ISIMA

Deuxième année F4

Apprentissage Statistique

Vincent BARRA

Institut Supérieur d'Informatique, de Modélisation et de leurs Applications

Table des matières

Introduction

La notion d'apprentissage est claire et intuitive pour les humains ou les animaux : c'est une procédure cognitive qui doit faire en sorte que l'individu réalise, de manière autonome, une tâche donnée. Typiquement, cette procédure s'effectue à partir d'exemples : ainsi, pour apprendre à lire à un enfant, on lui présente des exemples de lettres et de chiffres, écrits avec des écritures et des fontes différentes. À la fin de l'apprentissage, on attend de l'enfant qu'il soit capable de lire non seulement tous les chiffres et lettres de son livre de lecture, mais également tous les chiffres et lettres qu'il est susceptible de voir : en d'autres termes, on attend de lui qu'il ait une capacité de généralisation à partir des exemples qui lui ont été présentés, sans qu'il soit jamais nécessaire de lui fournir une description analytique et discursive de la forme et de la topologie des chiffres et des lettres.
L'apprentissage numérique poursuit exactement le même objectif : il s'agit de faire en sorte, à l'aide d'une procédure numérique programmée et exécutée sur un ordinateur, d'inférer un modèle d'un processus que l'on observe et sur lequel on peut effectuer des mesures, c'est-à-dire un ensemble d'équations qui décrivent le processus observé et qui permettent de faire des prédictions concernant le comportement de celui-ci. À cette fin, on fait l'hypothèse que le processus peut être décrit avec la précision désirée par une ou plusieurs fonctions qui contiennent des paramètres, et l'on ajuste ces derniers pour que cette ou ces fonctions s'ajustent aux données.

Nous nous intéressons dans ce cours à des problémes d'apprentissage dits **supervisés** : l'information d'entrée est représentée par un tableau individus/variables $[X, y]$. Chaque individu i est décrit par le vecteur ligne (x_i, y_i) où x_i est formé de d caractéristiques (mesures effectuées sur i) et y_i est une valeur discrète ou continue. Les n couples $(x_i, y_i)_{1 \leq i \leq n}$ forment **l'échantillon d'apprentissage**, noté E_n. L'apprentissage supervisé a pour objectif de prédire la valeur y d'un individu x en fonction de l'information dont on dispose sur cet individu. Le résultat d'un tel algorithme est une fonction f_n estimée à partir de E_n. f_n est un **modèle de régression** si $y_i \in \mathbb{R}$ et de **classification** si $y_i \in \mathbb{N}$. Les données d'apprentissage n'étant que des exemples qui

comportent leur part d'incertitude (liée au bruit, à l'imprécision des mesures), il est concevable, si la f_n la prend en compte, qu'elle accorde trop de confiance aux données d'apprentissage et ses capacités de généralisation en seront alors affectées : c'est le **sur-apprentissage** (ou overfitting). Au contraire, si f_n n'est pas assez fidèle aux données, les capacités de généralisation en seront également affectés : c'est le **sous-apprentissage** (ou underfitting). La fonction f_n issue de l'algorithme d'apprentissage doit donc rester fidèle à E_n sans donner trop d'importance aux données prises individuellement. Il s'agit donc de régler la complexité du modèle pour obtenir le meilleur compromis entre biais du modèle et variance. Pour mesurer la qualité du compromis, l'idéal est de disposer d'un échantillon indépendant de E_n, dit **échantillon de test**, sur lequel se mesure la précision des prédictions de f_n. On parle alors de **validation**. Ce compromis prend en compte les notions de régularisation, de réduction de dimension ou encore de minimisation du risque structurel. Son étude constitue le fil conducteur de ce cours.

Chapitre 1

Le problème de l'apprentissage

Sommaire

1.1 Définitions

Les données sont deux ensembles de variables aléatoires X et Y . On supposera dans la suite que X est un espace euclidien complet, et Y est un sous-ensemble fermé.
A partir de ces données, on souhaite apprendre une fonction $f : X \to Y$.
Dans la suite, nous nous restreignons à deux types de sous-espaces Y, amenant deux types de problèmes :

1. $Y \subset \mathbb{N}$: f est une fonction de classification
2. $Y \subset \mathbb{R}^k$: f est une fonction de régression

L'ensemble $E_n = [XY]$ est l'ensemble d'apprentissage, et consiste en n échantillons (x_i, y_i), tirages indépendants et identiquement distribués selon une distribution de probabilité fixe et inconnue $\mu(z)$ sur l'espace $Z = X \times Y$.

1.1.1 Algorithmes, espaces d'hypothèses, et fonctions de perte

Définition 1.1. *Un algorithme d'apprentissage $\mathcal{A}$ est une application d'un ensemble d'apprentissage E_n vers une fonction f : $\mathcal{A} : E_n \rightarrow f$*

Définition 1.2. *Un espace d'hypothèses $\mathcal{H}$ est un espace de fonctions qu'un algorithme d'apprentissage $\mathcal{A}$ cherche à atteindre.*

Le but général de l'apprentissage supervisé est d'utiliser l'ensemble E_n pour apprendre une fonction f qui, étant donnée une valeur x_{new} n'appartenant pas à X, fournit une valeur $y_{new} = f(x_{new})$. Une fonction de perte peut être utilisée pour préciser à quel point une fonction se comporte bien vis à vis de l'ensemble d'apprentissage E_n, ou à quel point elle est capable de prédire sur de nouvelles observations.

Définition 1.3. *Soit E_n un ensemble d'apprentissage, f une fonction apprise par $\mathcal{A}$ à partir de E_n, et (x, y) une donnée n'appartenant pas à E_n. Une fonction de perte est une fonction*

$$\begin{array}{rcl} V : \mathbb{R} \times \mathbb{R} & \rightarrow & \mathbb{R}^+ \\ (f(x), y) & \rightarrow & V(f(x), y) \end{array}$$

Exemples : pour des problèmes de régression, les fonctions de perte classiques sont :

1. fonction de perte charnière : $V(f(x), y) = (1 - yf(x))_+ = max\,(0, 1 - yf(x))$
2. fonction de perte quadratique, ou perte L_2. : $V(f(x), y) = (f(x) - y)^2$
3. fonction de perte L_1 : $V(f(x), y) = |f(x) - y|$:
4. fonction de perte de Huber : $V(f(x), y) = \begin{cases} \frac{1}{2\epsilon}(f(x) - y)^2 & \text{si } |f(x) - y| \geq \epsilon \\ 0 & \text{sinon} \end{cases}$
5. fonction de perte de Vapnik : $V(f(x), y) = \begin{cases} 0 & \text{si } |f(x) - y| \leq \epsilon \\ |f(x) - y| - \epsilon & \text{sinon} \end{cases}$

Exemples : pour des problèmes de classification , les fonctions de perte classiques sont :

1. fonction indicatrice : $V(f(x), y) = \mathbb{1}_{-yf(x) \leq 0}$
2. fonction de perte logistique : $V(f(x), y) = ln\left(1 + e^{-yf(x)}\right)$

1.1.2 Erreur empirique, erreur moyenne et généralisation

Définition 1.4. *Soit $z = (x, y)$. L'erreur moyenne d'une fonction f, étant donné une fonction de perte V et une distribution μ est*

$$I[f] = \mathbb{E}_z V(f(x), y) = \int V(f(x), y) d\mu_z$$

Le meilleur candidat pour minimiser le risque moyen en classification est donné par (Bayes)

$$f(x) = \begin{cases} 1 & \text{si} \quad \mu(Y = 1|X = x) > 0.5 \\ -1 & \text{sinon} \end{cases}$$

ce qui nécessite la connaissance de la loi μ ayant permis le tirage, qui reste inconnue. On doit donc construire un classifieur à partir du seul ensemble d'apprentissage, qui minimise le risque empirique défini par :

Définition 1.5. *L'erreur empirique d'une fonction f, étant donné une fonction de perte V et un ensemble d'apprentissage E_n est*

$$I_E[f] = \frac{1}{n} \sum_{i=1}^{n} V(f(x_i), y_i)$$

On espère alors que le classifieur ainsi construit aura une bonne capacité de généralisation

Définition 1.6. *On dit qu'un algorithme d'apprentissage $\mathcal{A}$ a une bonne capacité de généralisation si son erreur empirique est proche de son erreur moyenne. On note alors*

$$\mathcal{A}_{gen} = \{\mathcal{A}, |I_E[f] - I[f]| < \epsilon\}$$

L'avantage d'un algorithme $\mathcal{A} \in \mathcal{A}_{gen}$ est que si son erreur empirique est petite sur E_n, alors on peut prédire que $\mathcal{A}$ prédira de manière efficace sur des observations n'étant pas dans E_n. Un algorithme de $\mathcal{A}_{gen}$ ayant une erreur empirique élevée est au contraire sans intérêt.

1.1.3 Minimisation du risque empirique (MRE)

Définition 1.7. *Soit $\mathcal{H}$ un espace d'hypothèses. Une fonction f minimise le risque empirique si*

$$f \in arg \min_{f \in \mathcal{H}} I_E[f] = arg \min_{f \in \mathcal{H}} \left[\frac{1}{n} \sum_{i=1}^{n} V(f(x_i), y_i) \right]$$

Dans le cas où aucune fonction ne minimise le risque empirique, on utilise une variante de la définition

Définition 1.8. *Soit $\mathcal{H}$ un espace d'hypothèses. Une fonction f est un ϵ-minimum du risque empirique si pour tout $\epsilon > 0$:*

$$I_E[f] \leq \inf_{f \in \mathcal{H}} I[f] + \epsilon$$

1.1.4 Propriétés attendues du MRE

consistance

La première propriété désirée pour le MRE est celle de la consistance : on attend de l'algorithme MRE qu'il trouve une meilleure fonction $f \in \mathcal{H}$ lorsque le nombre d'observations augmente. Ceci est formalisé par la définition suivante :

Définition 1.9. *MRE est universellement consistant si*

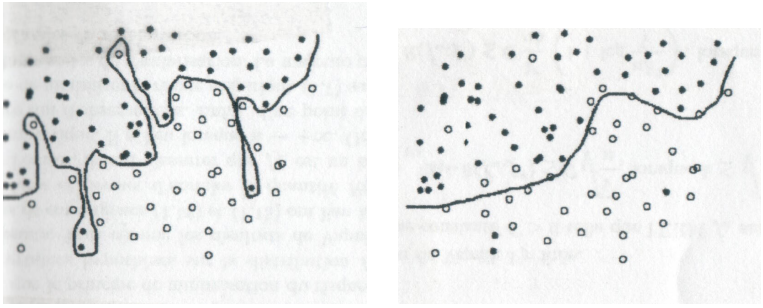
$$(\forall \epsilon > 0) \lim_{n \to \infty} \sup_{\mu} \mathbb{P} \left\{ I[f] > \inf_{f \in \mathcal{H}} I_E[f] + \epsilon \right\} = 0$$

Algorithme bien posé

Définition 1.10. *Le problème de trouver un algorithme d'apprentissage est bien posé si sa solution $\mathcal{A}$:*

1. *existe*
2. *est unique*
3. *est stable, i.e. une petite variation de l'ensemble E_n entraîne une petite variation de f.*

Le point essentiel pour l'algorithme MRE est la notion de stabilité, puisqu'existence et unicité sont assurées par la définition. Nous verrons que stabilité et consistance sont complémentaires et même équivalentes dans le cas de la minimisation du risque empirique. En fait, en tant que tel, MRE n'est pas bien posé. Pour imposer cette propriété, il est nécessaire de procéder à une **régularisation** de l'algorithme, qui consiste à contraindre l'espace des hypothèses à des classes de fonctions déterminées. Si $\mathcal{H}$ est suffisamment riche, à E_n fixé, le minimiseur du risque empirique tient compte de tout l'ensemble d'apprentissage (figure 1.1-a) et peut posséder une faible capacité de généralisation : on parle de sur apprentissage ou overfitting. Un comportement plus naturel serait celui de la figure 1.1-b, dans laquelle le classifieur est



(a) Overfitting (b)- Régularisation

FIGURE 1.1 – Overfitting et régularisation

"lissé", ou plus précisément régularisé. Ce comportement est lié à la faible connaissance du phénomène observé (loi μ). Ainsi, il s'agit d'obtenir un compromis entre fidélité aux données et régularisation, ce qui est connu dans le domaine de l'apprentissage sous le nom du dilmne biais/variance.

Plusieurs méthodes de régularisation existent, et citons d'ores et déjà deux exemples :

1. régularisation d'Ivanov, opérée directement sur $\mathcal{H}$:

$$\min_{f\in\mathcal{H}} \left[\frac{1}{n}\sum_{i=1}^{n} V(f(x_i), y_i)\right] \quad \text{sous } \Omega(f) \leq \tau$$

2. régularisation de Tikhonov, opérée indirectement sur $\mathcal{H}$ en ajoutant un terme de pénalité :

$$\min_{f\in\mathcal{H}} \left[\frac{1}{n}\sum_{i=1}^{n} V(f(x_i), y_i) + \lambda\Omega(f)\right]$$

La fonction $\Omega(f)$ est la fonction de régularisation, et est typiquement une norme d'un espace de Hilbert. Les paramètres τ et λ sont des paramètres de régularisation qui contrôlent le compromis entre l'attache aux données de E_n et la contrainte portée sur $\mathcal{H}$.

1.2 Construction d'algorithmes bien posés

Une autre alternative pour imposer à un algorithme d'être bien posé, est de construire directement de tels algorithmes. Les réseaux de neurones, que nous détaillerons plus loin, font partie de cette classe de méthodes.

%inputtikhonov.tex

Chapitre 2

Réseaux de neurones

Le but de ce chapitre n'est pas de présenter de manière exhaustive les réseaux de neurones, mais seulement de définir les notions essentielles permettant leur application dans le problème d'apprentissage. On s'intéresse uniquement dans la suite à deux modèles simples de réseaux de neurones, le perceptron et le perceptron multicouches. Une brève introduction à trois autres types classiques de réseaux est proposée en fin de chapitre.

2.1 Introduction aux réseaux de neurones

2.1.1 Réseaux de neurones et apprentissage automatique

Les réseaux de neurones artificiels sont des techniques numériques issues du domaine du connexionisme. Le courant connexionniste insiste sur le grand nombre de connexions (sous forme de réseau) réalisées entre les différents automates que sont les neurones. Le connexionisme permet :

— de disposer de nouveaux moyens de calcul : conversion de l'information des systèmes avec des applications pratiques ;

— de modéliser des phénomènes biologiques pour en apprendre davantage sur le cerveau en l'observant comme si c'était une machine de traitement électrique.

La démarche des réseaux de neurones s'oppose en certains points à celle de l'intelligence artificielle classique. En I.A., on utilise des règles qui sont manipulées selon les techniques de la logique formelle afin de fournir une représentation explicite du raisonnement. L'I.A. implique une approche " descendante " : elle part de l'analyse de la manière dont l'être humain procède pour résoudre des problèmes ou pour les apprendre, et tente de restituer cette

démarche en la décomposant en unités élémentaires. Les réseaux de neurones, eux, procèdent selon une approche " ascendante " qui tente de produire des phénomènes complexes comme l'apprentissage ou la reconnaissance de formes à partir d'opérations très élémentaires.
D'autre part, l'intelligence artificielle classique suit une approche cognitiviste et utilise l'outil informatique comme "manipulateur de symboles" sans se soucier de la vraisemblance biologique. L'approche connexionniste utilise les réseaux de neurones artificiels, plus vraisemblables au niveau biologique, mais un apprentissage automatique réalisé avec un tel outil ne renseigne pas vraiment sur les connaissances apprises puisque le résultat d'un tel apprentissage se manifestera par la modification de poids dans une matrice de connexions entre les différentes cellules du réseau.

2.1.2 Du neurone biologique au neurone formel

La reconnaissance du fait que le cerveau fonctionne de manière entièrement différente de celle d'un ordinateur conventionnel a joué un rôle très important dans le développement des réseaux de neurones artificiels. Les travaux effectués pour essayer de comprendre le comportement du cerveau humain ont menés à représenter celui-ci par un ensemble de composants structurels appelés neurones, massivement interconnectés entre eux. Le cerveau humain en contiendrait plusieurs centaines de milliards, et chacun de ceux-ci serait, en moyenne, connecté à dix mille autres.
Le neurone biologique est composé de quatre parties distinctes (fig 2.1) :

- le corps cellulaire, qui contient le noyau de la cellule nerveuse ; c'est en cet endroit que prend naissance l'influx nerveux, qui représente l'état d'activité du neurone ;
- les dendrites, ramifications tubulaires courtes formant une espèce d'arborescence autour du corps cellulaire ; ce sont les entrées principales du neurone, qui captent l'information venant d'autres neurones ;
- l'axone, longue fibre nerveuse qui se ramifie à son extrémité ; c'est la sortie du neurone et le support de l'information vers les autres neurones ;
- la synapse, qui communique l'information, en la pondérant par un poids synaptique, à un autre neurone ; elle est essentielle dans le fonctionnement du système nerveux.

La transmission de l'information d'un neurone à l'autre s'effectue au moyen de l'influx nerveux, qui est constitué d'une impulsion électrique, d'une durée d'environ 2 ms et d'une amplitude de 100 mV. Une cellule nerveuse standard

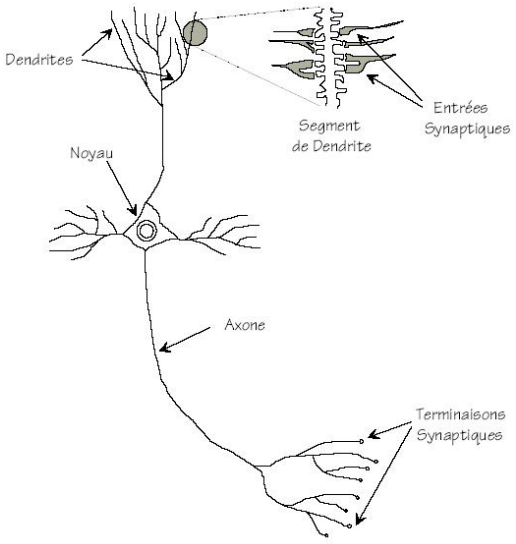


FIGURE 2.1 – le neurone biologique

non sollicitée en émet en moyenne cinquante à la seconde (activité spontanée). La probabilité d'émettre une impulsion est accrue ou réduite selon que la somme pondérée des entrées du neurone est globalement excitatrice ou inhibitrice. Cette fréquence peut ainsi être portée jusqu'à 100 impulsions par seconde, pour un neurone bombardé d'effets synaptiques excitateurs ; dans le cas contraire, elle peut être réduite à néant (le neurone reste silencieux). Les effets synaptiques qui agissent sur le neurone entraînent donc une modulation de la fréquence d'émission de l'influx nerveux. Le message transmis est précisément contenu dans le nombre d'influx nerveux émis, défini par une moyenne sur quelques dizaines de ms. L'information contenue dans le cerveau, quant à elle, est représentée par les poids synaptiques attribués aux entrées de chaque neurone. Le cerveau est capable d'organiser ces neurones, selon un assemblage complexe, non-linéaire et extrêmement parallèle, de manière à pouvoir accomplir des tâches très élaborées. Du fait du grand nombre de neurones et de leurs interconnexions, ce système possède une propriété de tolérance aux fautes. Ainsi, la défectuosité d'un neurone n'entraînera aucune perte réelle d'information, mais seulement une faible dégradation en qualité de toute l'information contenue dans le système. C'est pourquoi, nous pouvons reconnaître le visage d'une personne même si celle-ci a vieilli, par exemple, alors que c'est là une tâche quasiment impossible pour un ordinateur classique.

C'est la tentative de donner à l'ordinateur les qualités de perception du cerveau humain qui a conduit à une modélisation électrique de celui-ci. C'est

cette modélisation que tentent de réaliser les réseaux de neurones artificiels, dont l'élaboration repose sur base de la définition suivante, proposée par Haykin :
Un réseau de neurones est un processus distribué de manière massivement parallèle, qui a une propension naturelle à mémoriser des connaissances de façon expérimentale et de les rendre disponibles pour utilisation. Il ressemble au cerveau en deux points :

1. **la connaissance est acquise au travers d'un processus d'apprentissage ;**
2. **les poids des connections entre les neurones sont utilisés pour mémoriser la connaissance**

La première étude systématique du neurone artificiel est due au neuropsychiatre McCulloch et au logicien Pitts qui s'inspirèrent de leurs travaux sur les neurones biologiques.

2.1.3 Classification des réseaux de neurones

Un réseau de neurones formels est constitué d'un grand nombre de cellules de base interconnectées. De nombreuses variantes sont définies selon le choix de la cellule élémentaire, de l'architecture et de la dynamique du réseau.

Une cellule élémentaire peut manipuler des valeurs binaires ou réelles. Les valeurs binaires sont représentées par 0 et 1 ou -1 et 1. Différentes fonctions peuvent être utilisées pour le calcul de la sortie. Le calcul de la sortie peut être déterministe ou probabiliste.

L'architecture du réseau peut être sans rétroaction, c'est à dire que la sortie d'une cellule ne peut influencer son entrée. Elle peut être avec rétroaction totale ou partielle.

La dynamique du réseau peut être synchrone : toutes les cellules calculent leurs sorties respectives simultanément. La dynamique peut être asynchrone. Dans ce dernier cas, on peut avoir une dynamique asynchrone séquentielle : les cellules calculent leurs sorties chacune à son tour en séquence ou avoir une dynamique asynchrone aléatoire.

Par exemple, si on considère des neurones à sortie stochastique -1 ou 1 calculée par une fonction à seuil basée sur la fonction sigmoïde, une interconnection complète et une dynamique synchrone, on obtient le modèle de Hopfield et la notion de mémoire associative.

Si on considère des neurones déterministes à sortie réelle calculée à l'aide de la fonction sigmoïde, une architecture sans rétroaction en couches successives avec une couche d'entrées et une couche de sorties, une dynamique asynchrone

séquentielle, on obtient le modèle du Perceptron multi-couches (PMC) qui sera étudié dans les paragraphes suivants.

2.1.4 Applications

Les principales applications des réseaux de neurones sont l'optimisation et l'apprentissage. En apprentissage, les réseaux de neurones sont essentiellement utilisés pour :

— l'apprentissage supervisé ;
— l'apprentissage non supervisé ;
— l'apprentissage par renforcement.

Pour ces trois types d'apprentissage, il y a également un choix traditionnel entre :

— l'apprentissage off-line : toutes les données sont dans une base d'exemples d'apprentissage qui sont traités simultanément ;
— l'apprentissage on-line : les exemples sont présentés les uns après les autres au fur et à mesure de leur disponibilité.

Nous nous limitons, dans ce cours, à l'apprentissage supervisé à partir d'une base d'exemples. Dans ce cadre, l'apprentissage à l'aide de réseaux de neurones est bien adapté pour l'apprentissage à partir de données complexes (images sur une rétine, sons, ...) mais aussi à partir de données symboliques. Les entrées peuvent être représentées par de nombreux attributs à valeurs réelles ou symboliques, les attributs pouvant être dépendants ou non. La ou les sorties peuvent être réelles ou discrètes. L'apprentissage à l'aide de réseaux de neurones est tolérant au bruit et aux erreurs. Le temps d'apprentissage peut être long, par contre, après apprentissage, le calcul des sorties à partir d'un vecteur d'entrée est rapide. La critique principale est que le résultat de l'apprentissage, c'est-à-dire le réseau de neurones calculé par l'algorithme d'apprentissage, n'est pas interprétable par l'utilisateur : on ne peut pas donner d'explication au calcul d'une sortie sur un vecteur d'entrée. On parle de boîte noire.

Nous n'étudions que le perceptron, brique de base des modèles plus complexes, et le perceptron multi-couches (PMC). L'accent sera mis sur les algorithmes d'apprentissage pour ces deux modèles, en particulier sur l'algorithme de rétropropagation du gradient appliqué aux PMC. Cet algorithme est, en effet, le premier algorithme d'apprentissage convaincant dans un modèle suffisamment puissant et cet algorithme a de nombreuses applications.

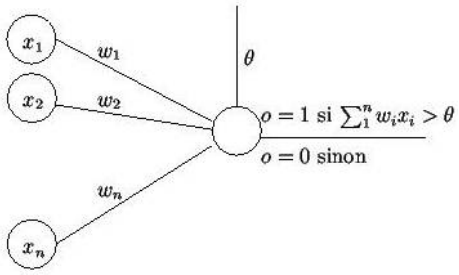


FIGURE 2.2 – représentation graphique d'un perceptron - première ébauche

2.2 Le perceptron

Le perceptron est un modèle de réseau de neurones avec algorithme d'apprentissage (Rosenblatt en 1958). L'idée sous-jacente de ce modèle est le fonctionnement de la rétine, l'étude de la perception visuelle.

2.2.1 Définition

Un perceptron linéaire à seuil prend en entrée n valeurs $x_1 \cdots x_n$ (la rétine) et calcule une sortie o. Un perceptron est défini par la donnée de $n+1$ constantes : les **coefficients synaptiques** $w_1, \cdots, w_n$ et le seuil (ou le biais) θ. La sortie o (Fig 2.2) est calculée par

$$o = \begin{cases} 1 \quad \text{si} \quad w^T x = \sum_{i=1}^{n} w_i x_i > \theta \\ 0 \qquad\qquad\qquad\qquad \text{sinon} \end{cases}$$

Les entrées $x_1, \cdots x_n$ peuvent être à valeurs dans {0,1} (ou {-1,1}) ou réelles, les poids peuvent être entiers ou réels. Il existe également des modèles pour lesquels le calcul de la sortie est probabiliste.

Pour simplifier les notations et certaines preuves, on remplace souvent le seuil par une entrée supplémentaire $x_0 = 1$. À cette entrée est associée un coefficient synaptique w_0. On peut décomposer le calcul de la sortie o en un premier calcul de la quantité $\sum_{i=0}^{n} w_i x_i$ appelée **potentiel post-synaptique** ou **entrée totale**, suivi d'une application d'une **fonction d'activation** sur cette entrée totale (Fig 2.3). La fonction d'activation est la fonction de Heaviside définie par $f(x) = 1_{\{x>0\}}$ lorsque la sortie est en {0,1}, et $g(x) = 2f(x)-1$

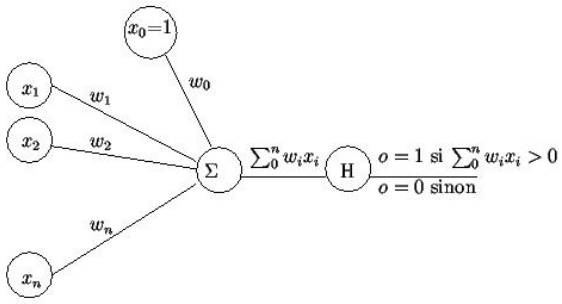


FIGURE 2.3 – représentation graphique d'un perceptron - seconde ébauche

lorsque la sortie est en {-1,1}.
Bien que considérant une entrée supplémentaire x_0, un perceptron est toujours considéré comme associant une sortie o aux n entrées $x_1 \cdots x_n$. L'équivalence entre le modèle avec seuil et le modèle avec entrée supplémentaire à 1 est immédiate : le coefficient w_0 est l'opposé du seuil θ. Nous considérerons toujours ce dernier modèle de perceptron linéaire à seuil par la suite.

Plus généralement, un perceptron linéaire à seuil à n entrées réelles (respectivement binaires) est une fonction de $\mathbb{R}^n$ (respectivement $\{0,1\}^n$) dans {0,1}.

2.2.2 Utilisation : discrimination linéaire

Soit $\mathcal{E}_a$ un ensemble d'exemples dans $\mathbb{R}^n \times \{0,1\}$. On note

$$\mathcal{E}_a^0 = \{s \in \mathbb{R}^n / (s,0) \in \mathcal{E}_a\}$$

et

$$\mathcal{E}_a^1 = \{s \in \mathbb{R}^n / (s,1) \in \mathcal{E}_a\}$$

On dit que $\mathcal{E}_a$ est **linéairement séparable** s'il existe un hyperplan H de $\mathbb{R}^n$ tel que les ensembles $\mathcal{E}_a^0$ et $\mathcal{E}_a^1$ soient situés de part et d'autre de cet hyperplan.
On montre qu'un perceptron linéaire à seuil à n entrées divise l'espace des entrées $\mathbb{R}^n$ en deux sous-espaces délimités par un hyperplan. Réciproquement, tout ensemble linéairement séparable peut être discriminé par un perceptron. Un perceptron est donc un discriminant linéaire. On montre facilement qu'un échantillon de $\mathbb{R}^n$ est séparable par un hyperplan si et seulement si l'échantillon de $\mathbb{R}^{n+1}$ obtenu en rajoutant une entrée toujours égale à 1 est séparable par un hyperplan passant par l'origine.

Toute fonction de $\mathbb{R}^n$ dans {0,1} n'est bien sur pas calculable par perceptron.

2.2.3 Algorithme d'apprentissage par correction d'erreur

Étant donné un échantillon d'apprentissage $\mathcal{E}_a$ de $\mathbb{R}^n \times$ {0,1} (respectivement $\{0,1\}^n \times$ {0,1}), c'est-à-dire un ensemble d'exemples dont les descriptions sont n attributs réels (respectivement binaires) et la classe est binaire, il s'agit de trouver un algorithme qui infère à partir de $\mathcal{E}_a$ un perceptron qui classifie correctement les éléments de $\mathcal{E}_a$ au vu de leurs descriptions si c'est possible, ou au mieux sinon.
L'algorithme d'apprentissage peut être décrit succinctement de la manière suivante. On initialise les poids du perceptron à des valeurs quelconques. A chaque fois que l'on présente un nouvel exemple, on ajuste les poids selon que le perceptron l'a correctement classé ou non. L'algorithme s'arrête lorsque tous les exemples ont été présentés sans modification d'aucun poids.
Dans la suite, on note x une description qui est un élément de $\mathbb{R}^n$ ou $\{0,1\}^n$. La i$^{\text{ème}}$ composante de x est notée x_i. Un échantillon $\mathcal{E}_a$ est un ensemble de couples (x, c) où c est la classe de x. Lorsqu'il sera utile de désigner un élément particulier de $\mathcal{E}_a$, nous noterons (x^s, c^s) le $s^{\text{ème}}$ élément de $\mathcal{E}_a$. x_i^s désignera la i$^{\text{ème}}$ composante du vecteur d'entrée x^s. Si une entrée x^s est présentée en entrée d'un perceptron, nous noterons o^s la sortie binaire calculée par le perceptron. Rappelons qu'il existe une $(n+1)^{\text{ème}}$ entrée x_0 de valeur 1 pour le perceptron.
L'algorithme d'apprentissage par correction d'erreur du perceptron linéaire à seuil suit alors le schéma suivant :

Algorithm 1: Algorithme d'apprentissage par correction d'erreur du perceptron linéaire

```
Initialisation aléatoire des w_i
répéter
    Prendre un exemple (x, c) dans E_a
    Calculer la sortie o du perceptron pour l'entrée x
    pour i ∈ {0 ··· n} faire
        w_i ← w_i + (c − o)x_i
jusqu'à test ;
```

La procédure d'apprentissage du perceptron est une procédure de correction d'erreur puisque les poids ne sont pas modifiés lorsque la sortie attendue c

est égale à la sortie calculée o par le perceptron courant.
Étudions les modifications sur les poids lorsque c diffère de o, lorsque $x \in \{0,1\}^n$:

- si o=0 et c=1, cela signifie que le perceptron n'a pas assez pris en compte les neurones actifs de l'entrée (c'est-à-dire les neurones ayant une entrée à 1) ; dans ce cas, $w_i \leftarrow w_i + x_i$; l'algorithme ajoute la valeur de la rétine aux poids synaptiques (renforcement).
- si o=1 et c=0, alors $w_i \leftarrow w_i - x_i$; l'algorithme retranche la valeur de la rétine aux poids synaptiques (inhibition).

Remarquons que, en phase de calcul, les constantes du perceptron sont les poids synaptiques alors que les variables sont les entrées. Tandis que, en phase d'apprentissage, ce sont les coefficients synaptiques qui sont variables alors que les entrées de l'échantillon $\mathcal{E}_a$ apparaissent comme des constantes.

Certains éléments importants ont été laissés volontairement imprécis.

- en premier lieu, il faut préciser comment est fait le choix d'un élément de $\mathcal{E}_a$: aléatoirement ? En suivant un ordre prédéfini ? Doivent-ils être tous présentés ?
- le critère d'arrêt de la boucle principale de l'algorithme n'est pas défini : après un certain nombre d'étapes ? Lorsque tous les exemples ont été présentés ? Lorsque les poids ne sont plus modifiés pendant un certain nombre d'étapes ?

Nous reviendrons sur toutes ces questions par la suite.

Exemple : apprentissage du OU binaire (tableau 2.1)
Les descriptions appartiennent à $\{0,1\}^2$, les entrées du perceptron appartiennent à $\{0,1\}^3$, la première composante correspond à l'entrée x_0 et vaut toujours 1, les deux composantes suivantes correspondent aux variables x_1 et x_2 . On suppose qu'à l'initialisation, les poids suivants ont été choisis : w_0=0 ; $w_1 = 1$ et $w_2 = -1$. On suppose que les exemples sont présentés dans l'ordre lexicographique.

Aucune entrée ne modifie le perceptron à partir de l'itération 10.

On peut montrer que si l'échantillon $\mathcal{E}_a$ est linéairement séparable et si les exemples sont présentés équitablement (c'est-à-dire que la procédure de choix des exemples n'en exclut aucun), la procédure d'apprentissage par correction d'erreur converge vers un perceptron linéaire à seuil qui sépare linéairement

étape	w_0	w_1	w_2	Entrée	$\sum_{i=0}^{2} w_i x_i$	o	c	w_0	w_1	w_2
Init								0	1	-1
1	0	1	-1	100	0	0	0	0	1	-1
2	0	1	-1	101	-1	0	1	1	1	0
3	1	1	0	110	2	1	1	1	1	0
4	1	1	0	111	2	1	1	1	1	0
5	1	1	0	100	1	1	0	0	1	0
6	0	1	0	101	0	0	1	1	1	1
7	1	1	1	110	2	1	1	1	1	1
8	1	1	1	111	3	1	1	1	1	1
9	1	1	1	100	1	1	0	0	1	1
10	0	1	1	101	1	1	1	0	1	1

TABLE 2.1 – Apprentissage du OU binaire

$\mathcal{E}_a$.
L'inconvénient majeur de cet apprentissage est que si l'échantillon présenté n'est pas linéairement séparable, l'algorithme ne convergera pas et l'on aura aucun moyen de le savoir. On pourrait penser qu'il suffit d'observer l'évolution des poids synaptiques pour en déduire si l'on doit arrêter ou non l'algorithme. En effet, si les poids et le seuil prennent deux fois les mêmes valeurs sans que le perceptron ait appris et alors que tous les exemples ont été présentés, cela signifie que l'échantillon n'est pas séparable. Et l'on peut penser que l'on peut borner les poids et le seuil en fonction de la taille de la rétine. C'est vrai mais les résultats de complexité suivants montrent que cette idée n'est pas applicable en pratique.

1. Toute fonction booléenne linéairement séparable sur n variables peut être réalisée par un perceptron dont les poids synaptiques entiers w_i sont tels que $\lceil w_i \rceil \leq (n+1)^{\frac{n+1}{2}}$
2. Il existe des fonction booléennes linéairement séparables sur n variables qui requièrent des poids entiers supérieurs à $2^{\frac{n+1}{2}}$

Ces résultats sont assez décevants. Le premier montre que l'on peut borner les poids synaptiques en fonction de la taille de la rétine, mais par un nombre tellement grand que toute application pratique de ce résultat semble exclue. Le second résultat montre en particulier que l'algorithme d'apprentissage peut nécessiter un nombre exponentiel d'étapes (en fonction de la taille de la rétine) avant de s'arrêter. En effet, les poids ne varient qu'au plus d'une unité à chaque étape. Même lorsque l'algorithme d'apprentis-

sage du perceptron converge, rien ne garantit que la solution sera robuste, c'est-à-dire qu'elle ne sera pas remise en cause par la présentation d'un seul nouvel exemple. Pire encore, cet algorithme n'a aucune tolérance au bruit : si du bruit, c'est-à-dire une information mal classée, vient perturber les données d'entrée, le perceptron ne convergera jamais. En effet, des données linéairement séparables peuvent ne plus l'être à cause du bruit. En particulier, les problèmes non-déterministes, c'est-à-dire pour lesquels une même description peut représenter des éléments de classes différentes ne peuvent pas être traités à l'aide d'un perceptron.

2.2.4 Algorithme d'apprentissage par descente de gradient

Plutôt que d'obtenir un perceptron qui classifie correctement tous les exemples, il s'agira maintenant de calculer une erreur et d'essayer de minimiser cette erreur. Pour introduire cette notion d'erreur, on utilise des poids réels et on élimine la notion de seuil (ou d'entrée supplémentaire), ce qui signifie que la sortie sera égale au potentiel post-synaptique et sera donc réelle.
Un perceptron linéaire prend en entrée un vecteur x de n valeurs $x_1 \cdots x_n$ et calcule une sortie o. Un perceptron est défini par la donnée d'un vecteur w de coefficients synaptiques. La sortie o est définie par $o = w^T x$
L'erreur d'un perceptron P défini par $w = (w_1 \cdots w_n)$ sur un échantillon d'apprentissage $\mathcal{E}_a$ d'exemples (x^s, c^s) est définie en utilisant par l'erreur quadratique :

$$E(w) = \frac{1}{2} \sum_{(x^s,c^s)\in\mathcal{E}_a} (c^s - o^s)^2$$

où o^s est la sortie calculée par P sur l'entrée x^s. L'erreur mesure donc l'écart entre les sorties attendue et calculée sur l'échantillon complet. On remarque que $E(w) = 0$ si et seulement si le perceptron classifie correctement l'échantillon complet. On suppose $\mathcal{E}_a$ fixé, le problème est donc de déterminer, par descente de gradient, un vecteur $\tilde{w}$ qui minimise $E(w)$. On

a alors :

$$\begin{aligned}\frac{\partial E(w)}{\partial w_i} &= \frac{\partial}{\partial w_i}\left(\frac{1}{2}\sum_{(x^s,c^s)\in\mathcal{E}_a}(c^s-o^s)^2\right)\\ &= \frac{1}{2}\sum_{\mathcal{E}_a}\frac{\partial}{\partial w_i}(c^s-o^s)^2\\ &= \sum_{\mathcal{E}_a}(c^s-o^s)\frac{\partial}{\partial w_i}(c^s-w^Tx^s)\\ &= \sum_{\mathcal{E}_a}(c^s-o^s)(-x_i^s)\end{aligned}$$

L'application de la méthode du gradient invite donc à modifier le poids w_i après une présentation complète de $\mathcal{E}_a$ d'une quantité Δw_i définie par :

$$\Delta w_i = -\epsilon\frac{\partial E(w)}{\partial w_i}$$

L'algorithme d'apprentissage par descente de gradient du perceptron linéaire peut maintenant être défini :

Algorithm 2: Algorithme d'apprentissage par descente de gradient du perceptron linéaire

```
Initialisation aléatoire des w_i
répéter
    pour i ∈ {1 ··· n} faire
        Δw_i ← 0
    pour tout exemple (x^s, c^s) ∈ E_a faire
        Calculer o^s
        pour i ∈ {1 ··· n} faire
            Δw_i ← Δw_i + ε(c^s − o^s)x_i^s
    pour i ∈ {1 ··· n} faire
        w_i ← w_i + Δw_i
jusqu'à test ;
```

La fonction erreur quadratique ne possède qu'un minimum (la surface est une paraboloïde). La convergence est assurée, même si l'échantillon d'entrée n'est pas linéairement séparable, vers un minimum de la fonction erreur pour un ϵ bien choisi, suffisamment petit. Si ϵ est trop grand, on risque d'osciller

autour du minimum. Pour cette raison, une modification classique est de diminuer graduellement la valeur de ϵ en fonction du nombre d'itérations. Le principal défaut est que la convergence peut être très lente et que chaque étape nécessite le calcul sur tout l'ensemble d'apprentissage.

Au lieu de calculer les variations des poids en sommant sur tous les exemples de $\mathcal{E}_a$, l'idée est alors de modifier les poids à chaque présentation d'exemple. La règle de modification des poids devient :

$$\Delta w_i = \epsilon(c^s - o^s)x_i^s$$

Cette règle est appelée **règle delta, ou règle Adaline, ou encore règle de Widrow-Hoff**, et l'algorithme correspondant s'écrit :

Algorithm 3: Apprentissage par règle Adaline

```
Initialisation aléatoire des w_i
répéter
    Prendre un exemple (x, c) ∈ E_a
    Calculer o
    pour i ∈ {1 ··· n} faire
        w_i ← w_i + ε(c − o)x_i
jusqu'à test ;
```

En général, on parcourt l'échantillon dans un ordre prédéfini. Le critère d'arrêt généralement choisi fait intervenir un seuil de modifications des poids pour un passage complet de l'échantillon.
Au coefficient ϵ près dans la règle de modification des poids, on retrouve l'algorithme d'apprentissage par correction d'erreur. Pour l'algorithme de Widrow-Hoff, il y a correction chaque fois que la sortie totale (qui est un réel) est différente de la valeur attendue (égale à 0 ou 1). Ce n'est donc pas une méthode d'apprentissage par correction d'erreur puisqu'il y a modification du perceptron dans (presque) tous les cas. Rappelons également que l'algorithme par correction d'erreur produit en sortie un perceptron linéaire à seuil alors que l'algorithme par descente de gradient produit un perceptron linéaire. L'avantage de l'algorithme de Widrow-Hoff par rapport à l'algorithme par correction d'erreur est que, même si l'échantillon d'entrée n'est pas linéairement séparable, l'algorithme va converger vers une solution optimale (sous réserve du bon choix du paramètre ϵ). L'algorithme est, par conséquent, plus robuste au bruit.
L'algorithme de Widrow-Hoff s'écarte de l'algorithme du gradient sur un point important : on modifie les poids après présentation de chaque exemple

en fonction de l'erreur locale et non de l'erreur globale. Rien ne prouve donc que la diminution de l'erreur en un point ne va pas être compensée par une augmentation de l'erreur pour les autres points. La justification empirique de cette manière de procéder est commune à toutes les **méthodes adaptatives** : le champ d'application des méthodes adaptatives est justement l'ensemble des problèmes pour lesquels des ajustements locaux vont finir par converger vers une solution globale.

L'algorithme de Widrow-Hoff est très souvent utilisé en pratique et donne de bons résultats. La convergence est, en général, plus rapide que par la méthode du gradient. Il est fréquent pour cet algorithme de faire diminuer la valeur de ϵ en fonction du nombre d'itérations comme pour l'algorithme du gradient.

2.2.5 Pour en finir avec le perceptron

L'apprentissage par perceptron ou par la méthode du gradient ne sont rien d'autre que des techniques de séparation linéaire qu'il faudrait comparer aux techniques utilisées habituellement en statistiques (discriminant linéaire, machines à vecteurs de support). Ces méthodes sont non paramétriques, c'est-à-dire qu'elles n'exigent aucune autre hypothèse sur les données que la séparabilité.

On peut montrer que presque tous les échantillons de moins de $2n$ exemples sont linéairement séparables lorsque n est le nombre de variables. Une classification correcte d'un petit échantillon n'a donc aucune valeur prédictive. Par contre, lorsque l'on travaille sur suffisamment de données et que le problème s'y prête, on constate empiriquement que le perceptron appris par un des algorithmes précédents a un bon pouvoir prédictif.

Il est bien évident que la plupart des problèmes d'apprentissage qui se posent naturellement ne peuvent pas être résolus par des méthodes aussi simples : il n'y a que très peu d'espoir que les exemples naturels se répartissent sagement de part et d'autre d'un hyperplan. Deux manières de résoudre cette difficulté peuvent être envisagées : soit mettre au point des séparateurs non-linéaires, soit (ce qui revient à peu près au même) complexifier l'espace de représentation de manière à linéariser le problème initial.
Les réseaux multicouches abordent ce type de problème.

2.3 Les réseaux multicouches

2.3.1 Architecture

Un perceptron linéaire à seuil est bien adapté pour des échantillons linéairement séparables. Cependant, dans la plupart des problèmes réels, cette condition n'est pas réalisée. Un perceptron linéaire à seuil est constitué d'un seul neurone. On s'est très vite rendu compte qu'en combinant plusieurs neurones le pouvoir de calcul était augmenté. La notion de perceptron multi-couches (PMC) a ainsi été définie.
On considère une couche d'entrée qui correspond aux variables d'entrée, une couche de sorties, et un certain nombre de couches intermédiaires. Ces couches sont constituées de **cellules**, qui comportent chacune un neurone à p entrées (sauf pour la couche d'entrée), un calcul de potentiel synaptique et une fonction d'activation. Les liens n'existent qu'entre les cellules d'une couche avec les cellules de la couche suivante, et on aboutit à la définition suivante de l'architecture d'un réseau multicouches :
un réseau de neurones à couches cachées est défini par une architecture vérifiant les propriétés suivantes :

- les cellules sont réparties de façon exclusive dans des couches $C_0, \cdots, C_q$
- la première couche C_0 est la rétine composée des cellules d'entrée qui correspondent aux n variables d'entrée ; les couches $C_1, \cdots, C_{q-1}$ sont les couches cachées ; la couche C_q est composée de la (ou les) cellule(s) de décision,
- Les entrées d'une cellule d'une couche $C_i, i \geq 1$ sont toutes les cellules de la couche C_{i-1} et aucune autre cellule.

La dynamique du réseau est synchrone (toutes les cellules calculent leurs sorties respectives simultanément).

2.3.2 Propriétés

On peut montrer que toute fonction booléenne peut être calculée par un PMC linéaire à seuil comprenant une seule couche cachée.
Supposons par exemple que les cellules élémentaires soient des perceptrons linéaires à seuil, on parle alors de perceptrons multi-couches (PMC) linéaire à seuil. Soit n variables binaires, il est facile de montrer que le OU n-aire est calculable par un perceptron linéaire à seuil et que toute conjonction sur les littéraux définis à partir des n variables est calculable par un perceptron linéaire à seuil. Étant donnée une fonction booléenne sur n variables, cette fonction peut être mise sous forme normale disjonctive, il suffit alors que chaque cellule de la couche cachée calcule une conjonction et que la cellule

de sortie calcule la disjonction des résultats.
Cependant, si l'on utilise cette méthode pour construire un réseau de neurones pour calculer une fonction booléenne quelconque, la couche cachée pourra contenir jusqu'à 2^n neurones (où n est la taille de la rétine), ce qui est inacceptable en pratique. On peut montrer par ailleurs que cette solution est loin d'être la meilleure.

Pour pouvoir utiliser les réseaux multi-couches en apprentissage, deux choses sont indispensables :

— une méthode indiquant comment choisir une architecture de réseau pour résoudre un problème donné. C'est-à-dire, pouvoir répondre aux questions suivantes : combien de couches cachées ? combien de neurones par couche cachée ?
— une fois l'architecture choisie, un algorithme d'apprentissage qui calcule, à partir de l'échantillon d'apprentissage, les valeurs des coefficients synaptiques pour construire un réseau adapté au problème.

Le premier point est encore un sujet de recherche actif : quelques algorithmes d'apprentissage auto-constructifs ont été proposés. Leur rôle est double :

— apprentissage de l'échantillon avec un réseau courant,
— modification du réseau courant, en ajoutant de nouvelles cellules ou une nouvelle couche, en cas d'échec de l'apprentissage.

Il semble assez facile de concevoir des algorithmes auto-constructifs qui classent correctement l'échantillon, mais beaucoup plus difficile d'en obtenir qui aient un bon pouvoir de généralisation.

Il a fallu attendre le début des années 80 pour que le deuxième problème trouve une solution : l'algorithme de **rétropropagation du gradient**, découvert simultanément par des équipes française et américaine. Cet algorithme est, comme son nom l'indique, basé sur la méthode du gradient. Il est donc nécessaire de considérer des fonctions d'erreur dérivables. Ceci implique qu'il n'est pas possible de considérer comme cellule élémentaire un perceptron linéaire à seuil. L'idée est alors de prendre comme cellule élémentaire un perceptron linéaire. Malheureusement, dans ce cas, l'introduction de cellules supplémentaires n'augmente pas l'expressivité (une combinaison linéaire de fonctions linéaires est une fonction linéaire).
Il est donc nécessaire de considérer une nouvelle cellule élémentaire. La sortie de cette cellule sera une fonction de variable réelle dérivable qui est une approximation de la fonction de Heaviside. On donne alors la définition d'un perceptron multicouches :

le perceptron multicouche est un réseau de neurones à couches cachées, avec des cellules élementaires à n entrées $x = (x_1 \cdots x_n)$, défini par un vecteur de poids synaptiques réels w et la sortie o calculée par la fonction d'activation :

$$o(x) = \frac{1}{1 + e^{-y}}$$

où le potentiel post-synaptique est

$$y = w^T x$$

la fonction d'activation σ est une sigmoïde. Puisque l'algorithme de rétropropagation du gradient va s'appuyer sur un calcul de gradient de cette fonction, il est essentiel que le calcul de la dérivée soit simple et rapide, et on utilise parfois à la place de la fonction précédente la fonction $th(x)$, dont la dérivée vaut $1 - th^2(x)$.

2.3.3 Rétropropagation du gradient

Algorithme

Le principe de l'algorithme est, comme dans le cas du perceptron linéaire, de minimiser une fonction d'erreur. Il s'agit ensuite de calculer la contribution à cette erreur de chacun des poids synaptiques. C'est cette étape qui est difficile. En effet, chacun des poids influe sur la cellule correspondante, mais la modification pour cette cellule va influer sur toutes les cellules des couches suivantes. Ce problème est parfois désigné sous le nom de Credit Assignment Problem .
Soit un PMC défini par une architecture à n entrées et à p sorties, soit w le vecteur des poids synaptiques associés à tous les liens du réseau. L'erreur du PMC sur un échantillon d'apprentissage $\mathcal{E}_a$ d'exemples (x^s, c^s) est définie par :

$$E(w) = \frac{1}{2} \sum_{(x^s, c^s) \in \mathcal{E}_a} \sum_{k=1}^{p} (c_k^s - o_k^s)^2$$

où o_k^s est la k$^{\text{ème}}$ composante du vecteur de sortie o^s calculé par le PMC sur l'entrée x^s. L'erreur mesure donc l'écart entre les sorties attendues et calculées sur l'échantillon complet. On suppose $\mathcal{E}_a$ fixé, le problème est donc de déterminer un vecteur $\tilde{w}$ qui minimise $E(w)$. Cependant, de la même façon que pour le perceptron avec la règle de Widrow-Hoff, plutôt que de chercher à minimiser l'erreur globale sur l'échantillon complet, on cherche à

minimiser l'erreur sur chaque présentation individuelle d'exemple. L'erreur pour un exemple est :

$$E_{(x,c)}(w) = \frac{1}{2}\sum_{k=1}^{p}(c_k - o_k)^2$$

Nous notons E la fonction $E_{(x,c)}$ et E est une fonction des poids synaptiques. Dans la suite, nous utilisons les notations suivantes (fig 2.4) :

- chaque cellule est définie par un indice,
- le réseau comporte p cellules de sortie,
- si i est l'indice d'une cellule de sortie, c_i est la sortie attendue pour cette cellule sur l'entrée x,
- w_{ij} est le poids synaptique associé au lien entre la cellule j et la cellule i, ce qui implique qu'elles se trouvent sur deux couches successives par définition de l'architecture,
- x_{ij} est l'entrée associée au lien entre la cellule j et la cellule i,
- $Pred(i)$ est l'ensemble des cellules dont la sortie est une entrée de la cellule i. La cellule n'est donc pas une cellule d'entrée, et tous les éléments de $Pred(i)$ appartiennent à la couche précédente de celle à laquelle appartient la cellule i,
- $Succ(i)$ est l'ensemble des cellules qui prennent comme entrée la sortie de la cellule i. La cellule n'est donc pas une cellule de sortie, et tous les éléments de $Succ(i)$ appartiennent à la couche suivante de celle à laquelle appartient la cellule i.
- y_i l'entrée totale de la cellule i, soit $y_i = \sum_{j \in Pred(i)} w_{ij}x_{ij}$. Remarquons que toutes les cellules de la couche précédant celle de i ne sont pas nécessairement impliquées dans y_i.
- o_i est la sortie de la cellule i, soit $o_i = \sigma(y_i)$,

On note enfin $\frac{\partial E(w)}{\partial w_{ij}} = \frac{\partial E}{\partial w_{ij}}$.
On remarque que w_{ij} ne peut influencer la sortie du réseau qu'à travers le calcul de y_i, et donc :

$$\begin{aligned}\frac{\partial E}{\partial w_{ij}} &= \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial w_{ij}} \\ &= \frac{\partial E}{\partial y_i}x_{ij}\end{aligned}$$

Il suffit donc de calculer $\frac{\partial E}{\partial y_i}$. Pour cela, on distingue deux cas : le cas où la cellule i est une cellule de sortie et le cas où c'est une cellule interne.

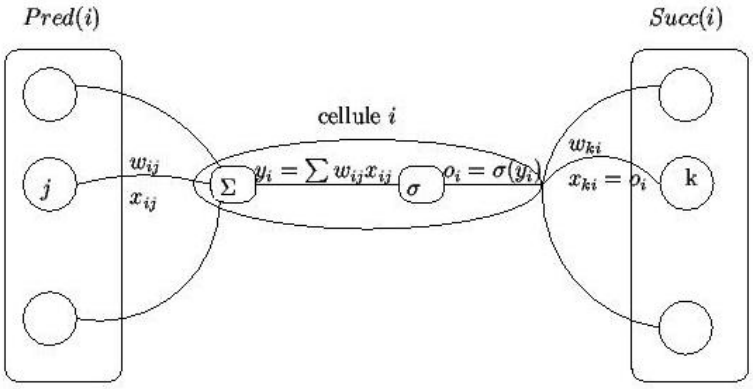


FIGURE 2.4 – Notations pour le PMC

— La cellule i est une cellule de sortie : dans ce cas, y_i ne peut influencer la sortie du réseau que par le calcul de o_i et $\frac{\partial E}{\partial y_i} = \frac{\partial E}{\partial o_i}\frac{\partial o_i}{\partial y_i}$.
La première dérivée est calculée par :

$$\frac{\partial E}{\partial o_i} = \frac{\partial}{\partial o_i}\frac{1}{2}\sum_{k=1}^{p}(c_k - o_k)^2$$

Seul le terme correspondant à $k = i$ a une dérivée non nulle et :

$$\frac{\partial E}{\partial o_i} = -(c_i - o_i)$$

La seconde dérivée est calculée en utilisant la définition du calcul de la sortie d'une cellule élémentaire et de la dérivée de la fonction sigmoïde :

$$\frac{\partial o_i}{\partial y_i} = o_i(1 - o_i)$$

et finalement :

$$\frac{\partial E}{\partial y_i} = -(c_i - o_i)o_i(1 - o_i)$$

— La cellule i est une cellule interne : dans ce cas, y_i va influencer le réseau

par tous les calculs des cellules de l'ensemble $Succ(i)$ et :

$$\begin{aligned}\frac{\partial E}{\partial y_i} &= \sum_{k \in Succ(i)} \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial y_i} \\ &= \sum_{k \in Succ(i)} \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial o_i}\frac{\partial o_i}{\partial y_i} \\ &= \sum_{k \in Succ(i)} \frac{\partial E}{\partial y_k} w_{ki} o_i (1 - o_i)\end{aligned}$$

L'étude de ces deux cas permet de calculer les dérivées partielles $\frac{\partial E}{\partial y_i}$ pour toute cellule i. Le calcul devra être fait pour les cellules de sortie, puis pour les cellules de l'avant-dernière couche jusqu'aux cellules de la première couche. C'est pour cette raison que l'on parle de rétropropagation. Il est alors possible de calculer toutes les dérivées partielles $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_i} x_{ij}$, ce qui permet d'en déduire la modification à effectuer sur les poids synaptiques :

$$\Delta w_{ij} = -\epsilon \frac{\partial E}{\partial w_{ij}}$$

ce qui amène à l'algorithme suivant, en notant $\delta_i = -\frac{\partial E}{\partial y_i}$:

Algorithm 4: Algorithme de rétropropagation du gradient

Initialisation aléatoire des w_i dans $[-\frac{1}{2}, \frac{1}{2}]$
répéter
  Prendre un exemple $(x, c) \in \mathcal{E}_a$
  Calculer o
  pour *toute cellule de sortie* i **faire**
    $\delta_i \leftarrow o_i(1 - o_i)(c_i - o_i)$
  pour *chaque couche* $q - 1$ *à* 1 **faire**
    for *chaque cellule* i *de la couche courante* **do**
      $\delta_i \leftarrow o_i(1 - o_i) \sum_{k \in Succ(i)} \delta_k w_{ki}$
  pour *tout* i, j **faire**
    $w_{ij} \leftarrow w_{ij} + \epsilon \delta_i x_{ij}$
jusqu'à *test* ;

Pour une cellule i de sortie, δ_i correspond à l'erreur usuelle $(c_i - o_i)$ multipliée par la dérivée de la fonction sigmoïde. Pour une cellule i interne, δ_i dépend

de la somme pondérée des erreurs des cellules de la couche suivante, et après présentation de l'entrée x et calcul de la sortie o , le calcul des erreurs δ_i sera effectué de la couche de sortie vers la couche d'entrée.
L'algorithme de rétroprogation du gradient est une extension de l'agorithme de Widrow-Hoff. En effet, dans les deux cas, les poids sont mis à jour à chaque présentation d'exemple et donc on tend à minimiser l'erreur calculée pour chaque exemple et pas l'erreur globale. La méthode donne de bons résultats pratiques. Dans la plupart des cas, on rencontre peu de problèmes dus aux minima locaux, notamment en raison du fait que l'on minimise une erreur locale. Cependant, les problèmes de minima locaux existent. Pour améliorer l'algorithme vis à vis de ce problème, mais aussi pour essayer d'améliorer la vitesse de convergence, une variante couramment utilisée consiste à pondérer la modification des poids en fonction du nombre d'itérations déja effectué. Plus formellement, on fixe une constante $\alpha \in [0, 1[$ appelée moment. Si t est un compteur du nombre d'itérations de la boucle principale, la règle de modification des poids devient :

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}(t)$$

où $\Delta w_{ij}(t) = \epsilon \delta_i x_{ij} + \alpha \Delta w_{ij}(t-1)$. L'intérêt de cette règle est de prendre en compte les modifications antérieures des poids dans le but d'éviter des oscillations perpétuelles.

Critères d'arrêt

Plusieurs critères d'arrêt peuvent être utilisés avec l'algorithme de rétropropagation du gradient. Le plus commun consiste à fixer un nombre maximum de périodes d'entraînement, ce qui fixe effectivement une limite supérieure sur la durée de l'apprentissage. Ce critère est important car la rétropropagation n'offre aucune garantie quant à la convergence de l'algorithme. Il peut arriver, par exemple, que le processus d'optimisation reste pris dans un minimum local. Sans un tel critère, l'algorithme pourrait ne jamais se terminer. Un deuxième critère commun consiste à fixer une borne inférieure sur l'erreur quadratique moyenne. Dépendant de l'application, il est parfois possible de fixer *a priori* un objectif à atteindre. Lorsque l'indice de performance choisi diminue en dessous de cet objectif, on considère simplement que le PMC a suffisamment bien appris ses donnéees et on arrête l'apprentissage.
Les deux critères précédents sont utiles mais ils comportent aussi des limitations. Le critère relatif au nombre maximum de périodes d'entraînement n'est aucunement lié à la performance du réseau. Le critère relatif à l'erreur minimale obtenue mesure quant à lui un indice de performance mais ce

dernier peut engendrer un phénomène dit de sur-apprentissage qui n'est pas désirable dans la pratique, surtout si l'on ne possède pas une grande quantité de données d'apprentissage, ou si ces dernières ne sont pas de bonne qualité.

Un processus d'apprentissage par correction des erreurs, comme celui de la rétropropagation, vise à réduire autant que possible l'erreur que commet le réseau. Mais cette erreur est mesurée sur un ensemble de données d'apprentissage. Si les données sont bonnes, c'est-à-dire quelles représentent bien le processus physique sous-jacent que l'on tente d'apprendre ou de modéliser, et que l'algorithme a convergé sur un optimum global, alors il devrait bien se comporter sur d'autres données issues du même processus physique. Cependant, si les données d'apprentissage sont partiellement corrompues par du bruit ou par des erreurs de mesure, alors il n'est pas évident que la performance optimale du réseau soit atteinte en minimisant l'erreur, lorsqu'on la testera sur un jeu de données différent de celui qui a servi à l'entraînement. On parle alors de la capacité de **généralisation** du réseau, c'est-à-dire sa capacité à bien se comporter avec des données qu'il n'a jamais vu auparavant.

Une solution à ce problème consiste à utiliser un autre critère d'arrêt basé sur une technique de validation croisée. Cette technique consiste à utiliser deux ensembles indépendants de données. En pratique, il s'agit de partitionner $\mathcal{E}_a$ pour entraîner le réseau en un ensemble d'apprentissage (ajustement des poids) un ensemble de validation(calcul d'un indice de performance). Le critère d'arrêt consiste alors à stopper l'apprentissage lorsque l'indice de performance calculé sur les données de validation cesse de s'améliorer pendant plusieurs périodes d'entraînement. Lors de deux périodes successives d'entraînement, des exemples peuvent être échangés entre ensembles d'apprentissage et de validation.

Saturation

Une autre considération pratique dont on doit tenir compte lorsqu'on entraîne un PMC concerne le phénomène de saturation des neurones où, sous certaines conditions, les neurones peuvent à toute fin pratique cesser d'apprendre tellement leur convergence devient lente. Dans ce cas, il convient de normaliser les données à l'entrée d'un PMC, c'est-à-dire de les transformer de manière à éviter tout risque de saturation. Une autre manière de procéder est d'initialiser les poids sur la première couche en choisissant un intervalle de valeurs aléatoires ajusté aux stimuli d'apprentissage

2.3.4 Propriété fondamentale

Terminons par une dernière remarque sur la puissance de représentation des réseaux multi-couches. Nous avons vu que toute fonction booléenne peut être calculée par un PMC linéaire à seuil comprenant une seule couche cachée dont on rappelle qu'elle peut être de taille exponentielle. Ce résultat a été généralisé par Hornik en 1989 au cas des fonctions réelles et des PMC : la plupart des fonctions numériques peuvent être approximées avec une précision arbitraire par des réseaux à une seule couche cachée. Mais comme dans le cas booléen, cette couche cachée peut être démesurément grande et le théorème de Hornik est essentiellement un résultat théorique sur l'expressivité des réseaux multi-couches.

Plus formellement, la propriété fondamentale des réseaux de neurones est l'**approximation parcimonieuse**, qui traduit deux propriétés distinctes : d'une part les réseaux de neurones sont des approximateurs universels, et d'autre part, une approximation à l'aide d'un réseau de neurones nécessite, en général, moins de paramètres ajustables que les approximateurs usuels.

Approximateurs universels

Cybenko a énoncé en 1989 la propriété suivante : toute fonction bornée, suffisamment régulière, peut être approchée uniformément, avec une précision arbitraire, dans un domaine fini de l'espace de ses variables, par un réseau de neurones comportant une couche de neurones cachés en nombre fini, possédant tous la même fonction d'activation, et un neurone de sortie linéaire.

Parcimonie

Hornik a montré en 1994 que si la sortie d'un réseau de neurones est une fonction non linéaire des paramètres ajustables, elle est plus parcimonieuse que si elle était une fonction linéaire de ces paramètres. De plus, pour les réseaux dont la fonction d'activation des neurones est une sigmoïde, l'erreur commise dans l'approximation varie comme l'inverse du nombre de neurones cachés, et elle est indépendante du nombre de variables de la fonction à approcher. Ainsi, pour une précision donnée (*i.e.* étant donné un nombre de neurones cachés) le nombre de paramètres du réseau est proportionnel au nombre de variables de la fonction à approcher.

Modélisation statistique

Dans la plupart des cas d'utilisation des réseaux de neurones, il va s'agir d'établir un modèle d'une fonction inconnue à partir de mesures bruitées de l'ensemble d'apprentissage, permettant de reproduire les sorties à partir des entrées, et de proposer une généralisation à des données test. On cherche alors la **fonction de régression** du processus considéré, *i.e.* la fonction obtenue en calculant la moyenne d'une infinité de mesures effectuées en chaque point du domaine de validité du modèle. Le nombre de points de ce domaine étant lui-même infini, la connaissance de la fonction de régression nécessiterait donc une infinité de mesures en un nombre infini de points.
Les réseaux de neurones, en raison de leur propriété fondamentale, sont de bons candidats pour réaliser une approximation de la fonction de régression à partir d'un nombre fini de mesures. Ils entrent donc dans le cadre des méthodes statistiques d'apprentissage, et élargissent ce domaine déjà bien exploré pour des fonctions de régression linéaire au cas non linéaire.

2.4 Détermination automatique de l'architecture

Le problème principal dans le choix de l'architecture d'un réseau de neurones est la détermination du nombre de couches et de neurones cachés. En effet, connaissant l'ensemble d'apprentissage, les couches d'entrée et de sortie sont immédiatement déterminées. Des résultats théoriques montrent qu'une couche cachée suffit pour résoudre des problèmes ayant des données continues ou discrètes. On peut même montrer que l'écart entre l'approximation réalisée par un réseau de neurones et la fonction à approcher est inversement proportionnel au nombre de neurones cachés. Ce résultat ne donne que des estimations, ou des bornes larges, du nombre de neurones cachés nécessaires. Il n'existe pas, à l'heure actuelle, de résultat théorique permettant de prévoir le nombre de neurones cachés nécessaires pour obtenir une performance spécifiée du modèle, compte tenu des données disponibles. Il faut donc nécessairement mettre en oeuvre une procédure numérique de conception de modèle. L'idée la plus naturelle consisterait à choisir le nombre de neurones cachés le plus grand possible, puisque c'est ce qui assure la plus grande précision à l'approximation uniforme d'une fonction. Ce serait oublier que le problème réel que l'on se pose n'est pas un problème d'approximation uniforme, mais un problème d'ajustement d'une fonction à un nombre fini de points ; il faut donc, non seulement que la fonction réalisée par le réseau

de neurones passe au plus près, au sens des moindres carrés, des points de l'ensemble d'apprentissage, mais également qu'il soit capable de généraliser de manière satisfaisante. Si le réseau de neurones possède un nombre de paramètres excessif, en raison d'un nombre excessif de neurones cachés, sa sortie peut passer avec une très grande précision par tous les points d'apprentissage, mais fournir des résultats dépourvus de signification entre ces points ; s'il possède un nombre de paramètres trop restreint, le modèle n'est pas suffisamment riche pour rendre compte de la complexité de la fonction de régression inconnue. Ce dilemme (appelé **dilemme biais-variance**) est le problème essentiel que doit affronter le concepteur de modèles. En pratique, on s'efforcera toujours de faire en sorte que le nombre de paramètres ajustables soit petit devant le nombre d'exemples : la parcimonie intrinsèque aux réseaux de neurones à fonction d'activation sigmoïde permet de réaliser cette condition plus facilement que si l'on utilise une méthode de régression multilinéaire par exemple.
Les travaux de recherche pour la résolution du problème d'architecture des réseaux de neurones peuvent se classer en trois grandes catégories :

- utilisation d'un ensemble de connaissances du domaine pour définir l'architecture initiale du réseau, et apprentissage par l'exemple. Parmi ces techniques, on peut citer KBANN qui construit le réseau à partir d'un ensemble de règles d'inférence du domaine (clauses de Horn sans cycle)
- construction à partir des exemples d'un réseau capable de bien classer les exemples. A chaque étape, on ajoute un neurone (voir une couche) caché(e) pour corriger les erreurs produites par le réseau à l'étape précédente (méthodes MTiling, MTower, MUpstart, Distal par exemple)
- construction à partir d' un réseau de neurones surdimensionné, apprentissage sur ce réseau et suppression progressive des paramètres du réseau de moindre importance (élagage). Plusieurs possibilités sont envisageables pour la mesure de cette importance, comme le calcul de l'amplitude des paramètres, de l'accroissement de la fonction coût (méthodes OBD et OBS par exemple)

Le modèle et l'architecture étant choisie, un problème est également de choisir les bonnes valeurs pour les paramètres (ϵ et α par exemple). Pour cela, on découpe l'ensemble d'apprentissage en un ensemble d'apprentissage, un ensemble de validation et un ensemble test. Lors de la phase d'apprentissage, on arrête périodiquement l'apprentissage, on estime l'erreur réelle sur l'ensemble de validation, on met à jour les paramètres en fonction de la variation de cette erreur estimée. L'ensemble test sert à estimer l'erreur réelle à la fin de l'apprentissage.

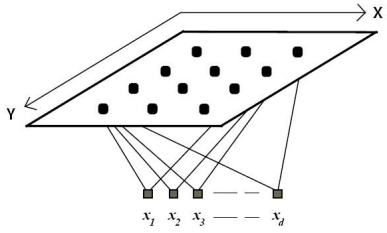


FIGURE 2.5 – Carte de Kohonen à deux dimensions

2.5 Quelques autres types de réseaux

Terminons par la présentations de trois autres types de réseaux de neurones couramment utilisés.

2.5.1 Cartes de Kohonen

Les réseaux de Kohonen, ou cartes auto-organisatrices (Self Organizing Maps, SOM) sont des réseaux d'apprentissage non supervisés, qui regroupent les observations en classes, en respectant la topologie des observations.

Architecture

Les neurones d'une carte auto-organisatrice sont disposés en une seule couche, de laquelle émane une topologie définie par une notion de voisinage des cellules (2.5). Les principes de base de fonctionnement de ce réseau, inspirés des constatations biologiques relatives à l'ordonnancement des neurones, furent introduits en 1981 par Teuvo Kohonen. L'entrée du réseau, unique et commune à tous les neurones, est le vecteur de caractéristiques X qui représente un objet. A chaque neurone sont associées des coordonnées (x, y) indiquant sa position sur la carte, ainsi qu'un vecteur de poids synaptiques

Apprentissage

L'apprentissage de la carte auto-organisatrice est basé sur deux constatations biologiques essentielles, à savoir que, dans le cerveau :

1. chaque cellule nerveuse correspond à un stimulus spécifique ;
2. il existe une région d'intense activité autour de la cellule la plus stimulée.

Le modèle d'entraînement de la carte consistera donc à :

1. sélectionner le neurone correspondant le mieux à un signal d'entrée donné ;
2. induire dans un voisinage de l'élu une région d'intense activité.

Dans un premier temps, l'apprentissage est non-supervisé et consiste à répéter les étapes suivantes : Le terme quadratique du calcul de distance

---

Présenter un objet X à l'entrée du réseau, sans préciser la classe à laquelle il appartient
Rechercher le neurone q^* dont le vecteur de poids W_q est le plus proche, au sens de la distance euclidienne, du vecteur d'entrée :

$$q^* = arg \min_q \frac{1}{2}(X - W_q)^T(X - W_q)$$

Adapter le vecteur de poids de ce neurone ainsi que ceux de ses voisins topologiques $q \in \mathcal{V}(q^*)$ de manière à ce qu'ils se rapprochent davantage du vecteur d'entrée :

$$\begin{cases} W_q(t+1) & = W_q(t) + \eta(X - W_q(t)) & \text{si q}\in \mathcal{V}(q^*) \\ W_q(t+1) & = W_q(t) & \text{sinon} \end{cases}$$

η étant le taux d'apprentissage.

---

peut être omis car il est constant pour tous les neurones. La recherche de celui dont le vecteur de poids synaptiques est le plus proche du vecteur d'entrée s'effectue alors selon :

$$q^* = arg \max_q \left(W_q^T X - \frac{1}{2} W_q^T W_q \right)$$

Il s'agit donc de rechercher le neurone dont la valeur de sortie est la plus élevée, sous la condition d'imposer que le seuil de chaque neurone soit tel que :

$$w_{q_0} = \frac{1}{2} W_q^T W_q$$

et donc

$$q^* = arg \max_q (W_q^T X)$$

Plusieurs méthodes peuvent être envisagées pour réaliser l'adaptation des poids synaptiques. Par exemple, celle qui, par expérience, s'est avérée la plus performante pour organiser des cartes destinées à des tâches de reconnaissance porte le nom de règle d'adaptation centrale. Le taux d'apprentissage

η décroît à la fois avec le temps et avec la distance d par rapport au neurone élu q^*. La notion de voisinage est donc implicitement contenue dans la définition du taux d'apprentissage. Le fait de débuter l'adaptation des poids avec un taux d'apprentissage plus élevé permet d'obtenir assez rapidement une organisation rudimentaire générale des vecteurs de poids, tandis que sa décroissance progressive permet ensuite d'affiner la résolution de la carte. Les valeurs du taux d'apprentissage ainsi que celles de sa vitesse de décroissance dépendent du problème à traiter et doivent être déterminées empiriquement. La règle d'adaptation centrale permet de réduire les formules de mise à jour des poids à l'expression :

$$W_q(t+1) = W_q(t) + \eta(t,d)(X - W_q(t))$$

A l'issue d'un nombre suffisant d'itérations, les vecteurs de poids synaptiques se sont accordés spécifiquement sur les divers aspects des variables d'entrée. Un quantificateur vectoriel a été créé (la quantification vectorielle consiste à substituer à un vecteur X, l'indice du vecteur le plus proche, au sens de la distance Euclidienne, déterminé parmi un ensemble restreint de prototypes possibles. Utilisée en mode de classification, la quantification consiste à rechercher la classe du prototype, plutôt que son indice) : les vecteurs de poids, qui sont devenus les centres des vecteurs d'entrée, ont migré dans l'espace de représentation R^d de ceux-ci, afin de représenter au mieux l'espace des signaux d'entrée. La carte auto-organisatrice présente cependant une différence par rapport à un quantificateur vectoriel conventionnel, puisque les cellules topologiquement proches sont ici sensibles à des signaux physiquement similaires.

Ce quantificateur n'est toutefois pas encore utilisable pour une tâche de reconnaissance. Le processus d'apprentissage n'ayant en effet pas été, jusqu'ici, supervisé, il convient à présent de déterminer quelles cellules se sont accordées sur tel ou tel type de stimulus : c'est l'objet de l'étiquetage. Cette procédure consiste à attribuer à chaque cellule une étiquette qui correspond à la classe d'entrée dont elle est la plus proche. A cette fin, l'ensemble des vecteurs de caractéristiques des objets d'apprentissage sont à nouveau présentés au système, et la fréquence avec laquelle chaque neurone est le plus proche d'une classe particulière est calculée. A la suite de cette étape, chaque neurone se voit associé à la classe pour laquelle il a été le plus fréquemment choisi. Il peut se produire que certains neurones ne soient jamais sélectionnés en tant que le plus proche d'un vecteur d'entrée d'un prototype d'apprentissage. Ils doivent, dans ce cas, être écartés lors de l'utilisation de la carte en reconnaissance, afin d'éviter toute indétermination.

A l'issue de la procédure d'étiquetage, la carte auto-organisatrice peut enfin être utilisée en classificateur. Un vecteur de caractéristiques est présenté à

l'entrée du réseau, et la classe reconnue est celle qui est associée au neurone dont le vecteur de poids synaptiques est le plus proche du vecteur d'entrée.

Algorithme LQV

L'inconvénient de l'apprentissage non supervisé décrit précédemment est que les poids des neurones sont organisés de manière à minimiser une erreur de quantification, alors que l'objectif réel est de reconnaître la classe du vecteur d'entrée. Seules les frontières entre les classes importent donc vraiment. Une meilleure estimation de celles-ci peut être obtenue en introduisant une phase d'apprentissage supplémentaire, au cours de laquelle la classification des objets, ignorée jusqu'ici, est prise en considération. Cet apprentissage supervisé consiste à présenter à nouveau des vecteurs de caractéristiques au réseau, et à réajuster les vecteurs de poids de manière à minimiser le taux global d'erreurs de classification. Cet algorithme, appelé Learning Vector Quantization (LVQ), a également été développé par Kohonen. L'algorithme initial consistait à renforcer, à chaque itération, la ressemblance entre le vecteur de poids (centroïde) du neurone élu et celui du vecteur de caractéristiques présenté lorsque les classes de ceux-ci étaient équivalentes, et à éloigner les deux vecteurs dans le cas contraire. Le vecteur de poids du neurone le plus proche du vecteur d'entrée est alors adapté selon :

$$\begin{cases} W_{q^*}(t+1) & = W_{q^*}(t) + \eta(t)(X - W_{q^*}(t)) \quad \text{si } \omega_{q^*} = \omega_X \\ W_{q^*}(t+1) & = W_{q^*}(t) - \eta(t)(X - W_{q^*}(t)) \quad \text{sinon} \end{cases}$$

où

— $\eta(t)$ est un gain scalaire qui décroît de manière monotone avec le temps
— ω_X et ω_{q^*} désignent respectivement la classe de l'objet représenté par le vecteur de caractéristiques X et celle associée à la cellule élue q^*

L'algorithme connu sous le nom de LVQ2 est une évolution de cet algorithme initial. Trois conditions doivent à présent être remplies pour qu'il y ait effectivement réajustement des centroïdes :

1. le centroïde le plus proche du vecteur d'entrée doit être de classe différente de celui-ci ;
2. le centroïde le plus proche suivant doit appartenir à la même classe que le vecteur d'entrée ;
3. le vecteur d'entrée doit se trouver dans une fenêtre symétrique définie autour du plan médian entre ces deux centroïdes.

Si ces trois conditions sont simultanément vérifiées, le vecteur de référence incorrect est éloigné du vecteur d'entrée alors que le second vecteur en est

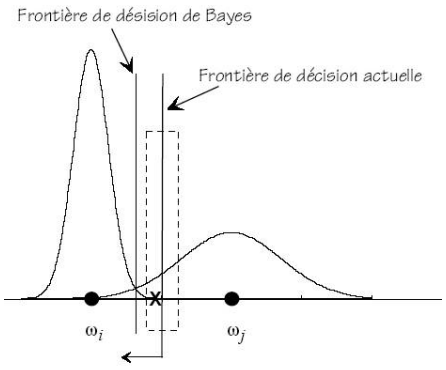


FIGURE 2.6 – Adaptation des centoïdes par LVQ2

rapproché. Les équations qui assurent un tel apprentissage sont les suivantes :

$$\begin{cases} W_{q_1}(t+1) &= W_{q_1}(t) - \eta(t)(X - W_{q_1}(t)) \\ W_{q_2}(t+1) &= W_{q_2}(t) + \eta(t)(X - W_{q_2}(t)) \end{cases}$$

q_1 et q_2 désignant respectivement les neurones dont le vecteur de poids est le plus proche du vecteur d'entrée et le suivant le plus proche.

A titre d'exemple, la figure 2.6 illustre une application de l'algorithme LVQ2 dans un cas monodimensionnel. Les deux classes ω_i et ω_j sont séparées par une frontière qui, à la suite de l'apprentissage non supervisé, se situe à mi-distance entre les deux centroïdes i et j associés à ces classes. La frontière optimale de décision se situe en réalité là où l'expression $P(X|\omega_i)P(\omega_i) = P(X|\omega_j)P(\omega_j)$ est vérifiée. Lorsqu'un objet X appartenant à la classe ω_j apparaît dans la partie gauche d'une fenêtre centrée sur la région de décision actuelle, donc mal placé, les trois conditions exigées par l'algorithme LVQ2 sont vérifiées, et il y aura donc réajustement des centroïdes. Le centroïde correspondant à la classe incorrecte sera éloigné du vecteur X, tandis que celui qui correspond à la classe correcte en sera rapproché. La frontière de décision entre les deux centroïdes se rapproche, dans ce cas, de la frontière optimale de Bayes.

Propriétés

Hormis le fait que les neurones topologiquement proches répondent à des stimuli physiquement semblables, la carte auto-organisatrice de Kohonen n'apporte rien de plus qu'un quantificateur vectoriel conventionnel. Bien que cette particularité puisse être intéressante à observer, elle est en soi inutile à la classification, puisque seule la classe associée au neurone dont le vecteur de poids

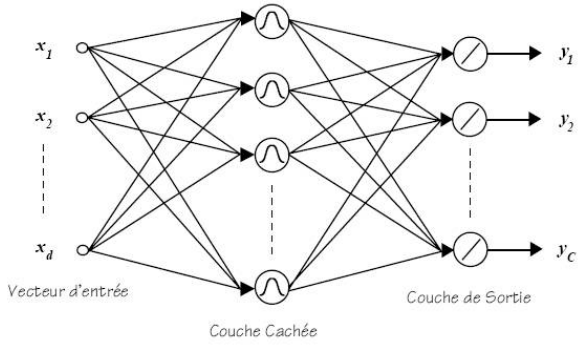


FIGURE 2.7 – Exemple de réseau à fonction de base radiale

synaptiques est le plus proche du vecteur d'entrée est alors requise.
On montre facilement que les frontières de décision entre deux centroïdes sont linéaires par morceaux. Un tel classificateur, qui utilise un nombre multiple mais restreint, de vecteurs de prototypes pour chaque classe, peut être considéré comme un système intermédiaire entre le classificateur euclidien, qui utilise un seul prototype pour chaque classe, et le classificateur du plus proche voisin, qui utilise tous les vecteurs de caractéristiques des objets d'apprentissage comme prototype pour chaque classe. L'apprentissage discriminant, que subit dans un second temps la carte auto-organisatrice, lui confère toutefois un certain avantage. Le nombre optimal de cellules du réseau dépend essentiellement de l'application envisagée, et, comme dans le cas d'un quantificateur vectoriel conventionnel, doit être déterminé empiriquement.

2.5.2 Réseaux à base radiale

Architecture

Les réseaux de fonctions à base radiale comportent deux couches de neurones (figure 2.7). Les cellules de sortie effectuent une combinaison linéaire de fonctions non linéaires, fournies par les neurones de la couche cachée. Ces fonctions produisent une réponse différente de zéro seulement lorsque l'entrée se situe dans une petite région bien localisée de l'espace des variables. Bien que plusieurs modèles de telles fonctions existent, le plus courant est de type gaussien :

$$y_{1,i}(X) = e^{-\frac{(X-W_{1,i})^T(X-W_{1,i})}{2\sigma_i^2}}$$

où :

— X est le vecteur d'entrée du réseau
— $y_{1,i}$ est la sortie du neurone i de la première couche
— $W_{1,i}$ est le vecteur de poids synaptiques du neurone i
— σ_i^2 est le paramètre de normalisation du neurone i

La sortie d'un neurone de la seconde couche est simplement donnée par :

$$y_{2,i} = W_{2,i}^T Y_1$$

où Y_1 le vecteur des sorties des neurones de la première couche.
Plus le vecteur d'entrée est proche du centre d'une gaussienne, plus la sortie du neurone de la première couche qui lui correspond est élevée. la dénomination "fonction à base radiale" vient du fait que la gaussienne est symétrique radialement, c'est-à-dire que la valeur de sortie obtenue est la même pour toutes les entrées situées à une même distance du centre de la gaussienne.

Apprentissage

L'apprentissage d'un tel réseau est généralement scindé en deux parties. Dans un premier temps, les poids des neurones de la couche cachée sont adaptés au moyen d'une quantification vectorielle. Il existe plusieurs manières d'effectuer cette dernière, mais celle qui est probablement la plus efficace et la plus simple à mettre en oeuvre est l'algorithme K-moyennes. Lorsque les poids des cellules cachées sont fixés, les paramètres de normalisation σ_i^2 sont déterminés en calculant la dispersion des données d'apprentissage associées à chaque centroïde. La seconde couche du réseau peut alors être entraînée. L'apprentissage est cette fois supervisé (les valeurs de sortie désirées sont fournies), et s'effectue typiquement à l'aide d'un algorithme basé sur un critère des moindres carrés. L'entraînement de la seconde couche est très rapide, car, d'une part, les sorties de la couche cachée peuvent être calculées une fois pour toutes pour tous les exemples d'apprentissage, et d'autre part, les sorties des neurones de la seconde couche sont linéaires. Les méthodes classiques d'apprentissage de transformations linéaires, tel le critère d'apprentissage par correction d'erreur du perceptron, peuvent donc être appliquées.

Comparaison au PMC

Le fait que l'apprentissage de la couche cachée soit non supervisé est toutefois un inconvénient de ce modèle de réseau de neurones, vis-à-vis d'un perceptron multicouches. Pour résoudre partiellement ce problème, des méthodes d'apprentissage supervisé de réseaux de fonctions à base radiale ont également

été développées. Un avantage du réseau de fonctions à base radiale est que sa phase d'apprentissage est plus rapide que celle du perceptron multicouches. Ceci ne constitue toutefois qu'un avantage utile à l'étude et à la mise au point d'un système (de classification ou autre). Dans l'optique d'une utilisation pratique, c'est en effet le volume de calcul à effectuer en phase d'exploitation qu'il est préférable d'optimiser. En théorie, le réseau de fonctions à base radiale est capable, tout comme le perceptron multicouches, d'effectuer une approximation arbitrairement proche de n'importe quelle transformation non linéaire. La principale différence entre les deux est la nature de la fonction d'activation des neurones de la couche cachée. La non-linéarité sigmoïdale utilisée dans le perceptron multicouches fournit une sortie différente de zéro pour une région infiniment grande de l'espace d'entrée, ce qui lui confère un certain pouvoir de généralisation dans des régions où peu de données d'apprentissage sont disponibles. Au contraire, la non-linéarité radiale utilisée ici, ne fournit une réponse différente de zéro que localement. Le nombre d'unités cachées nécessaire pour permettre au réseau de recouvrir l'ensemble de l'espace d'entrée peut dès lors s'avérer parfois très élevé.
Une autre différence entre ces deux modèles de réseaux de neurones est que la non-linéarité, présente dans la couche de sortie du perceptron multicouches, est inexistante dans le réseau de fonctions à base radiale, ce qui constitue un désavantage de ce dernier vis-à-vis du premier. La caractéristique linéaire de la couche de sortie d'un réseau de fonctions à base radiale rend ce dernier plus proche du perceptron que ne l'est le perceptron multicouches. Le réseau de fonctions à base radiale diffère cependant du perceptron, dans le sens où il est capable de réaliser des transformations non linéaires de l'espace d'entrée.

2.5.3 Réseaux dynamiques

Introduisons pour finir les réseaux dynamiques, autre type de réseaux de neurones aux applications importantes. Les opérations réalisées par les neurones sont ici régies par des équations différentielles, à temps continu ou à temps discret, ce qui permet de mieux modéliser les problèmes réels où l'on rencontre des systèmes dynamiques non-linéaires. Ceci est particulièrement vrai pour des problèmes de prédiction, d'identification, ou encore de contrôle. Une manière très simple et fréquemment utilisée de prendre en compte l'information temporelle est d'utiliser un perceptron multicouches statique, pour lequel le vecteur de caractéristiques fourni à l'entrée est constitué d'une suite de vecteurs dynamiques (figure 2.8). Ce modèle de réseau porte alors le nom de réseau de neurones à délai temporel. Comme il ne comporte pas de rétro-action, ce réseau de neurones peut toujours être entraîné à l'aide de l'algorithme de rétro-propagation du gradient standard. Les réseaux dyna-

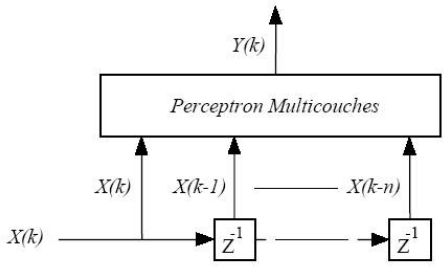


FIGURE 2.8 – Exemple de réseau dynamique

miques proprement dit, sont ceux dans lesquels les sorties du perceptron multicouches, voire même celles des neurones cachés, sont réinjectées à l'entrée du réseau, avec un certain délai, en sus des entrées conventionnelles. L'algorithme d'apprentissage, quoique toujours basé sur le calcul du gradient d'une fonction de coût, doit alors être revu afin d'être adapté à la structure particulière du modèle de réseau utilisé.
Un modèle particulièrement intéressant de réseau de neurones dynamique sont les réseaux récurrents dynamiques. Le modèle à la base de ceux-ci consiste en une seule couche, dont les neurones sont complètement interconnectés. Généralement, une partie de ces neurones est utilisée en tant qu'entrée du système, et une autre partie en tant que sortie. La dynamique de ce réseau est régie, dans le cas discret, par des équations de type :

$$y_i(k+1) = \phi\left(W_i^T Y(k) + X(k)\right)$$

Où k désigne l'instant discret.
Les réseaux récurrents dynamiques se révèlent en pratique particulièrement efficaces dans beaucoup de problèmes de modélisation, d'identification, ou encore de contrôle non-linéaire. En règle générale, cependant, seuls les réseaux de neurones à délai temporel, qui ne sont donc pas réellement dynamiques au sens strict du terme, s'avèrent être utilisés en pratique pour des tâches de classification.