi}^{\rho}$ si i est égal à j. Nous appliquerons ce critère lors de toutes nos simulations.

- **Mécanisme de rejet**
  La base de données *NIST* contient quelques écritures exécrables. Nous avons demandé à plusieurs personnes d'identifier une série de caractères dûment choisis. Régulièrement, elles répondaient qu'elles hésitaient entre deux possibilités. Dans un système de tri de courrier, nous rejetterions de tels chiffres. Ces diverses alternatives se révèlent parfois utiles lors de la transcription de documents sous forme électronique[4]. Définissons un seuil θ. Si toutes les sorties du réseau sont inférieures à θ, nous rejetons le motif. Si plusieurs sorties sont supérieures à θ, nous proposons ces diverses alternatives. Lorsqu'une seule

[1] Nous anticipons un peu. Nous n'avons pas encore réalisé d'expériences avec des lettres et des chiffres.

[2] Ou lorsque plusieurs solutions sont envisageables. Supposons que le mot erroné "matim" apparaisse dans le texte. Le système proposera alors "matin" ou "matir".

[3] Si, par exemple, le mot "rance" est transformé en "ranch", l'erreur est indétectable.

[4] Reprenons notre exemple. Le mot "matim" figure dans le texte et le système indique que la dernière lettre est un "m" ou un "n". Il est maintenant possible d'effectuer la bonne correction.

sortie est supérieure au seuil θ, elle représente évidemment la classe du motif d'entrée. Diverses expériences sont nécessaires afin d'évaluer les performances de ce mécanisme. Nous serons particulièrement attentif aux détails suivants :

- Combien d'alternatives le système fournit-il ? Il est souhaitable que le réseau ne suggère que deux ou trois possibilités.
- La classe du motif figure-t-elle régulièrement parmi les propositions ? Si aucune des sorties actives ne représente la solution correcte, notre mécanisme est inutile !
- Quel est le pourcentage de motifs pour lesquels le réseau hésite et suggère plusieurs possibilités ? Actuellement, les taux d'erreur et de rejet acceptés pour un système de tri de courrier sont respectivement de 1% et 9% [46].

Nous n'appliquerons ce critère qu'au réseau fournissant les meilleurs résultats.

15.2 Initialisation des réseaux

L'initialisation des poids synaptiques et des biais a d'importantes répercussions sur la vitesse d'apprentissage et les capacités de généralisation. Ces valeurs sont uniformément distribuées dans un intervalle I judicieusement choisi.

15.2.1 Initialisation d'un *perceptron* multicouche

Plusieurs chercheurs proposent des algorithmes déterminant les bornes de I. Ainsi, F. J. Śmieja définit, pour un *perceptron* multicouche, l'intervalle $[-2/\sqrt{d_{\text{in}}}, 2/\sqrt{d_{\text{in}}}]$, où d_{in} est le *fan-in* d'un neurone. Lors d'une étude d'algorithmes d'initialisation, G. Thimm [18] a constaté que la méthode de Śmieja fournissait de bons résultats. Nous l'utiliserons lors de nos expériences. Il est parfois possible de déterminer un meilleur intervalle en effectuant plusieurs expériences.

15.2.2 Initialisation d'un *high order perceptron*

Nous utilisons les résultats des expériences de G. Thimm. Le procédé d'initialisation dépend de la fonction d'activation $\varphi(x)$:

- Si $\varphi(x)$ est la fonction identité, les poids sont initialisés aléatoirement avec une variance d'environ 10^{-4}. (Ce procédé revient à initialiser les poids dans l'intervalle $[-0.017, 0.017]$.)
- Si $\varphi(x)$ est une tangente hyperbolique, les meilleures performances s'obtiennent en initialisant les poids dans l'intervalle $[-a/\sqrt{d_{\text{in}}}, a/\sqrt{d_{\text{in}}}]$, où a est choisi de manière à ce que la variance des poids corresponde au tiers de la distance entre les points de courbure maximale de $\varphi(x)$.

15.3 Conventions

Nous utiliserons les conventions suivantes lors des descriptions des simulations :

- Sauf mention explicite, les réseaux sont entraînés avec l'algorithme de *backpropagation online*.

- Dans les tableaux de résultats, trois chiffres décrivent la topologie des *perceptrons* multi-couches. Ils dénotent respectivement le nombre de neurones des couches d'entrée, cachée et de sortie. Nous n'envisageons pas l'entraînement de réseaux comportant plus d'une couche cachée.

- Lorsque nous travaillerons avec des *high order perceptrons*, nous nous restreindrons à des réseaux de second ordre totalement connectés.

- Nous numéroterons parfois les expériences dans les tableaux de résultats. Nous pourrons ainsi y faire plus facilement référence dans le texte.

Chapitre 16

Apprentissage à partir de l'image prétraitée et normalisée

L'expérience la plus simple consiste à présenter au réseau des chiffres prétraités, normalisés, puis lissés. Cette dernière opération s'avère cruciale. Lors d'une étude de divers algorithmes d'élagage [4], nous obtenions des résultats médiocres pour des problèmes à entrées et sorties booléennes.

Rappelons que l'algorithme de normalisation utilisé inscrit le caractère dans une portion de l'image de 128 pixels sur 128. Connaissant l'angle inférieur gauche de cette zone et le paramètre k définissant la hauteur standard (paragraphe 12.3), il est trivial d'extraire l'image de k pixels sur k représentant le caractère normalisé. La figure 16.1 décrit cette opération.

Pour de telles expériences, nous n'utilisons que des *perceptrons* multicouches. Le nombre d'entrées du réseau étant relativement élevé, seuls des algorithmes constructifs permettent d'entraîner des *high order perceptrons* dans ce genre de problème [47].

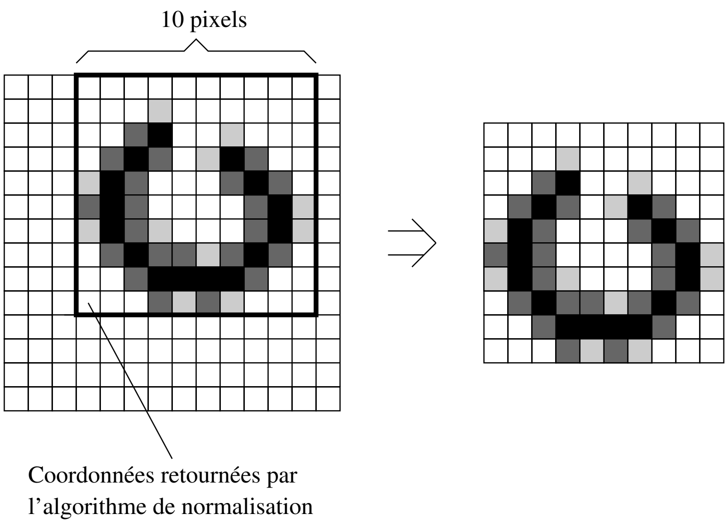


Figure 16.1: Extraction du caractère normalisé (dix pixels sur dix)

16.1 Images de huit pixels sur huit

Le tableau 16.1 présente les résultats de deux expériences. La reconnaissance des chiffres de la base de validation s'avère médiocre. Ces résultats s'expliquent peut-être par l'importante déformation induite par la normalisation de la taille.

Topologie	Apprentissage	Validation
64-100-10	98.34%	92.64%
64-50-10	97.28%	92.1%

Tableau 16.1: Résultats des expériences effectuées avec l'image de huit pixels sur huit

16.2 Images de seize pixels sur seize

Le tableau 16.2 présente les excellents résultats[1] obtenus en travaillant avec des chiffres de seize pixels sur seize.

Topologie	Apprentissage	Validation
256-128-10	99.79%	**97.39%**

Tableau 16.2: Résultats de l'expérience effectuée avec l'image de seize pixels sur seize

16.2.1 Etude des données de validation

Nous proposons une étude détaillée des données de validation[2]. Le tableau 16.3 présente les pourcentages d'erreurs calculés pour chacune des écritures. Nous constatons que dix écritures sont parfaitement traitées. Le taux d'erreur se révèle, à quelques exceptions, acceptable.

La figure 16.2 illustre les motifs incorrectement classifiés de l'écriture n° 174. Le réseau confond la majorité des “2” avec des “3”[3]. Ce style d'écriture n'est pas représenté dans les exemples d'apprentissage. Il est certainement possible d'améliorer les performances des réseaux en intégrant d'autres calligraphies à nos données d'entraînement.

Le réseau se révèle incapable de reconnaître les chiffres de la figure 16.3. Remarquons que même pour un être humain, la tâche n'est pas aisée ! Il commet cependant des erreurs surprenantes (figure 16.4). Observons le premier chiffre de la seconde ligne. En cachant la moitié supérieure de la boucle de ce “9”, nous obtenons un “7” fort respectable.

16.2.2 Intégration du mécanisme de rejet

Le tableau 16.4 présente les performances du système lorsque nous lui adjoignons le mécanisme de rejet ($\theta = 0.1$). Le pourcentage de classifications erronées passe ainsi de 2.61 à 1.27 pour

[1] Afin de les confirmer, nous avons constitué une seconde base de validation comportant 4941 chiffres écrits par 47 personnes différentes. Nous obtenons un taux de classification correcte de 96.86%.

[2] Effectuer ce travail lors de toutes les simulations s'avérerait fastidieux et allongerait inutilement ce rapport ! Nous ne le ferons donc que pour cette expérience. Cette étude nous permettra de présenter au lecteur quelques caractères particuliers de *NIST*.

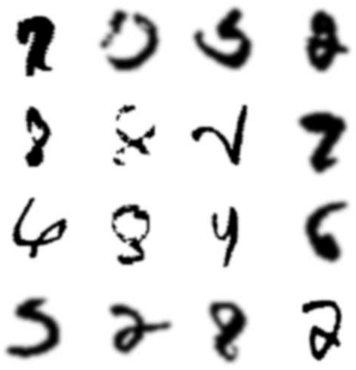
[3] Les “2”, très particuliers, ressemblent à des “3” dont la boucle inférieure serait tronquée.

Code	Erreur	Code	Erreur	Code	Erreur	Code	Erreur
140	2.94%	141	2.52%	142	7.14%	**143**	**0%**
144	0.78%	145	3.0%	146	5.49%	147	0.85%
148	4.35%	149	3.16%	150	2.13%	**151**	**0%**
152	5.65%	153	4.76%	**154**	**0%**	155	0.79%
156	2.88%	157	0.84%	**158**	**0%**	159	2.61%
160	**0%**	161	4.13%	162	8.04%	163	0.91%
164	0.82%	**165**	**0%**	166	1.89%	167	2.88%
168	2.52%	**169**	**0%**	170	0.92%	171	1.54%
172	2.4%	173	4.76%	174	13.16%	175	0.89%
176	**0%**	177	1.09%	178	0.94%	179	6.73%
180	5.26%	181	0.89%	**182**	**0%**	183	0.96%
184	9.0%	185	1.59%	186	3.64%	**187**	**0%**

Tableau 16.3: Pourcentages d'erreurs calculés pour chacune des écritures de la base de validation. Le code identifie la personne ayant écrit les caractères.

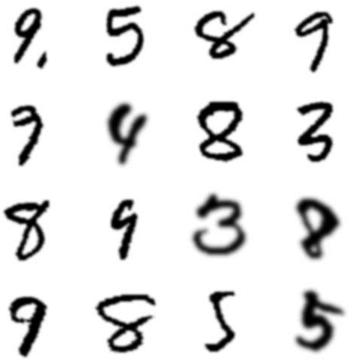
8	2	5	2
2	3	3	2
8	2	2	2
2	2	2	

Figure 16.2: Une écriture problématique. Le tableau indique, pour chaque chiffre, la classe à laquelle il appartient.



7	0	5	2
8	8	2	9
6	8	4	6
5	2	8	2

Figure 16.3: Quelques chiffres méconnaissables. Le tableau indique, pour chaque chiffre, la classe à laquelle il appartient.



9/8	5/3	8/4	9/7
9/7	4/9	8/2	3/8
8/9	9/4	3/8	8/7
9/7	8/5	5/3	5/3

Figure 16.4: Quelques erreurs surprenantes. Le tableau indique, pour chaque chiffre, la classe à laquelle il appartient ainsi que celle proposée.

la base de validation.

Etudions les chiffres pour lesquels le réseau suggère deux possibilités[4].

- Généralement, la sortie dont l'activation est la plus élevée correspond à la classe du motif (la figure 16.5 propose quelques exemples).

- Toutes les propositions du réseau se révèlent erronées pour six chiffres (figure 16.6).

Notre mécanisme de rejet satisfait ainsi les divers critères formulés au paragraphe 15.1. Afin de confirmer ces résultats, nous avons à nouveau utilisé une seconde base de validation. Notons qu'il est encore possible de réduire le pourcentage d'erreurs en modifiant le seuil θ de l'algorithme de rejet (tableau 16.5). Nous observons une diminution du taux de reconnaissance et une augmentation du nombre total de motifs rejetés ou pour lesquels le réseau hésite.

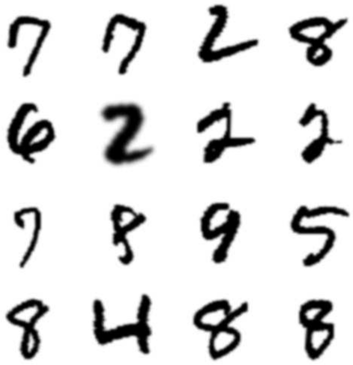
	Apprentissage	Validation
Motifs correctement classés	99.78%	94.77%
Erreurs	0.02%	1.27%
Motifs rejetés	0.19%	0.6%
Plusieurs solutions (la sortie la plus active ne correspond pas à la classe du motif)	0.01%	0.82%
Plusieurs solutions (la sortie la plus active correspond à la classe du motif)	0%	2.41%
Plusieurs solutions (aucune des sorties actives ne correspond à la classe du motif)	0%	0.12%

Tableau 16.4: Résultats de l'expérience effectuée avec l'image de seize pixels sur seize en intégrant le mécanisme de rejet ($\theta = 0.1$)

[4]Lors des simulations, le réseau n'a jamais proposé plus de deux solutions. Ce mécanisme serait totalement inintéressant si le système suggérait trop de possibilités !

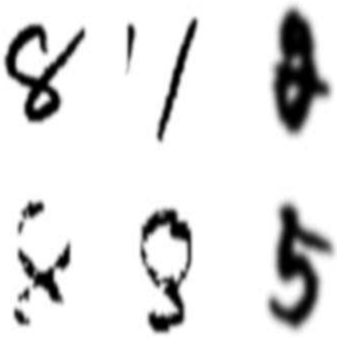
	Apprentissage	Validation
Motifs correctement classés	98.67%	92.04%
Erreurs	0.05%	**0.96%**
Motifs rejetés	0.14%	0.28%
Plusieurs solutions (la sortie la plus active ne correspond pas à la classe du motif)	0.01%	1.19%
Plusieurs solutions (la sortie la plus active correspond à la classe du motif)	1.12%	5.29%
Plusieurs solutions (aucune des sorties actives ne correspond à la classe du motif)	0.01%	0.24%

Tableau 16.5: Résultats de l'expérience effectuée avec l'image de seize pixels sur seize en intégrant le mécanisme de rejet ($\theta = 0.03$)



7/9	7/9	2/6	8/4
6/4	2/8	2/7	2/8
7/9	8/9	9/4	5/9
8/9	4/8	8/9	8/9

Figure 16.5: Motifs pour lesquels le réseau propose deux solutions (la sortie la plus active correspond à la classe du motif). Les alternatives se révèlent parfois logiques, parfois surprenantes. Le premier motif de la troisième ligne représente un "7". Nous avons montré ce chiffre à diverses personnes et plusieurs hésitèrent entre un "7" et un "9". Le réseau suggère ces deux solutions. Nous n'expliquons cependant pas pourquoi le réseau hésite entre un "4" et un "8" lorsque nous lui présentons le second motif de la dernière ligne. Le tableau indique, pour chaque chiffre, la classe à laquelle il appartient ainsi que celle proposée par le réseau.



8/4/0	1/4/7	2/0/1
8/4/2	8/2/7	5/3/8

Figure 16.6: Motifs pour lesquels le réseau propose deux solutions (aucune des sorties actives ne correspond à la classe du motif). Le second chiffre est un "1". Le réseau propose les alternatives "4" et "7". Cette solution est relativement logique : l'algorithme corrigeant l'inclinaison a redressé la totalité de "1" de la base d'apprentissage. Ici, l'important bruit perturbe le processus. Par contre, les chiffres "4" et "7" possèdent très souvent une barre oblique. Remarquons qu'en ajoutant une barre horizontale à ce chiffre, nous obtenons un "4" ou un "7". Le tableau indique, pour chaque chiffre, la classe à laquelle il appartient, puis les deux alternatives suggérées par le réseau.

16.2.3 Elagage du réseau

En étudiant les résultats des expériences, nous constatons que les performances du réseau semblent liées au nombre de neurones de la couche cachée. Lors de la classification d'un motif, certains neurones de la couche cachée possèdent une forte activité, alors que d'autres demeurent inactifs ($a^{\rho}_{l,i} \approx 0$). Diverses expériences suggèrent qu'à chaque classe ρ sont associées plusieurs configurations C^{ρ}_{i} distinctes de neurones actifs sur la couche cachée. Le réseau appliquerait ainsi des règles du type :

> Si la majorité des neurones de la configuration C^{ρ}_{i} sont actifs, le motif présenté au réseau appartient à la classe i.

Afin qu'une telle règle fonctionne, il est impératif qu'aucune configuration ne soit incluse dans une autre. Ainsi, un nombre de neurones élevé sur la couche cachée permettrait une plus grande variété de configurations. Remarquons finalement que ce mécanisme expliquerait les alternatives proposées par le réseau lors de la présentation de certains chiffres. Etudions le dernier chiffre de la figure 16.5. En supprimant une portion de la boucle inférieure de ce "8", nous obtenons un motif s'apparentant à un "9". Ainsi, plusieurs neurones appartenant à deux configurations distinctes seraient actifs simultanément.

La figure 16.7 décrit les portions d'image utilisées par les réseaux obtenus lors des deux premières expériences. L'algorithme *Skeletonization* a, par exemple, sélectionné les deux pixels de l'angle inférieur droit de l'image. Seule l'extémité de quelques chiffres "2" occupe cette portion de l'image. Ces pixels permettraient ainsi leur identification.

Exp.	Topologie	Connexions	Apprentissage	Validation
1	46-40-10	2240	99.69%	95.96%
2	**46-44-10**	**2464**	**99.75%**	**96.28%**
3	256-30-10	7980	99.75%	95.6%

Tableau 16.6: Apprentissage avec des images de seize pixels sur seize : topologies et performances de quelques réseaux obtenus après élagage (*Skeletonization*). Nous avons simplement modifié quelques paramètres lors des deux premières expériences. Lors de la dernière, nous interdisons à l'algorithme d'élaguer des neurones de la couche d'entrée.

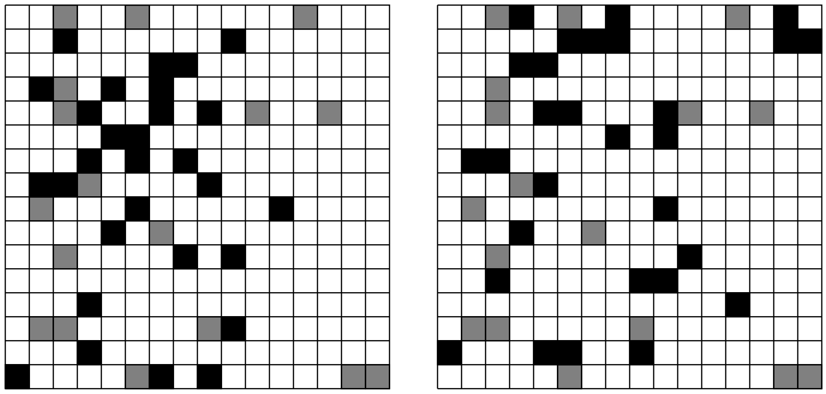


Figure 16.7: Portions de l'image de 16×16 pixels déterminées par l'algorithme *Skeletonization* lors des deux premières expériences. Les carrés gris (respectivement noirs) désignent les pixels sélectionnés lors des deux expériences (respectivement lors d'une seule expérience).

Chapitre 17

Apprentissage à partir des projections

Lors de cette série d'expériences, nous entraînons le réseau (*perceptron* multicouche) avec les projections *VH2D* de caractères prétraités, normalisés et lissés. Le tableau 17.1 présente les performances obtenues lors de deux expériences.

Les résultats médiocres, obtenus en calculant les projections à partir d'images de huit pixels sur huit, confirment les observations formulées au paragraphe 16.1. La normalisation des caractères à de très petites tailles engendre une importante déformation. Nous obtenons de bien meilleures performances en utilisant les projections d'images de seize pixels sur seize. Le nombre important d'entrées du réseau nous interdit cependant l'entraînement de *high order perceptrons*.

Topologie	Taille de l'image	Apprentissage	Validation
46-40-10	8×8	94.18%	90.19%
94-47-10	16×16	99.18%	96.34%

Tableau 17.1: Résultats obtenus en utilisant la méthode des projections *VH2D*

Chapitre 18

Apprentissage à partir du contour et des maxima des projections

Nous appliquons notre algorithme de recherche de maxima (paragraphe 13.2.2) à des images prétraitées, normalisées (quatre-vingts pixels sur quatre-vingts) et lissées. Une première série d'expériences a fourni des résultats très médiocres (environ 90% des données de validation correctement classifiées). Nous avons alors complété nos vecteurs d'entrées avec des informations sur le contour (paragraphe 13.3). En utilisant quatre maxima pour chacune des projections et huit domaines d'orientation pour l'analyse du contour, nous obtenons des vecteurs d'apprentissage de quarante composantes. Cette approche permet l'entraînement de *high order perceptrons* de second ordre totalement connectés ($C_2^{40} \cdot 10 + 40 \cdot 10 = 8200$ connexions).

18.1 Entraînement d'un *high order perceptron*

Après plusieurs jours de calculs sur une *Sparc Ultra*, un *high order perceptron* classifie correctement plus de 99% des données d'apprentissage. Malheureusement, ses performances de généralisation s'avèrent médiocres (environ 92%). L'application d'algorithmes d'élagage (*Smallest weights* et *Smallest variance*) n'a pas produit l'effet souhaité. Le temps d'entraînement entre deux élagages était trop important (environ une semaine en effectuant la simulation sur une *Sparc Ultra*). Les algorithmes constructifs permettraient certainement d'apprendre notre base de données à un *high order perceptron*.

18.2 Entraînement d'un *perceptron* multicouche

Lors d'une première série d'expériences, nous observions les motifs d'apprentissage que le réseau ne reconnaissait pas correctement. Les données d'entraînement contenaient 153 chiffres de piètre qualité dont la classification était toujours erronée[1].

Nous avons alors supprimé ces 153 vecteurs et obtenu une nouvelle base d'apprentissage. Son utilisation accélère la convergence du processus d'entraînement et s'avère spécialement

[1] Nous avons effectué une dizaine de simulations afin de déterminer ces motifs.

intéressante lors de l'élagage. Il est ainsi possible de restreindre le nombre de cycles séparant deux phases d'élagage successives. Nous obtenons les mêmes performances de généralisation, quelle que soit la base d'apprentissage choisie (tableau 18.1).

Notons finalement que l'algorithme de rétropropagation non linéaire aboutit à des résultats similaires.

Exp.	Topologie	Apprentissage	Validation
1	40-50-10	99.08%	96.66%
2	40-50-10	99.88%	96.64%

Tableau 18.1: Résultats obtenus en utilisant la méthode des maxima (quatre maxima par projection) et le codage du contour (huit domaines d'orientation). Lors de la première expérience, la totalité de la base d'apprentissage est utilisée. 153 motifs ont été supprimés des données d'entraînement lors de la seconde expérience.

18.3 Elagage du réseau

Nous avons appliqué divers algorithmes d'élagage afin de simplifier le réseau (tableau 18.2). *Autoprune* détermine la meilleure topologie. Malheureusement, l'algorithme n'enlève qu'un neurone de la couche d'entrée. Il est impossible de comprendre le fonctionnement du réseau en observant l'enchevêtrement des 717 connexions.

Seul l'algorithme *Skeletonization* permet de sélectionner un sous-ensemble des entrées du réseau :

- Le réseau exploite au moins une information (position ou hauteur) relative à chacun des maxima des projections horizontale, verticale et diagonale 45°. Il n'utilise quasiment plus la projection sur la diagonale 135°.
- L'algorithme d'élagage élimine les entrées relatives au pourcentage du contour composé de segments horizontaux.

Nous n'avons cependant pas effectué assez de simulations pour confirmer ces observations.

Le réseau obtenu lors de la cinquième expérience possède des capacités de généralisation très légèrement supérieures. Quelques cycles d'apprentissage supplémentaires permettraient certainement d'obtenir les mêmes résultats avec le réseau de 717 connexions.

Remarquons finalement qu'*OBD* nécessite un nombre important de cycles d'apprentissge entre deux élagages successifs[2]. De telles expériences nécessitent un temps prohibitif (plusieurs semaines sur une station *Dec Alpha 130 Mhz*).

[2] Nous effectuons environ dix cycles d'entraînement entre deux élagages avec *Smallest weights*. *OBD* nécessite jusqu'à cent cycles afin de fournir des résultats intéressants !

Exp.	Elagage	Topologie	Connexions	Apprentissage	Validation
1	*OBD*	40-41-10	1249	99.16%	96.26%
2	*Autoprune*	**39-50-10**	**717**	**99.08%**	**96.36%**
3	*Skeletonization*	29-34-10	1326	98.34%	96.0%
4	*Smallest weights*	40-50-10	1431	99.07%	96.28%
5	*Smallest weights*	40-50-10	1274	99.1%	96.42%
6	*Smallest weights*	40-50-10	1197	99.09%	96.18%

Tableau 18.2: Topologies et performances de quelques réseaux obtenus après élagage.

Chapitre 19

Analyse en composantes principales

19.1 Calcul des axes principaux

Les vecteurs propres associés aux m plus grandes valeurs propres de la matrice de covariance $C = (c_{ij})$ de la distribution déterminent les axes principaux. Nous travaillons avec des images prétraitées, normalisées (256 pixels sur 256) et lissées. Nous obtenons ainsi une matrice de covariance 256×256 définie par :

$$c_{ij} = \frac{1}{N} \cdot \sum_{\rho=1}^{N} (\xi_i^\rho - \overline{\xi_i}) \cdot (\xi_j^\rho - \overline{\xi_j}) \tag{19.1}$$

où N dénote le nombre de motifs d'apprentissage et

$$\overline{\xi_i} = \frac{1}{N} \sum_{\rho=1}^{N} \xi_i^\rho \tag{19.2}$$

La matrice étant symétrique, nous appliquons l'algorithme de Jacobi [48] afin de résoudre le problème de valeurs propres.

19.2 *APEX*

Les expériences effectuées avec *APEX* ne sont pas concluantes. Nous obtenions des résultats beaucoup plus rapidement en optimisant le calcul de la matrice de covariance et des vecteurs propres. Supposons que nous doublions le nombre de motifs d'apprentissage. Nous augmentons évidemment le temps de calcul de $C = (c_{ij})$. Le nombre d'opérations intervenant dans le calcul des valeurs propres n'est pas modifié. Nous multiplions cependant par deux la durée de chaque cycle d'apprentissage d'*APEX* ...

19.3 Performances du système

Les tableaux 19.1 et 19.2 présentent les performances du système en fonction de la dimension m de l'espace de projection. Lors des premières simulations (tableau 19.1), nous avons normé les vecteurs d'apprentissage et de validation. Nous obtenons ainsi une meilleure généralisation. L'analyse de ces résultats suggère que les projections sur les axes 33 à 48 n'apportent aucune information[1].

Topologie	m	Apprentissage	Validation
8-50-10	8	93.45%	89.91%
16-50-10	16	98.76%	94.32%
32-50-10	32	99.29%	96.06%
32-100-10	32	99.37%	96.02%
48-50-10	48	99.27%	95.62%
64-50-10	64	99.28%	96.30%

Tableau 19.1: Résultats obtenus avec l'analyse en composantes principales (1). Les vecteurs d'apprentissage et de validation sont normés. m désigne la dimension intrinsèque.

Topologie	m	Apprentissage	Validation
8-50-10	8	93.52%	89.91%
16-50-10	16	98.84%	93.15%
32-50-10	32	99.32%	95.34%
48-50-10	48	99.43%	95.36%
64-50-10	64	99.55%	95.52%

Tableau 19.2: Résultats obtenus avec l'analyse en composantes principales (2). Les vecteurs d'apprentissage et de validation ne sont pas normés. m désigne la dimension intrinsèque.

Nous souhaitions encore effectuer diverses expériences :

- En augmentant la dimension intrinsèque m lors d'expériences successives, nous pourrions déterminer le nombre optimal de projections. Théoriquement, pour $m = 256$, nous devrions obtenir les mêmes résultats qu'en présentant au réseau des images de seize pixels sur seize.

- L'élagage du réseau optimal à l'aide de l'algorithme *Skeletonization* permettrait certainement d'éliminer les entrées superflues[2]. Nous obtiendrions ainsi un réseau comportant m^* entrées.

[1] Les projections sur ces axes ne permettent certainement plus la distinction des différentes classes (figure 14.4)

[2] Une première simulation (avec $m = 64$) a fourni des résultats encourageants.

- Finalement, nous souhaiterions entraîner un réseau à l'aide de projections dans un espace de dimension m^* déterminé par l'analyse discriminante. Nous devrions observer des performances analogues à celles de l'expérience précédente.

Partie IV

Conclusion

Chapitre 20

Performances des systèmes réalisés

Nous avons utilisé diverses méthodes d'extractions de caractéristiques lors de nos simulations. C'est en présentant l'image prétraitée, normalisée (seize pixels sur seize) et lissée au réseau que nous obtenons les meilleurs résultats. Cette approche nécessite cependant un temps de calcul considérable lors de l'apprentissage et de l'élagage. Une semaine de calculs (sur une station *Dec Alpha 130 Mhz*) s'avéra ainsi nécessaire afin de déterminer chacun des réseaux comportant 46 entrées. Remarquons toutefois que le processus nécessitait une importante quantité de mémoire (environ 25 Mb). Lorsqu'un autre utilisateur exécutait un programme réclamant les mêmes ressources, la machine utilisait son *swap space*, ralentissant ainsi l'exécution des programmes.

Les autres expériences n'excédaient que rarement les 24 heures de calcul (ce temps dépendait toujours du nombre de processus exécutés sur la machine). Les performances observées étant quasiment identiques, il est difficile de déterminer quel est le meilleur algorithme d'extractions de caractéristiques.

- **Projections** *VH2D*.
  Cet algorithme d'extractions se révèle extrêmement rapide lors de réalisations matérielles [21]. Son principal inconvénient réside dans le nombre élevé d'entrées du réseau. Combiné à l'algorithme de rétropropagation non linéaire, il permettrait la réalisation d'un processeur extrêmement rapide spécialisé dans la reconnaissance de caractères manuscrits.

- **Recherche des maxima des projections** *VH2D* **et informations sur le contour.**
  Cette méthode se révèle la plus coûteuse en temps de calcul. Il serait évidemment possible d'analyser le contour et les diverses projections en parallèle. Remarquons encore que l'algorithme d'extraction de caractéristiques repose sur de multiples observations et ne connaît aucune justification mathématique.

- **Analyse en composantes principales.**
  Une fois la matrice de projection W déterminée, l'extraction de caractéristiques s'avère très rapide. Il suffit de multiplier par W le vecteur codant les niveaux de gris des pixels

de l'image prétraitée, normalisée et lissée. Remarquons que cette opération correspond à l'adjonction d'une couche au réseau neuromimétique[1] (figure 20.1).

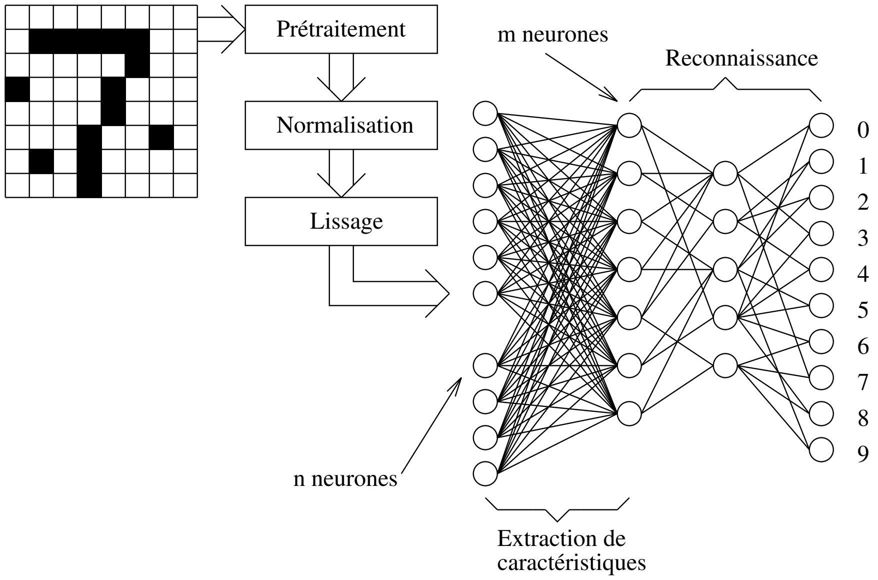


Figure 20.1: Adjonction d'une couche de connexions effectuant l'extraction de caractéristiques (analyse en composantes principales)

Une fois l'apprentissage achevé, la reconnaissance se révèle extrêmement rapide. Actuellement, nous utilisons *SNNS* lors de la validation[2]. Les coefficients synaptiques étant déterminés, nous pourrions écrire un petit programme implantant le réseau de neurones. Soient W_1 et W_2 les matrices contenant les poids des connexions reliant respectivement les entrées à la couche cachée et la couche cachée aux sorties. Soit encore $\boldsymbol{a}^\rho = [a^\rho_{L,1}, \ldots, a^\rho_{L,10}]^T$ le vecteur contenant les activations des neurones de la couche de sortie. La propagation des signaux est déterminée par l'équation :

$$\boldsymbol{a}^\rho = \varphi(W_2 \cdot \varphi(W_1 \cdot \boldsymbol{\xi}^\rho)) \tag{20.1}$$

où la fonction d'activation φ est appliquée à chacune des composantes d'un vecteur. Il suffit de programmer un algorithme de multiplication matricielle. Nous obtiendrions ainsi les résultats encore plus rapidement.

20.1 Amélioration des résultats

Lors de notre analyse des données de validation, nous avons constaté que certaines écritures, très particulières, n'étaient pas correctement reconnues par le système. En complétant judicieusement notre base d'apprentissage, il serait certainement possible d'améliorer les performances des réseaux neuronaux. Il faudrait observer les écritures figurant dans la base actuelle

[1] Il s'agit de la couche de connexions excitatrices du modèle *APEX*.

[2] Les structures de données de ce logiciel sont relativement complexes et comportent énormément de pointeurs.

et n'ajouter que des styles différents. Une autre solution consisterait à étudier les particularités des chiffres n'étant pas correctement reconnus. Nous compléterions alors la base de données avec des caractères du même type.

Le réseau se révèle incapable d'apprendre des motifs fortement bruités. Leur suppression ne perturbe pas les performances du réseau et réduit la durée de l'apprentissage (chapitre 18). Nous pourrions ainsi compenser l'augmentation du temps de calcul liée à l'adjonction de nouveaux motifs d'entraînement.

20.2 Segmentation et reconnaissance

Nos systèmes ne reconnaissent aucun caractère fortement bruité. Etudions l'image de la figure 20.2. Représente-t-elle “9” ou “91” ? Nous pourrions résoudre ce problème en effectuant simultanément la segmentation et la reconnaissance. Ce procédé permettrait de repérer le “9”, puis de constater que l'image comporte encore de l'information.

Figure 20.2: Un chiffre fortement bruité

20.3 Performances de l'algorithme de rétropropagation non linéaire

Nous avons utilisé cet algorithme lors de l'entraînement à partir des maxima et des orientations et obtenu des performances identiques à celles de l'algorithme de *backpropagation*. Ce résultat confirme l'intérêt de cette règle d'apprentissage[3]. Il serait intéressant de développer des algorithmes d'élagage spécifiques à la rétropropagation non linéaire, puis de les implanter en *hardware*.

20.4 Performances des algorithmes d'élagage

Parmi les différentes méthodes utilisées, seuls deux algorithmes permettent de réduire notablement la taille des réseaux sans altérer leurs performances :

[3]Comme l'article de Hertz [27] ne propose qu'une application, nous avons testé l'algorithme avec divers problèmes et obtenu de très bons résultats [5].

- *Autoprune* élimine un grand nombre de connexions. Cependant, lors des expériences effectuées, l'algorithme n'élaguait aucun neurone. Nous obtenions ainsi un enchevêtrement de connexions impossible à analyser.
- L'algorithme *Skeletonization* sélectionne un sous-ensemble des entrées. L'observation des réseaux obtenus permet d'émettre quelques hypothèses concernant le fonctionnement des réseaux. Il serait intéressant d'appliquer des algorithmes d'extraction de règles (*rule extraction*) [35] afin de les confirmer.

Les autres algorithmes proposent des réseaux nettement plus complexes ou nécessitent un important nombre de cycles d'entraînement entre deux élagages.

20.5 Les *high order perceptrons*

Les expériences effectuées avec des *high order perceptrons* n'ont pas fourni les résultats escomptés. Le principal problème réside dans le temps de calcul nécessaire. L'utilisation d'algorithmes constructifs nous semble indispensable afin d'entraîner un réseau et de découvrir une topologie optimale dans un temps raisonnable.

Chapitre 21

Comparaison avec d'autres méthodes

Divers chercheurs proposent des systèmes de reconnaissance de chiffres manuscrits. Nous présentons ici les résultats de deux travaux et les comparons aux nôtres.

21.1 *"Handwritten Digit Recognition by Neural Networks with Single-Layer Training"*

Cet article [46] présente les travaux de S. Knerr et de ses collègues. L'originalité de la solution réside dans l'architecture du réseau de neurones utilisé. Knerr a démontré que tout problème de classification défini dans $\mathbb{R}^N$ est décomposable en plusieurs problèmes linéairement séparables. Il suffit dès lors d'entraîner des réseaux monocouches.

Knerr propose une variante de l'algorithme *LMS* (paragraphe 5.2) déterminant un hyperplan séparateur, quel que soit le problème linéairement séparable[1]. Une fois les réseaux entraînés, les fonctions d'activation de type sigmoïde sont remplacées par des fonctions "Seuil" fournissant des valeurs booléennes. Le système de reconnaissance s'obtient finalement en combinant les sorties des neurones avec des portes logiques *AND* et en intégrant un mécanisme de rejet dépendant d'un seuil μ.

Deux bases de données interviennent lors des simulations. Pour chacune d'elles, Knerr a utilisé cinq partitions différentes en données d'apprentissage et de validation et a moyenné les résultats des simulations.

- La base française contient 8700 chiffres écrits par 13 collègues de Knerr.
- 9000 chiffres extraits de *U.S. Postal Service OAT Handwritten Zip Code Database* (base de données des services postaux américains) constituent la seconde base de données.

Knerr extrait le contour des caractères prétraités et normalisés (huit pixels sur huit) à l'aide de masques de Kirsch [38] et obtient quatre images, codant respectivement la présence de verticales, d'horizontales ou de diagonales (45° ou 135°). Il effectue aussi des expériences en présentant au réseau des images prétraitées et normalisées (seize pixels sur seize). Les

[1] Nous avons présenté les inconvénients de l'algorithme *LMS* au paragraphe 5.2.

tableaux 21.1 et 21.2 présentent les performances des systèmes conçus pour les deux bases de données. Le système neuromimétique se compose de 45 réseaux monocouches comportant chacun 256 entrées. Dix portes logiques *AND* constituent les sorties du dispositif.

	Image 16×16	Extraction du contour
Apprentissage ($\mu = 0$)	99.3% / 0.1% / 0.6%	99.6% / 0.1% / 0.3%
Validation ($\mu = 0$)	97.6% / 0.7% / 1.7%	97.7% / 0.4% / 1.9%
Validation ($\mu = 0.3$	95.1% / 3.9% / 1.0%	96.3% / 2.7% / 1.0%

Tableau 21.1: Résultats obtenus avec la base de données française (% de motifs correctement traités / % de motifs rejetés / % de classifications erronées)

	Image 16×16	Extraction du contour
Apprentissage ($\mu = 0$)	98.5% / 0.5% / 1.0%	98.9% / 0.3% / 0.8%
Validation ($\mu = 0$)	93.5% / 2.4% / 4.1%	**96.5%**/ **1.0%** / **2.5%**
Validation ($\mu = 1.2$)	70.9% / 28.1% / 1.0%	
Validation ($\mu = 0.4$)		90.3% / 8.7% / 1.0%

Tableau 21.2: Résultats obtenus avec la base de données des services postaux américains (% de motifs correctement traités / % de motifs rejetés / % de classifications erronées).

Les performances du modèle de Knerr dépendent de la base de données utilisée. Cette constatation suggère une grande prudence lors de la comparaison avec nos résultats. De plus, le mécanisme de rejet est différent du nôtre. La difficulté du problème *U.S. Postal Service OAT Handwritten Zip Code Database* nous semble cependant identique à celle de *NIST* (divers exemples de chiffres sont présentés dans l'article). Nous proposons par conséquent quelques remarques :

- Lors des expériences réalisées avec $\mu = 0$, la méthode d'extraction de caractéristiques de Knerr fournit des résultats similaires aux nôtres. L'architecture du réseau se révèle cependant nettement plus complexe (11520 connexions). Rappelons que nous résolvons le problème à l'aide d'un *perceptron* multicouche partiellement connecté (717 connexions).

- Notre mécanisme de rejet, ainsi que celui de Knerr, permettent l'obtention d'un taux d'erreur inférieur ou égal à un pour-cent. Rappelons qu'il s'agit des performances actuellement exigées par les services postaux.

21.2 *"Experiments with robust similarity measures for OCR"*

L'objectif de ce projet, réalisé par T. M. Breuel et G. Maître [31], consistait à développer des méthodes de reconnaissance de chiffres manuscrits, peu sensibles aux dégradations et distorsions. T. M. Breuel et G. Maître proposent une représentation des caractères prétraités, normalisés et lissés basée sur leur contour. Ils ont constitué un ensemble de 534 prototypes à l'aide de leur algorithme d'extraction de caractéristiques.

Le processus de classification comporte les étapes suivantes (figure 21.1) :

- Après le prétraitement du caractère inconnu, sa représentation, basée sur le contour, est déterminée.
- Cette dernière est alors comparée à tous les prototypes de la base de données intégrée au système. Les informations ainsi récoltées sont fournies à un algorithme des k plus proches voisins (*k-Nearest Neighbours*) [40]. Ce dernier détermine la classe à laquelle appartient le chiffre.

Nous invitons le lecteur intéressé par les détails à se référer à [31].

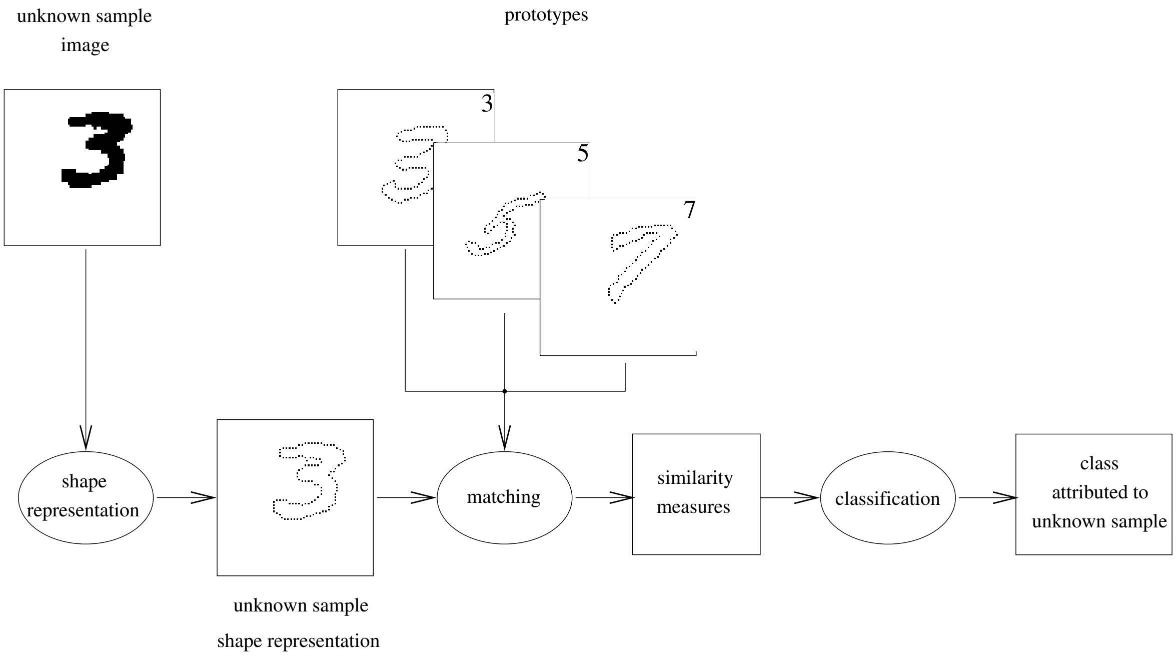


Figure 21.1: Le système de reconnaissance d'écriture proposé par T. M. Breuel et G. Maître(cette image est issue de [31])

Nous avons utilisé exactement les mêmes bases de données que T. M. Breuel et G. Maître[2] afin de comparer les performances des systèmes respectifs. Leur méthode se révèle très performante et reconnaît 98.37% des données de validation. Rappelons que notre meilleur réseau ne classifiait correctement que 97.39% de ces données.

Le temps de calcul constitue l'inconvénient majeur du système de T. M. Breuel et G. Maître (environ 90 secondes pour reconnaître un caractère sur une *Sparc IPX*). Notre système se révèle évidemment nettement plus rapide.

[2] Ils ont développé un algorithme d'apprentissage constituant un ensemble de prototypes à partir de la base de données contenant 15025 caractères.

Chapitre 22

Vers un système de reconnaissance de textes

La figure 22.1 illustre le schéma général d'un système de reconnaissance d'écriture.

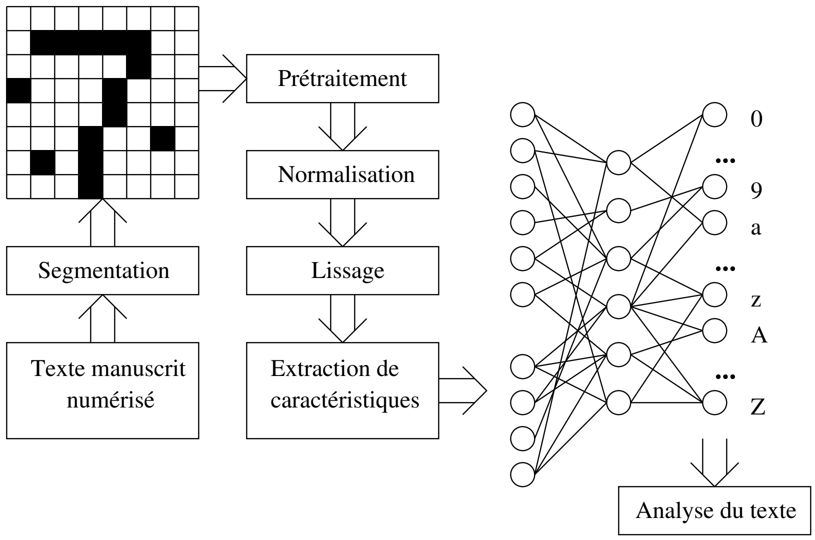


Figure 22.1: Schéma général du système de reconnaissance d'écriture complet

22.1 Reconnaissance de caractères quelconques

Notre système ne reconnaît actuellement que des chiffres. Il suffit de compléter notre base d'apprentissage avec des exemples de lettres minuscules et majuscules. Quelques expériences seraient certainement nécessaires afin de déterminer la topologie du réseau[1]. *NIST* ne contient malheureusement aucun caractère accentué. Afin de traiter des textes français, espagnols ou suédois, nous devrions constituer une nouvelle base de données.

[1]Nous conseillons l'utilisation d'un *perceptron* multicouche ou d'un algorithme constructif proposant l'architecture d'un *high order perceptron*.

Lors du prétraitement, il paraît judicieux d'éviter d'éliminer le bruit. Cet algorithme détruirait probablement les accents, et signes de ponctuation. Citons encore quelques difficultés :

- Une barre verticale surmontée d'un amas de pixels connaît plusieurs interprétations :
  - la lettre "i" ou
  - le chiffre "1" bruité.
- Si le texte étudié est rédigé en anglais, nous considérons un groupe de pixels surmontant une lettre comme du bruit. Cependant, cette méthode ne s'applique pas à des langues telles que le français ou l'italien. Nous devons discerner les accents du bruit. Les signes de ponctuation offrent les mêmes difficultés, quelle que soit la langue.

La meilleure solution à ces divers problèmes consisterait à effectuer simultanément l'élimination du bruit, la segmentation et la reconnaissance.

22.2 Reconnaissance d'expressions mathématiques

Les textes mathématiques offrent d'autres difficultés :

- reconnaissance des lettres grecques, abondamment utilisées;
- traitement de symboles graphiques particuliers (flèches surmontant les vecteurs, intégrales, dérivées partielles, opérateurs);
- transcription de ces divers symboles dans un formalisme aisément compréhensible par l'être humain.

R. Fateman, T. Tokuyasu, B. P. Berman, N. Mitchell ont réalisé un système adapté au traitement d'expressions mathématiques [36], transcrivant le texte manuscrit en un document LaTeX.

22.3 Réalisation d'un système de correction orthographique

La parfaite reconnaissance de caractères s'avère impossible. Diverses méthodes permettent la correction des erreurs [42] :

- L'utilisation d'un dictionnaire permet la détection d'un mot erroné. Lorsque plusieurs alternatives sont possibles, l'intervention d'un être humain est indispensable.
- Les méthodes sémantiques nous semblent appropriées lors de l'étude de documents scientifiques mais ne s'appliquent que partiellement à des oeuvres littéraires. De tels algorithmes corrigeraient par exemple les syllepses, modifiant ainsi le texte original.
- Certains chercheurs suggèrent des méthodes statistiques. Le principe consiste à calculer la probabilité qu'une suite de n caractères soit correcte.

Nous souhaitons présenter quelques difficultés surgissant lors de la réalisation d'un tel système. Bien qu'écrit en français, ce document comporte deux citations (paragraphes 3.2 et 14.2) et divers termes anglais. L'algorithme de correction doit impérativement déterminer la langue à laquelle appartient un ensemble de mots[2]. Certaines personnes suggéreront d'utiliser conjointement les dictionnaires de plusieurs langues, par exemple le français et l'anglais. Cette solution simpliste n'est évidemment pas satisfaisante. Supposons que nous rencontrions le mot "abreviation". Notre système proposera les alternatives "abréviation" et "*abbreviation*" ... La recherche dans plusieurs dictionnaires se révélera particulièrement lente. Les méthodes statistiques sont évidemment liées à une langue spécifique. Par exemple, la séquence "zer", fréquente en allemand n'apparaît que rarement en français.

Les textes historiques offrent une difficulté supplémentaire. Les dictionnaires et méthodes statistiques sont incapables de traiter les dates. Une base de données de faits historiques permettrait de résoudre ce délicat problème. Nous pourrions ainsi repérer certains événements en parcourant le texte.

[2]Trois langues apparaissent dans un seul alexandrin de Baudelaire : "*Remember* ! Souviens-toi, prodigue ! *Esto memor* !" (L'horloge). Déterminer automatiquement à quelles langues appartiennent ces divers mots nous paraît très difficile. Nous ne disposons d'aucun accent ou caractère particulier à une langue (par exemple, une cédille sous un "c") dans cet extrait. Le problème est plus simple lorsqu'au moins une phrase est écrite dans une langue étrangère.

Annexe A

L'algorithme de *backpropagation* : calculs détaillés

Dans ce chapitre, nous présentons les calculs permettant d'obtenir la règle d'apprentissage présentée au chapitre 8. L'erreur quadratique en sortie s'écrit (équation 8.6) :

$$E = \frac{1}{2} \sum_{\rho} \sum_{j=1}^{N_L} \left(t_j^{\rho} - O_j^{\rho} \right)^2 \tag{A.1}$$

où

$$\begin{aligned} O_j^{\rho} &= \varphi_{L,j} \left(h_{L,j} \right) \\ &= \varphi_{L,j} \left(\sum_{i=0}^{C_l} w_{L,i,j} \cdot c^{\rho}_{L,i,\{(l_1,n_1),(l_2,n_2),\ldots\}} \right) \end{aligned} \tag{A.2}$$

Calculons la dérivée de cette expression par rapport au poids $w_{L,i,j}$:

$$\frac{\partial E}{\partial w_{L,i,j}} = - \sum_{\rho} \left(t_j^{\rho} - O_j^{\rho} \right) \cdot \varphi'_{L,j} \left(h^{\rho}_{L,j} \right) \cdot c^{\rho}_{L,i,\{(l_1,n_1),(l_2,n_2),\ldots\}} \tag{A.3}$$

Nous définissons ainsi

$$\delta^{\rho}_{L,j} = \varphi'_{L,j} \left(h^{\rho}_{L,j} \right) \cdot \left(t_j^{\rho} - O_j^{\rho} \right) \tag{A.4}$$

Dérivons maintenant l'équation A.2 par rapport à $w_{L-1,k,i}$. Le réseau comporte ainsi une couche cachée. Par hypothèse, ce dernier ne possède que des connexions de premier ordre.

Ainsi, $c^{\rho}_{l,j} = a^{\rho}_{l-1,j}$. Cette constatation permet de simplifier le calcul de la dérivée.

$$\frac{\partial E}{\partial w_{L-1,k,i}} = -\sum_{\rho}\sum_{j=1}^{N_L}\left(t^{\rho}_j - O^{\rho}_j\right)\cdot\frac{\partial O^{\rho}_{L,j}}{\partial h^{\rho}_{L,j}}\cdot\frac{\partial h^{\rho}_{L,j}}{\partial c^{\rho}_{L,i}}\cdot\frac{\partial c^{\rho}_{L,i}}{\partial h^{\rho}_{L-1,i}}\cdot\frac{\partial h^{\rho}_{L-1,i}}{\partial w_{L-1,k,i}} \tag{A.5}$$

$$= -\sum_{\rho}\sum_{j=1}^{N_L}\left(t^{\rho}_j - O^{\rho}_j\right)\cdot\varphi'_{L,j}\left(h^{\rho}_{L,j}\right)\cdot w_{L,i,j} \cdot\varphi'_{L-1,i}\left(h^{\rho}_{L-1,i}\right)\cdot c^{\rho}_{L-1,k}$$

Nous déduisons de l'équation ci-dessus que :

$$\delta^{\rho}_{l,j} = \varphi'_{l,j}\left(h^{\rho}_{l,j}\right)\cdot\sum_{k=1}^{N_{l+1}}\delta^{\rho}_{l+1,k}w_{l+1,j,k} \tag{A.6}$$

Nous obtenons ainsi la règle d'apprentissage proposée au chapitre 8 :

$$\delta^{\rho}_{m,j} = \begin{cases} \varphi'_{m,j}\left(h^{\rho}_{m,j}\right)\cdot\left(t^{\rho}_j - O^{\rho}_j\right) & \text{si } m = L \\ \\ \varphi'_{m,j}\left(h^{\rho}_{m,j}\right)\cdot\displaystyle\sum_{k=1}^{N_{m+1}}\delta^{\rho}_{m+1,k}w_{m+1,j,k} & \text{si } 2 \leq m < L \end{cases} \tag{A.7}$$

Annexe B

Modification du test statistique proposé par Finnoff en vue d'une implantation efficace d'*Autoprune*

L'implantation de l'équation 10.8 :

$$T(w_{l,i,j}) = \log \frac{\left| \sum_{\rho=1}^{n} \left(w_{l,i,j} + \eta \cdot \delta_{l,j}^{\rho} \cdot c_{l,i}^{\rho} \right) \right|}{\eta \sqrt{\sum_{\rho=1}^{n} \left(\overline{\delta_{l,j} \cdot c_{l,i}} - \delta_{l,j}^{\rho} \cdot c_{l,i}^{\rho} \right)^2}}$$

s'avère problématique. Cette formule nécessite en effet la mémorisation de tous les termes $\delta_{l,j} \cdot c_{l,i}$. Une expérience effectuée dans ce projet consiste à présenter des images de 256 pixels au réseau. Nous utilisons un *multilayer perceptron* comportant 128 neurones sur la couche cachée. Le réseau est ainsi composé de 34048 connexions. La base d'apprentissage étant constituée de 15025 caractères, il faut mémoriser $5.1157 \cdot 10^8$ valeurs. En supposant maintenant que nous utilisions 4 octets pour chacune de ces dernières, nous nécessitons $2.0461 \cdot 10^9$ octets. Une telle expérience est inconcevable sur les ordinateurs actuels. La formule 10.8 nécessite de plus deux parcours des valeurs mémorisées :

- calcul de la moyenne $\overline{\delta_{l,j} \cdot c_{l,i}}$
- calcul du numérateur et du dénominateur

En appliquant la formule ci-dessous à l'équation 10.8,

$$\sqrt{\sum_{i=1}^{n} (x_i - \overline{x})^2} = \sqrt{\frac{n \cdot \sum_{i=1}^{n} (x_i)^2 - \left(\sum_{i=1}^{n} x_i \right)^2}{n}} \tag{B.1}$$

nous obtenons :

$$T(w_{l,i,j}) = \log \frac{\left| n \cdot w_{l,i,j} + \eta \cdot \sum_{\rho=1}^{n} \left(\delta_{l,j}^{\rho} \cdot c_{l,i}^{\rho} \right) \right|}{\eta \sqrt{\dfrac{n \cdot \sum_{\rho=1}^{n} \left(\delta_{l,j}^{\rho} \cdot c_{l,i}^{\rho} \right)^2 - \left(\sum_{\rho=1}^{n} \left(\delta_{l,j}^{\rho} \cdot c_{l,i}^{\rho} \right) \right)^2}{n}}} \tag{B.2}$$

Cette formule permet d'effectuer la somme des $\delta_{l,j}^{\rho} \cdot c_{l,i}^{\rho}$ ainsi que celle des $\left(\delta_{l,j}^{\rho} \cdot c_{l,i}^{\rho} \right)^2$ durant l'apprentissage. La mémoire nécessaire à notre exemple est de $34048 \cdot 2 \cdot 4 = 2.7238 \cdot 10^5$ octets. Nous avons implanté l'équation B.2 dans *SNNS*.

Annexe C

Quelques chiffres de la base de validation correctement reconnus

Nous présentons quelques motifs de la base de validation correctement classifiés par le réseau décrit au paragraphe 16.2.

Figure C.1: Seize “0” de la base de validation

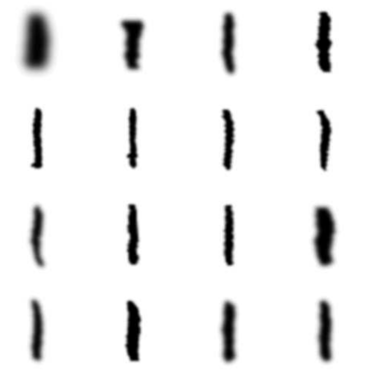


Figure C.2: Seize “1” de la base de validation

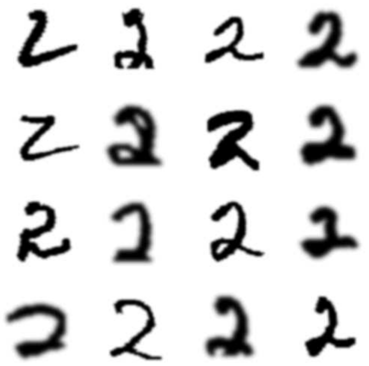


Figure C.3: Seize “2” de la base de validation

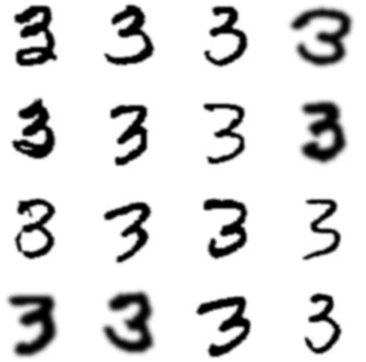


Figure C.4: Seize “3” de la base de validation

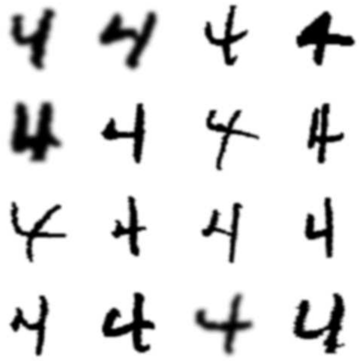


Figure C.5: Seize “4” de la base de validation

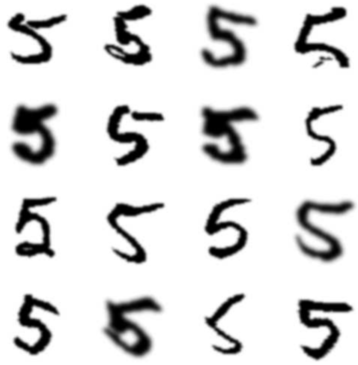


Figure C.6: Seize “5” de la base de validation

Figure C.7: Seize “6” de la base de validation

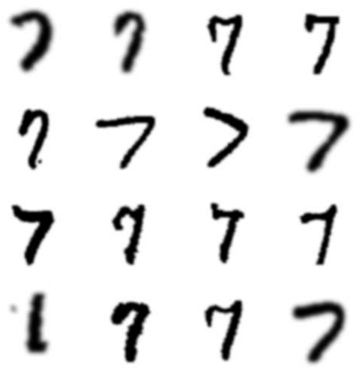


Figure C.8: Seize “7” de la base de validation

Figure C.9: Seize “8” de la base de validation

Figure C.10: Seize “9” de la base de validation

Bibliographie

[1] UCI Repository of Machine Learning Databases and Domain Theories. ftp://ics.uci.edu/pub/machine-learning-databases.

[2] M. A. Lehr B. Widrow. 30 Years of Adaptative Neural Networks : Perceptron, Madaline and Backpropagation. *Proc. IEEE*, 78(9):1415–1442, September 1990.

[3] B. Widrow, M. E. Hoff. Adaptative Switching Circuits. *IRE Western Electric Show and Convention Record*, pages 96–104, 1960.

[4] Jean-Luc Beuchat. Optimisation de réseaux neuronaux. Technical report, IDIAP, Av. du Simplon 4, 1920 Martigny, Switzerland, October 1995.

[5] Jean-Luc Beuchat. Réalisation de réseaux neuronaux en logique stochastique. Technical report, Laboratoire de Systèmes Logiques, Ecole Polytechnique Fédérale de Lausanne, June 1996.

[6] C. Fyfe, R. Baddeley. Non-Linear Data Structure Extraction using Simple Hebbian Networks. *Biological Cybernetics*, 72(6):533–541, 1995.

[7] R. Cairoli. *Algèbre linéaire*. Presses Polytechniques et Universitaires Romandes, 1991.

[8] B. N. Chatterji. Feature Extraction Methods for Character Recognition. *IETE Technical Review*, 3(1):9–22, 1986.

[9] D. O. Hebb. *The Organization of Behavior*. Wiley, 1949.

[10] B. Hassibi D. Stork. Second Order Derivatives for Network Pruning : Optimal Brain Surgeon. *Advances in Neural Information Processing Systems*, 5:164–171, 1993.

[11] F. Blayo, M. Verleysen. *Les Réseaux de Neurones Artificiels*. Presses Universitaires de France, 1996. Que sais-je ? 3042.

[12] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, 1962.

[13] E. Fiesler. Neural Network Classification and Formalization. Preprint available from IDIAP, 1994. http://www.idiap.ch/idiap-othertext.html.

[14] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, Inc., 1990.

[15] G. Coray, A. Ballim. Reconnaissance des Formes. Cours polycopié, Laboratoire d'Informatique Théorique (LITH), Ecole Polytechnique Fédérale de Lausanne, 1995.

[16] G. J. Gibson, C. F. N. Cowan. On the Decision Regions of Multilayer Perceptrons. *Proc. IEEE*, 78(10):1590–1594, 1989.

[17] G. Maître, S. Brunet, G. Pante. Traitement préliminaire de l'image d'un texte manuscrit en vue de sa reconnaissance : une méthode de sur-segmentation, 1995.

[18] G. Thimm, E. Fiesler. High Order and Multilayer Perceptron Initialization. Preprint available from IDIAP, 1996. Accepted for publication by IEEE Trans. on Neural Networks, 1996.

[19] G. Thimm, R. Grau, E. Fiesler. Modular Object-Oriented Neural Network Simulators and Topology Generalization. In M. Marinaro and P. G. Morasso, editors, *Proceedings of the International Conference on Artificial Neural Networks (ICANN 94)*, pages 747–750. Springer Verlag, 1994.

[20] F. Gruau. Automatic Definition of Modular Networks. *Adaptative Behaviour*, 3(2):151–183, 1995.

[21] H. D. Cheng, D. C. Xia. A Novel Parallel Approach to Character Recognition and its VLSI Implementation. *Pattern Recognition*, 29(1):97–119, 1996.

[22] S. Haykin. *Neural Networks : A Comprehensive Foundation.* MacMillan College Publishing Company, Inc., 1994.

[23] Institute for Parallel and Distributed High Performance Systems, University of Stuttgart. *SNNS User Manual, Version 4.0.* ftp://ftp.informatik.uni-stuttgart.de/pub/SNNS.

[24] J. Hérault, C. Jutten. *Réseaux neuronaux et traitement du signal.* Hermès, 1994.

[25] R. G. Palmer J. Hertz, A. Krogh. *Introduction to the Theory of Neural Computation.* Addison-Wesley Publishing Company, 1991.

[26] J. Sietsma, R. J. F. Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4(1):67–69, 1991.

[27] Benny Lautrup John Hertz, Anders Krogh and Torsten Lehmann. Non-Linear Back-Propagation : Doing Back-Propagation without Derivatives of the Activation Function. Preprint available from Neuroprose, March 1994. ftp://archive.cis.ohio-state.edu/pub/neuroprose.

[28] K. I. Diamantaras, S. Y. Kung. *Principal Component Neural Networks.* Wiley, 1996.

[29] M. D. Garris, R. A. Wilkinson. NIST Special Database 3 Handwritten Segmented Characters. Technical report, National Institute of Standards and Technology, Advanced Systems Division, Image Recognition Group, February 1992.

[30] O. Maischberger. Die Implementation neuronaler Netze mittels FPGAs, September 1995.

[31] Gilbert Maître. Experiments with robust similarity measures for OCR. Technical Report 95-03, IDIAP, Av. du Simplon 4, 1920 Martigny, Switzerland, June 1995.

[32] Michael C. Mozer, Paul Smolensky. Using Relevance to Reduce Network Size Automatically. *Connection Science*, 1(1):3–16, 1989.

[33] N. Kambhatla, T. K. Leen. Fast Non-Linear Dimension Reduction. In Cowan, Tesauro, Alspector, editor, *Advances in Neural Information Processing Systems*, volume 6. Morgan Kaufmann Publishers, 1994.

[34] Lutz Prechelt. Adaptative Parameter Pruning in Neural Networks. Technical Report 95-009, International Computer Science Institute, Berkley, CA, March 1995.

[35] R. Andrews, J. Diederich, A. B. Tickle. A Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks. Technical report, Neurocomputing Research Center, Queensland University of Technology, Box 2434 GPO Brisbane 4001, Queensland, Australia, 1995.

[36] R. Fateman, T. Tokuyasu, B. P. Berman, N. Mitchell. Optical Character Recognition and Parsing of Typeset Mathematics. Technical report, Computer Science Division, EECS Department, University of California at Berkeley, 1995.

[37] J. Lyall R. H. Davis. Recognition of handwritten characters - a review. *Image and Vision Computing*, 4(4):208–218, 1986.

[38] R. M. Haralick, L. G. Shapiro. *Computer and Robot Vision*, volume 1. Addison-Wesley, 1992.

[39] Russell Reed. Pruning Algorithms - A Survey. *IEEE Transactions on Neural Networks*, 4:740–747, September 1993.

[40] B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[41] F. Rosenblatt. The Perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, (65):386–408, 1958.

[42] S. Impedovo, L. Ottaviano, S. Occhinegro. Optical Character Recognition - A Survey. *International Journal of Pattern Recognition and Artificial Intelligence*, 5(1&2):1–24, 1991.

[43] S. Mori, C. Y. Suen, K. Yamamoto. Historical Review of OCR Research and Development. *Proc. IEEE*, 80(7):1029–1058, 1992.

[44] J. Denker S. Solla, Y. Le Cun. Optimal Brain Damage. *Advances in Neural Information Processing Systems*, 2:598–605, 1990.

[45] A. W. Senior. Off-line Handwriting Recognition: a Review and Experiments. Technical Report CUED/F-INFENG/TR 105, Cambridge University Engineering Department, Cambridge, England, December 1992.

[46] Stefan Knerr, Léon Personnaz, Gérard Dreyfus. Handwritten Digit Recognition by Neural Networks with Single-Layer Training. *IEEE Transactions on Neural Networks*, 3(6):962–968, 1992.

[47] Georg Thimm. *Optimization of High Order Perceptrons*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 1997. Proposed at the Signal Processing Laboratory.

[48] W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1988.

[49] W. McCulloch, W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bull. Math. Bioph.*, 3:115–133, 1943.

[50] William Finnoff, Ferdinand Hergert, Hans Georg Zimmermann. Improving model selection by nonconvergent methods. *Neural Networks*, 6:771–783, 1993.

Liste des figures

Liste des tableaux

Table des matières