Cours de

Neurocomputing - Réseaux de Neurones

mars-mai 2008
v 1.0
Benoît Decoux
(benoit.decoux@wanadoo.fr)

Sommaire

Introduction

Le domaine des Réseaux de Neurones concerne une catégorie d'algorithmes basés sur l'obtention de fonctionnements complexes à partir d'opérations simples réalisées par des "processeurs" élémentaires. Il s'agit d'un domaine récent (années 80) qui est un principalement un domaine de Recherche mais qui a donné lieu à quelques applications pratiques et industrielles.

Ces processeurs élémentaires constituent des modèles ultra-simplifiés des neurones naturels.

Les principales propriétés des réseaux de neurones sont leur capacité d'apprentissage à partir d'exemples. Ils peuvent ensuite généraliser à des données qui ne leur ont pas été présentées. Cet apprentissage peut être supervisé (données d'entrée associées aux sorties désirées correspondantes) ou non. La résolution des problèmes n'est donc pas préalablement effectuée par l'homme puis codée sous forme de règles, mais ces règles sont déduites d'un apprentissage.

I) Généralités

I.1) Historique

Les débuts

1943 : Neurone binaire de Mc Culloch et Pitts. Leur raisonnement était qu'un neurone binaire peut réaliser une fonction logique simple ET et OU ; ils peuvent donc réaliser n'importe quelle fonction logique complexe ou arithmétique. Von Neumann s'est inspiré de leurs travaux.

1949 : publication du livre de Hebb : "Organization of Behavior". Hebb a cherché à modéliser le conditionnement opérant et à le relier à l'activité neuronale. Il a proposé une règle d'apprentissage très simple ayant été utilisée par la suite dans de nombreux modèle de RN (notamment le modèle de Hopfield).

Les premiers succès

1958 : Perceptron de Rosenblatt. Il s'agissait d'un d'un réseau à 1 couche ne pouvant traiter des problèmes simples, linéairement séparables (par exemple, la fonction OU logique).

1960 : Adaline de Widrow et Hoff

Les années de disette

1969 : Purblication du livre "Perceptrons" par Minsky et Pappert. Ces chercheurs mettaient en lumière les limitation du perceptron de Rosenblatt (aux problèmes linéairement séparables). Cette publication a eu un impact négatif fort sur l'ensemble de la recherche dans les RN.

Les années 70 ont vu le développement d'autres techniques d'intelligence artificielle :systèmes experts, traitement de symboles, systèmes à base de règles, développement des langages Prolog et Lisp, etc).

L'explosion

1982-1984 :

- 2 articles fondateurs d'Hopfield
- articles fondateurs et livre de Kohonen : "Self-organization and associative memory"

1986 : Publication d'un recueil d'articles par le PDP Research Group par Rumelhart et Mc Clelland.

1987 : Première conférence Internationale à San Diego.

I.2) Applications

I.2.1) Applications par types de problèmes

a) Analyse de données

Certains types de Réseaux de Neurones (comme la carte auto-organisatrice de Kohonen) permettent d'analyser des propriétés des données, qui sont la plupart du temps multi-dimensionnelles, notamment la détection de regroupement (clustering).

b) Classification

La classification est l'étape ultérieure à l'analyse de données. Il s'agit d'une prise de décision, sur l'appartenance ou non à une catégorie. On parle également de Reconnaissance de formes. Les formes dont il est question peuvent être des signaux, des images ou des éléments caractéristiques extraits de formes du monde réel, codés sous forme de vecteurs.

c) Modélisation de fonctions / prédiction

Il s'agit de détecter des propriétés sous-jacentes aux évolutions temporelles. Il y a un fort besoin d'algorithmes en prédiction dans l'industrie : prédiction du temps par la météo, de la consommation d'énergie, de la fréquentation d'aéroports, etc.

Les Réseaux de Neurones donnent de bons résultats dans ce domaine, et il existe des logiciels commerciaux les exploitant dans ce domaine.

d) Optimisation, satisfaction de contraintes

Dans l'industrie, il existe de nombreux problèmes de satisfaction de contrainte (comme la gestion de tâches dans un certain ordre) et d'optimisation (détermination du schéma des pistes optimal sur les circuits imprimés, problème du voyageur de commerce, etc).

Les Réseaux de Neurones présentent l'avantage de proposer rapidement (sans calculs longs) des solutions presqu'optimales, suffisantes pour la plupart des applications.

e) Mémorisation associative

La mémorisation associative consiste à rappeler une forme à partir d'une version incomplète ou dégradée d'elle-même. Cette propriété est utile à l'application de reconnaissance de formes.

C'est une des propriétés "spectaculaire" du modèle de Hopfield, qui a participé à l'explosion des RN dans les années 80, même si ce modèle est assez limité en pratique car le nombre de formes mémorisables est environ égal à 15% du nombre de neurones, et très coûteux en nombre de connexions.

I.2.2) Applications par domaines

a) Biométrie

- Reconnaissance biométrique : empreinte digitale, iris de l'œil, visage (par exemple pour l'accès à des lieux sécurisés) ;
- Classification automatique d'objets divers.

b) Interface Homme-Machine

- Détection de visages
- Reconnaissance de parole (par exemple pour la conversion parole vers texte), de l'écriture (par exemple traitement ou archivage de documents manuscrits) ;

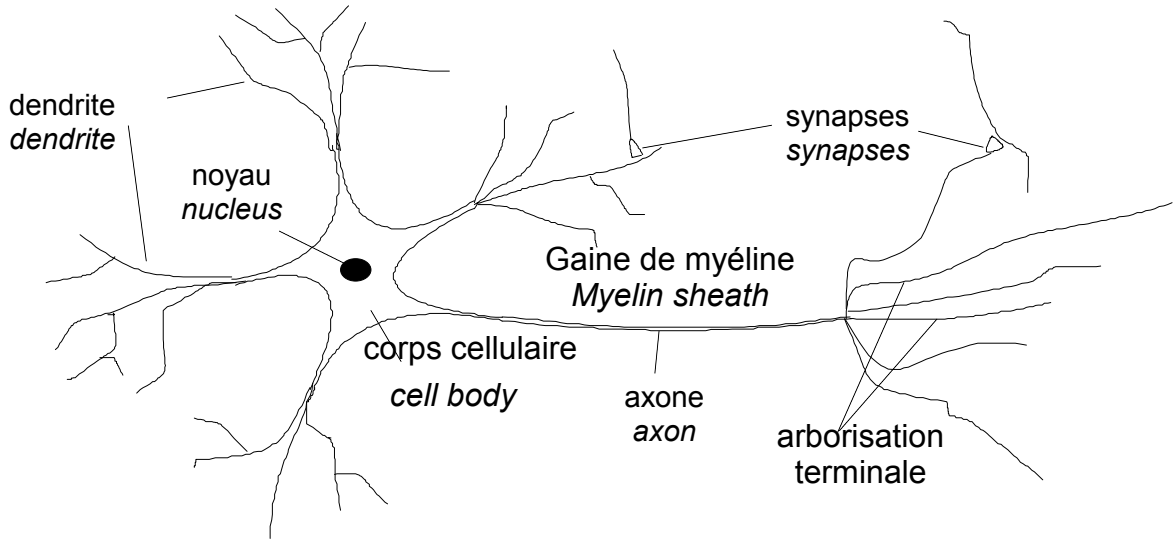
c) Finance

Dans le domaine de la finance, il y a un fort besoin en algorithmes permettant de prévoir l'évolution futures des marchés en fonction des évolutions et évènements passés.

Les Réseaux de Neurones, qui apprennent par l'exemple et donnent de bons résultats en prédiction, suscitent l'intérêt et ont déjà donné naissance à des applications concrète dans ce domaine.

I.3) Les neurones biologiques

Il existe un grand nombre de types de neurones biologiques différents. La figure ci-dessous représenté un neurone typique.



Les neurones communiquent entre eux par l'intermédiaire des synapses.

Les synapses d'un neurones sont connectées aux dendrites d'autres neurones.

L'information transitent dans les neurones par l'intermédiaires des axones, des dendrites vers l'arborisation terminale.

La longueur de l'axone peut aller jusqu'à 10000 fois celle du soma (de l'ordre de la dizaine de micromètres).

Au niveau de la connexion entre 2 neurones, la communication s'effectue par un neurotransmetteur chimique.

Les synapses peuvent être excitatrices ou inhibitrices.

Un neurone, quand il est actif, émet des potentiels d'actions (impulsions électriques qui traversent l'axone des dendrites vers les synapses) Quand un neurone est excité par suffisamment d'autres neurones actifs, il devient lui-même actif.

Le nombre de neurones dans le cerveau humain est de l'ordre de 100 milliards (10^{11}). Chaque neurone peut avoir de l'ordre de 10000 connexions avec les autres neurones.

I.4) Modélisation

Le domaine des réseaux de neurones est également appelé *connexionnisme*.

Un neurone est un processeur élémentaire (ultra-simple) qui modélise grossièrement un neurone naturel. L'opération qu'il réalise est une somme pondérée d'un certain nombre de signaux d'entrée (ce nombre peut être variable), et applique cette somme à une fonction de transfert qui peut avoir différentes formes (voir figure ci-dessous). Les coefficients de pondération sont appelés coefficients synaptiques, ou poids synaptiques.

Les signaux d'entrée sont constitués par les signaux de sortie d'autres neurones connectés au neurone considéré, ou des signaux d'entrée. Les coefficients synaptiques modélisent les "forces" de connexion entre les neurones. Ces forces de connexion sont adaptées par *apprentissage*.

I.4.1) Le neurone

a) Modèle courant

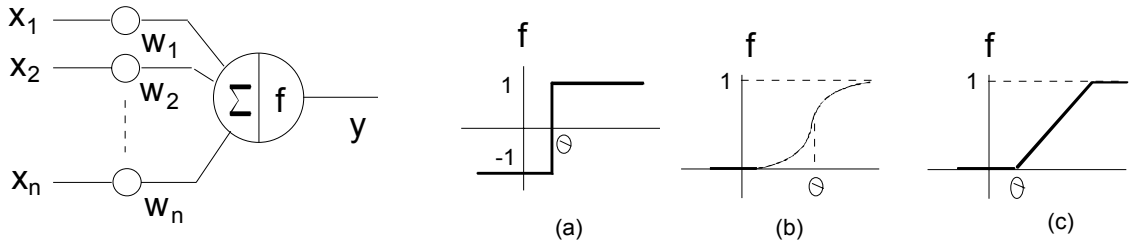
L'opération réalisée par un modèle de neurone est :

$$y = f\left(\sum_{i=1}^{n} w_i x_i\right)$$

ou

$$y = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$$

où θ est appelé biais. Il correspond à un décalage de la fonction f sur l'axe des abscisses. L'argument de f est appelé "état interne" du neurone ou encore potentiel post-synaptique. F est appelée fonction d'activation ou encore fonction de transfert.



La fonction d'activation peut avoir diverses formes :

(a) *seuil logique*, (neurone de McCulloch et Pitts) ; les états de sortie peuvent alors être 0 et 1 ou –1 et +1

(b) *logistique* ; elle a une forme similaire à la fonction seuil, mais est continue : elle peut varier entre –1 et +1, ou 0 et +1

(c) *linéaire* avec ou sans saturation

La fonction de la figure (b) ci-dessus s'appelle sigmoïde. Elle s'exprime par :

$$f(x) = \frac{1}{1 + e^{-(x-\theta)/a}}$$

Le paramètre *a* est la pente de la sigmoïde au niveau de son point d'inflexion.

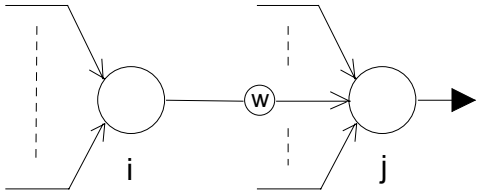
Le décalage de la fonction de transfert sur l'axe des abscisses θ peut être vu comme un lien d'entrée supplémentaire du neurone, dont l'entrée est une valeur fixe. Cette entrée constante peut être notée par exemple x_0, et le poids du lien connecté à cette entrée w_0. Dans ce cas on peut écrire la sortie du neurone de la façon suivante :

$$y = f\left(\sum_{i=0}^{n} w_i x_i\right)$$

où f est centrée sur 0, et n est le nombre d'entrées du neurone.

Il y a 2 cas possibles pour une entrée d'un neurone : être constituée par :

- la sortie d'un autre neurone
- une entrée, c'est à dire une composante d'un vecteur de données



Les poids synaptiques sont les paramètres libres du réseau. Ils sont déterminés automatiquement en fonction du problème par apprentissage, une fois la structure du réseau établie.

b) Relation entrées/sortie

Produit scalaire

Le produit scalaire entre 2 vecteurs est défini par :

$$p_s(v_1, v_2) = \sum_{i=1}^{n} x_{1i}.x_{2i}$$

Lien avec covariance et corrélation

Le produit de corrélation est défini par :

$$p_c(v_1, v_2) = \frac{\sum_{i=1}^{n} (x_{1i} - \overline{x_1}).(x_{2i} - \overline{x_1})}{\sqrt{\sum_{i=1}^{n} (x_{1i} - \overline{x_1})^2}.\sqrt{\sum_{i=1}^{n} (x_{2i} - \overline{x_2})^2}}$$

En statistique, on l'utilise pour comparer 2 variables entre elles, à partir de leurs différentes valeurs. On peut également l'utiliser pour comparer 2 vecteurs entre eux. Les différentes composantes de ces vecteurs sont alors considérées comme différentes valeurs d'une même variable.

La corrélation ne donne une mesure de similarité que si les vecteurs sont centrés et normés (longueur=1). Quand c'est le cas, elle est alors maximale pour une ressemblance maximale.

Lien avec la distance euclidienne

Pour éviter de normaliser et centrer les vecteurs (d'entrée et de poids), la mesure de distance est le plus souvent utilisée. Les algorithmes ne sont alors plus très "neuronaux".

Les principales mesures de similarité permettant de comparer 2 vecteurs entre eux sont la distance et le produit de corrélation. Ils ne peuvent être calculés que si les vecteurs ont la même dimension.

La distance euclidienne entre 2 vecteurs x_1 et x_2 est définie par la racine carrée de la somme des différences entre les composantes des vecteurs deux à deux, au carré. Elle est définie par :

$$d_E(v_1, v_2) = \sqrt{(v_2 - v_1)^t.(v_2 - v_1)}$$

que l'on peut détailler de la manière suivante :

$$d_E(v_1, v_2) = \sqrt{(x_{21} - x_{11})^2 + (x_{22} - x_{12})^2 + ... + (x_{2n} - x_{1n})^2}$$

$$= \sqrt{\sum_{i=1}^{n} (x_{2i} - x_{1i})^2}$$

Parfois on utilise simplement la somme des différences au carré comme mesure de distance.

Dans le cas de vecteurs binaires, on utilise souvent la distance de Hamming, définie par :

$$d_H(v_1, v_2) = |x_{21} - x_{11}| + |x_{22} - x_{12}| + ... + |x_{2n} - x_{1n}|$$
$$= \sum_{i=1}^{n} |x_{2i} - x_{1i}|$$

Cette distance représente le nombre de bits différents entre les composantes des 2 vecteurs prises 2 à 2.

Si les 2 vecteurs sont normés (=leur norme est égale à 1) et centrés (à moyenne nulle) ; la distance euclidienne est alors minimale lorsque le produit de corrélation est maximal.

En pratique, par souci de simplification, c'est souvent la distance qui est utilisée à la place du produit de corrélation. Cela évite en effet de normaliser et centrer les vecteurs.

Classification

La sortie d'un neurone à deux entrées *x1* et *x2* est définie par

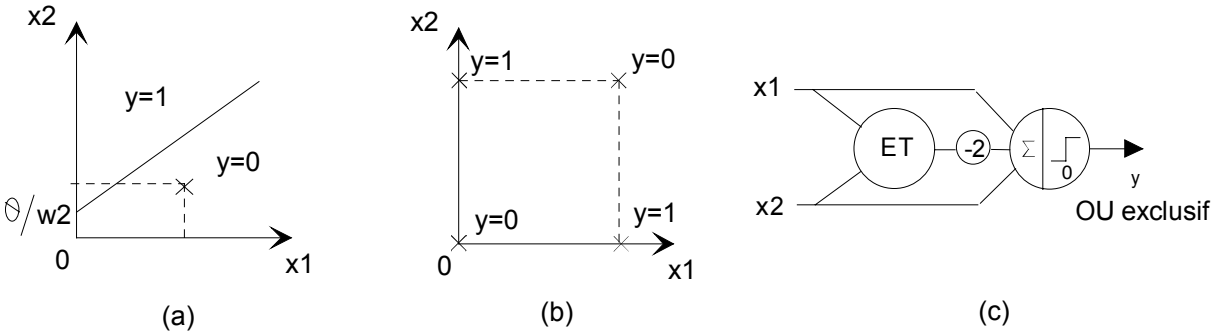
$$y=f(x1.w1+x2.w2-\theta)$$

où *f()* est un seuil logique dans le cas du neurone de MacCulloch et Pitts. Représentée dans un repère à deux dimensions (*x1*, *x2*), cette opération correspond à une séparation du plan en deux régions dont la frontière est une droite d'équation

$$x2=-w1.x1/w2+\theta/w2$$

puisque si $x1.w1+x2.w2 \geq \theta$, y=1, et si $x1.w1+x2.w2 < \theta$, y=0. Les deux variables d'entrée *x1* et *x2* correspondent aux coordonnées des points dans ce plan. Toute entrée située sous la droite provoquera une sortie égale à 0, toute entrée située au dessus une sortie égale à 1. Dans le cas de 3 entrées, la frontière correspond à un plan dans un repère à 3 dimensions ; pour un nombre supérieur d'entrées *n*, il s'agit alors d'un hyperplan dans un espace de dimension *n*.

L'opération effectuée par un tel neurone correspond à une *prise de décision*, ou encore une *classification*. Un neurone à seuil est donc capable de distinguer des formes linéairement séparables. Or, comme toutes les formes ne sont pas linéairement séparables, un neurone seul ne peut pas résoudre tous les problèmes possibles. Par exemple, la fonction OU exclusif n'est pas linéairement séparable. Deux neurones situés sur deux étages sont nécessaires pour réaliser cette fonction.



Classification par un neurone à seuil logique possédant deux entrées *x1* et *x2*. **(a)** Séparation du plan (*x1*, *x2*) par une droite. L'ordonnée à l'origine de cette droite est $\theta/w2$, sa pente $-w1/w2$. **(b)** Représentation de la fonction OU exclusif dans le même plan. Les deux classes (y=0 et y=1) ne peuvent pas être séparées par une droite. **(c)** Groupe de deux unités pour le calcul de la fonction OU exclusif.

La pente de la droite séparatrice des deux classes peut être ajustée par apprentissage, plus précisément en utilisant la règle d'apprentissage supervisée étudiée précédemment.

La détermination du paramètre θ également par apprentissage présente l'avantage de pouvoir obtenir n'importe quelle droite.

I.4.2) Connexion entre neurones

Plusieurs neurones peuvent être reliés entre eux selon divers schémas de connexion, pour définir des réseaux.

Le schéma de connexion peut définir des couches, et dans ce cas la connexion d'une couche à l'autre peut être totale (les neurones d'une couche sont connectés à tous les neurones de la couche en amont) ou locale (les neurones d'une couche sont connectés à une partie des neurones de la couche en amont).

Si tous les neurones sont connectés entre eux, on parle de réseaux récurrents car les sorties d'un neurone sont reliées à ses entrées, de manière directe ou indirecte.

I.4.3) Apprentissage

a) 2 grands types d'apprentissage

Il existe deux grands types d'apprentissage utilisés avec les Réseaux de Neuroones : supervisé (ou "avec professeur") et non-supervisé (ou "sans professeur").

L'apprentissage supervisé est utilisé pour la classification ou la prédiction : l'information d'appartenance à une classe est utilisée pendant l'apprentissage, pour générer une information d'erreur exploitée dans la modification des poids.

Dans l'apprentissage non-supervisé, cette information de classe d'appartenance n'est pas utilisée. Ce type d'apprentissage est donc plutôt utilisé pour l'analyse de données, l'optimisation, etc.

b) Règles

Les règles d'apprentissage indiquent la quantité de modification à appliquer à chaque poids, en fonction des exemples d'entrée (et des sorties désirées associées, dans le cas de l'apprentissage supervisé) :

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

où t désigne un instant avant la modification des poids, et t+1 un instant après.

Il existe un grand nombre de règles d'apprentissage différentes. Les plus répandues sont les suivantes :

- $\Delta w_{ij} = \alpha . x_i . y_j$ règle de Hebb
- $\Delta w_{ij} = \alpha . (x_i - w_{ij}) . y_j$ règle d'apprentissage compétitif
- $\Delta w_{ij} = \alpha . x_i . (y_j^d - y_j)$ règle d'apprentissage supervisé
- $\Delta w_{ij} = \alpha . \delta_j . y_i$ règle de la rétro-propagation du gradient

La 1ère règle est une règle d'apprentissage *non-supervisé* (ou *sans professeur*), c'est à dire dans laquelle n'apparaît pas de terme de sortie désirée des neurones. Ceci est équivalent aux

méthodes de classification sans professeur, dans lesquelles les classes correctes ne sont pas connues. Hebb était un neurobiologiste qui a cherché à modéliser les phénomènes d'apprentissage dans les neurones naturels.
où :

- w_{ij} est le poids de la connexion entre les neurones i et j ; cette connexion est un lien d'entrée du neurone j ;
- x_i et y_j sont respectivement les sorties des neurones source i et cible j ;
- α est un coefficient d'apprentissage, en général inférieur à 1 : il règle la quantité d'adaptation des poids à chaque présentation d'un exemple ;

La 2^e peut être utilisée pour un apprentissage supervisé ou non-supervisé (par exemple VQ ou LVQ).

- y_j^d est la sortie désirée du neurone j.

La 3^e est une règle d'apprentissage *supervisé* (ou *avec professeur*). La classe correcte est connue au moins pour une partie des exemples disponibles. Dans le cadre des réseaux de neurones ces informations constituent des sorties désirées des neurones. Les 2 termes de la règle correspondent à :

$$\Delta w_{ij} = \text{taux d'apprentissage} \times \text{entrée} \times \text{erreur de sortie}$$

Parfois la combinaison entre le vecteur d'entrée et le vecteur de poids d'un neurone n'est pas constitué d'un produit de corrélation mais d'une distance (en général euclidienne) entre ces deux vecteurs (voir plus loin : réseaux VQ et LVQ, réseau RCE). En fait, quand le produit de corrélation entre deux vecteurs est maximal, la distance entre eux est minimale dans le cas où ces vecteurs sont normalisés (c'est à dire que leur norme est égale à 1). Donc dans certaines conditions, les méthodes qui utilisent les produits de corrélation et celles qui utilisent les distances sont équivalentes.

La 4^e règle est celle de l'algorithme de rétro-propagation du gradient. $\delta_j^{(c)}$ est un paramètre qui dépend de la dérivée de fonction d'activation (d'où la nécessité qu'elle soit dérivable) et des poids des couches en aval. Cet algorithme sera abordé en détail par la suite.

c) Test et évaluation des algorithmes de classification

Pour pouvoir quantifier les performances d'un algorithme de classification, il faut disposer d'un ensemble de vecteurs d'exemples dont il faut connaître la classe correcte.

Taux de reconnaissance/erreur

On peut calculer un taux de reconnaissance, ou taux de réussite, indépendamment des classes : t_r. Ce taux est défini comme étant le rapport du nombre de classifications correctes, sur le nombre d'exemples utilisés dans la phase de reconnaissance :

$$t_r = \frac{\text{nombre d'exemples correctement classés}}{\text{nombre total d'exemples de test}}$$

Il est compris entre 0 et 1.

Le **taux d'erreur** est égal à $1-t_r$.

Les pourcentages sont obtenus par multiplication de ces taux par 100.

On peut définir ces mêmes taux pour toutes les classes séparément. Pour chaque classe, on calcule le nombre d'exemples de cette classe correctement classés sur le nombre d'exemples total de cette classe. Pour la classe i :

$$(t_r)_i = \frac{\text{nombre d'exemples de la classe i correctement classés}}{\text{nombre total d'exemples de test de la classe i}}$$

Validation croisée

Lorsque les performances d'un algorithme de RdF sont évaluées avec les mêmes données que celles utilisées pour l'apprentissage, cela ne renseigne en rien sur la capacité de **généralisation** du classifieur. Il se peut que les données aient été **apprises par coeur**, c'est à dire que la généralisation soit mauvaise.

Il est donc plus judicieux de diviser l'ensemble des données en deux parties distinctes : une partie pour l'apprentissage et l'autre pour le test. On parle alors de **validation croisée**.

Certaines méthodes peuvent comporter plusieurs cycles d'apprentissage (1 cycle=présentation de toute la base d'apprentissage), et parfois il arrive que le taux de reconnaissance sur la base de test augmente d'abord puis diminue ensuite. Dans ce cas il faut arrêter l'apprentissage quand ce taux est maximal.

La séparation de l'ensemble de données en un sous-ensemble d'apprentissage et un sous-ensemble de test peut s'effectuer de différentes manières :

Matrice de confusion

La matrice de confusion permet d'analyser les erreurs de classification. Elle permet de voir à quelles autres classes ont été affectées les entrées incorrectement classées.

Pour n classes, elle est de taille n^2.

Par exemple, dans le cas de 3 classes : chien, chat et girafe, une ligne de la matrice de confusion suivante indique les classes ont été affectées les formes à classer. La somme des nombres sur une ligne est donc égale à 100 :

	chien	chat	girafe
chien	75	20	5
chat	16	76	8
girafe	7	3	90

I.5) Gestion des données

I.5.1) Affichage de Sammon

Intérêt

Il n'est pas possible de visualiser mentalement, ni même de représenter graphiquement, des données de dimension supérieure à 3. Dans certains cas il est pourtant nécessaire de les visualiser, par exemple pour repérer des regroupements et initialiser certains algorithmes.

Il est possible de réaliser des projections en 2 ou 3 dimensions qui conservent les relations existant entre les points dans l'espace multidimensionnel dans lequel ils sont définis. Un algorithme bien connu est celui de Sammon.

Principe

Le principe de cette méthode est de définir un critère d'erreur qui mesure la somme des différences au carré entre les distances entre tous les points considérés deux à deux dans l'espace de départ, et les distances entre les mêmes points dans l'espace d'arrivée, et de minimiser cette erreur.

Soit n le nombre de points, D la dimension de l'espace de départ, d la dimension de l'espace d'arrivée (en général, 2), $D_{i,j}\ \forall$ i, j=1,....,n, i≠j, la distance (en général, euclidienne) entre 2 points dans l'espace de départ, $d_{i,j}\ \forall$ i, j=1,....,n, i≠j la distance entre 2 points dans l'espace d'arrivée. Le critère à minimiser est :

$$E = \frac{1}{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} D_{ij}} \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{(D_{ij} - d_{ij})^2}{D_{ij}}$$

Ce critère varie entre 0 et 1.

$$c = \frac{1}{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} D_{ij}} \text{ est une constante}$$

Les coordonnées des points dans l'espace d'arrivée constituent les variables à déterminer par l'algorithme. N'étant initialement pas connues, elles sont initialisées à des valeurs aléatoires. Les points sont déplacés à chaque itération de l'algorithme par petites quantités, jusqu'à ce que le critère E atteigne un minimum. Ce minimum est local, c'est à dire que l'algorithme peut converger vers cette valeur alors qu'il existe un ou plusieurs minima plus petit(s). La modification s'effectue selon la descente la plus pentue (steepest descent) du gradient du critère *E* (c'est également la méthode qui est à la base de l'apprentissage par rétropropagation du gradient).

A chaque itération de l'algorithme, chaque coordonnée dans l'espace d'arrivée est modifiée d'une quantité proportionnelle au négatif du gradient du critère :

$$x_{pq}(t+1) = x_{pq}(t) - \alpha . \frac{\partial E(t)}{\partial x_{pq}(t)}$$

où x_{pq} est la q^e coordonnée du p^e point dans l'espace d'arrivée. Une valeur usuelle du taux de modification α est 0,3 à 0,4.

Quand la visualisation des données est faite en dimension 2, x et $\frac{\partial E}{\partial x}$ sont des vecteurs de dimension 2. Pour le p^e point :

$$x_p = \left(x_{p1}, x_{p2}\right)^t$$

et

$$\frac{\partial E(t)}{\partial x_p} = \left(\frac{\partial E(t)}{\partial x_{p1}}, \frac{\partial E(t)}{\partial x_{p2}} \right)^t$$

On peut démontrer que les dérivées par rapport à chacune des variables sont définies par :

$$\frac{\partial E}{\partial x_{pq}} = -2c.\sum_{\substack{j=1 \\ j \neq p}}^{n} \frac{D_{pj} - d_{pj}}{D_{pj}.d_{pj}} \left(x_{pq} - x_{jq}\right)$$

C'est la relation que l'on peut alors programmer.

I.5.2) Répartition en base d'apprentissage/base de test

Il y a plusieurs manières de subdiviser la base de vecteurs d'exemples disponibles pour l'apprentissage et le test des algorithmes.

a) Holdout

L'ensemble des exemples est divisé en deux parties : une partie pour la construction du classifieur et l'autre pour la reconnaissance. Les deux parties ne sont pas forcément égales en taille. La division, la construction du classifieur et son test peuvent être répétés plusieurs fois, et la moyenne des taux de reconnaissance calculée. La division et le choix des exemples sont aléatoires.

b) Leave-K-out (rotation)

L'ensemble des exemples est divisé en un nombre N de sous-ensembles comportant tous le même nombre d'exemples. Un classifieur est construit en laissant de côté un des groupes, et le test a lieu avec ce dernier. La même séquence construction-test est répétée pour les N sous-ensembles, et la moyenne des taux de reconnaissance est calculée.

c) Leave-one-out

Il s'agit d'un cas particulier de la méthode Leave-K-out avec K=1.

Le classifieur est construit à partir de tous les exemples sauf 1. Le test de reconnaissance est réalisée avec cet exemple initialement laissé de côté. Cette opération est réalisée autant de fois qu'il y a d'exemples. A chaque fois un échantillon est laissé de côté. Le nombre de mauvaises classifications est compté, puis divisé par le nombre d'échantillons. Ce résultat constitue le pourcentage de mauvaises classifications. Le résultat obtenu est **statistiquement représentatif**, mais au prix d'un **coût de calcul important**.

I.5.3) Normalisation

Dans certaines bases de données caractéristiques, en général chacune des caractéristiques possède une plage de variation différente. Si une caractéristique possède une grande plage de variation et une autre une petite, la première aura tendance à avoir plus d'influence sur les résultats de la classification que la seconde (pour les méthodes basées sur l'utilisation d'une distance). A la limite tout se passerait comme si cette deuxième caractéristique n'était pas utilisée.

Par exemple, dans les données Wine classées par les une méthode simple comme les KPPV, le résultat sans normalisation est de l'ordre de 70%, et avec normalisation de l'ordre de 90 à 95%.

Pour éviter cet inconvénient, il est nécessaire de normaliser les données, en divisant par exemple chacune des composantes des vecteurs de données par la valeur maximale de cette composante sur tous les exemples d'apprentissage.

Voir l'annexe pour la définition de la normalisation d'un vecteur ou d'un ensemble de vecteurs.

II) Apprentissage compétitif

L'apprentissage compétitif désigne la façon dont l'apprentissage est appliqué aux Réseaux de Neurones. Le neurone dont le vecteur poids est le plus proche (au terme d'une mesure de similarité comme la distance euclidienne ou le produit scalaire) d'un vecteur d'entrée est désigné comme vainqueur d'une compétition.

L'apprentissage est alors appliqué uniquement à ce neurone, comme dans les méthodes VQ et LVQ, ou à ce neurone et ses voisins, comme dans les cartes auto-organisatrices de Kohonen.

II.1) Vector Quantization (VQ)

II.1.1) Description de la méthode

Cette méthode est également appelée quantification vectorielle. Elle désigne le fait de remplacer un vecteur dont les composantes peuvent être quelconques, par un vecteur d'un ensemble discret. Cette opération est analogue à la quantification d'un signal échantillonné : la valeur des échantillons constitue alors une variable mono-dimensionnelle, alors que dans la RdF on raisonne sur des vecteurs multi-dimensionnels.

a) Principe

La méthode de quantification vectorielle est très proche des nuées dynamiques. Mais plutôt que d'utiliser la moyenne des vecteurs affectés à une classe pour représenter celle-ci, les vecteurs représentants sont adaptés par une règle d'apprentissage, à chaque classification d'un nouveau vecteur.

De plus, cette méthode est souvent formulée dans le cadre des Réseaux de Neurones, qui fait que les termes employés sont différents de ceux de la méthode des nuées dynamiques, tout en désignant des choses très proches. (voir l'introduction aux Réseaux de Neurones en

annexe). Les prototypes sont alors codés dans les vecteurs poids des neurones. Dans la version "classique", on recherche par exemple une distance minimale comme critère de similarité maximale, mais dans la version neuronale on recherche un produit scalaire maximal, entre un vecteur à classer et les vecteurs poids des neurones. On peut montrer que ces deux critères sont équivalents, dans certaines conditions (voir annexe sur calcul matriciel).

L'apprentissage est de type non-supervisé, c'est à dire que l'information d'appartenance des vecteurs d'exemple à une classe n'est par utilisée pendant l'apprentissage, sauf pour ceux qui sont utilisés pour initialiser l'algorithme (comme dans le cas des nuées dynamiques).

b) Paramètres

Une fois choisis les représentants initiaux pour chaque classe et la mesure de similarité (ex. : distance euclidienne), le seul paramètre de cet algorithme est le coefficient d'apprentissage (α).

c) Algorithme

L'algorithme de cette méthode est le suivant :

- Choisir les représentants initiaux pour chaque classe
- Initialiser les poids de chaque neurone à un exemple représentatif de chaque classe (1)
- Répéter un certain nombre de fois (2) :
  - Appliquer un vecteur en entrée du réseau
  - Sélectionner le neurone dont le vecteur poids est le plus proche du vecteur d'entrée (neurone "vainqueur") (3)
  - Modifier les poids de ce neurone par la règle d'apprentissage (4) :

$$\Delta w_{ij} = \alpha.(x_i - w_{ij})y_j$$

(1) ou à des valeurs aléatoires, mais cela peut poser le problème des neurones "morts".
(2) En général, on présente une fois tous les vecteurs de la base d'apprentissage, puis on recommence. On peut fixer un nombre de présentations de manière arbitraire et empirique, ou définir un critère de convergence. Par exemple, utiliser la base de test après chaque présentation de la base d'apprentissage ; le critère d'arrêt est alors atteint quand le taux de reconnaissance sur la base de test passe au dessus d'un seuil.
(3) La mesure de similarité utilisée peut être basée sur le calcul d'une distance euclidienne, de Hamming, ou d'un produit de corrélation.
(4) La sortie du neurone vainqueur est fixée à 1 et celle des autres à 0 (c'est le principe de l'apprentissage compétitif). Par application de la règle d'apprentissage, seuls les poids du neurone vainqueur sont modifiés. C'est équivalent à appliquer la règle

$$\Delta w_{ij} = \alpha.(x_i - w_{ij})$$

uniquement au neurone ayant répondu le plus fortement.

L'effet de cette règle d'apprentissage est de rapprocher le vecteur poids du neurone vainqueur du vecteur à classer.

d) Implémentation en langage C

Il y a plusieurs manières d'implémenter cet algorithme en langage C.

Dans l'implémentation qui suit, les vecteurs sont d'abord chargés dans une liste. Puis ces vecteurs sont retirés un à un de cette liste : les premiers pour constituer les vecteurs représentatifs, les suivants pour l'apprentissage puis pour le test de classification.

De plus, on utilise la moitié des vecteurs pour l'apprentissage et l'autre moitié pour le test de classification.

Algorithme détaillé

```
Initialisations générales
Charger les vecteurs du fichier de données dans une liste
Initialisations spécifiques aux nuées dynamiques, VQ et LVQ
Pour chaque classe
    Répéter un nombre de fois égal au nombre de vecteurs choisi pour initialiser les représentants
        Choisir aléatoirement un vecteur de la classe en cours, dans la liste
        Calculer la moyenne entre ce vecteur et les vecteurs déjà tirés pour cette classe
        Retirer ce vecteur de la liste
Adaptation des vecteurs représentants
Pour chaque épisode d'apprentissage
    Pour chaque vecteur de la liste des vecteurs restant
        Tirage aléatoire d'un vecteur
        Recherche du représentant la plus proche de ce vecteur
        Si le nombre de vecteurs tirés est inférieur à la moitié du nombre total de vecteurs de la liste
            Adaptation du vecteur prototype : rapprochement du représentant sélectionné du vecteur à classer
    Sinon
        Incrémenter le compteur des vecteurs correctement classés,
        ou celui des vecteurs incorrectement classés, selon le cas
```

Calcul des taux de reconnaissance

A chaque classification d'un nouveau vecteur, on regarde s'il a été correctement classé ou non (en comparant sa classe réelle avec la classe décidée par l'algorithme).

Pour chacune des classes, on utilise un compteur de classifications correctes $compt\_id_i$ et un compteur de classifications incorrectes $compt\_diff_i$.

Pour chaque vecteur à classer, on incrémente le 1er ou le 2e selon le cas.

Pour la classe i, le taux de reconnaissance et le taux d'erreurs sont donc respectivement égaux à :

$$(t_r)_i = \frac{compt\_id_i}{compt\_id_i + compt\_diff_i} \quad \text{et} \quad (t_e)_i = \frac{compt\_diff_i}{compt\_id_i + compt\_diff_i}$$

e) Propriétés

- Méthode peu coûteuse en calculs car chaque vecteur à classer n'est comparé qu'à un seul autre vecteur pour chaque classe : le noyau.
- Méthode ayant la propriété de minimiser un critère global : la somme des distances des points au prototype le plus proche d'eux.
- D'un point de vue géométrique, cette propriété correspond à un découpage de l'espace des entrées appelé *pavage* (ou *tessellation*) *de Voronoï*, dans lequel chaque pavé est

associé à un vecteur représentant : il est constitué par l'ensemble des points les plus proches de lui que de tous les autres représentants.

- Méthode sensible à l'ordre de présentation des données à classer : si cet ordre est aléatoire, le résultat de la classification sera différent à chaque exécution de l'algorithme.
- Avec la distance euclidienne, méthode ayant tendance à créer des classes hyper-sphériques (des cercles en dimension 2). Donc fonctionne bien avec des classes hypersphériques mais moins avec les formes plus complexes. Pour pallier à cet inconvénient, on peut utiliser plusieurs représentants par classe (voir variante ci-dessous).

f) Variante

Une variante consiste à utiliser plusieurs vecteurs représentants par classe. Elle permet de traiter le cas des classes dont la forme est plus complexe que simplement hypersphérique.

Comme dans les nuées dynamiques, la structure générale de l'algorithme reste la même.

D'un point de vue neuronal, dans le cas où il y a plusieurs représentants par classe, cela correspond à avoir 2 couches de neurones, dont la 2^e (neurones de codage des classes) implémentent des fonctions OU.

II.1.2) Lien avec l'algorithme des nuées dynamiques

L'algorithme des nuées dynamiques est un algorithme de classification de données bien connu. Il ne s'agit pas d'une méthode appartenant à la catégorie des Réseaux de Neurones, mais on va voir qu'elle est très proche de l'algorithme VQ. Il est intéressant de l'aborder rapidement, en raisonnant par analogie avec ce dernier.

La seule différence avec VQ est que le noyau est constitué par la moyenne des vecteurs qui ont été à la classe qu'il représente par la classification. La moyenne pouvant s'exprimer de manière récursive, il n'est pas nécessaire de re-calculer la moyenne de tous les vecteurs. La moyenne de n+1 vecteurs peut s'exprimer de la façon suivante :

$$m_{n+1} = \frac{n.m_n + x_{n+1}}{n+1}$$

où m_n est la moyenne de n vecteurs, x_{n+1} est le $(n+1)^e$ vecteur et m_{n+1} la moyenne de n+1 vecteurs. Cette relation reste valable lorsque x et m sont des vecteurs de dimension quelconque.

II.2) Learning vector quantization (LVQ)

LVQ (pour Learning Vector Quantization) est une version supervisée de l'algorithme de quantification vectorielle. Comme dans toutes les méthodes supervisées, pendant l'apprentissage, on utilise l'information d'appartenance des exemples d'apprentissage à une classe.

Il existe plusieurs versions de cet algorithme, appelées : LVQ1, LVQ2, LVQ2.1, LVQ3, OLVQ1. Le package LVQ_PAK (disponible sur Internet) regroupe les sources de ces programmes pour toutes les versions, sauf LVQ2.

Principe

Pour un vecteur d'entrée à classer, si le vecteur poids du neurone vainqueur est un prototype de la classe correcte, il est modifié de la même façon qu'avec la méthode VQ (version non-supervisée de l'algorithme) ; il est donc rapproché du vecteur d'entrée. A la différence de VQ, si le vecteur poids du neurone vainqueur n'est pas un prototype de la classe correcte, il est éloigné du vecteur d'entrée.

On peut avoir plusieurs représentants par classe. En effet, comme dans le cas des Nuées Dynamiques, avec un seul représentant par classe la méthode de bons résultats pour les classes hypersphériques et moins bons pour les autres formes de classes. Le fait de prendre plusieurs représentants par classe permet de mieux traiter le cas de classes de formes complexes.

Avec plusieurs représentants par classe, le réseau se compose de 2 couches : la première est constituée de neurones codant chacun un représentant d'une classe, la deuxième de cellules réalisant des fonctions OU sur les sorties de neurones de la première. Les neurones de la dernière couche codent chacun une classe.

LVQ1

La modification des poids du neurone vainqueur est définie par :

$$w_k(t+1) = w_k(t) + s(t).\alpha\big(x(t) - w_k(t)\big)$$

où *c* est l'indice de classe, α un coefficient d'apprentissage (fixe), et

s(t)= +1 si la classification est correcte (le neurone vainqueur représente la bonne classe)

-1 si la classification est incorrecte (le neurone vainqueur ne représente pas la bonne classe)

LVQ2

Dans cette version on ne modifie pas uniquement le prototype le plus proche du vecteur à classer mais les deux prototypes les plus proches, quand les conditions suivantes sont réunies :

le vecteur à classer est proche de la frontière (hyperplan) qui sépare les 2 prototypes, c'est à dire qu'il est à peu près à égale distance du prototype correct et du prototype incorrect ;

le prototype le plus proche correspond à une classe incorrecte et le second prototype le plus proche à la classe correcte.

La première condition peut être implémentée de la manière suivante : soient d_i et d_j les distances euclidienne entre une entrée *x* et respectivement w_i, le représentant de la classe incorrecte, et w_j, celui de la classe correcte. *x* est défini comme tombant dans une fenêtre de largeur L si

$$\min\left(\frac{d_i}{d_j}, \frac{d_j}{d_i}\right) > s \text{ avec } s = \frac{1-L}{1+L}$$

L'algorithme éloigne alors w_i (le prototype incorrect) du vecteur à classer x et en rapproche w_j (le prototype correct), comme dans la version de base.

Les valeurs recommandées pour L sont 0,2 à 0,3.

L'intérêt de cette variante est de déplacer les vecteurs prototypes uniquement dans les cas où il y a mauvaise classification, et où le vecteur à classer se trouve proche de la frontière entre 2 classes, zone d'ambiguïté.

LVQ2.1

Dans cette version on modifie les deux prototypes les plus proches comme dans LVQ2, mais l'un des deux peut indifféremment représenter la classe correcte et l'autre une mauvaise classe. La modification n'est appliquée que si le vecteur à classer est proche de la frontière qui sépare les deux prototypes.

LVQ3

Dans le cas où les 2 prototypes les plus proches du vecteur à classer représentent une classe différente, ils sont modifiés de la même manière que dans LVQ2.1.

Mais dans le cas où ces 2 prototypes appartiennent à la même classe que le vecteur à classer, on ajoute un facteur constant dans la règle d'apprentissage :

$$w_k(t+1) = w_k(t) + \varepsilon.\alpha\big(x(t) - w_k(t)\big),\ k \in \{i,j\}$$

où i et j sont les indices de ces 2 prototypes.

L'effet de cette règle supplémentaire est d'éviter les dérives des prototypes à long terme (c'est à dire dans le cas où les mêmes entrées sont présentées indéfiniment, cycliquement), possibles avec LVQ2.1.

Des valeurs empiriques pour ε vont de à 0,1 à 0,5.

OLVQ1

Il s'agit d'une variante de l'algorithme de base LVQ1, dans laquelle les valeurs optimales des taux d'apprentissage sont déterminées. Chaque classe possède son propre taux d'apprentissage.

Le taux d'apprentissage est défini par :

$$\alpha_c(t) = \frac{\alpha_c(t-1)}{1 + s(t).\alpha_c(t-1)}$$

où *c* est l'indice de classe,

et s(t)=+1 si la classification est correcte ;

-1 si la classification est incorrecte.

Il diminue quand le nombre de bonnes classifications augmente (s(t)=1), et augmente quand le nombre de mauvaises classification augmente (s(t)=-1). Ainsi, quand la classification s'améliore, les prototypes ont tendance à se stabiliser.

Empiriquement, on observe que le taux de reconnaissance maximal est atteint au bout d'un nombre de présentations de toute la base d'apprentissage égal à 30 à 50 fois le nombre de vecteurs prototypes.

On ajoute simplement l'indice de la classe dans la règle d'apprentissage :

$$w_k(t+1) = w_k(t) + s(t).\alpha_c(t)\big(x(t) - w_k(t)\big)$$

II.3) Cartes auto-organisatrices de Kohonen

Les cartes auto-organisatrices de Kohonen (ou SOM pour Self-Organizing Map) sont principalement appliquées à l'**analyse de données**, mais elles peuvent également donner de très bons résultats en **classification**.

Dans le 1er cas, l'apprentissage est non-supervisé ; dans le 2e cas, d'apprentissage non-supervisé.

Il s'agit d'un algorithme classé dans la catégorie des réseaux de neurones même si, tout comme les méthodes LVQ, il peut être décrit complètement sans faire référence à ce domaine.

Cet algorithme est très répandu et a donné lieu à de très nombreuses applications pratiques (pour s'en convaincre, il suffit d'effectuer une recherche sur Internet avec les mots-clefs "Kohonen", "SOM", "carte auto-organisatrice", etc).

II.3.1) Principe et propriétés

Principe

L'algorithme SOM est proche de la méthode non-supervisée VQ : chaque classe est représentée par un ensemble de vecteurs prototypes (=les vecteurs-poids des neurones dans la version neuronale), et ces derniers se rapprochent des vecteurs à classer lorsqu'ils sont sélectionnés. Mais en plus de cela, dans les SOM les prototypes ne sont pas indépendants : il existe des relations de voisinage entre eux, et quand un prototype est déplacé, ses voisins le sont également (dans une moindre mesure).

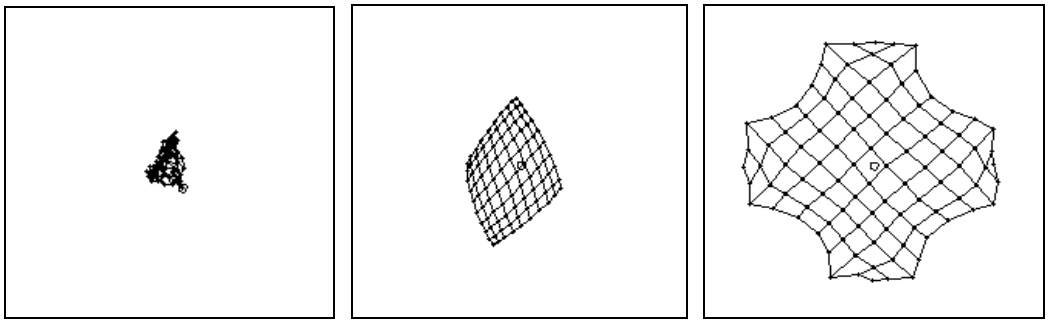
Codage topologique

Le résultat de ce processus est que, après apprentissage, ces prototypes sont répartis "équitablement" pour couvrir tout l'espace d'entrée en reflétant également la densité des entrées (il y aura plus de neurones pour coder les régions de l'espace où les vecteurs de donnée sont plus nombreux). Deux neurones voisins répondent de manière maximale pour des vecteurs d'entrée proches. Il en résulte un **codage topologique,** une **cartographie** (d'où l'expression "carte") de l'espace d'entrée. Pour parvenir à cette propriété, il faut que la taille du voisinage et le taux d'apprentissage diminuent progressivement au fil de l'apprentissage.

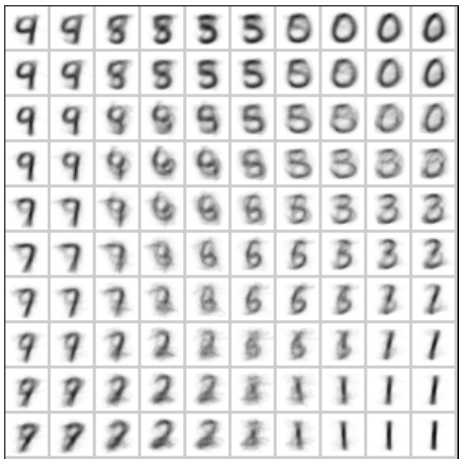
Cette propriété est similaire à celle de l'algorithme de visualisation de Sammon, qui réalise une projection d'une dimension quelconque vers une dimension réduite 2 ou 3, conservant la topologie de l'espace d'entrée. La différence ici est que en plus de cette projection, il y a une quantification d'un grand nombre de vecteurs (les vecteurs d'entrée) vers un nombre plus petit de vecteurs représentatifs (=les vecteurs poids des neurones), dont le nombre est choisi par l'utilisateur.

La dimension de ces cartes peut être quelconque, mais en général on utilise des réseaux à 2 ou 1 dimensions. A 1 dimension, les neurones définissent une "ficelle", à deux dimensions une grille.

L'exemple de la figure ci-dessus illustre cette propriété : l'espace d'entrée est à 2 dimensions et possède une forme de croix (c'est à dire que tous les vecteurs d'entrée sont pris dans cette croix) ; l'espace de sortie est également à 2 dimensions puisqu'il s'agit d'une grille (de neurones). Chaque neurone de la grille est représenté à l'emplacement de son vecteur-poids. Les vecteurs-poids sont initialement aléatoires, et petit à petit ils couvrent tout l'espace d'entrée (il s'agit de la représentation des vecteurs poids en début, en cours et en fin d'algorithme).



Cet autre exemple ci-dessous illustre bien la propriété de codage topologique : il s'agit cette fois de vecteurs de donnée correspondant à des petites images de chiffres manuscrits, de taille 28x28 pixels. La dimension de l'espace d'entrée est donc ici de 28x28=784, celle de l'espace de sortie est prise égale à 2. La dimension de l'espace d'entrée étant supérieure à 2, on ne peux pas le représenter comme dans l'exemple précédent. Par contre, il est sensé de représenter les vecteurs-poids sous forme d'images. On voit que l'on retrouve les 10 chiffres possibles (de 0 à 9) dans les poids des neurones, et que des neurones voisins codent des formes similaires.



Il est intéressant de constater que la motivation initiale des recherches ayant mené aux cartes auto-organisatrices était la modélisation de certaines propriétés du cerveau. Or il s'est avéré par la suite que ce modèle possédait des applications pratiques très intéressantes.

Minimisation d'un critère global

Ce critère est la somme des distances entre les vecteurs d'entrée et le vecteur poids le plus proche, comme LVQ, mais également (pas un minimum global mais une "bonne" solution).

On verra que cette propriété peut être exploitée dans les applications d'optimisation.

Le résultat de l'apprentissage était un étalement des vecteurs poids des neurones dans cet espace pour en réaliser un pavage de Voronoï (ce qui était déjà une propriété de l'algorithme VQ). Mais en plus du pavage de Voronoï, la somme des distances entre les vecteurs poids des neurones est minimisée.

II.3.2) Algorithme d'apprentissage

a) Algorithme

Initialisation des paramètres :
- poids d'entrée (petites valeurs aléatoires)
- taux d'apprentissage et règle de sa décroissance
- forme et taille du voisinage et règle de décroissance

Répéter jusqu'à stabilisation du réseau (*)
- Présenter au réseau un vecteur x de l'ensemble d'apprentissage
- Déterminer le neurone dont le vecteur-poids est le plus proche du vecteur d'entrée x :

  le neurone d'indice *j* tel que $\|w_j - x\| = \min_n \|w_n - x\|$ (**), où *n* est le nombre de neurones.
- Mise à jour des poids du neurone *j* et de ses voisins : le vecteur poids du neurone *j'* est modifié par la règle :

$$w_{j'}(t+1) = w_{j'}(t) + \alpha(t) \times h_{jj'}(t) \times (x - w_{j'}(t))$$

  où $h_{jj'}$ est une fonction de voisinage du neurone *j*, centré sur lui, de forme carrée ou gaussienne (***)
- Réduction du voisinage et du taux d'apprentissage

(*) la stabilisation du réseau est obtenue lorsque la variation des poids a atteint une valeur seuil, (par le haut, car elle diminue forcément).
(**) || || désigne une distance (par exemple euclidienne)
(***) dans le cas du neurone vainqueur, $h_{jj}=h_{jj}=1$

b) Réglage des paramètres

Forme du voisinage

La forme la plus simple du voisinage est un voisinage carré. Dans ce cas, tous les neurones du voisinage sont modifiés par la même quantité à chaque itération.

Une autre forme possible est la forme gaussienne, dans laquelle l'écart-type σ est diminué au cours de l'apprentissage :

$$h_{jj'} = \exp\left(\frac{-d_{jj'}^{\,2}}{2\sigma(n)^2} \right)$$

où j est l'indice du neurone vainqueur, j' l'indice d'un autre neurone, et $d_{jj'}$ la distance entre les coordonnées de ces 2 neurones dans la couche, celles-ci étant leur position dans la grille. n est l'indice des itérations d'apprentissage.

Ces 2 types de voisinages donnent des résultats similaires.

Diminution de la taille du voisinage au cours de l'apprentissage

La taille du voisinage est réduite au cours de l'apprentissage (à la fin il peut ne concerner que le neurone vainqueur de la compétition et dans ce cas on est ramené à l'algorithme de quantification vectorielle VQ). Le voisinage peut être initialement très grand, puis être réduit à zéro en fin d'apprentissage, c'est à dire qu'à la fin un seul neurone voit ses poids modifiés à chaque étape d'apprentissage.

Le grand voisinage (qui peut aller jusqu'à la totalité de la carte) initial permet une structuration globale cohérente de la carte ; le petit voisinage final permet une spécialisation de chaque neurone.

Diminution du taux d'apprentissage

Pour assurer la convergence de l'apprentissage on peut faire varier le taux d'apprentissage en fonction du nombre d'itérations. La valeur initiale peut être grande, par exemple égale à 1.

Si la décroissance du taux d'apprentissage est trop rapide, il peut se produire un phénomène de torsion de la grille.

II.3.3) Exploitation

II.3.3.1) Analyse des données

Comme dans le cas de VQ, l'apprentissage provoque une quantification vectorielle. La position finale des neurones reflète la répartition des classes dans l'espace d'entrée, et leur densité : des régions plus denses (en terme d'exemples) seront codées par un nombre plus important de neurones que les autres.

II.3.3.2) Classification

Il existe de nombreuses tentatives d'application des cartes de Kohonen à la classification. C'est encore un sujet de recherche.

Ce paragraphe présente une méthode simple, donnant néanmoins de bons résultats.

a) Principe

Si l'on force chaque neurone de la carte à ne répondre qu'aux vecteurs de données d'une classe donnée, chaque vecteur-poids d'un neurone va constituer un vecteur représentatif de cette classe, et les différents vecteurs-poids vont s'auto-organiser pour la représenter "au mieux", c'est à dire se positionner en fonction de son étalement et de la densité des données.

On pourrait également utiliser une carte différente pour chaque classe, ce qui reviendrait à peu près au même.

L'algorithme se modifie alors simplement :

- d'une part il faut ajouter une étape d'affectation des neurones aux classes ;
- d'autre part, lors de l'apprentissage, on ne modifie le vecteur poids d'un neurone que si celui-ci est affecté à la classe du vecteur à classer.

b) Algorithme

On reprend l'algorithme de base et on effectue les ajouts nécessaires (en gras ci-dessous) :

Initialisation des paramètres :
- poids d'entrée (petites valeurs aléatoires)
- taux d'apprentissage et règle de sa décroissance
- forme et taille du voisinage et règle de décroissance

Affectation des neurones à l'une des classes

Répéter jusqu'à stabilisation du réseau (*)
- Présenter au réseau un vecteur x de l'ensemble d'apprentissage
- Déterminer le neurone dont le vecteur-poids est le plus proche du vecteur d'entrée x :

  le neurone d'indice *j* tel que $\|w_j - x\| = \min_n \|w_n - x\|$ (**), où *n* est le nombre de neurones.

- **Si ce vecteur appartient à la classe affectée à ce neurone**
- Mise à jour des poids du neurone *j* et de ses voisins : le vecteur poids du neurone *j'* est modifié par la règle :

$$w_{j'}(t+1) = w_{j'}(t) + \alpha(t) \times h_{jj'}(t) \times (x - w_{j'}(t))$$

  où $h_{jj'}$ est une fonction de voisinage du neurone *j*, centré sur lui, de forme carré ou gaussien (***)
- Réduction du voisinage et du taux d'apprentissage

(*) la stabilisation du réseau est obtenue lorsque la variation des poids a atteint une valeur seuil, (par le haut, car elle diminue forcément).
(**) $\|\ \|$ désigne une distance (par exemple euclidienne)
(***) dans le cas du neurone vainqueur, $h_{jj'}=h_{jj}=1$

Remarque : l'apprentissage étant ici supervisé, les données sont réparties en données d'apprentissage et données de test. L'algorithme peut fonctionner par cycles, un cycle étant la présentation de tous les vecteurs de la base d'apprentissage une fois. La boucle se sépare alors en 2 :

- pour chaque cycle
  - pour chaque vecteur de la base d'apprentissage

II.3.3.3) Optimisation

Dans l'application d'analyse de données, la carte a été testée avec des données tirées au hasard dans l'espace d'entrée.

Dans l'application d'optimisation du problème du voyageur de commerce (Traveling Salesman Problem, TSP), les entrées ne sont pas prises au hasard mais sont constituées par les coordonnées des villes par lesquelles le voyageur doit passer. Elles sont présentées plusieurs fois au réseau, jusqu'à convergence de ses poids.

Si le nombre de neurones est égal au nombre de villes, chacun peut se spécialiser sur une ville et son vecteur poids (qui représente les 2 coordonnées dans le plan) va finir par être égal aux coordonnées de la ville.

Il faut cependant ajouter un mécanisme d'affectation unique entre les villes et les neurones, sinon les vecteurs poids de certains d'entre eux peuvent osciller entre 2 villes, et d'autres n'être attirés par aucune.

Au final, si la correspondance est unique, la somme des distances entre les vecteurs poids des neurones représente la somme des distances entre les villes. Or, ce critère est minimisé par l'apprentissage ; la somme des distances parcourues entre les villes est donc minimisée.

II.3.4) Interprétation des résultats

a) Comparaison avec les autres méthodes d'apprentissage compétitif

Comme dans le cas du VQ, l'apprentissage est **non-supervisé**, c'est à dire que l'on laisse le réseau évoluer librement. D'où le terme "auto-organisatrice" pour qualifier la carte. Le neurone ayant gagné la compétition (=le plus proche du vecteur d'entrée présenté au réseau) est déplacé en direction du vecteur d'entrée, comme dans le cas de l'algorithme VQ. Chaque neurone de la carte constitue un vecteur représentant.

Mais contrairement à VQ, il n'est pas déplacé seul : ses voisins sont déplacés de la même manière que lui.

b) Comparaison avec affichage de Sammon

La propriété de réduction de dimension est la même que celle de l'affichage de Sammon. Par contre, dans l'affichage de Sammon on utilise un point par vecteur, alors que dans la SOM un neurone (qui correspond à un point de l'affichage de Sammon) peut coder plusieurs vecteurs : le nombre de neurones est fixé et chacun d'entre eux peut coder toute une partie de l'espace d'entrée, c'est à dire tous les vecteurs plus proches de son vecteur poids que du vecteur poids de tous les autres neurones.

II.3.5) Outils

En pratique, il existe plusieurs outils pour test des cartes auto-organisatrices :

- Matlab : fonction *newsom* (Neural Networks Toolbox)
- Som Toolbox : boîte à outils développée par l'inventeur du SOM et son équipe. Disponible à l'adresse : *http://www.cis.hut.fi/projects/somtoolbox/* ; elle contient des utilitaires comme un module d'affichage de Sammon
- SOM_PAK : version C de la Som Toolbox (avec certaines différences). Disponible à l'adresse *http://www.cis.hut.fi/research/som_lvq_pak.shtml*

III) Réseaux de neurones multicouches avec rétropropagation du gradient

La rétropropagation du gradient est un algorithme d'apprentissage applicable aux réseaux de neurones multicouches non-linéaires. Ce type de réseau est historiquement appelé perceptron.

Cette méthode est basée sur les propriétés de classification des neurones non-linéaires, sans les limitations des réseaux mono-couche.

III.1) Propriétés d'un neurone à seuil

Un neurone à seuil à 2 entrées peut séparer un plan en 2 parties par une droite, et donc réaliser une classification en 2 classes.

En effet, la sortie de ce neurone est définie par :

$$y = s(w_1 x_1 + w_2 x_2)$$

où *s* est la fonction seuil (voir annexe sur les réseaux de neurones).

Les vecteurs d'entrée de ce neurone sont de dimension 2 et donc représentent des points dans un plan.

On peut distinguer 2 cas :

$$w_1x_1+w_2x_2>0 \ (1) \rightarrow y=1$$
$$w_1x_1+w_2x_2<0 \ (2) \rightarrow y=0$$

$$(1) \Leftrightarrow x_2 > -\frac{w_1}{w_2} x_1 \ (2)$$

$x_2 = -\frac{w_1}{w_2} x_1$ est l'équation d'une droite passant par l'origine et de pente $-w_1/w_2$.

(2) $\Leftrightarrow$ le point est au dessus de la droite

En rajoutant un 3e lien d'entrée à ce neurone (entrée x_0 et poids w_0), relié à une entrée constante, on peut avoir une droite quelconque :

$$y = s(w_0 x_0 + w_1 x_1 + w_2 x_2)$$

L'équation de la droite devient alors :

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2} x_0$$

En prenant par exemple $x_0=1$ (choix arbitraire), l'ordonnée à l'origine de la droite est $-w_0/w_2$. La pente reste la même.

Un perceptron mono-couche permet donc de réaliser une classification en 2 classes dans le cas d'un problème linéairement séparable, c'est à dire pour lequel tous les exemples d'une classe peuvent être séparés de tous les exemples de l'autre classe par une droite.

III.2) Apprentissage

III.2.1) Règle du perceptron

Le but de l'apprentissage est d'obtenir automatiquement les droites de séparation, à partir d'exemples d'entrées/sorties. Les paramètres de ces droites (pente et ordonnée à l'origine) sont codés dans les poids du réseau.

L'apprentissage est de type supervisé car il utilise l'information de classe correcte des exemples.

La règle dite d'apprentissage supervisé est définie par :

$$\Delta w_{ij} = \text{ taux d'apprentissage} \times \text{erreur de sortie} \times \text{entrée}$$

soit

$$\Delta w_{ij} = \alpha.(y_j^{(d)} - y_j).x_i$$

où

- $y_j^{(d)}$ est la valeur désirée (c'est à dire la valeur exacte) de la sortie du neurone j ;
- y_j est la valeur fournie par le neurone ;
- x_i est l'entrée i des neurones.

Elle peut être appliquée à un neurone unique ou plusieurs. L'erreur est locale à chaque neurone.

III.2.2) Apprentissage par descente de gradient

Il s'agit d'une autre méthode d'apprentissage, basée sur la diminution du gradient d'une erreur par rapport aux poids du réseau.

La règle d'apprentissage est définie par :

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

où :

- α est le taux d'apprentissage, positif et compris entre 0 et 1 (la plupart du temps très inférieur à 1).
- E est l'erreur de sortie : dans le cas de plusieurs neurones et d'un apprentissage incrémental (voir paragraphe suivant), elle est égale à la somme des erreurs individuelles des neurones de sortie :

  $$E = \frac{1}{2} \sum_{i=1}^{m} (y_i^{(d)} - y_i)^2$$

  Le coefficient 1/2 est arbitraire.
- $y_j^{(d)}$ est la sortie désirée du neurone j et y_j sa sortie effective. L'apprentissage est du type supervisé.
- *m* est le nombre de neurones.

Le calcul de la dérivée de $\partial E / \partial w_{ij}$ permet de déterminer l'expression de la règle d'apprentissage :

$$\Delta w_{ij} = \alpha.(y_j^{(d)} - y_j).x_i$$

Elle est identique à la règle d'apprentissage du perceptron énoncée dans le paragraphe précédent. Ce qui signifie que réduire l'erreur locale à chaque neurone est équivalent à réduire la somme des erreurs des neurones de sortie.

III.2.3) Mode incrémental et mode "par cycles"

Selon que l'on modifie les poids des neurones après chaque présentation d'une entrée ou après la présentation de tous les exemples de la base d'apprentissage (en fonction de l'erreur cumulée), on est en mode d'apprentissage instantané ou différé.

L'algorithme est un peu différent dans les deux cas.

a) Apprentissage incrémental

Ce mode d'apprentissage est également appelé temps-réel, en ligne, ou instantané.

L'algorithme correspondant est le suivant :

```
Initialisation des poids
Pour chaque cycle d'apprentissage (jusqu'à convergence)
Début
    Pour chaque exemple de la base d'apprentissage
    Début
        Présentation de l'exemple au réseau
        Activation du réseau (=propagation d'activité directe=calcul de sa sortie)
        Calcul de l'erreur instantanée de sortie
        Modification des poids en fonction de l'erreur
    Fin
Fin
```

b) Apprentissage par cycles

Ce mode d'apprentissage est également appelé hors-ligne ou différé (batch).
La présentation de tous les exemples une fois est appelée cycle (epoch).
L'algorithme correspondant est le suivant :

```
Initialisation des poids
Pour chaque cycle d'apprentissage (jusqu'à convergence)
Début
   Pour chaque exemple de la base d'apprentissage
   Début
      Présentation de l'exemple au réseau
      Activation du réseau
      Calcul de l'erreur de sortie et cumul
   Fin
   Modification des poids en fonction de l'erreur cumulée
Fin
```

Dans l'apprentissage en temps différé, la règle d'apprentissage n'est appliquée qu'une fois que tous les vecteurs d'apprentissage ont été présentés au réseau. L'erreur est la valeur moyenne calculée sur tous les exemples de la base d'apprentissage :

$$E = \frac{1}{2F} \sum_{f=1}^{F} \sum_{i=1}^{m} (y_i^{(d,f)} - y_i^{(f)})$$

où F est le nombre de formes présentes dans cette base.

III.3) Algorithme de rétropropagation du gradient

Les réseaux monocouches ne peuvent traiter que les problèmes linéairement séparables, ce qui n'est pas le cas de la plupart des problèmes réels. Par exemple, le problème pourtant simple du OU-Exclusif n'est pas linéairement séparable. L'extension de l'apprentissage par descente de gradient aux réseaux multi-couches a donc été développée pour traiter les problèmes dans lesquels les classes peuvent avoir des formes quelconques.

III.3.1) Principe

Le nom de cette méthode désigne la façon dont le gradient de l'erreur de sortie est utilisé pour adapter les poids synaptiques du réseau, de la couche de sortie vers la ou les couche(s) en amont, dans le but de réduire ses erreurs.

Les algorithmes en mode incrémental et en mode "par cycle" sont les mêmes que ceux étudiés précédemment dans le cas d'un réseau monocouche, excepté le fait que la modification des poids est elle-même composée d'une boucle :

```
…
Pour chaque couche i, de la couche de sortie vers la couche d'entrée
   Modification des poids de la couche i par rétropropagation du gradient de l'erreur de sortie
…
```

III.3.2) Formules de la rétro-propagation

Comme dans le cas mono-couche (décrit plus haut), la règle d'apprentissage par descente de gradient est définie par :

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

Dans le cas des réseaux multicouches, elle peut donner naissance à la règle d'apprentissage dite "delta généralisée" (generalized delta-rule), définie par :

$$\Delta w_{ij}^{(c)} = \alpha . \delta_j^{(c)} . y_i^{(c-1)}$$

avec

$$\delta_j = e_j . f'(a_j) \text{ pour le neurone j de la couche de sortie}$$

$$\delta_j^{(c)} = f'(a_j^{(c)}) . \sum_k \delta_k^{(c+1)} . w_{jk}^{(c+1)} \text{ pour le neurone j d'une couche cachée}$$

et

- c est l'indice de la couche courante, c-1 l'indice de la couche directement en amont (en général à gauche) et c+1 l'indice de la couche directement en aval (en général à droite)
- e_j est l'erreur de sortie au neurone j, définie par : $e_j = y_j^{(d)} - y_j$ ($y_j^{(d)}$ sortie désirée)
- $a_j = \sum_i w_{ij} y_i$.
- $\delta_j^{(c)}$ est appelé gradient local au neurone j (caché ou de sortie).
- c est l'indice de la couche
- f(x) est une fonction non-linéaire pour au moins une des couches, mais ne peut pas être la fonction seuil

Remarque sur les notations : le réseau étant composé de plusieurs couches, les sorties des neurones sont toutes désignées par y ; l'indice c est ajouté pour préciser à quelle couche ces derniers appartiennent.

III.3.3) Démonstrations des formules

Le principe de base utilisé dans cet algorithme est la descente de gradient, qui s'exprime par :

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

Le principe de calcul pour obtenir ces formules est le suivant : puisque ce que l'on cherche est la dérivée de l'erreur de sortie *E* par rapport à w_{ij}, il faut faire apparaître la dépendance entre eux. En mode d'apprentissage en temps-réel (ou instantané), *E* est définie par exemple par :

$$E = \frac{1}{2} \sum_{j=1}^{n} (y_j^{(d)} - y_j)^2$$

où *n* est le nombre de neurones de sortie, y_j la sortie du neurone de sortie *j*, définie par :

$$y_j = f\left(\sum_i w_{ij} y_i \right)$$

où f est la fonction d'activation (dérivable) des neurones de la couche de sortie, w_{ij} les poids des neurones de sortie et y_i les sorties des neurones de la couche précédente.

On voit que la dépendance entre y_j et w_{ij} est à 2 niveaux : la fonction d'activation $f()$ et la somme. On fait donc apparaître une variable intermédiaire : a_j, l'activation du neurone j :

$$a_j = \sum_i w_{ij} y_i$$

et

$$y_j = f(a_j)$$

On effectue donc la décomposition du gradient de l'erreur par rapport aux poids, en dérivées partielles faisant apparaître toutes ces dépendances :

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial a_j} \cdot \frac{\partial a_j}{\partial w_{ij}}$$

Les paramètres y_j, a_j et w_{ij} concernent la couche de sortie.

On peut également effectuer cette décomposition pour le cas de neurones de couches cachées. E représente toujours l'erreur de sortie, et les autres paramètres y_j, a_j et w_{ij} sont ceux de la couche cachée concernée.

Les 2 derniers termes de cette expression s'exprimeront de la même manière quelle que soit la couche que l'on est en train de calculer : y_j, a_j et w_{ij} sont exprimés par rapport à cette même couche, et donc la dépendance entre eux sera la même d'une couche à l'autre.
Par contre le premier sera différent pour la couche de sortie et pour les autres couches (couches cachées), puisque E est exprimée par rapport à la couche de sortie.

Calcul du 2e terme

Le lien entre y_j et a_j est : $y_j = f(a_j)$. On a donc :

$$\frac{\partial y_j}{\partial a_j} = f'(a_j)$$

où le prime désigne la dérivée par rapport à l'argument :

$$f'(a_j) = \frac{\partial f(a_j)}{\partial a_j}$$

Calcul du 3e terme

L'activation a_j est définie par :

$$a_j = \sum_i w_{ij} y_i$$

donc :

$$\frac{\partial a_j}{\partial w_{ij}} = y_j$$

Calcul du 1er terme

1er cas : neurone de sortie

Dans le cas d'un neurone de sortie, la dépendance entre E et y_j est :

$$E = \frac{1}{2}\sum_{i=1}^{n}(y_i^{(d)} - y_i)^2$$

ou si l'on fait apparaître l'erreur de sortie du neurone i : e_i

$$E = \frac{1}{2}\sum_{i=1}^{m} e_i^2 \text{ avec } e_i = y_i^{(d)} - y_i$$

donc

$$\frac{\partial E}{\partial y_j} = -e_j$$
$$= -(y_j^{(d)} - y_j)$$

En effet, si l'on détaille :

$$\frac{\partial E}{\partial y_j} = \frac{1}{2}\left(\frac{\partial e_1^2}{\partial y_j} + \frac{\partial e_2^2}{\partial y_j} + \ldots + \frac{\partial e_j^2}{\partial y_j} + \ldots\right)$$

Finalement, si l'on réunit tous ces résultats partiels :

$$\boxed{\frac{\partial E}{\partial w_{ij}} = -e_j.f'(a_j).y_j}$$

Finalement :

$$\Delta w_{ij} = -\alpha\frac{\partial E}{\partial w_{ij}} = \alpha e_j.f'(a_j).y_j = \alpha.\delta_j.y_j$$

avec :

$$\delta_j = e_j.f'(a_j)$$

Cette grandeur est appelée gradient local au neurone *j*.

2^e^ cas : neurone de la dernière couche cachée

Pour un neurone de cette couche, le calcul est le même que le cas d'un neurone de sortie, jusqu'à :

$$\frac{\partial E}{\partial y_j} = \frac{1}{2}\sum_k \frac{\partial e_k^2}{\partial y_j}$$

k est un indice sur tous les neurones de la couche de sortie. y_j représente maintenant la sortie d'un neurone de la couche cachée, ce qui change tout : contrairement au cas d'un neurone se sortie, où seule la dérivée de e_j par rapport à y_j était non-nulle, maintenant aucune des dérivées n'est nulle : en effet, e_k représente toujours une erreur de sortie (celle du *k*ième neurone de sortie), alors que y_j représente la sortie d'une couche en amont. Comme tous les neurones de cette couche sont connectés aux neurones de la couche de sortie, l'erreur de sortie du k^e neurone e_k dépend de toutes les sorties des couches en amont de la couche de sortie. Nous devons donc écrire :

$$\frac{\partial E}{\partial y_j} = \sum_k e_k \frac{\partial e_k}{\partial y_j}$$

On doit encore décomposer cette expression car la dépendance entre e_k et y_j est à deux niveaux :

$$\begin{aligned} e_k &= y_k^{(d)} - y_k \\ &= y_k^{(d)} - f(a_k) \\ &= y_k^{(d)} - f(\sum_j w_{jk} y_j) \end{aligned}$$

où j est un indice sur tous les neurones de la dernière couche cachée.

La décomposition utile est donc :

$$\frac{\partial E}{\partial y_j} = \sum_k e_k \frac{\partial e_k}{\partial a_k} . \frac{\partial a_k}{\partial y_j}$$

avec

$$\frac{\partial e_k}{\partial a_k} = -f'(a_k)$$

et

$$\frac{\partial a_k}{\partial y_j} = w_{jk}$$

car

$$a_k = \sum_j w_{jk} y_j$$

Finalement on obtient :

$$\frac{\partial E}{\partial y_j} = -\sum_k e_k f'(a_k) w_{jk}$$

et donc

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} . \frac{\partial y_j}{\partial a_j} . \frac{\partial a_j}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} f'(a_j) . y_j = -\left[\sum_k e_k f'(a_k) w_{jk} \right] f'(a_j) . y_j$$

On remarque que le gradient local apparaît dans cette expression. Posons :

$$\delta_k = e_k . f'(a_k)$$

On peut alors écrire :

$$\frac{\partial E}{\partial y_j} = -\sum_k \delta_k w_{jk}$$

et

$$\boxed{\frac{\partial E}{\partial w_{ij}} = -f'(a_j) . y_j . \sum_k \delta_k w_{jk}}$$

Souvenons-nous que l'on a aussi, d'après la définition du gradient local δ_j :

$$\frac{\partial E}{\partial w_{ij}} = -\delta_j . y_j$$

donc, par identification :

$$\delta_j = f'(a_j) . \sum_k \delta_k . w_{jk}$$

On a donc obtenu une récurrence : pour la dernière couche cachée (donc l'avant-dernière couche de neurones) le gradient local est exprimé en fonction des gradients locaux de la

couche de sortie. On retrouvera cette récursivité entre les autres couches, et donc la méthode permet d'adapter les poids de n'importe quelle couche (en partant de la dernière et en remontant jusqu'à la première) du réseau, menant à une diminution progressive de l'erreur de sortie.

III.4) Caractéristiques et paramètres de l'algorithme

III.4.1) Caractéristiques

L'algorithme de rétropropagation du gradient permet de résoudre des problèmes de classification dans lesquels les classes ne sont pas linéairement séparables.

L'erreur de sortie est une fonction d'un grand nombre de variables : les poids. Ils définissent une surface dans l'espace des poids. Cette surface possède une forme complexe, composée de nombreux minima locaux. Le principe de descente de gradient sur lequel est basé l'algorithme garantit que l'erreur évolue toujours vers un minimum, mais pas forcément le minimum global.

Ce modèle est sujet à l'apprentissage par cœur, c'est à dire avec une erreur sur les données d'apprentissage qui diminue continuellement au cours de l'apprentissage, mais avec une erreur sur les données de test, qui diminue dans un premier temps, puis augmente ensuite. On utilise donc la validation croisée (alternance des cycles d'apprentissage avec des données et des cycles de test avec d'autres données), et on utilise le taux d'erreur sur la base de test comme critère d'arrêt. En pratique, on mémorise les poids du réseau à chaque cycle d'apprentissage, et on ne conserve que le fichier de poids correspondant à l'erreur minimale sur la base de test.

III.4.2) Paramètres

Dans le cas de la rétropropagation du gradient comme dans le cas des réseaux de neurones en général, la plupart des paramètres sont déterminés de manière empirique.

a) Structure du réseau

Nombre de couches cachées

Une seule couche cachées permet de traiter la plupart des problèmes.

Le fait d'en utiliser 2 ou 3, voire plus, permet de définir des régions plus complexes dans l'espace des entrées.

Nombre de neurones dans les couches cachées

Si le nombre de neurones dans les couches cachées est trop grand, le réseau va avoir tendance à réaliser un apprentissage par cœur des données d'apprentissage, et donc à mal généraliser à de nouvelles données.
S'il est trop petit, il ne possédera pas assez de variables internes pour résoudre le problème à traiter.
Le choix du nombre de neurones est donc un compromis entre ces 2 aspects.

Connectivité

En général, on utilise une connectivité complète entre les entrées et les différentes couches, c'est à dire que les neurones sont connectés à toutes les neurones de la couche en amont, et à toutes les entrées pour le cas de la 1ère couche cachée.

b) Fonction d'activation

La fonction d'activation des neurones peut a priori être quelconque pourvu qu'elle soit dérivable, ce qui exclus la fonction seuil. La dérivée de la fonction d'activation intervient en effet dans la règle de modification des poids.

De plus, il faut au moins qu'il y ait une couche avec une fonction d'activation non-linéaire, car si toutes les couches étaient linéaires, cela serait équivalent à un réseau à une seule couche linéaire ; or un réseau monocouche ne peut pas traiter un problème non-linéairement séparable.

En général, c'est la sigmoïde qui est utilisée. Elle peut être uniquement positive ou positive et négative (symétrique par rapport au centre du repère). Dans le 2e cas, l'apprentissage est plus rapide car les poids sont modifiés même pour des valeurs d'entrées de la sigmoïde négatives.

c) Taux d'apprentissage

Il s'agit du paramètre α présent dans la règle d'apprentissage.

S'il est très faible, l'apprentissage est lent.

S'il est grand, on risque d'avoir des oscillations des valeurs des poids.

On peut le faire diminuer en cours d'apprentissage (comme dans le cas des cartes de Kohonen), pour améliorer la stabilisation du réseau.

d) Valeurs initiales des poids

Si les valeurs initiales des poids sont très faibles, l'apprentissage est lent car ils interviennent dans la règle de modification.

Si elles sont grandes, l'apprentissage est également lent car la somme pondérée des entrées d'un neurone est grande, et la dérivée de la fonction de transfert est faible. Or celle-ci intervient dans la règle de modification des poids.

III.5) Variantes

Les variantes décrites dans ce paragraphe ont toutes pour objectif d'améliorer les performance de la version de base de l'algorithme de la rétropropagation.

Cette amélioration concerne le taux d'erreur sur la base de test ou la rapidité de convergence.

III.5.1) Ajout d'un terme de moment

Une méthode pour accélérer la convergence de l'apprentissage est d'ajouter un terme appelé "moment" (momentum) dans la règle d'apprentissage, défini par :

$$\eta.\Delta w_{ij}(n-1)$$

La règle d'apprentissage devient donc :

$$\Delta w_{ij}(n) = -\alpha \frac{\partial E}{\partial w_{ij}} + \eta.\Delta w_{ij}(n-1)$$

où *n* est l'indice de l'itération d'apprentissage actuelle (et donc *n*-1 l'indice de l'itération précédente) et avec

$$0 < \eta < 1 \quad \text{(typiquement : } \eta \approx 0{,}5 - 0{,}9\text{)}$$

Ce terme permet d'augmenter le taux d'apprentissage sans provoquer d'oscillations des poids, et donc d'éviter des minima locaux.

La variation des poids ne dépend donc plus que du gradient mais également de leur variation précédente.

Ce moment introduit une certaine "douceur" dans l'apprentissage, par le biais d'une récurrence : pour un exemple donné, si la variation des poids est grande, elle aura tendance à le rester à l'exemple suivant.

Il est nécessaire que η soit inférieur à 1 en valeur absolue pour éviter une divergence.

Il permet ainsi de s'affranchir des petites discontinuités de la surface d'erreur, et donc d'éviter de rester bloqué dans un minimum local.

En général le moment est diminué au cours de l'apprentissage.

III.5.2) Ajout de bruit

Le fait d'ajouter des valeurs aléatoires aux poids et/ou aux entrées peut permettre à la fonction d'énergie de sortir d'un minimum local.

Ces valeurs aléatoires doivent rester petites, pour éviter de "rater" des minima locaux intéressants.

Le fait de sélectionner aléatoirement les exemples d'entrée dans la base d'apprentissage constitue également l'apport d'un autre aspect aléatoire au processus d'apprentissage.

III.5.3) RPROP (Résilient Propagation)

Proposée par Riedmiller et Braun en 1993, le principe de cette variante de la rétropropagation est de n'utiliser que le signe du gradient de l'erreur de sortie, au lieu de l'amplitude du gradient. Cela permet de s'affranchir des inconvénients dus à cette amplitude (notamment le fait qu'elle soit liée à la dérivée de la fonction de transfert, et peut donc être très faible).

L'idée est de d'augmenter la quantité de variation des poids quand le gradient ne change pas de signe (ce qui signifie que l'erreur diminue, et que les poids évoluent dans le bon sens), et de l'augmenter sinon (on a sauté un minimum de la surface d'erreur).

La version de l'apprentissage utilisée ici est le mode différé (batch mode) : cumul de l'erreur sur tous les vecteurs d'exemple de la base d'apprentissage (=1 cycle), suivi de la modification des poids. L'algorithme d'apprentissage ne change pas par rapport à la version de base de la rétropropagation en mode différé.

A chaque cycle d'apprentissage, les poids sont modifiés (de la couche de sortie vers la 1ère couche) des quantités suivantes :

$$\Delta w_{ij}(n) = \begin{cases} -\Delta_{ij}(n), & \text{si } \dfrac{\partial E}{\partial w_{ij}}(n) > 0 \\ +\Delta_{ij}(n), & \text{si } \dfrac{\partial E}{\partial w_{ij}}(n) < 0 \\ 0, & \text{sinon} \end{cases} \quad \text{avec} \quad \Delta_{ij}(n) = \begin{cases} \eta^{-}\Delta_{ij}(n-1), & \text{si } \dfrac{\partial E}{\partial w_{ij}}(n-1).\dfrac{\partial E}{\partial w_{ij}}(n) > 0 \\ \eta^{+}\Delta_{ij}(n-1), & \text{si } \dfrac{\partial E}{\partial w_{ii}}(n-1).\dfrac{\partial E}{\partial w_{ij}}(n) < 0 \\ 0, & \text{sinon} \end{cases}$$

et

$$0 < \eta^{-} < 1 < \eta^{+}$$

où n est l'indice de l'itération courante de l'algorithme, et n-1 l'indice de l'itération précédente.

Remarque : le choix de ces 2 paramètres influe très peu sur les résultats de l'algorithme.

Chaque poids est donc associé à un paramètre d'adaptation spécifique.

Il y a une exception à la règle ci-dessus : si le gradient change de signe, les poids sont remis à leur valeur précédent leur modification :

$$\Delta w_{ij}(n) = -\Delta w_{ij}(n-1) \qquad \text{si} \qquad \frac{\partial E}{\partial w_{ij}}(n-1).\frac{\partial E}{\partial w_{ij}}(n) < 0$$

Pour ne pas répéter la même étape, il faut alors laisser passer un cycle d'apprentissage sans modification du paramètre $\Delta_{ij}(n)$, en imposant $\dfrac{\partial E}{\partial w_{ij}}(n-1) = 0$ dans la règle de modification des $\Delta_{ij}(n)$ ci-dessus.

Cette version de la rétropropagation converge plus rapidement vers une solution que sa version de base.

III.5.4) QuickProp

Proposée par Fahlman en 1988, l'algorithme QuickProp consiste à observer l'évolution du signe du gradient $\partial E / \partial w_{ij}$.

La modification du poids reliant le neurone i au neurone j, à l'itération n, est définie par :

$$\Delta w_{ij}(n) = \frac{s(n)}{s(n-1) - s(n)} \Delta w_{ij}(n-1)$$

avec

$$s(n) = \frac{\partial E}{\partial w_{ij}}(n) \qquad \text{et} \quad s(n-1) = \frac{\partial E}{\partial w_{ij}}(n-1)$$

Cette méthode nécessite de mémoriser, pour chacun des poids, les valeurs du gradient, ainsi que la quantité de modification de l'itération précédente de l'apprentissage.

L'apprentissage est de type différé (batch).

Cette variante de la rétropropagation permet une convergence environ 4 fois plus rapide que la version de base.

Annexes

A.1) Quelques éléments de calcul vectoriel et matriciel

Vecteurs

La notation usuelle pour un vecteur est sous forme d'une colonne.

Norme

Soit $x = (x_1, x_2, ..., x_n)^t$ un vecteur de dimension n.

Sa norme euclidienne est définie par :

$$\|x\| = \sqrt{x_1^2 + x_2^2 + ... + x_n^2}$$

$$= \sqrt{\sum_{i=1}^{n} x_i^2}$$

Un vecteur est dit normal si sa norme vaut 1.

Produit scalaire

Le produit scalaire de 2 vecteurs $x = (x_1, x_2, ..., x_n)^t$ et $y = (y_1, y_2, ..., y_n)^t$ est défini par :

$$x.y = \sum_{i=1}^{n} x_i . y_i$$

La norme d'un vecteur x est également la racine du produit scalaire de son transposé avec lui-même :

$$\|x\|^2 = x^t.x$$

La normalisation d'un vecteur consiste à rendre sa norme égale à 1. Ce résultat est obtenu en divisant toutes les composantes du vecteur par la norme de ce dernier :

$$x' = \frac{x}{\|x\|}$$

est normalisé. On dit également qu'il est normal.

Orthogonalité entre vecteurs

Le cosinus de 2 vecteurs est défini par :

$$\cos(x, y) = \frac{x^t.y}{\|x\|.\|y\|}$$

Il correspond au produit scalaire des versions normalisées des deux vecteurs.
Les deux vecteurs sont orthogonaux si le cosinus entre eux est nul.

Une matrice est dite orthogonale si toutes ses colonnes sont orthogonales entre elles deux à deux.
Une matrice orthogonale multipliée par sa transposée donne une matrice diagonale.

Projection

La projection d'un vecteur x sur un autre y est définie par :

$$\frac{x^t.y}{\|y\|}$$

Distance

La distance entre deux vecteurs est une grandeur souvent utilisée pour quantifier la similarité entre eux. Elle ne peut être calculée que si les deux vecteurs ont la même dimension.

La distance euclidienne entre deux vecteurs x et y est définie par :

$$d_E(x,y) = \sqrt{\|x-y\|} = \sqrt{(x-y)^t(x-y)} = \sqrt{\sum_i (x_i - y_i)^2}$$

Un autre exemple de distance est la distance de Hamming, définie par :

$$d_H(x,y) = \sum_{i=1}^{n} |x_i - y_i|$$

Lien entre distance euclidienne et produit scalaire

Soit d(x,y) la distance euclidienne entre les vecteurs x et y. On a :

$$d^2(x,y) = (x-y)^t(x-y)$$

$$\leftrightarrow \qquad d^2(x,y) = x^t x + y^t y - 2x^t y = \|x\|^2 + \|y\|^2 - 2x^t y$$

Si les vecteurs x et y sont normalisés, on a

$$d^2(x,y) = 2(1 - x^t y)$$

Si la distance est minimale, le produit scalaire est maximal.

On a également

$$d^2(x,y) = \|x\|^2 + \|y\|^2 - 2\|x\|\|y\|\cos(x,y)$$

Indépendance

Soient $\{x_1, x_2, \ldots, x_m\}$ un ensemble de vecteurs. Un vecteur y est une combinaison linéaire de ces vecteurs s'il peut être mis sous la forme

$$y = c_1x_1 + c_2x_2 + \ldots + c_mx_m$$

où c_i sont des nombres réels dont au moins l'un d'entre eux est non nul. Par exemple, $y = x_1$ est une combinaison linéaire.

L'ensemble des vecteurs x_i est linéairement indépendant si on ne peut pas trouver un ensemble de valeurs $\{c_1, c_2, \ldots, c_m\}$ tel que

$$c_1x_1 + c_2x_2 + \ldots + c_mx_m = \mathbf{0} \quad \text{(vecteur nul)}$$

dont au moins un des coefficients est non nul ; ce qui revient à dire qu'aucun des vecteurs n'est une combinaison linéaire des autres.

Ces notions s'appliquent également aux lignes et aux colonnes des matrices.

Par exemple, dans la matrice A suivante, les colonnes sont linéairement dépendantes :

$$A=\begin{pmatrix}1 & 2 & 1\\ 2 & 4 & 1\\ 3 & 6 & 0\end{pmatrix}$$

Dans la matrice B suivante également, bien que cela soit moins évident :

$$B=\begin{pmatrix}3 & 4 & 2\\ 1 & 0 & 2\\ 2 & 1 & 3\end{pmatrix}$$

En effet :

$$\begin{pmatrix}4\\ 0\\ 1\end{pmatrix}=2\times\begin{pmatrix}3\\ 1\\ 2\end{pmatrix}-\begin{pmatrix}2\\ 2\\ 3\end{pmatrix}$$

Rang d'une matrice

Le rang d'une matrice n'est pas forcément égal à sa dimension. Il est égal au plus grand nombre de lignes (ou de colonnes) constituant un ensemble linéairement indépendant.

Par exemple, la dimension de la matrice A est 3×3, et son rang est 3×2.

Inverse d'une matrice

Cette opération n'est applicable qu'aux matrices carrées, et uniquement de plein rang, c'est à dire des matrices dont le rang est égal à la dimension.

Propriétés :

- $A\times A^{-1}=I$
- $A\times A^{-1}= A^{-1}\times A$

Dans le cas d'une matrice diagonale $A=\text{diag}\{a_{i,i}\}$, l'inverse est égale à :

$$A^{-1}=\text{diag}\{1/a_{i,i}\}$$

Dans le cas de matrices non carrées, on définit la notion d'inverse généralisée (encore appelée pseudo-inverse ou inverse de Moore-Penrose).

Vecteurs propres et valeurs propres

Un vecteur u qui vérifie l'équation

$$A.u=\lambda.u \quad (1)$$

est un vecteur propre de la matrice A et λ est la valeur propre qui lui est associée. On a également :

$$(1) \leftrightarrow A.u-\lambda.u=0$$
$$\leftrightarrow (A-\lambda.I).u=0$$

où I est la matrice identité (des 1 sur la diagonale, des 0 partout ailleurs). Cette équation est appelée équation caractéristique de la matrice. Si u est un vecteur propre, tout vecteur k.u avec k constante est un autre vecteur propre.

Une matrice de dimension n×n possède n vecteurs propres (associés chacun à une valeur propre).

En général on rassemble les vecteurs propres dans une matrice, sous la forme de ses colonnes, et les valeurs propres dans une autre matrice, disposées en colonne, dans l'ordre correspondant à celui des vecteurs propres. Soient U et Λ ces deux matrices, respectivement. L'équation matricielle correspondant à (1) est :

$$A.U=\Lambda.U$$

A.2) Eléments de statistiques

Moyenne d'une variable

Par définition, une variable peut prendre différentes valeurs. Soient x_i n exemples (réalisations) de la variables x. La moyenne d'une variable x (par exemple la variable "longueur de pétale" pour les données Iris) est définie par :

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

On dit qu'une variable est centrée si sa moyenne est nulle.

Variance d'une variable

La variance d'une variable est égale à la somme des carrés des différences entre les exemples de cette variable et leur moyenne :

$$\mathrm{var}(x) = \frac{\sum_{i=1}^{n} (x_i - \overline{x})^2}{n}$$

La variance caractérise donc la somme des écarts d'une variable par rapport à sa moyenne.

Covariance entre des variables différentes

Dans le cas de données multi-dimensionnelles, il y a plusieurs variables (par exemple dans les données Iris il y a 4 variables). Ces variables peuvent être corrélées ou non. Si elles sont corrélées, leur covariance et leur produit de corrélation, définis ci-dessous, ne sont pas nuls.

Soient x_i et x_j deux variables et leurs moyennes respectivement $\overline{x_i}$ et $\overline{x_j}$. La covariance entre elles est définie par :

$$\mathrm{cov}(x_i, x_j) = \frac{1}{n} \sum_{k=1}^{n} (x_{i,k} - \overline{x_i})(x_{j,k} - \overline{x_j})$$

où n est le nombre de réalisations des variables (=nombre de vecteurs d'exemple), $x_{i,k}$ est le k^e exemple de la variable i (et $x_{j,k}$ est la k^e réalisation de la variable j).

Par exemple, dans les données *Iris*, la covariance entre la longueur et la largeur de pétale n'est pas nulle.

La covariance entre une variable et elle-même est égale à sa variance.

Corrélation entre des variables différentes

Quand la covariance entre des variables différentes n'est pas nulle, ces variables varient ensemble et elles sont également corrélées.

Le produit de corrélation est la version normalisée (il est compris entre 0 et 1) de la covariance :

$$\mathrm{cor}(x_i, x_j) = \frac{\mathrm{cov}(x_i, x_j)}{\sigma_i \sigma_j}$$

où

$$\sigma_i = \sqrt{\mathrm{var}(x)}\ ,\ \mathrm{i}=1,2$$

est appelé écart-type de la variable *x*.

Dans le cas d'un nombre plus grand de variables, celles-ci peuvent être corrélées 2 à 2. On définit alors une matrice de variance-covariance C :

$$C = \left\{\mathrm{cov}(x_i, x_j), i = 1,...,n; j = 1,...,n\right\}$$

où i est l'indice de la colonne dans la matrice C et j l'indice de la ligne, ou inversement puisque cette matrice est symétrique.

Cette notation signifie qu'à l'intersection de la ligne j et de la colonne i de cette matrice se trouve la covariance entre les variables x_i et x_j. La dimension de C est donc n×n.

La matrice de covariance peut également être obtenue de la façon suivante : à partir du produit d'une matrice A et de sa transposée, où A contient tous les vecteurs de données, auxquels on a soustrait leur moyenne (définie ci-dessus), disposés en colonnes. A^T contient donc les mêmes vecteurs mais disposés en lignes :

$$C = \frac{1}{n}.A.A^T$$

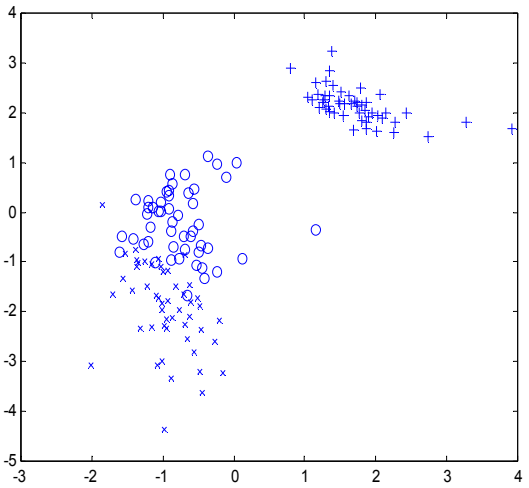
A.3) Exemples de données

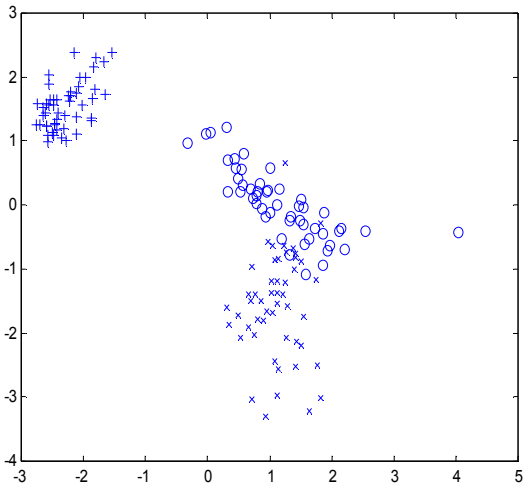
A.3.1) Iris

Contenu du fichier original :

5.1,3.5,1.4,0.2,Iris-setosa	7.0,3.2,4.7,1.4,Iris-versicolor	6.3,3.3,6.0,2.5,Iris-virginica
4.9,3.0,1.4,0.2,Iris-setosa	6.4,3.2,4.5,1.5,Iris-versicolor	5.8,2.7,5.1,1.9,Iris-virginica
4.7,3.2,1.3,0.2,Iris-setosa	6.9,3.1,4.9,1.5,Iris-versicolor	7.1,3.0,5.9,2.1,Iris-virginica
4.6,3.1,1.5,0.2,Iris-setosa	5.5,2.3,4.0,1.3,Iris-versicolor	6.3,2.9,5.6,1.8,Iris-virginica
5.0,3.6,1.4,0.2,Iris-setosa	6.5,2.8,4.6,1.5,Iris-versicolor	6.5,3.0,5.8,2.2,Iris-virginica
5.4,3.9,1.7,0.4,Iris-setosa	5.7,2.8,4.5,1.3,Iris-versicolor	7.6,3.0,6.6,2.1,Iris-virginica
4.6,3.4,1.4,0.3,Iris-setosa	6.3,3.3,4.7,1.6,Iris-versicolor	4.9,2.5,4.5,1.7,Iris-virginica
5.0,3.4,1.5,0.2,Iris-setosa	4.9,2.4,3.3,1.0,Iris-versicolor	7.3,2.9,6.3,1.8,Iris-virginica
4.4,2.9,1.4,0.2,Iris-setosa	6.6,2.9,4.6,1.3,Iris-versicolor	6.7,2.5,5.8,1.8,Iris-virginica
4.9,3.1,1.5,0.1,Iris-setosa	5.2,2.7,3.9,1.4,Iris-versicolor	7.2,3.6,6.1,2.5,Iris-virginica
5.4,3.7,1.5,0.2,Iris-setosa	5.0,2.0,3.5,1.0,Iris-versicolor	6.5,3.2,5.1,2.0,Iris-virginica
4.8,3.4,1.6,0.2,Iris-setosa	5.9,3.0,4.2,1.5,Iris-versicolor	6.4,2.7,5.3,1.9,Iris-virginica
4.8,3.0,1.4,0.1,Iris-setosa	6.0,2.2,4.0,1.0,Iris-versicolor	6.8,3.0,5.5,2.1,Iris-virginica
4.3,3.0,1.1,0.1,Iris-setosa	6.1,2.9,4.7,1.4,Iris-versicolor	5.7,2.5,5.0,2.0,Iris-virginica
5.8,4.0,1.2,0.2,Iris-setosa	5.6,2.9,3.6,1.3,Iris-versicolor	5.8,2.8,5.1,2.4,Iris-virginica
5.7,4.4,1.5,0.4,Iris-setosa	6.7,3.1,4.4,1.4,Iris-versicolor	6.4,3.2,5.3,2.3,Iris-virginica
5.4,3.9,1.3,0.4,Iris-setosa	5.6,3.0,4.5,1.5,Iris-versicolor	6.5,3.0,5.5,1.8,Iris-virginica
5.1,3.5,1.4,0.3,Iris-setosa	5.8,2.7,4.1,1.0,Iris-versicolor	7.7,3.8,6.7,2.2,Iris-virginica
5.7,3.8,1.7,0.3,Iris-setosa	6.2,2.2,4.5,1.5,Iris-versicolor	7.7,2.6,6.9,2.3,Iris-virginica
5.1,3.8,1.5,0.3,Iris-setosa	5.6,2.5,3.9,1.1,Iris-versicolor	6.0,2.2,5.0,1.5,Iris-virginica
5.4,3.4,1.7,0.2,Iris-setosa	5.9,3.2,4.8,1.8,Iris-versicolor	6.9,3.2,5.7,2.3,Iris-virginica
5.1,3.7,1.5,0.4,Iris-setosa	6.1,2.8,4.0,1.3,Iris-versicolor	5.6,2.8,4.9,2.0,Iris-virginica
4.6,3.6,1.0,0.2,Iris-setosa	6.3,2.5,4.9,1.5,Iris-versicolor	7.7,2.8,6.7,2.0,Iris-virginica
5.1,3.3,1.7,0.5,Iris-setosa	6.1,2.8,4.7,1.2,Iris-versicolor	6.3,2.7,4.9,1.8,Iris-virginica
4.8,3.4,1.9,0.2,Iris-setosa	6.4,2.9,4.3,1.3,Iris-versicolor	6.7,3.3,5.7,2.1,Iris-virginica

Affichage de Sammon des données (2 exécutions différentes de l'algorithme)





A.3.2) Wine

Il y a 3 classes, qui correspondent à 3 cultivateurs différents. Dans le cadre ci-dessous sont données les 4 premières lignes de chaque classe telles qu'elles se présentent dans le fichier de données original :

```
1,14.23,1.71,2.43,15.6,127,2.8,3.06,.28,2.29,5.64,1.04,3.92,1065
1,13.2,1.78,2.14,11.2,100,2.65,2.76,.26,1.28,4.38,1.05,3.4,1050
1,13.16,2.36,2.67,18.6,101,2.8,3.24,.3,2.81,5.68,1.03,3.17,1185
1,14.37,1.95,2.5,16.8,113,3.85,3.49,.24,2.18,7.8,.86,3.45,1480
……
2,12.37,.94,1.36,10.6,88,1.98,.57,.28,.42,1.95,1.05,1.82,520
2,12.33,1.1,2.28,16,101,2.05,1.09,.63,.41,3.27,1.25,1.67,680
2,12.64,1.36,2.02,16.8,100,2.02,1.41,.53,.62,5.75,.98,1.59,450
2,13.67,1.25,1.92,18,94,2.1,1.79,.32,.73,3.8,1.23,2.46,630
……
3,12.86,1.35,2.32,18,122,1.51,1.25,.21,.94,4.1,.76,1.29,630
3,12.88,2.99,2.4,20,104,1.3,1.22,.24,.83,5.4,.74,1.42,530
3,12.81,2.31,2.4,24,98,1.15,1.09,.27,.83,5.7,.66,1.36,560
3,12.7,3.55,2.36,21.5,106,1.7,1.2,.17,.84,5,.78,1.29,600
……
```

Remarque : le premier chiffre sur chaque ligne correspond à un numéro de classe (arbitraire)

Informations disponibles dans le fichier associé :

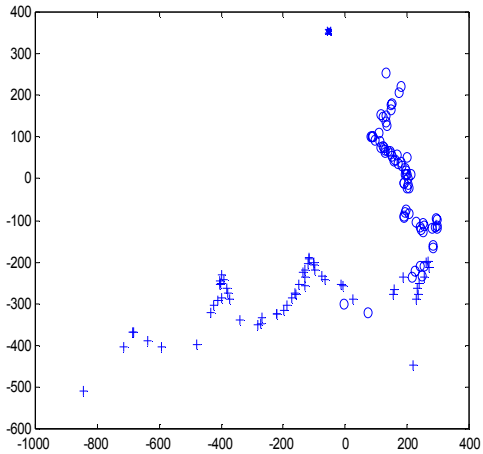
13 attributs (=caractéristiques) :

1) Alcohol
2) Malic acid
3) Ash
4) Alcalinity of ash
5) Magnesium
6) Total phenols
7) Flavanoids
8) Nonflavanoid phenols
9) Proanthocyanins
10) Color intensity
11) Hue
12) OD280/OD315 of diluted wines
13) Proline

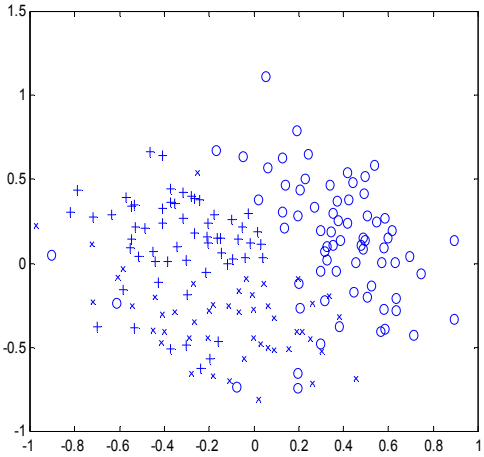
Répartition des exemples dans les 3 classes :

classe 1 : 59 classe 2 : 71 classe 3 : 48

Résultat de l'affichage de Sammon :



données non normalisées



données normalisées

A.3.3) Exemples de données : Caractères manuscrits

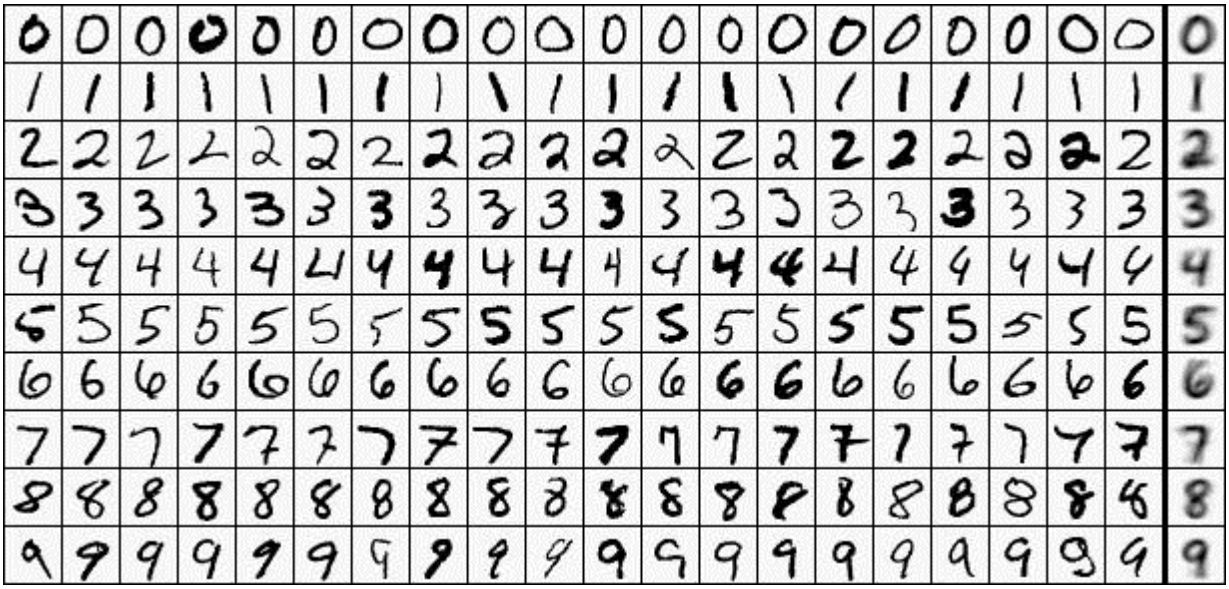
1er cas : Paramètres caractéristiques extraits d'images de caractères manuscrits (hollandais)

Extrait du fichier "caract_holland.don", qui contient 20000 exemples différents (1 exemple = 1 ligne) :

```
2 8 3 5 1 8 13 0 6 6 10 8 0 8 0 8 8 T
5 12 3 7 2 10 5 5 4 13 3 9 2 8 4 10 10 I
4 11 6 8 6 10 6 2 6 10 3 7 3 7 3 9 9 D
7 11 6 6 3 5 9 4 6 4 4 10 6 10 2 8 8 N
2 1 3 1 1 8 6 6 6 6 5 9 1 7 5 10 10 G
4 11 5 8 3 8 8 6 9 5 6 6 0 8 9 7 7 S
4 2 5 4 4 8 7 6 6 7 6 6 2 8 7 10 10 B
1 1 3 2 1 8 2 2 2 8 2 8 1 6 2 7 7 A
2 2 4 4 2 10 6 2 6 12 4 8 1 6 1 7 7 J
11 15 13 9 7 13 2 6 2 12 1 9 8 1 1 8 8 M
…
etc
…
```

2e cas : Images brutes (caractères normalisés en taille et en position)

Extrait du fichier "caract.don", qui contient 10000 exemples différents :



(la dernière colonne correspond aux images moyennes de chaque ligne)

A.4) Bibliographie sur les Réseaux de Neurones

- Hérault J., Jutten Ch., « Réseaux neuronaux et traitement du signal », Hermès, 1994
- Jodouin J. F., « Les réseaux de neurones. Principes et définitions », Hermès, 1994
- Jodouin J. F., « Réseaux neuromimétiques », Hermès, 1994
- Nadal J.P., « Réseaux de neurones. De la physique à la psychologie », Armand Colin, 1993