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.

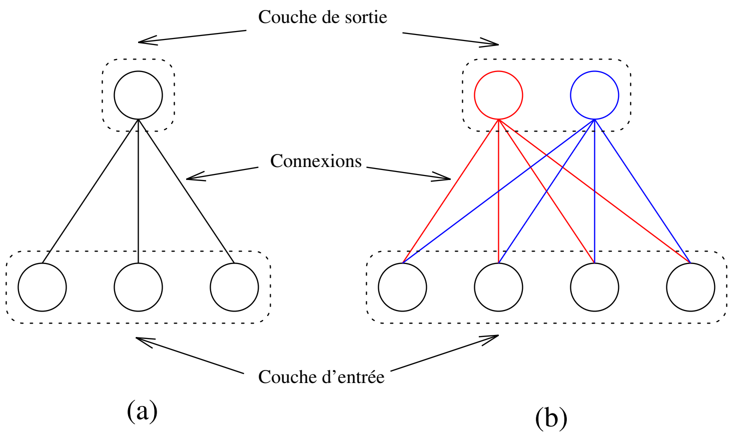

Figure 5.1: Deux réseaux monocouches. (a) Le réseau de gauche, adapté à l'apprentissage de fonctions booléennes, ne possède qu'un neurone de sortie. (b) Nous utiliserions par contre le réseau de droite pour un problème de classification. Remarquons qu'il se décompose en deux réseaux ne comportant qu'une 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 $\boldsymbol{\xi}^\rho = [\xi_1^\rho, \ldots, \xi_N^\rho]^T$, où

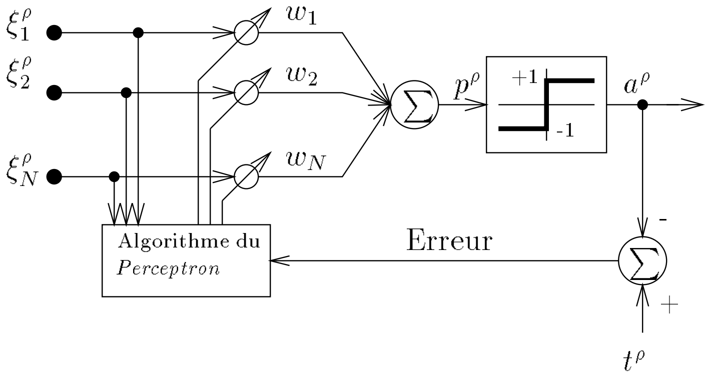

Figure 5.4: Le modèle du *Perceptron*

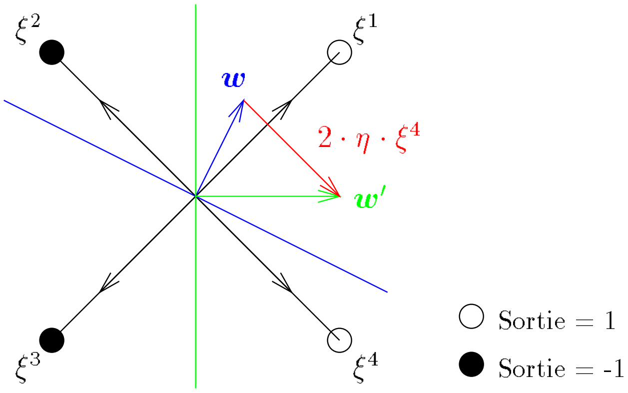

Figure 5.5: Illustration du fonctionnement de la règle du *Perceptron*. Le vecteur $\boldsymbol{w}$ est initialisé aléatoirement. Nous présentons alors au réseau les motifs $\boldsymbol{\xi}^1$, $\boldsymbol{\xi}^4$, $\boldsymbol{\xi}^3$ et finalement $\boldsymbol{\xi}^2$. Le vecteur $\boldsymbol{\xi}^1$ 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.

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 :

- 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.

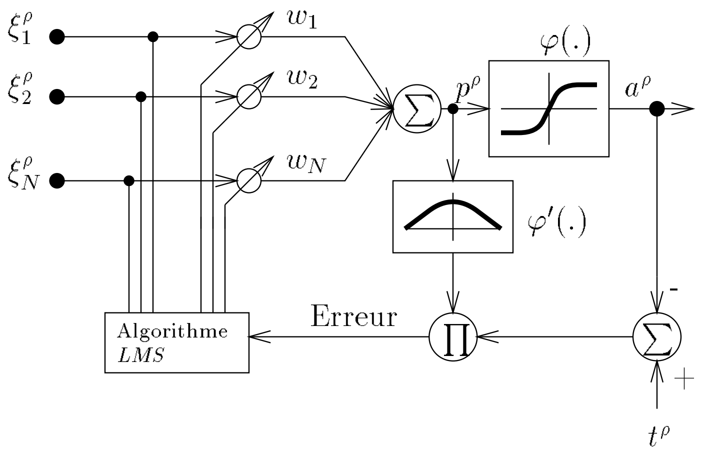

Figure 5.6: Le modèle de l'*Adaline* non linéaire

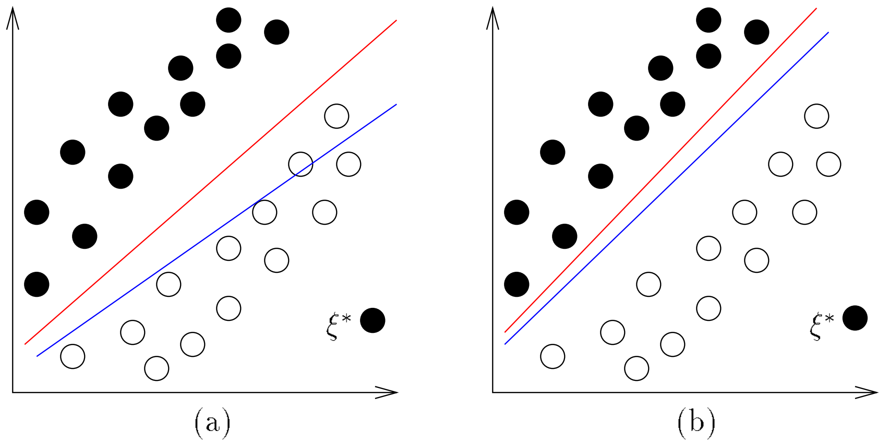

Figure 5.7: Effets de la non-linéarité. Les points noirs (respectivement blancs) appartiennent à la classe A (respectivement B).

[5] 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 $E^\rho \neq 0$.

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 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.

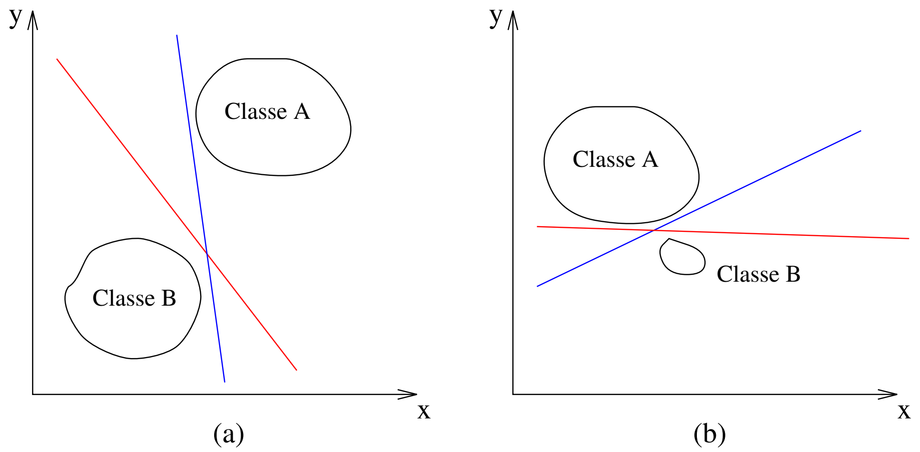

Figure 5.8: Solutions obtenues avec la règle du *Perceptron* dans les situations illustrées par la figure 5.2. Les lignes de différentes couleurs indiquent quelques-unes des solutions du problème.

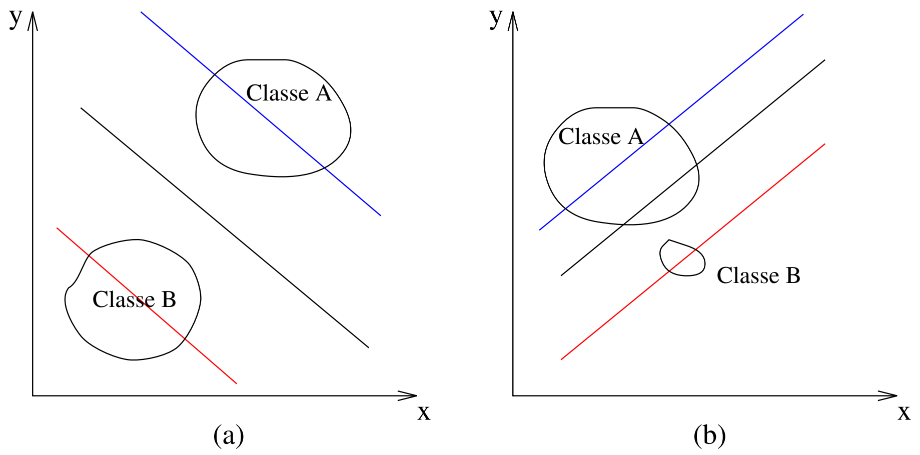

Figure 5.9: Solutions obtenues avec l'*Adaline* dans les situations illustrées par la figure 5.2. La ligne bleue (respectivement rouge) désigne la droite des moindres carrés de la classe A (respectivement B).

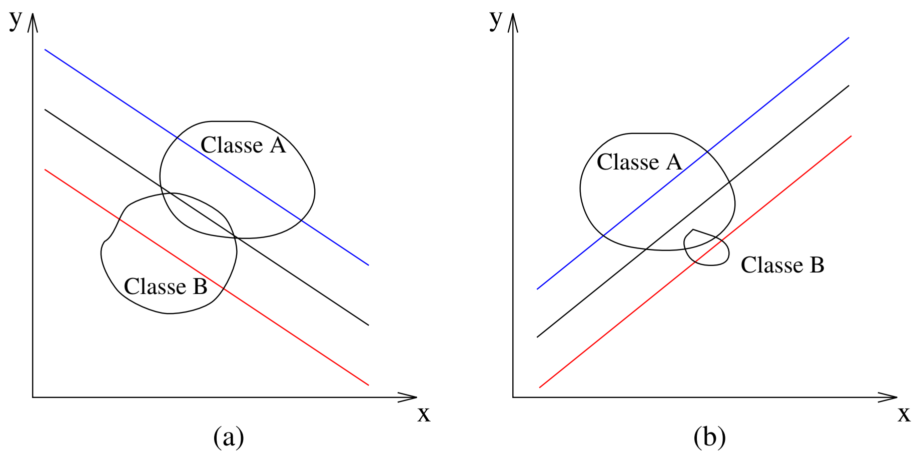

Figure 5.10: Solutions obtenues avec l'*Adaline* dans les situations illustrées par la figure 5.3. La ligne bleue (respectivement rouge) désigne la droite des moindres carrés de la classe A (respectivement B). Contrairement au *Perceptron*, l'*Adaline* détermine une solution lorsque le problème n'est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

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).

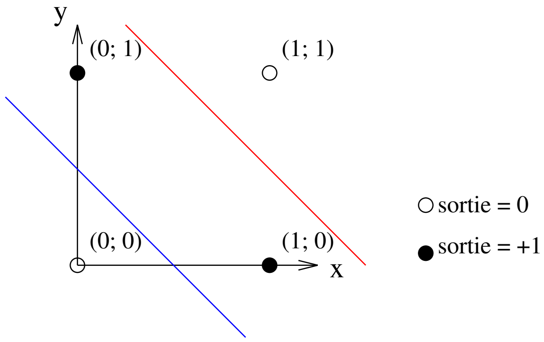

Figure 6.2: Séparation réalisée par un *perceptron* multicouche

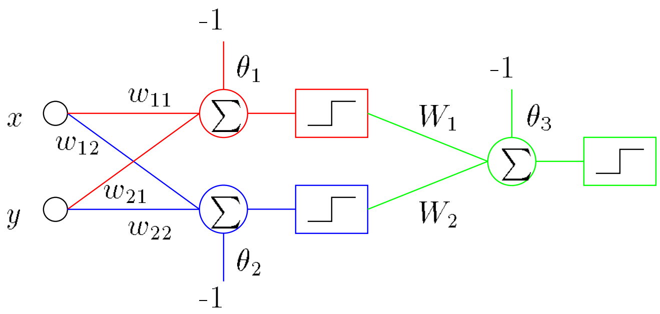

Figure 6.3: *Perceptron* multicouche réalisant la séparation de la figure 6.2. Le *perceptron* dessiné en bleu (respectivement en rouge) trace la séparatrice bleue (respectivement rouge) de la figure 6.2. Le neurone dessiné en vert combine alors ces deux lignes afin de réaliser la séparation désirée. 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

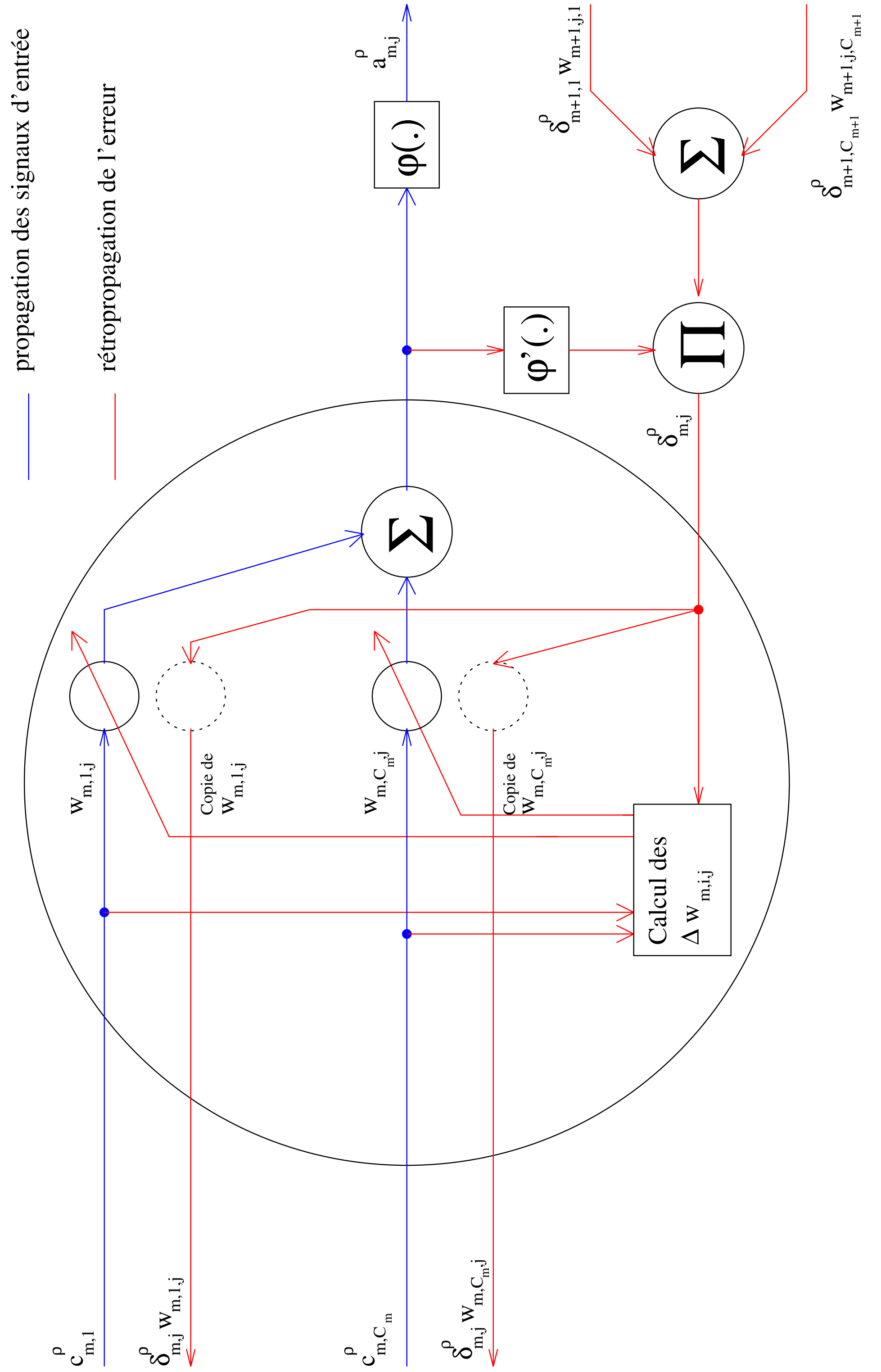

Figure 8.1: Détail de la propagation du signal d'entrée et de la rétropropagation de l'erreur dans un neurone j de la couche cachée m d'un *multilayer perceptron*.

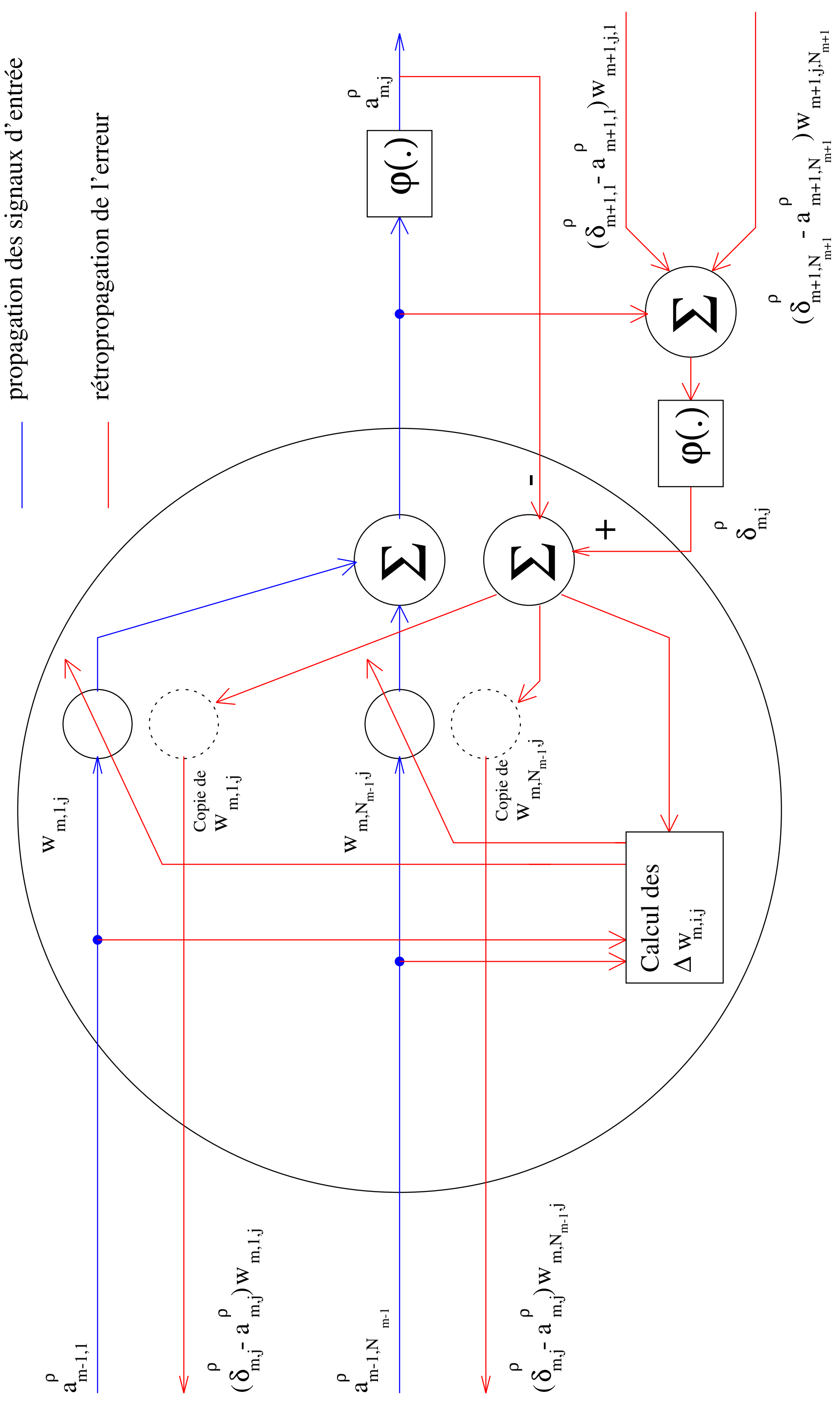

Figure 8.2: Détail de la propagation du signal d'entrée et de la rétropropagation de l'erreur dans un neurone j de la couche cachée m d'un *multilayer perceptron* (algorithme de rétropropagation non linéaire)

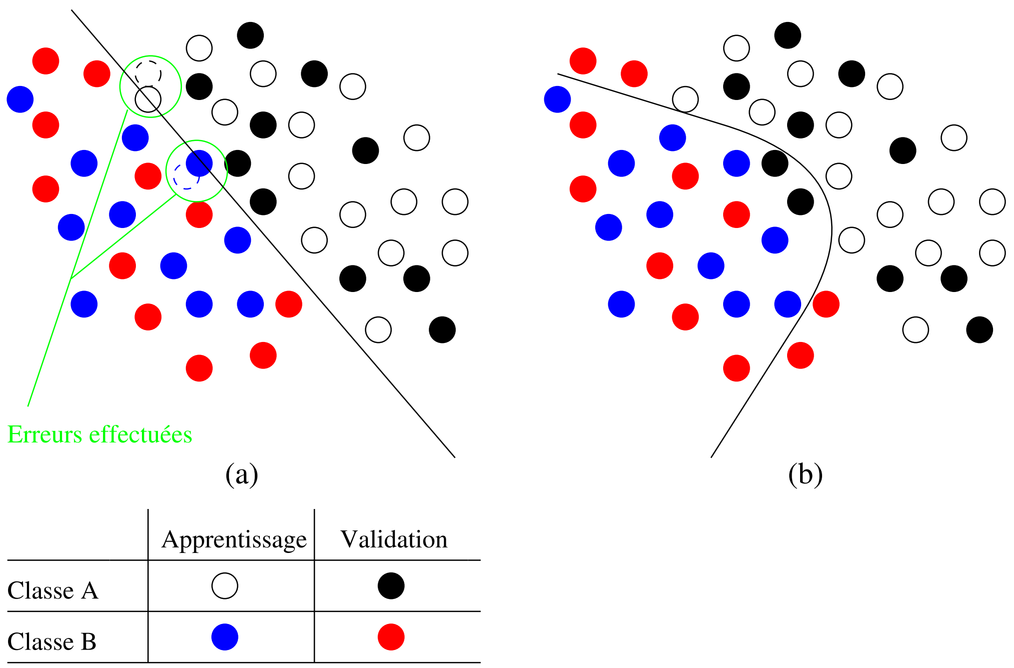

Figure 9.1: Un problème de classification. Nous commettons deux erreurs, repérées par les cercles verts, lors de la conception de la base d'apprentissage. Les cercles en pointillé indiquent les positions exactes de ces deux motifs.
(a) Séparatrice déterminée par l'*Adaline*. (b) Séparatrice obtenue en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

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.

- 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.

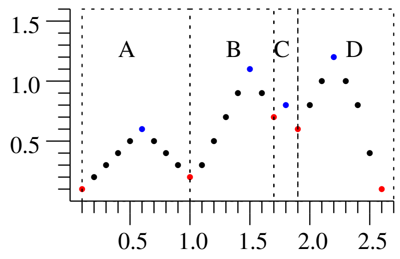

Figure 13.9: Recherche des maxima d'une fonction discrète. Les points bleus et rouges désignent respectivement les maxima et les minima.

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

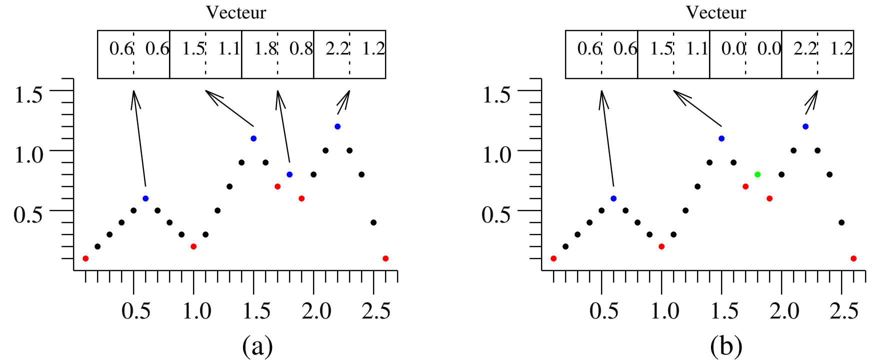

Figure 13.10: Organisation des maxima dans un vecteur. Nous souhaitons modéliser la fonction discrète avec au plus quatre maxima. Les points bleus, rouges et verts désignent respectivement les maxima, les minima et les maxima éliminés par notre algorithme. (a) Pour une certaine valeur de β, nous obtenons 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.

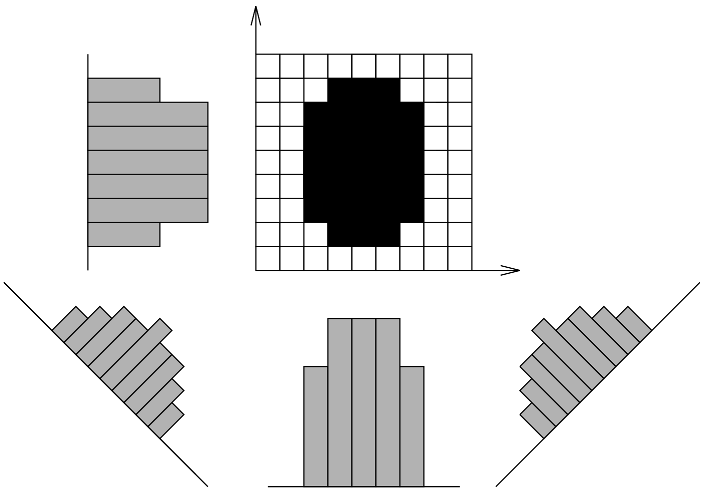

Figure 13.11: Projections du chiffre 0 écrit avec un feutre épais

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

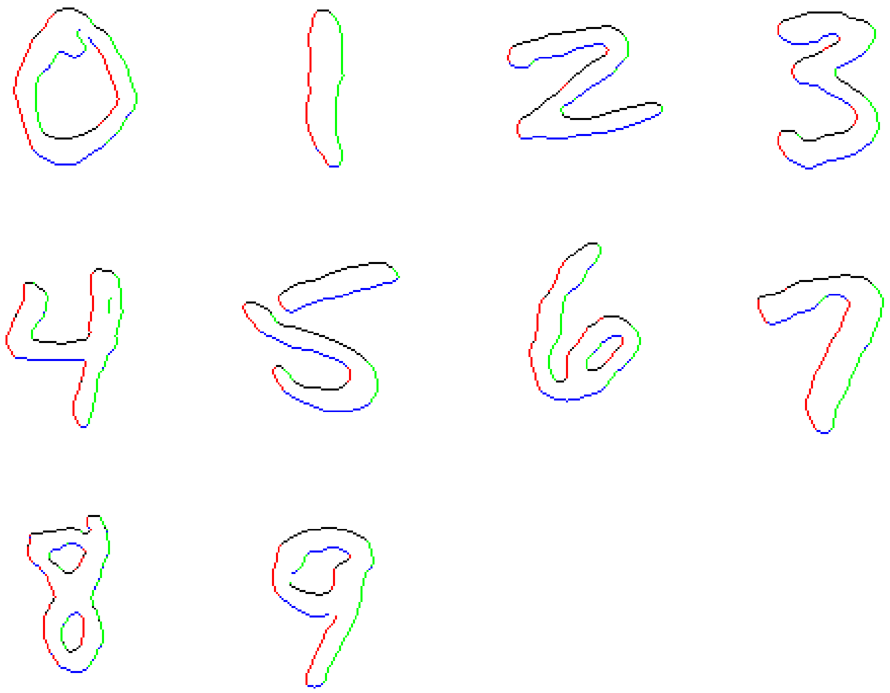

Figure 13.15: Classification selon quatre domaines d'orientation des points du contour de quelques chiffres

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].

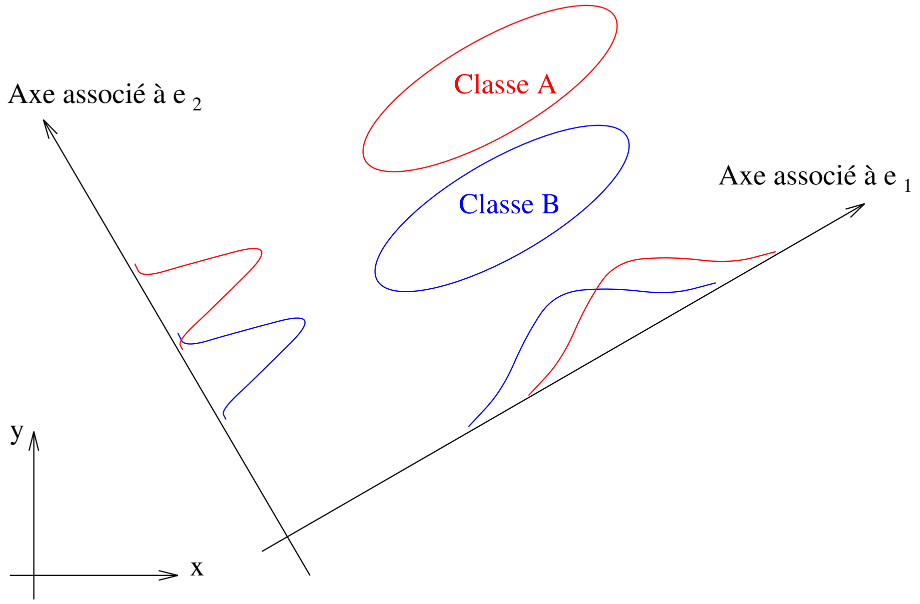

Figure 14.4: Limitations de l'analyse en composantes principales. La projection des données sur l'axe principal ne permet plus la distinction des deux classes.

[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).