Les réseaux de neurones
historique, méthodes et applications

Marie Cottrell

SAMOS-MATISSE

CNRS UMR 8595

Université Paris 1- Sorbonne

Introduction

Les premiers modèles

Les réseaux

Modèle de Hopfield

Le perceptron multicouches

Algorithme de Kohonen

Conclusion

Bref historique

1940 - 1960 concepts	Mac Cullogh & Pitts 43 Hebb 49 Rosenblatt 58	modèle de neurone loi d'adaptation perceptron
1960 - 1980 transition & déclin	Widrow-Hoff 58 Minsky & Papert 69	adaline limites aux perceptrons
1980 - … renouveau	Hopfield, 82 Kohonen 72, 82, 84 Rumelhart, Le Cun 86 (Werbos 74)	réseaux dynamiques auto-organisation rétro-propagation

Quelques dates, quelques noms

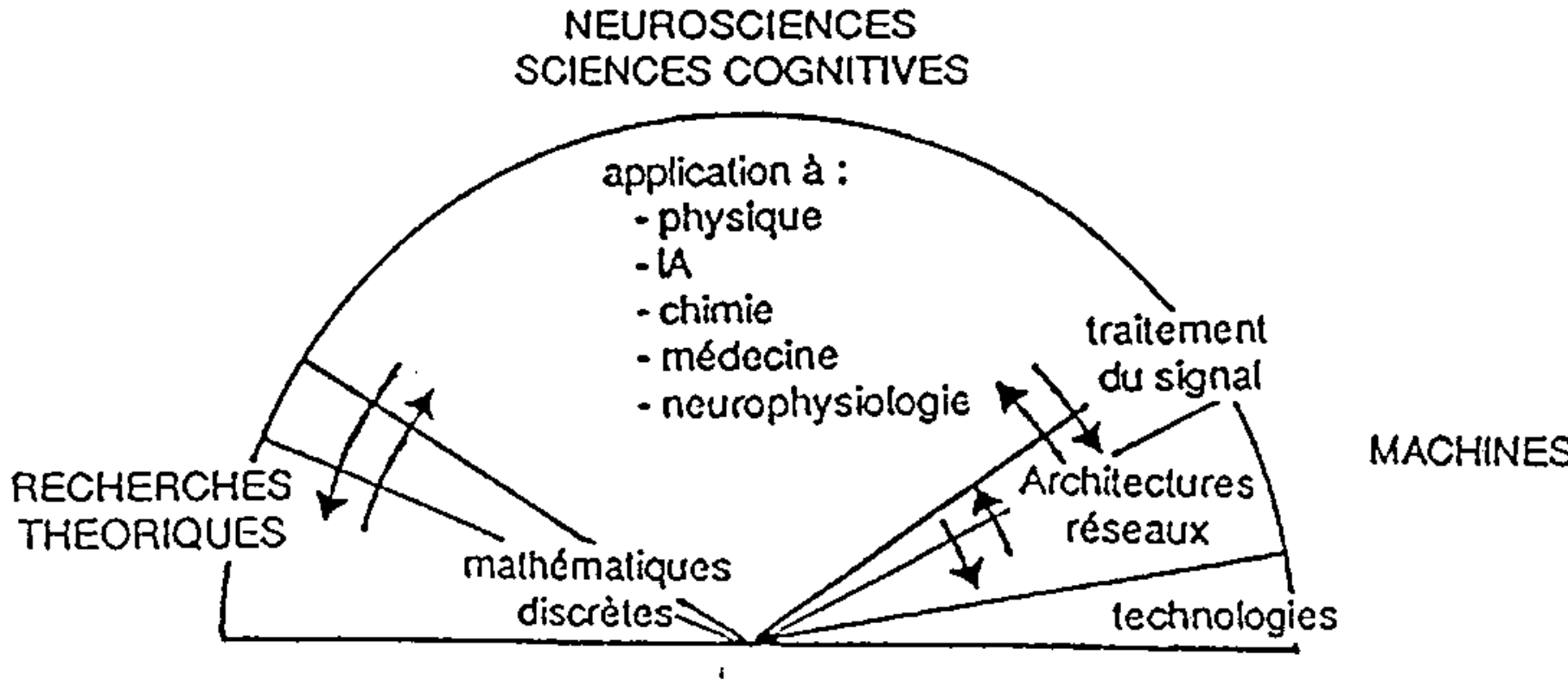
- Années 40, recherche opérationnelle, codage
- RAND Corporation (46-54), Von Neumann, automates

- Besoin de proposer des machines et/ou des modèles reproduisant des fonctions des cerveaux humains (calcul, mémoire, comportement intelligent)

- A la suite des pionniers de l'informatique (Turing)
- A la suite des chercheurs en intelligence artificielle (limites de l'IA symbolique)
- Chercheurs de différents domaines : Hérault, 70, Amari, 72, Von der Marlsburg, 73, Little, 74, Grossberg, 76…
- En France, journées NSI, Neuro-Nîmes

- Pendant ce temps là, progrès de la puissance de calcul

Pluridisciplinarité

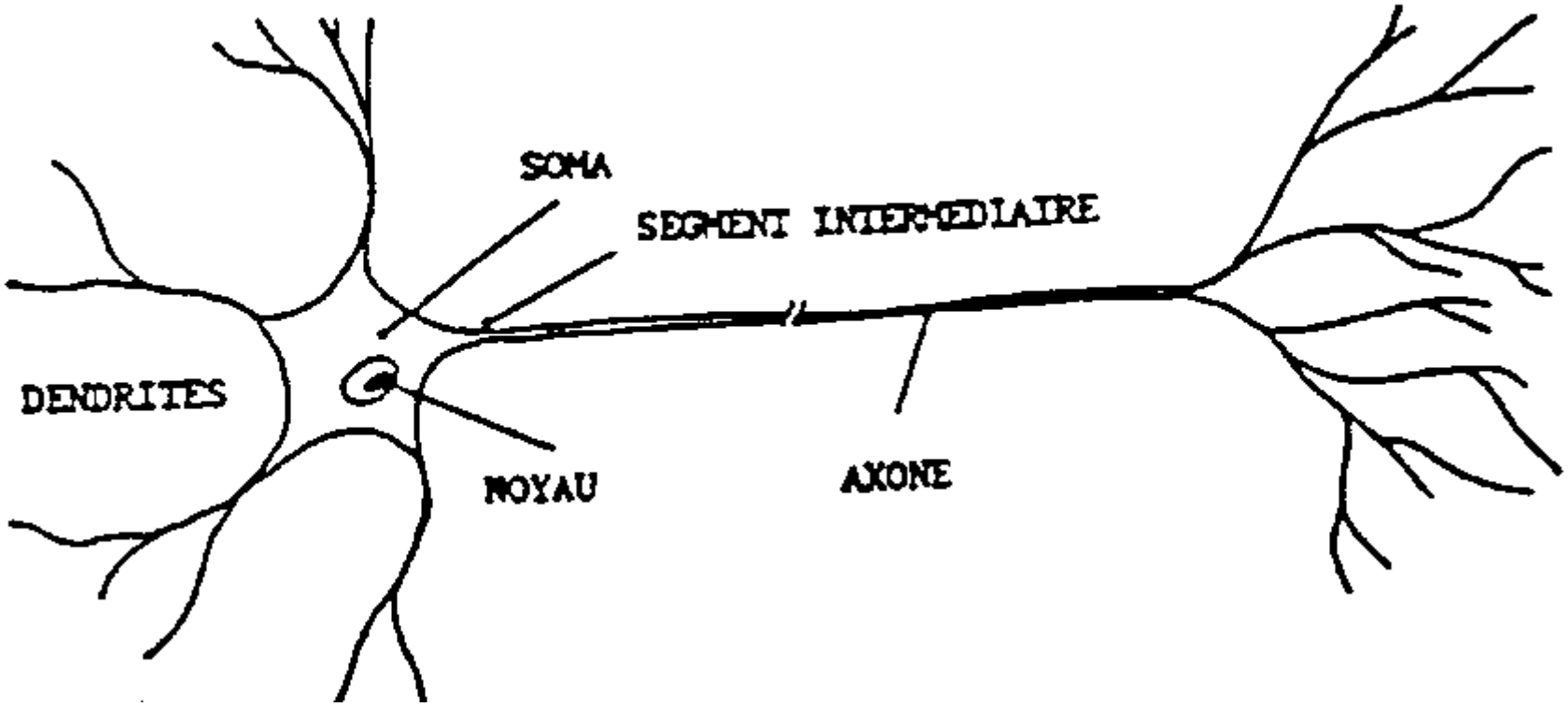


Amy et Décamp, Rapport CNRS 1987

Réseaux de neurones et neurosciences

- 10^{11} neurones, 10^{15} connexions
- On sait mesurer et enregistrer l'activité électrique d'un neurone ou d'un ensemble de neurones
- Propriétés collectives de réseaux, qui mettent en jeu des relations excitatrices ou inhibitrices
- Dans le cerveau, on distingue des sous-ensembles, des connexions internes et externes, des entrées et des sorties, réalisant certaines tâches
- Les recherches sur le fonctionnement du cerveau ont fait d'énormes progrès (exposition Cité des Sciences, 2002), tant du point de vue de la compréhension du fonctionnement qu'à l'échelle cellulaire et membranaire

Un neurone (d'après Jutten)



Des neurones (d'après Jutten)

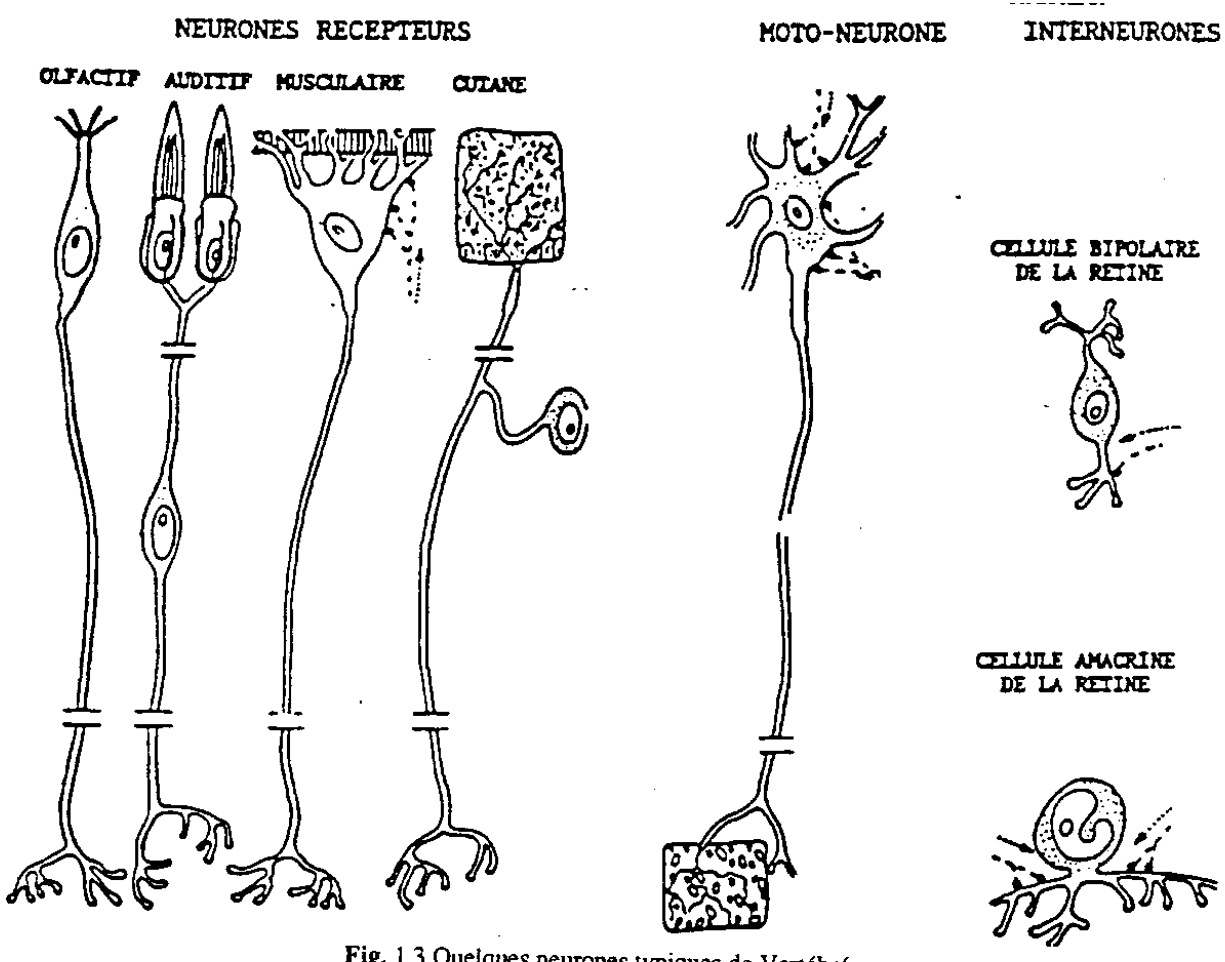
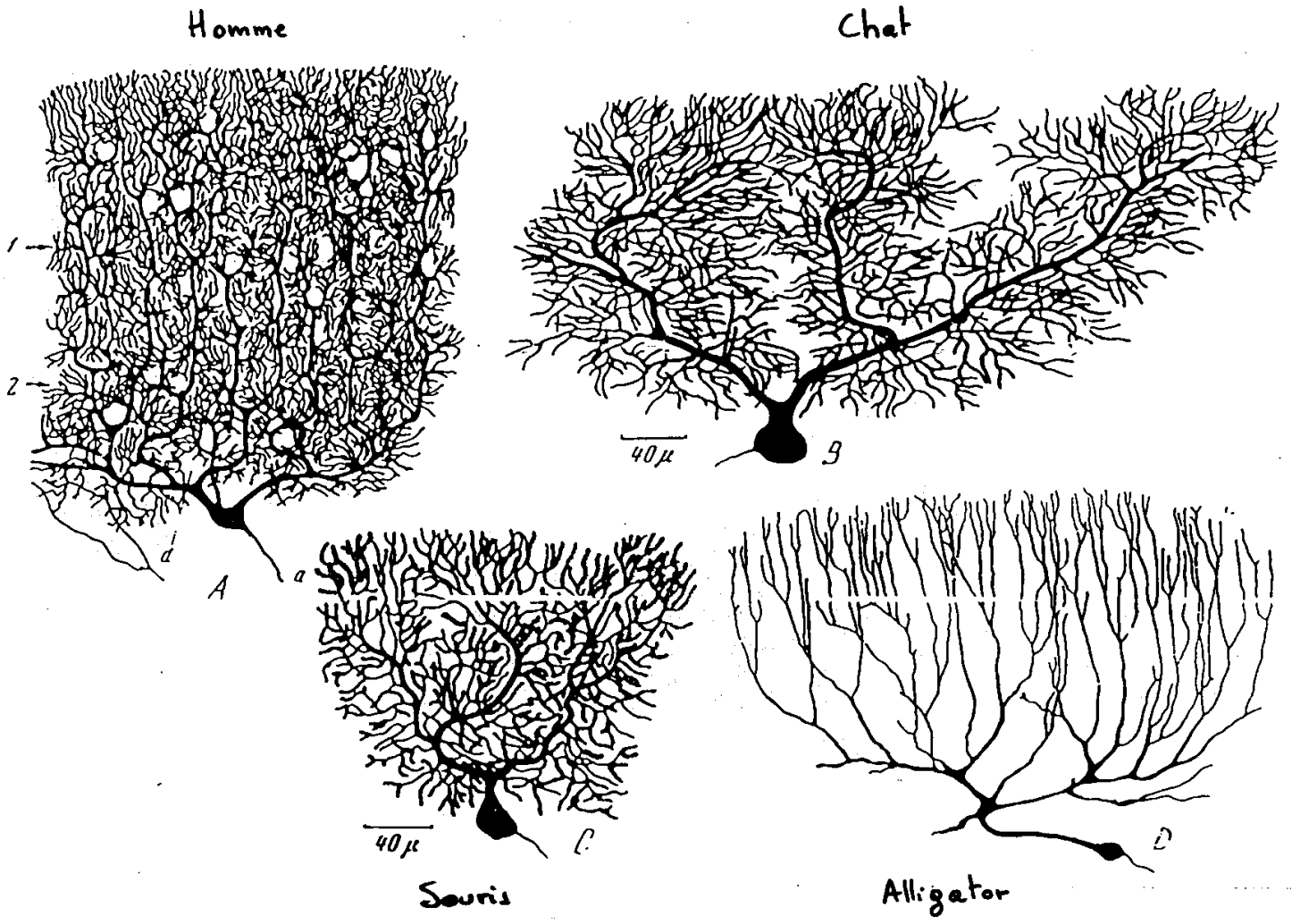


Fig. 1.3 Quelques neurones typiques de Vertébrés.

Cellules de Purkinie dans le cervelet (d'après Rospars)



Trois grandes caractéristiques du cerveau vu comme une « machine »

- Apprentissage
  - Adaptation
  - Plasticité synaptique
  - Reconversion
- Robustesse
  - Résistance à l'imprécision des entrées
  - Résistance à la détérioration
  - Distribution des informations
- Parallélisme
  - Interactions locales
  - Propriétés globales
  - Simultanéité du traitement
- Ces caractéristiques inspirent la construction des réseaux de neurones formels

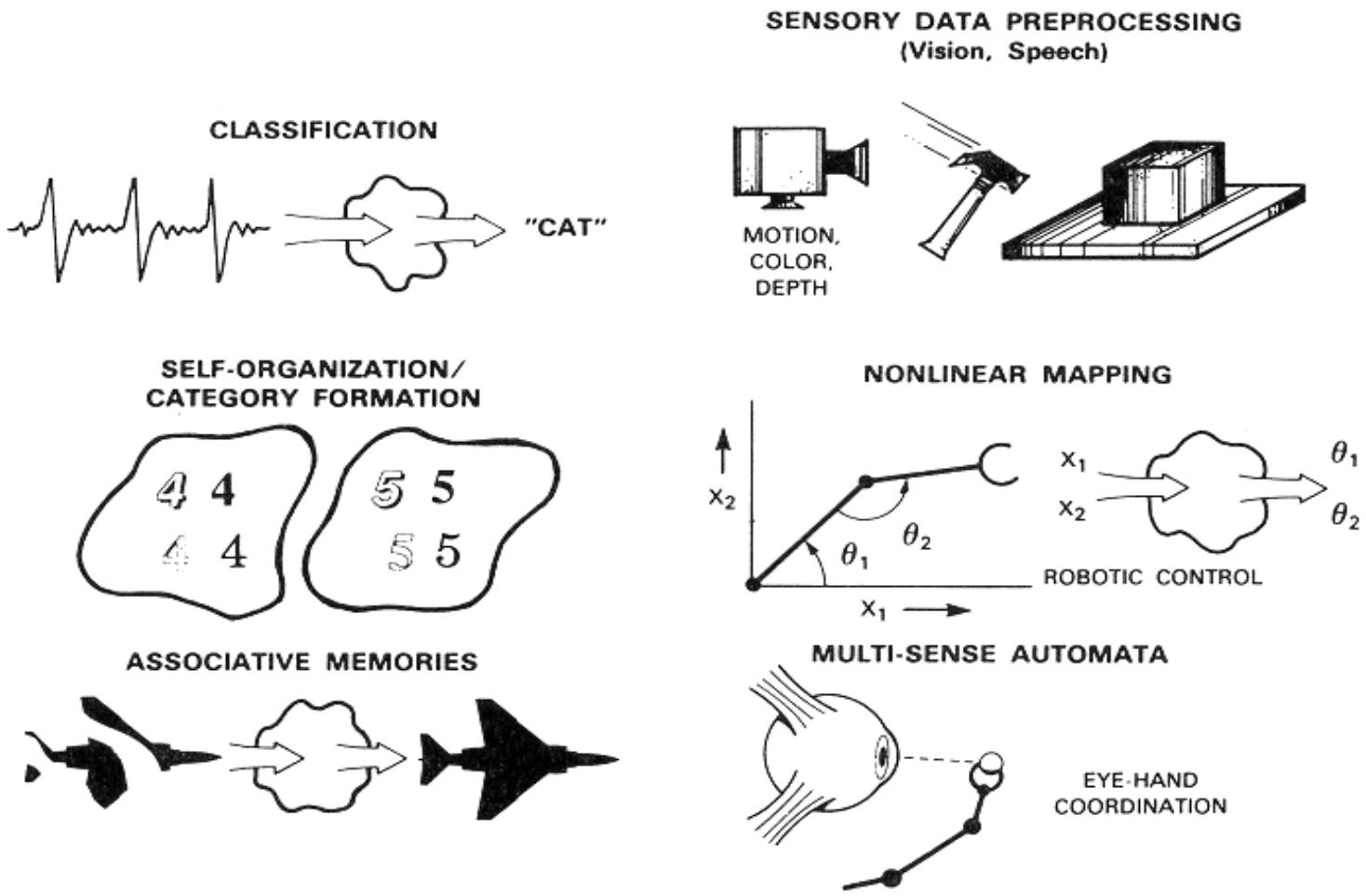
Ordinateur vs. cerveau

- Ordinateur de Von Neumann's
  - déterministe
  - séquences d'instructions
  - tâches répétitives
  - vitesse élevée
  - programmation
  - unicité des solutions
- ex: produit matriciel

- Cerveau (humain)
  - parallélisme
  - adaptation (apprentissage)
  - vitesse lente
  - comportement flou
  - différentes façons d'aborder un problème
  - différentes solutions
- ex: reconnaissance de visages

Il se révèle extrêmement difficile de construire un ordinateur qui imite réellement un cerveau

Exemples de tâches: faciles pour le cerveau, difficiles pour un ordinateur...



extrait de: DARPA neural network study, 1988

Difficultés

- La miniaturisation a une limite (au moins un électron par bit d'information)
- Difficultés de gérer les transferts
- Difficultés du fonctionnement en parallèle (mais gros progrès)
- Les ordinateurs sont câblés (architecture fixe)
- Ils ne reconnaissent rien si un bit est erroné, si une connexion est rompue

- Leur complexité est infiniment moindre que celle de n'importe quel animal

Un MX 1/16 fonctionne avec $120*10^6$ connexions actives par seconde

Un petit cafard met en jeu 10^9 connexions par seconde

Ordinateurs parallèles (La Recherche 88)

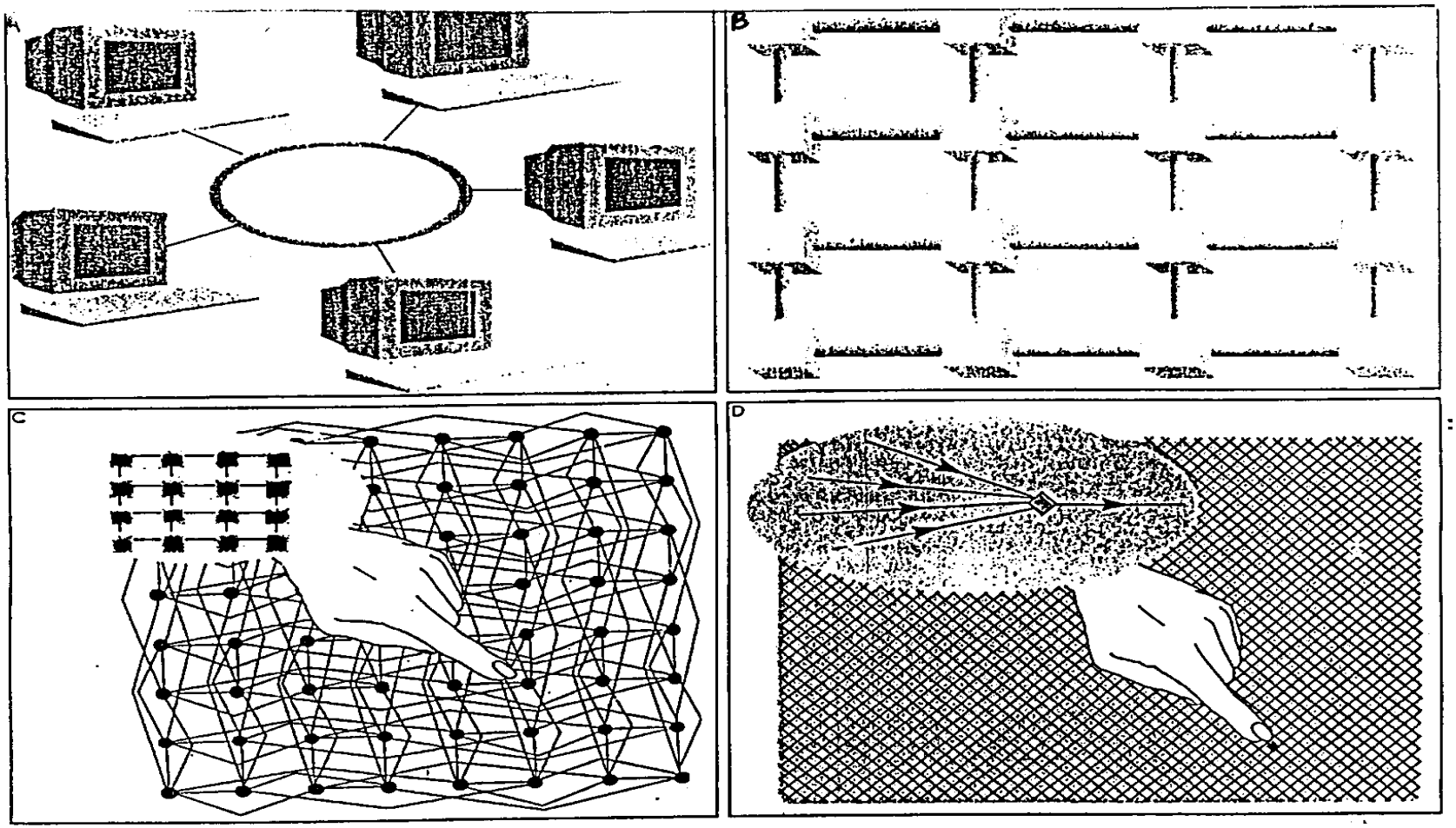


Figure 6. A mesure que le nombre de connexions et de processeurs augmente, le parallélisme devient massif, et le rôle du réseau d'interconnexions en tant que mémoire devient prédominant. Dans un système multiprocesseur classique (A), toute l'information est, bien sûr, contenue dans les processeurs, le réseau ne faisant qu'assurer leurs communications. Mais dans les grilles de « transputers », microprocesseurs de conception très récente (B), l'importance de l'information contenue dans les liens se fait déjà sentir : pour utiliser une telle architecture, il faut non seulement savoir programmer chaque transputer, mais aussi utiliser au mieux les énormes débits d'informations autorisés par chaque lien. Avec la Connection Machine et son réseau à deux niveaux, dont on voit une portion dessinée « à plat » en C, on franchit un seuil qualitatif : on atteint le parallélisme massif. Le stade ultime du parallélisme — le parallélisme que l'on pourrait qualifier de total — est atteint avec ce qu'on appelle les réseaux de neurones formels (D), où toute l'information est contenue

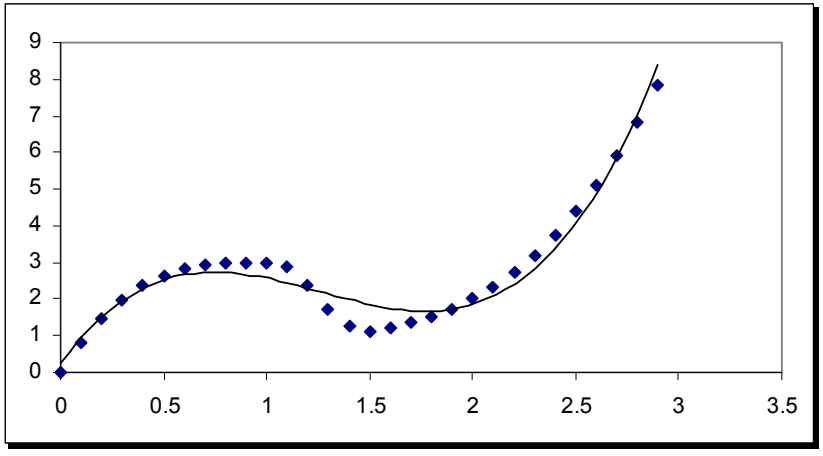
Les réseaux de neurones artificiels

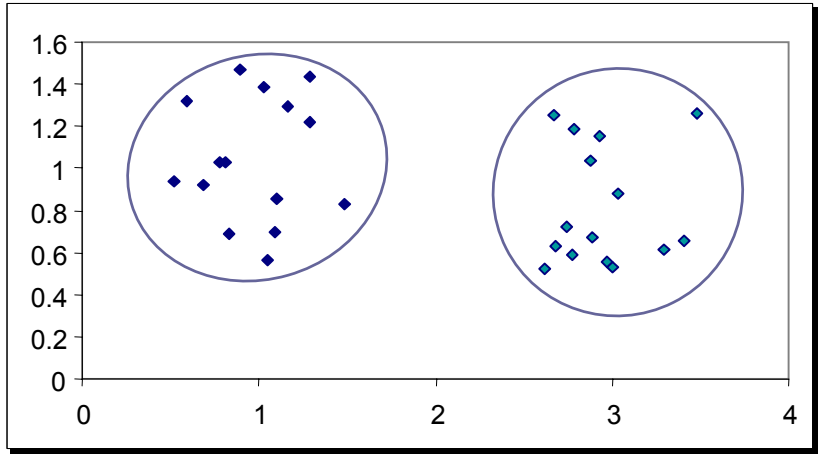
- Les informaticiens mettent au point des puces dédiées (en particulier pour les systèmes embarqués), des architectures d'ordinateurs de plus en plus complexes et performantes.
- En même temps, on s'est orienté vers la mise au point de modèles, d'algorithmes, qui se présentent sous la forme de programmes utilisables dans les ordinateurs classiques (de plus en plus rapides).
- On s'est éloigné assez radicalement des neurosciences (bien que certains laboratoires continuent comme RIKEN, à Tokyo).
- Les contacts restent étroits avec les sciences cognitives.
- Le domaine s'est considérablement rapproché des statistiques
- Il s'agit d'un ensemble d'outils permettant de résoudre des problèmes divers.
- **Ils sont essentiellement NON LINEAIRES et ADAPTATIFS**

Exemples de tâches

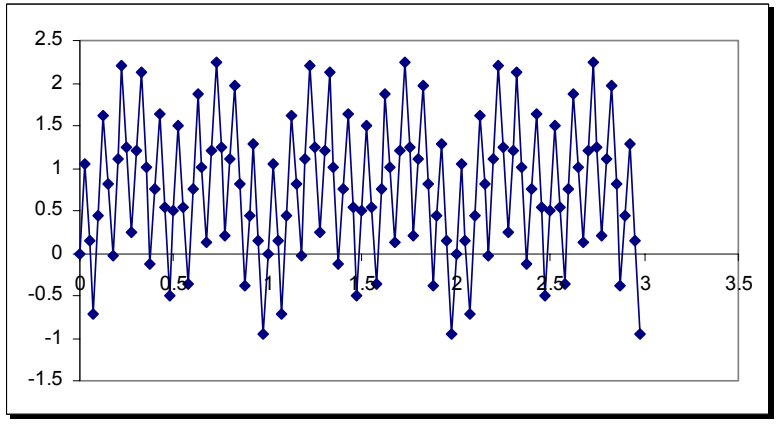
- Approximation de fonctions et de distributions
- Analyse exploratoire de données (fouille de données, visualisation)
- Régression non linéaire
- Identification et prévision de séries temporelles
- Classification établissement de topologie, scoring
- Reconnaissance de formes, de visages, d'écriture
- Contrôle de procédé
- Filtrage adaptatif, etc.…etc…

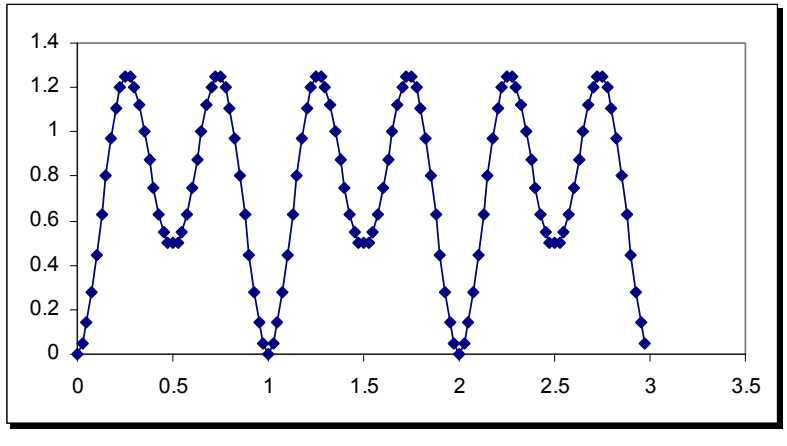
Régression, classification,



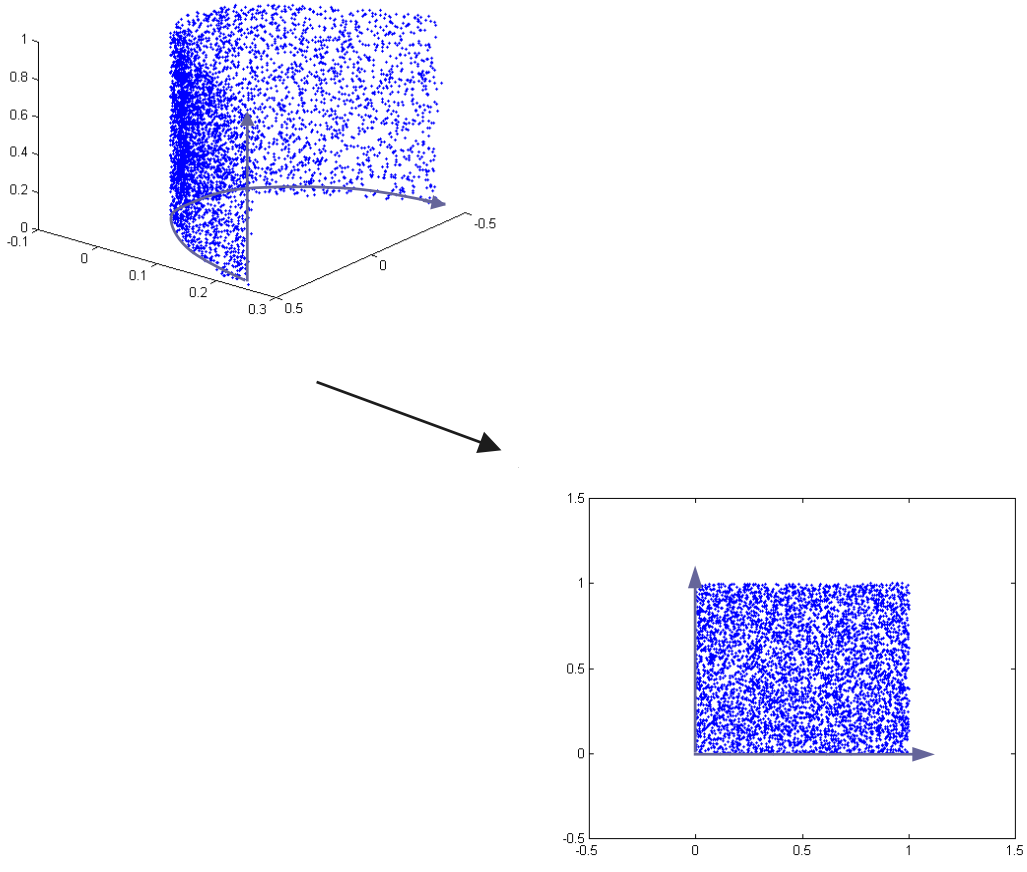


Filtrage adaptatif





Projection non linéaire



Introduction

Les premiers modèles

Les réseaux

Modèle de Hopfield

Le perceptron multicouches

Algorithme de Kohonen

Conclusion

Premiers modèles

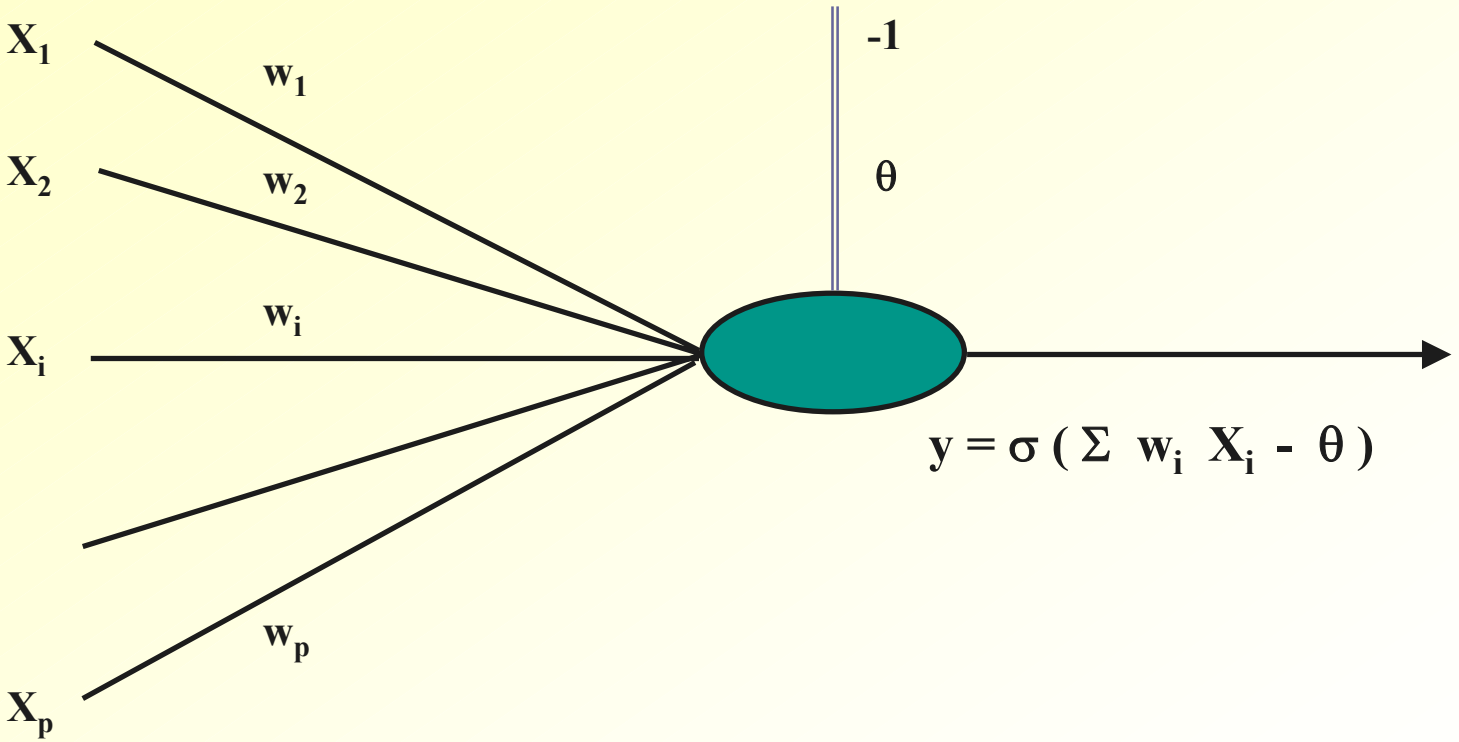
- Le neurone formel
  Mac-Cullogh & Pitts (1943)

- Le perceptron simple
  (Rosenblatt, 1958, Minsky-Papaert, 1969)

- L'ADALINE
  (Widrow-Hoff, 1958)

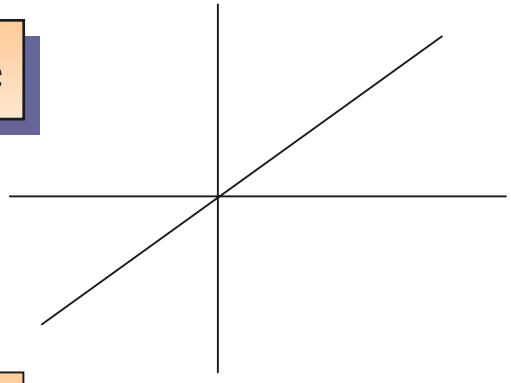
Le neurone formel

- Schématisation du neurone biologique

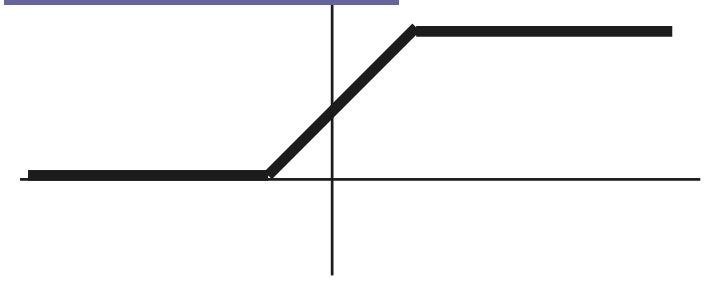


Fonctions d'activation

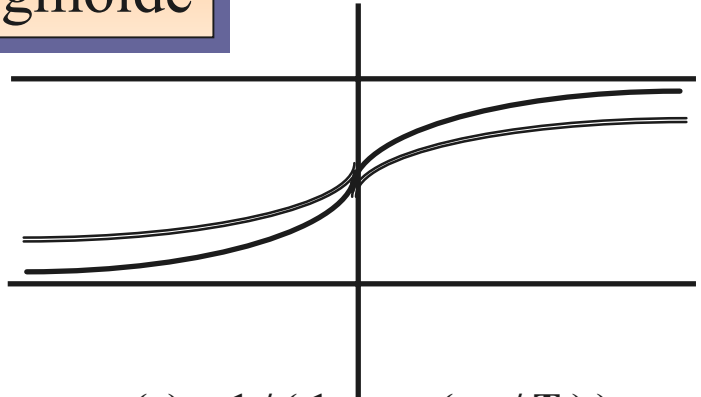
Linéaire



Par morceaux



Sigmoïde

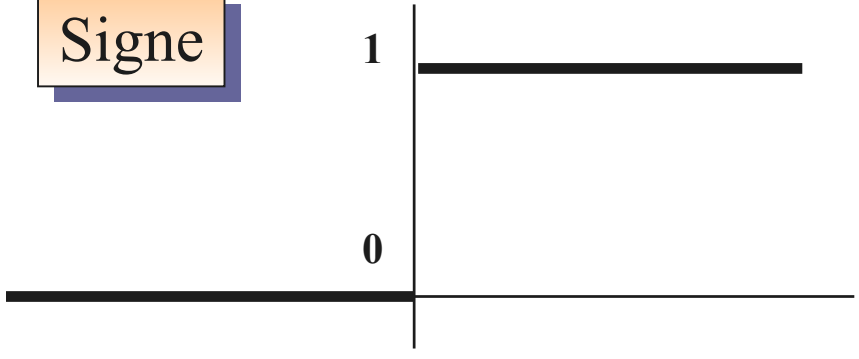


$\sigma(x) = 1 / (1 + \exp(- x / T))$

Quand T tend vers 0

s tend vers la fonction *signe*

Signe



Si $\Sigma w_i x_i \geq \theta$, $y = +1$

Si $\Sigma w_i x_i < \theta$, $y = 0$ ou -1

Neurone probabiliste

- On veut favoriser l'évènement (y = + 1), lorsque $\Sigma\, w_i\, x_i \geq \theta$
- y est une variable aléatoire binaire
- On prend par exemple,

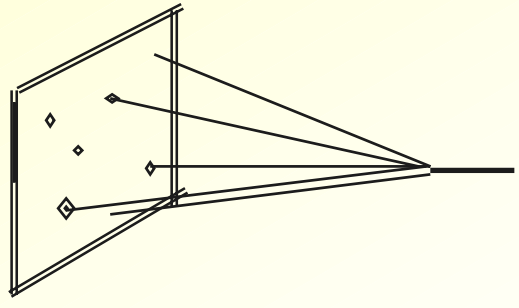
$$\mathbf{Prob}(\, y = +1) = \frac{1}{1 + exp(-(\Sigma w_i x_i - \theta\,) / T\,)}$$

- Le paramètre T (positif) est la température

- Quand l'argument est positif, Prob(y = + 1) > ½
- C'est ce genre de modèle qui est utilisé dans les machines de Boltzmann (Ackley, Hinton et Sejnowski, 1985)

Le perceptron simple (Rosenblatt, 1958)

- Algorithme supervisé (au sens de l'IA)
- Fonction non-linéaire (fonction signe)
- Objectif: classification (binaire)



$$\hat{y} = \sigma\left(\sum_j w_j x_j - \theta \right)$$

$$\sigma(x) = sign(x)$$

- 2 classes A et B

$$\begin{cases} \hat{y} = +1 & \text{si } x \in A \\ \hat{y} = 0 & \text{si } x \in B \end{cases}$$

- Il s'agit de déterminer les paramètres w_j, tels que les réponses soient correctes pour tous les objets.
- Apprentissage = estimation
- Mais processus itératif

Le perceptron (l'algorithme)

- APPRENTISSAGE

  Au temps 0, les paramètres sont aléatoires

  Au temps t, un objet est présenté

  Si y est correct, pas de changement

  Si y = -1 (au lieu de +1), $w_j(t+1)=w_j(t) + \varepsilon\, x_i$

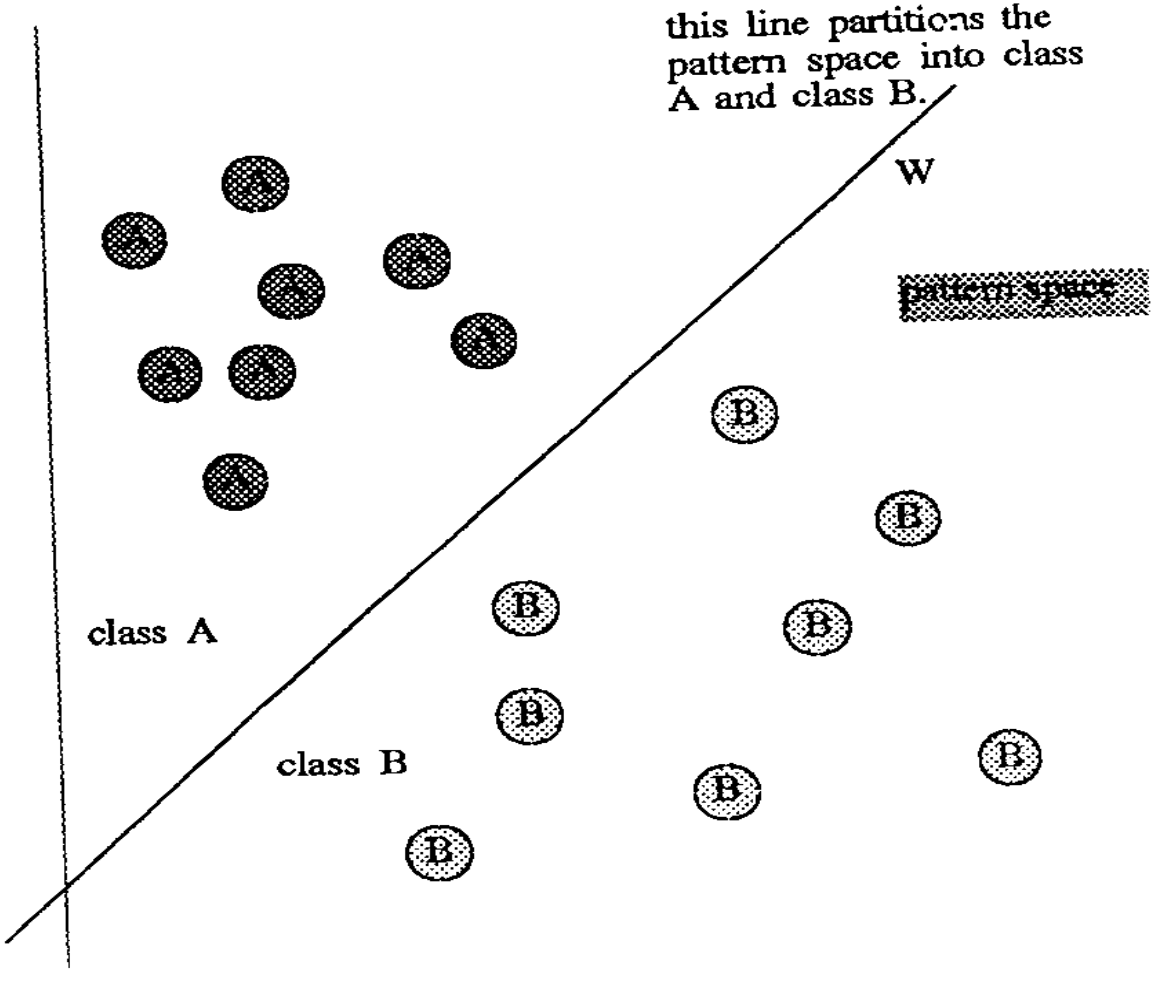
  Si y = +1 (au lieu de -1), $w_j(t+1)=w_j(t) - \varepsilon\, x_i$

- Cela peut s'écrire :

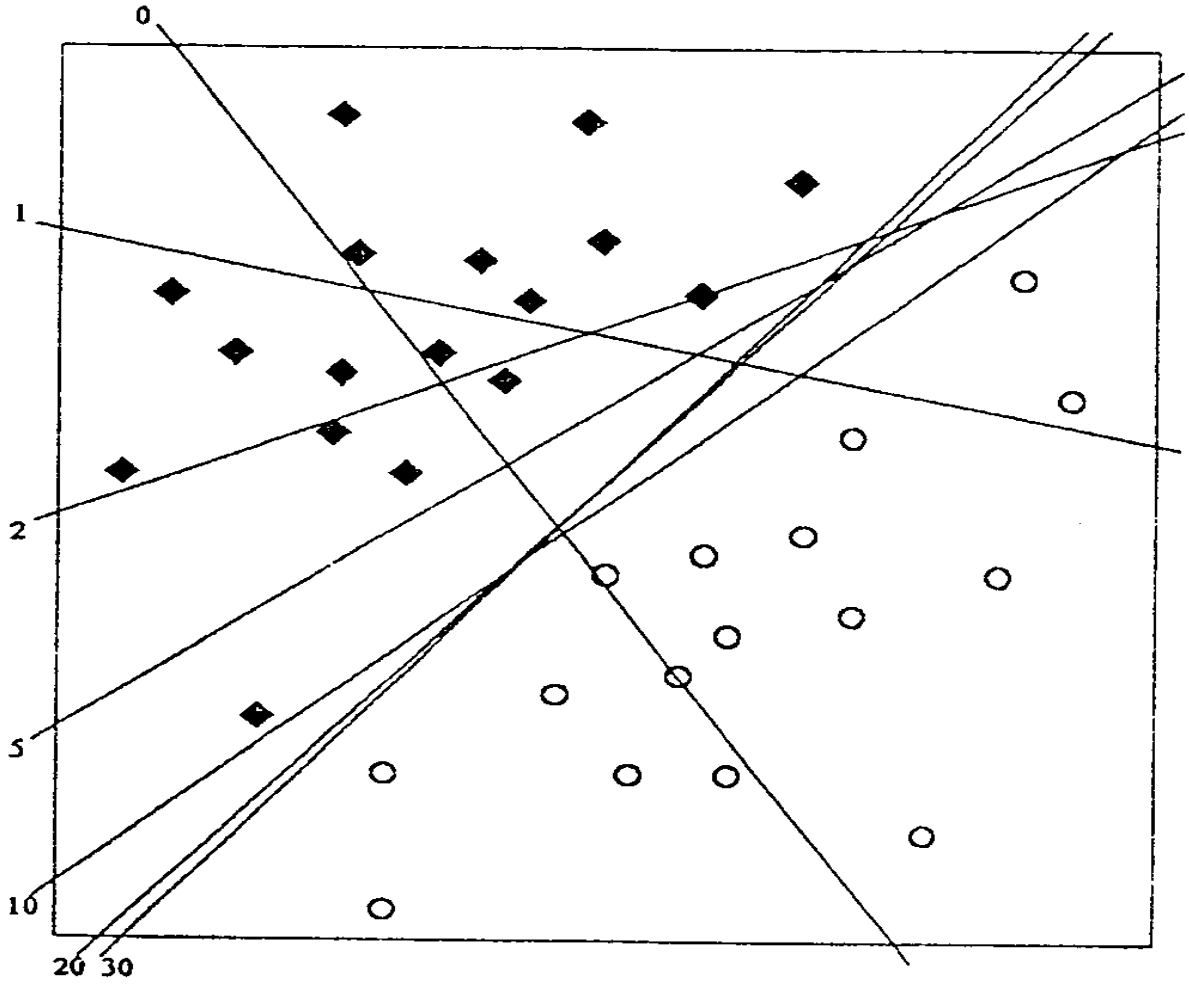
- $W(t+1) = W(t) + \varepsilon\, (d(t) - y(t))\, X(t)$

  où d est la sortie désirée, y la sortie calculée, $\varepsilon > 0$

Le perceptron simple



Le perceptron simple



Convergence

Si les 2 classes sont linéairement séparables (au sens strict), ce processus d'apprentissage est convergent, en un nombre fini d'étapes

Dém :

- On suppose les X normés, ce qui ne change rien
- Si l'entrée X est dans B, on la remplace par - X. Ainsi, pas d'erreur signifie : $W.X > 0$.
- L'hypothèse est que $\exists\ \delta$ et W^* (qu'on peut prendre normé) tels que pour toute entrée X, on ait

$$W^*.X > \delta$$

- On pose $g(W(t)) = W^*.W(t) / |W(t)| = \cos(W^*, W(t))$, et $g(W(t)) \leq 1$.

Convergence (suite)

A chaque changement, le numérateur

$$W^*.W(t+1) = W^*.W(t) + \varepsilon\, W^*.\, X(t)$$
$$\geq W^*.W(t) + \varepsilon\, \delta$$

- Après M changements,

$$W^*.W(M) \geq W^*.W(0) + \varepsilon\, M\, \delta$$

- Au dénominateur,

$$| W(t+1) |^2 = | W(t) |^2 + 2\, \varepsilon\, W(t).\, X(t) + \varepsilon^2\, |X(t)|^2$$

$$\leq |W(t)|^2 + \varepsilon^2\, |X(t)|2$$

puisque il y a changement quand $W(t).X\,(t) < 0$.

Convergence (Suite et fin)

- D'où, $|W(M)|^2 \leq |W(0)|^2 + \varepsilon^2 M$

- On pose $W(0) = 0$

Donc, $\varepsilon M \delta/\varepsilon\sqrt{M} \leq g(W(M)) \leq 1$,
d'où M est fini, majoré par $1/\delta^2$.

- Donc si W^* existe, au bout d'un nombre fini d'étapes, $W(t)$ reste fixe. Il y a donc convergence

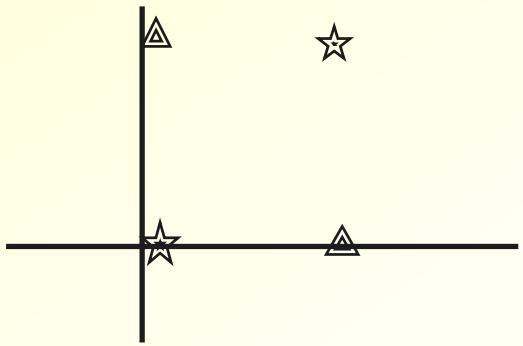
Problème

- MINSKY et PAPERT (1969) ont prouvé que le Perceptron ne peut pas résoudre une grande classe de problèmes non linéaires comme par exemple

- LE PROBLEME XOR :

  Séparer les points (1,1) et (0,0)

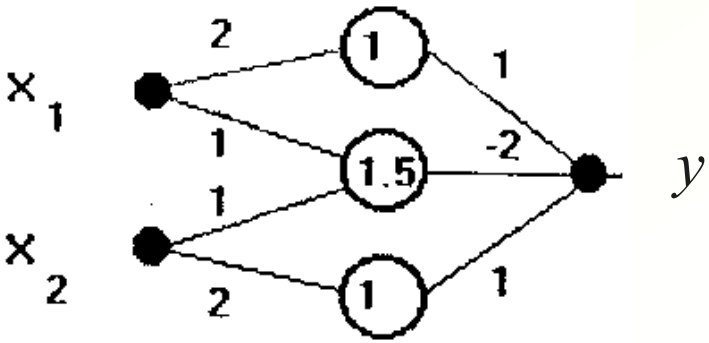
  des points (1,0) et (0,1)



- **Pour cela, on doit rajouter des couches cachées**

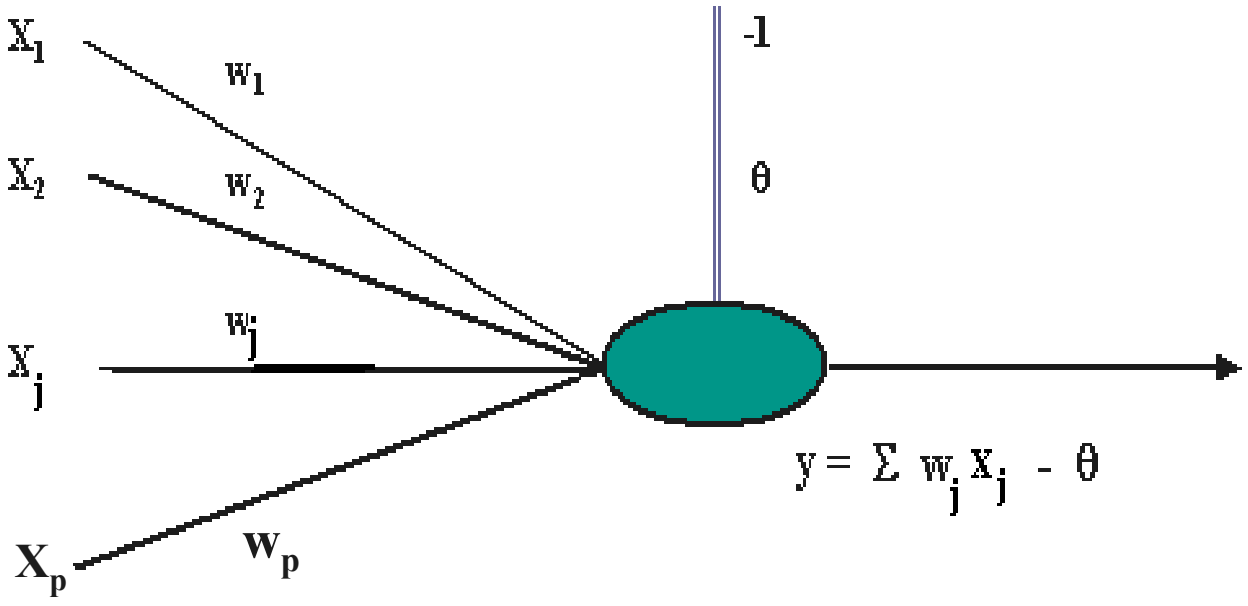
On a $y = 1$ pour (1,0) et (0,1)

$y = 0$ pour (1,1) et (0,0)



Le modèle ADALINE (Widrow-Hoff, 1958)

ADAptative LINEar model



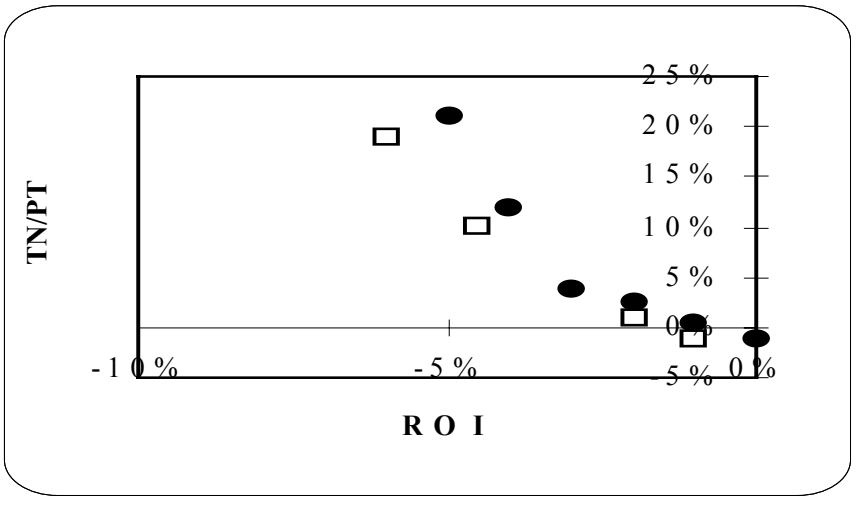
Il s'agit d'un simple modèle de régression linéaire

Neurone formel linéaire et régression

- Les entrées sont les variables explicatives
- Le seuil est la constante du modèle (ou son opposé)
- Le calcul des paramètres (ou poids) se fait par n'importe quelle méthode
  - Calcul direct
  - Gradient déterministe
  - Gradient stochastique
- Le minimum de l'erreur est unique, le fonction d'erreur est convexe

Neurone formel linéaire
= régression linéaire

Exemple (très) simple, 10 entreprises, 2 ratios (ROI, TN/TP)



$$y_i = \alpha + \beta_1 (ROI)_i + \beta_2 (TN/PT)_i + \varepsilon_i$$

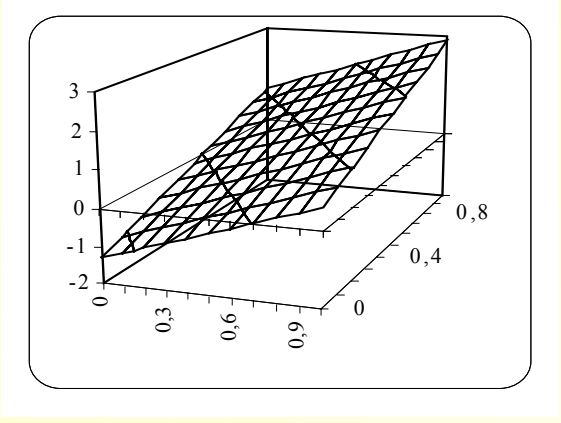
$$Rmse = \sqrt{\frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{n}}$$

Les données et les résultats

CAS (1)	ROI (2)	TN/PT (3)	Classement (4)	Linéaire (5)	2NL-1L (6)	2NL-1NL (7)	2NL-1NL-M (8)	3NL-1L (9)	3NL-1NL (10)
1	0,167	1,000	1,000	0,963	0,957	0,970	0,968	0,999	0,968
2	0,000	0,909	0,000	0,403	-0,168	0,001	0,054	0,007	0,002
3	0,333	0,591	1,000	0,591	1,043	0,996	0,997	1,241	0,997
4	0,250	0,500	0,000	0,227	0,332	0,046	0,023	0,074	0,051
5	0,500	0,227	1,000	0,304	0,613	0,936	0,881	0,762	0,931
6	0,667	0,091	0,000	0,441	0,356	0,077	0,185	0,419	0,070
7	0,667	0,159	1,000	0,568	1,054	0,988	0,947	1,084	0,986
8	0,833	0,068	1,000	0,789	1,023	0,953	0,897	1,019	0,951
9	0,833	0,000	0,000	0,662	0,186	0,022	0,091	0,289	0,013
10	1,000	0,000	1,000	1,052	1,003	0,974	0,945	1,114	0,971
Rmse				**0,42**	**0,21**	**0,04**	**0,09**	**0,20**	**0,04**

Régression linéaire

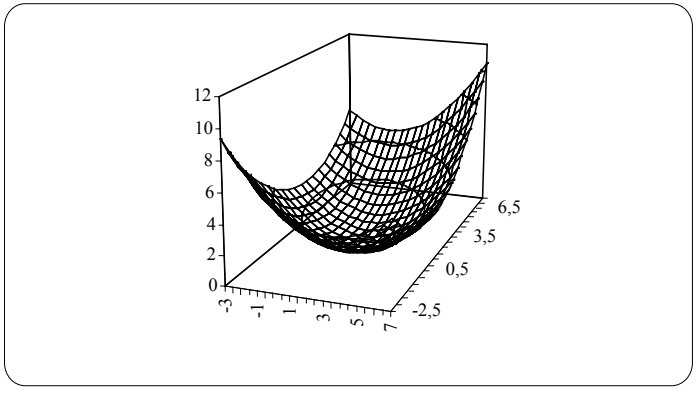
- Exemple (suite), fonction réponse



$$\hat{\theta} = (X'X)^{-1} X'Y$$

La solution est calculée en un coup par une formule déterministe

- Fonction d'erreur:



$$E = \frac{1}{2} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

$$\hat{y}_i = \sum_{j=0}^{2} \hat{w}_j x_{ji}$$

Gradient vs. gradient stochastique

- Gradient:
  - Fonction d'erreur sur tous les exemples

$$E = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

  - ajustement des poids

$$\nabla w_j = -\varepsilon \frac{\partial E}{\partial w_j}$$

$$= \varepsilon \sum_{i=1}^{N}(y_i - \hat{y}_i)x_{ji}$$

- Gradient stochastique
  - Fonction d'erreur sur un exemple

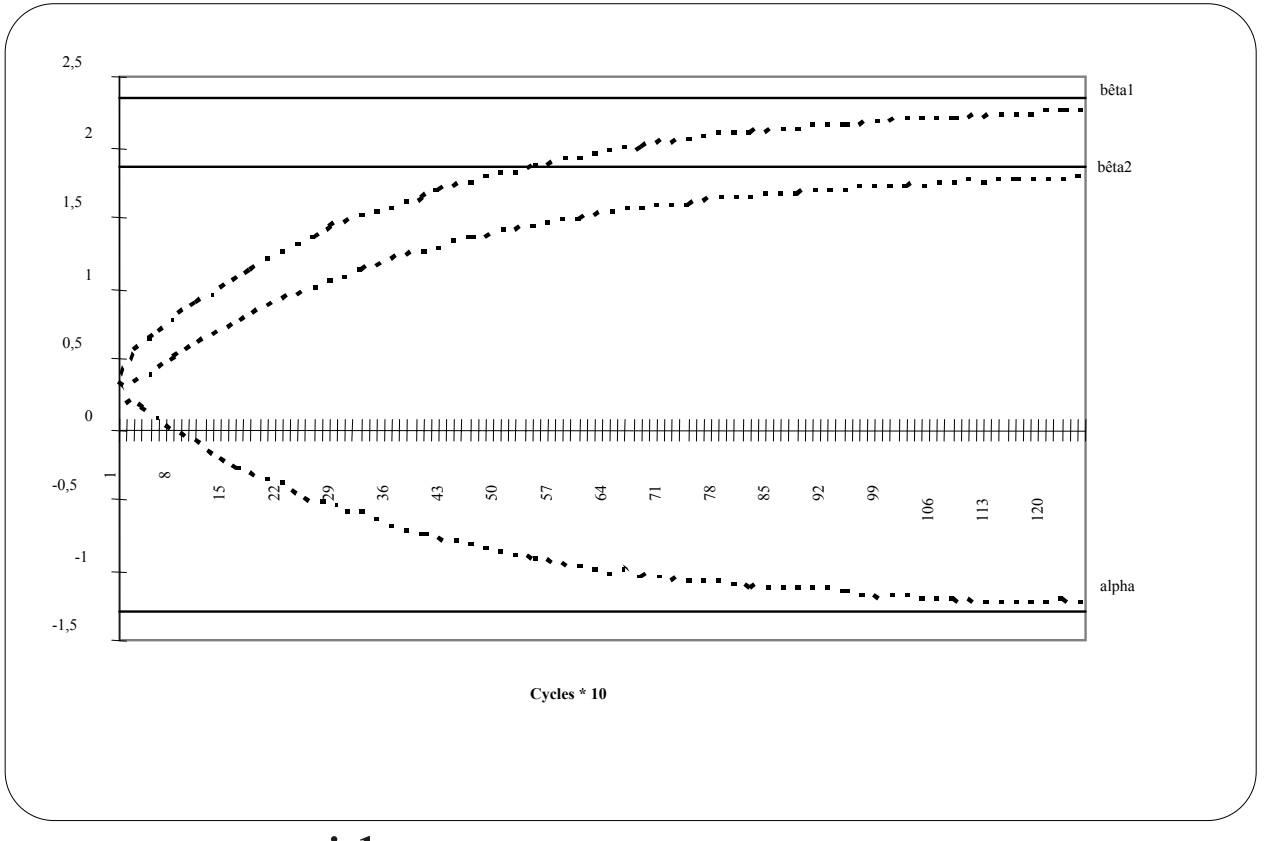
$$E_i = \frac{1}{2}(y_i - \hat{y}_i)^2$$

  - ajustement des poids

$$\nabla w_j = -\varepsilon \frac{\partial E_i}{\partial w_j}$$

$$= \varepsilon(y_i - \hat{y}_i)x_{ji}$$

Estimation des paramètres par l'ADALINE



Convergence rapide

Introduction

Les premiers modèles

Les réseaux

Modèle de Hopfield

Le perceptron multicouches

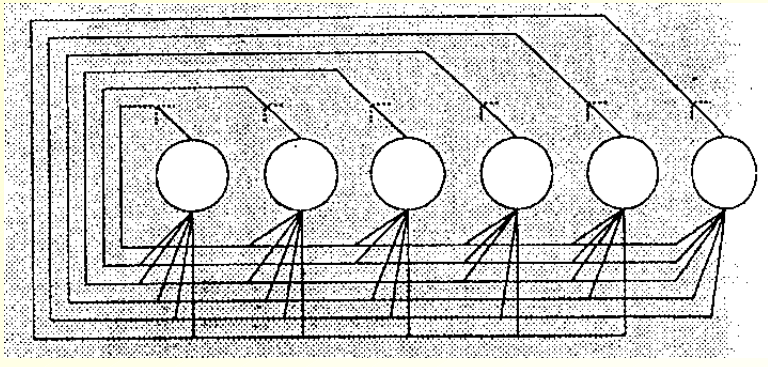
Algorithme de Kohonen

Conclusion

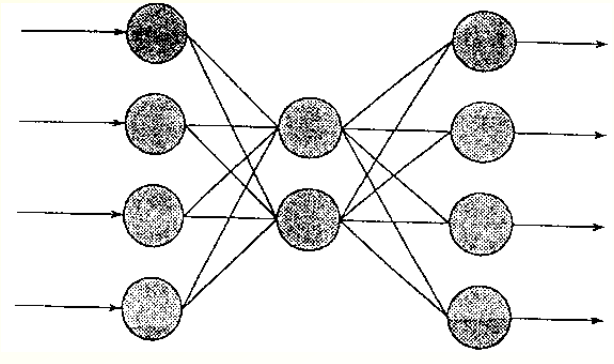
Architecture

- On va mettre ces neurones formels en réseau

- Monocouche
  complètement connecté



- Réseaux en couches
  sans connexions directes



- Toute autre architecture

Apprentissage

- On remarque que même dans les modèles simples précédents, la construction du réseau se fait par apprentissage
- Généralement, l'architecture (nombre de couches, nombre de neurones) est fixée, et l'apprentissage consiste à calculer progressivement les valeurs des connexions w_{ij}
- Les règles d'apprentissage s'inspirent de la règle de Hebb 1943

Règle de Hebb

Il y a renforcement d'une connexion lorsque les deux neurones qu'elle relie sont simultanément excités.

- Apprentissage supervisé
- Apprentissage non supervisé
- Les règles de modifications des connexions sont locales (dans le temps et dans le réseau).
- Le réseau peut être construit **ssi** l'algorithme d'apprentissage est convergent

Vocabulaire (RN/Statistique) (d'après Hastie et Tibshirani, 1994)

Les réseaux de neurones sont des modèles

- Réseaux de neurones
  - Apprentissage
  - Poids, efficacité synaptique
  - Connaissance
  - Apprentissage supervisé
  - Classification
  - Apprentissage non supervisé
  - Clustering
  - Réseau de neurones
  - Ensemble d'apprentissage

- Statistique
  - Estimation
  - Paramètres
  - Valeur des paramètres
  - Régression/classification
  - Discrimination/classement
  - Estimation de densité
  - Classement/typologie
  - Modèle
  - Echantillon

Introduction

Les premiers modèles

Les réseaux

Modèle de Hopfield

Le perceptron multicouches

Algorithme de Kohonen

Conclusion

Mémoire (humaine/ordinateur)

Mémoire humaine	Mémoire d'ordinateur
Reconstitution à partir d'une information partielle	Ne reconnaît qu'un mot écrit exactement
Pas de localisation, pas de neurone « grand-mère	L'information est localisée à une certaine adresse
Pas de déroulement, pas de circuit électrique pour comparer	Recherche par déroulement, reconnaissance par comparaison

- Nombreux travaux pionniers Amari, Kohonen

Modèle de Hopfield (1982)

- But : **mémoriser des formes, des motifs**

- Constitution d'une mémoire
  - distribuée (contenue par l'ensemble du réseau)
  - associative (permettant le rappel à partir d'une information partielle ou bruitée)
  - fabriquée par apprentissage

Réseau complètement connecté (mais peut être incomplet)

- *N* neurones binaires, totalement connectés
- Etat *S* dans $\{ -1, +1 \}^N$
- Connexions C_{ij}, symétriques, avec $C_{ii} = 0$
- Dynamique synchrone ou asynchrone

$$S_i\ (t+1) = \text{signe} (\Sigma\ C_{ij}\ S_j\ \ (t))$$

Modèle de Hopfield

- En général, connexions de HEBB
- Objets à mémoriser S^1, S^2, ..., S^m, ..., S^p (de {-1, +1} ^{N}

$$C_{ij} = \Sigma\ S_i^m\ S_j^m$$

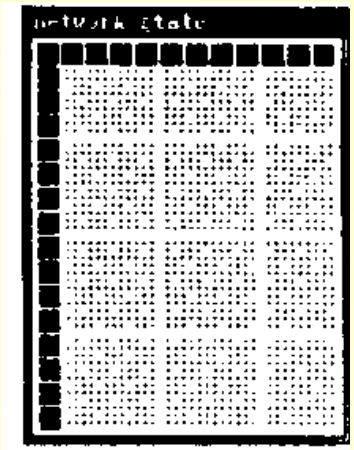
- D'autres choix sont possibles
- L'évolution correspond à la minimisation d'une fonction d'énergie (applications en optimisation)

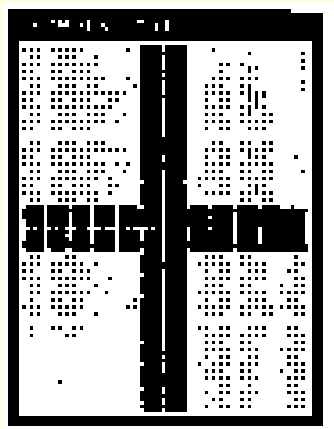
$$E(S(t) = -\frac{1}{2}\sum_{ij} C_{ij} S_j(t) S_i(t)$$

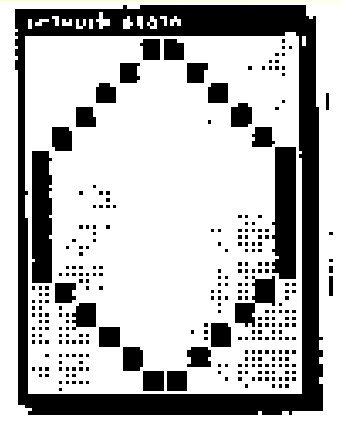
- **Pour de bonnes performances, il faut que le nombre d'objets à mémoriser soit inférieur à 0.14 N**
- Nombreux travaux (Hopfield, Gardner, Amit, etc...)
- Variantes pour améliorer les performances

Modèle de Hopfield

- Images mémorisées







- Image bruitée

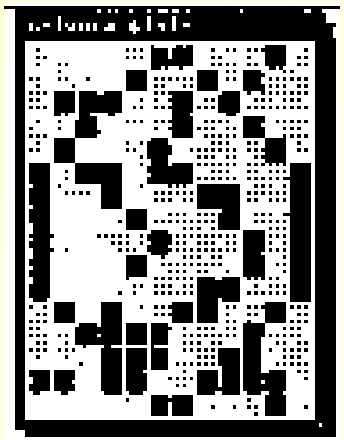
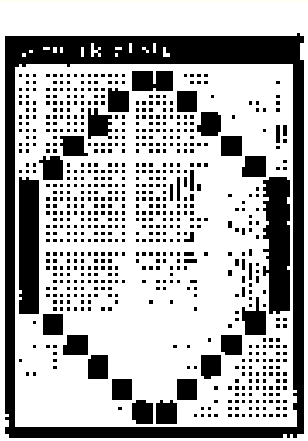
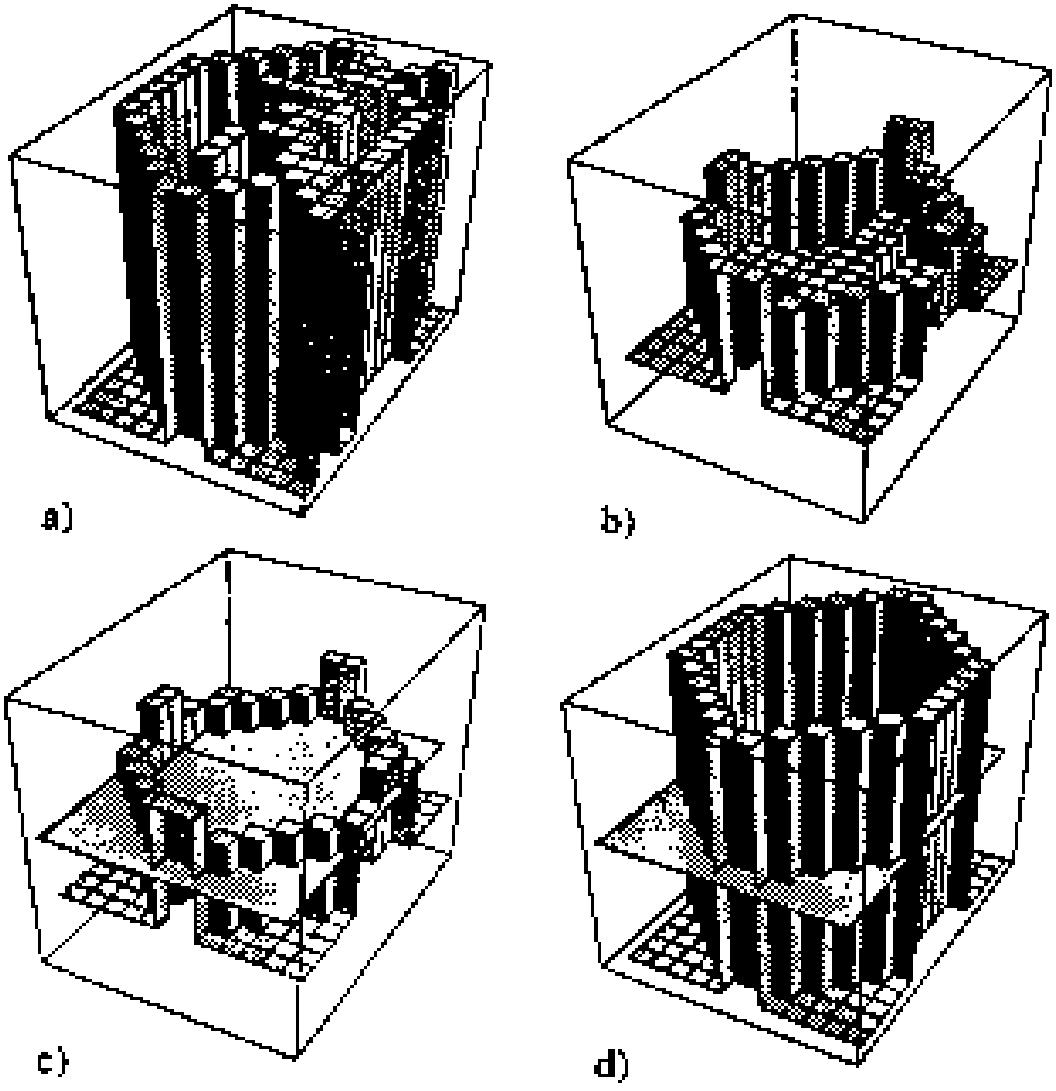


Image reconnue

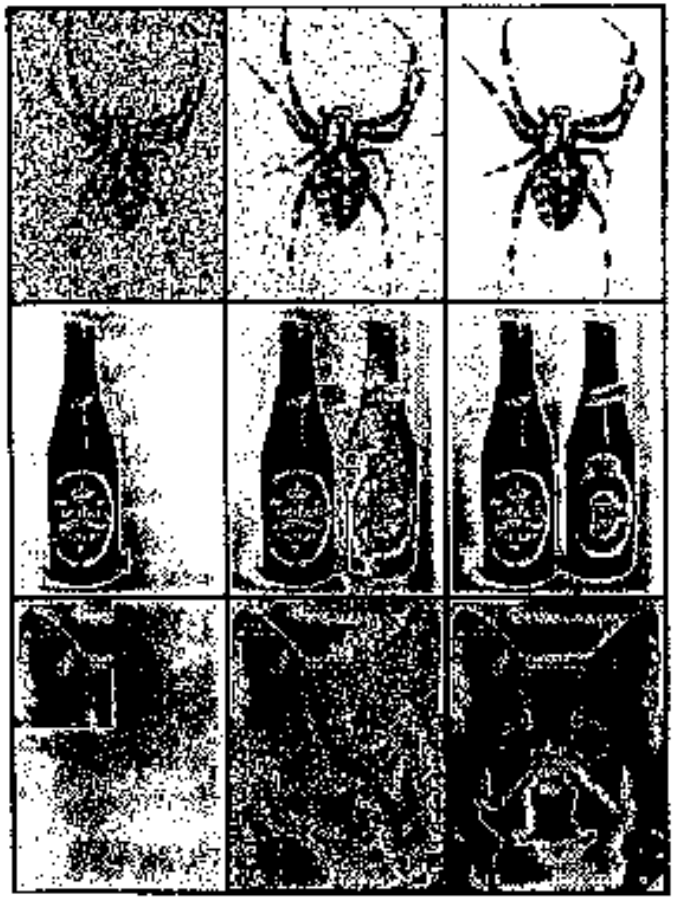


Le calcul



Modèle de Hopfield : exemple

- Hertz, Grogh et Palmer (1991)
- On ne stocke que 3 images
- On les reconnaît, à partir d'images partielles ou bruitées



Modèle de Hopfield : applications

- Reconnaissance de caractères (lecture automatique de caractères ou de chiffres manuscrits)
- Reconnaissance de séquences
- Mémorisation et reconnaissance de formes ou d'images
- La faible capacité de ces réseaux a conduit à d'autres modèles, comme des machines de Boltzmann (où la mise à jour des états des neurones est aléatoire)
- Application à la reconnaissance de profils de bateaux ou d'avions (Azencott)

Vocabulaire RN / physique

- Réseau de neurones
  - Cellule ou neurone
  - Actif ou inactif
  - Efficacité ou poids synaptique
  - Excitatrice
  - Inhibitrice
  - Seuil θ
  - Signal reçu (potentiel de membrane)

- Physique
  - Spin
  - Magnétisation +1 ou –1
  - Lien, couplage
  - Lien positif
  - Lien négatif
  - Champ local
  - Champ moléculaire

Introduction

Les premiers modèles

Les réseaux

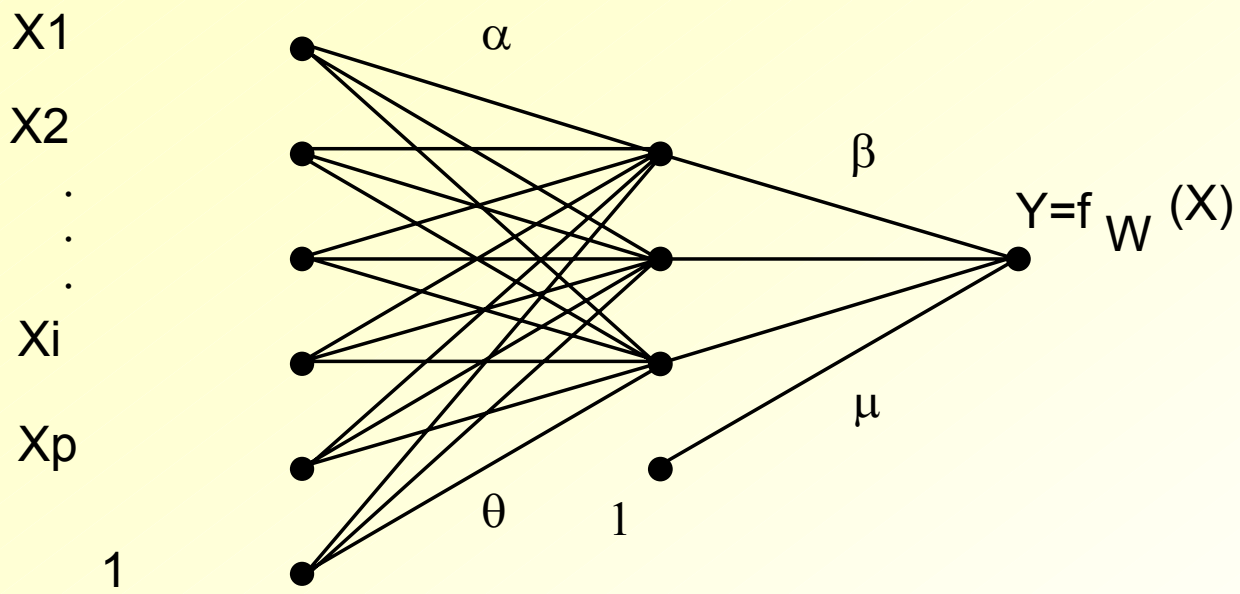
Modèle de Hopfield

Le perceptron multicouches

Algorithme de Kohonen

Conclusion

Perceptron multi-couches (MLP)
Régression



L'unité de sortie est linéaire

$$Y = Y_{out} = \Sigma\, \beta_j\, z_j + \mu$$

Les unités cachées sont non linéaires

$$z_j = \phi\,(\,\Sigma\, \alpha_{ji}\, X_i + \theta_j\,)$$

avec $\phi(t) = 1/(1 + \exp(-t))$

Le perceptron comme approximateur

Relation entrée sortie : $Y = f(X) + \eta$

Approximée par : $Y = f_W(X) + \delta + \eta$

Le modèle devient : $Y = f_w(X) + \varepsilon$

C'est un modèle de **régression non linéaire**, d'une forme particulière, appartenant à la classe des fonctions qui peuvent être représentées par un perceptron multicouches

*On sait que toute **fonction continue de $\mathbf{R}^n$ dans $\mathbf{R}$, définie sur un compact** peut être approchée avec la précision que l'on veut à l'aide d'un perceptron à une seule couche cachée, pourvu qu'on ait assez d'unités cachées.*

Les (ε) sont des v.a. i.i.d., de moyenne 0 et de variance σ^2

Perceptron multicouches

- Les fonctions d'activation des unités cachées sont dérivables, en général des sigmoïdes

- Le perceptron apparaît alors comme un modèle **paramétrique** particulier, non linéaire, où la sortie peut s'écrire (si la sortie est linéaire)

$$y = \mu + \sum_{j=1}^{k} \beta_j \phi\left(\sum_{i=1}^{p} \alpha_{ji} x_i + \theta_j\right)$$

- Les poids et les seuils sont les paramètres du modèle
- Le modèle est équivalent à un modèle linéaire, lorsque les fonctions d'activation sont linéaires

Les paramètres sont $W = (\mu, \alpha, \beta, \theta)$
Le nombre de paramètres est M

Perceptron multicouches

- Avec plus de couches, il est possible de résoudre n'importe quel problème de classification
- Avec **une seule couche cachée**, on peut approcher n'importe quelle fonction **continue définie sur un compact**, à condition de mettre assez d'unités sur la couche cachée

APPRENTISSAGE SUPERVISE

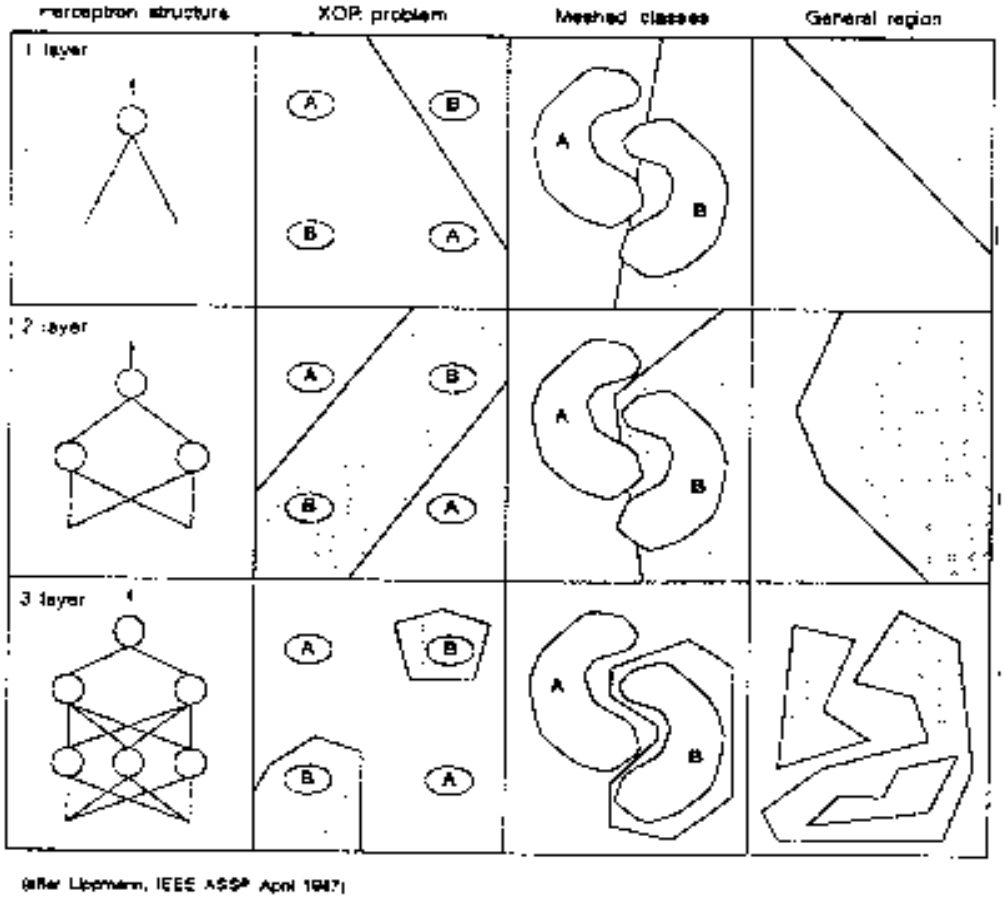
- L'apprentissage consiste à minimiser une fonction d'erreur (en général quadratique)

$$E = \sum_i \left\| y_i - f_W(X_i) \right\|^2$$

- On utilise des méthodes du gradient global ou du gradient stochastique (rétro-propagation du gradient), premier ou second ordre, etc.

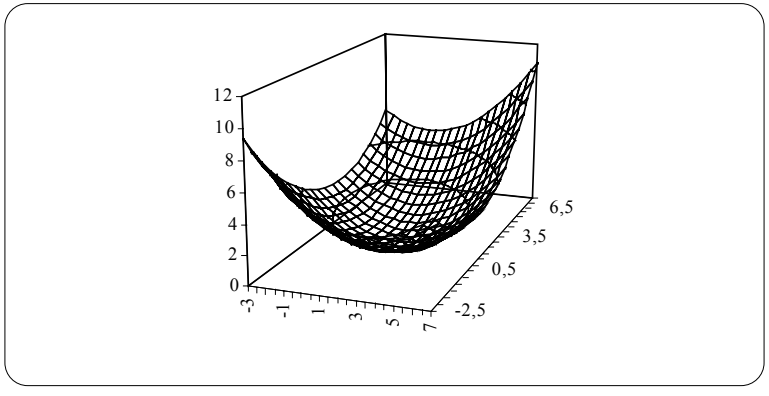
Réseaux à couches

Idée des propriétés, avant même les résultats théoriques :

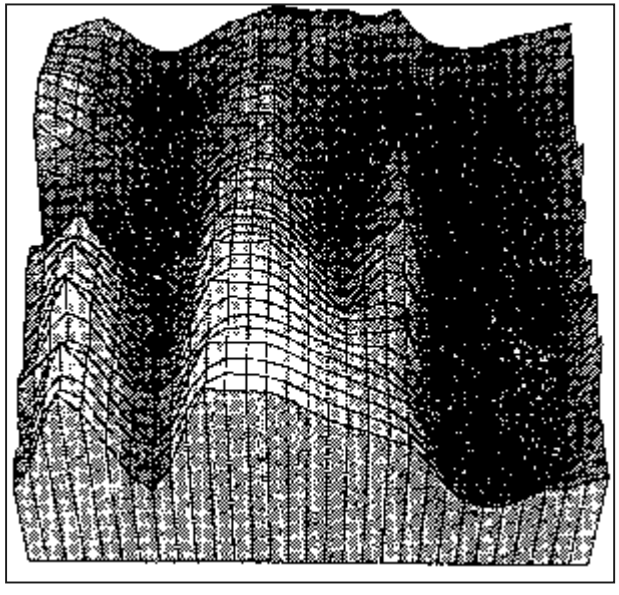


Fonctions d’erreur

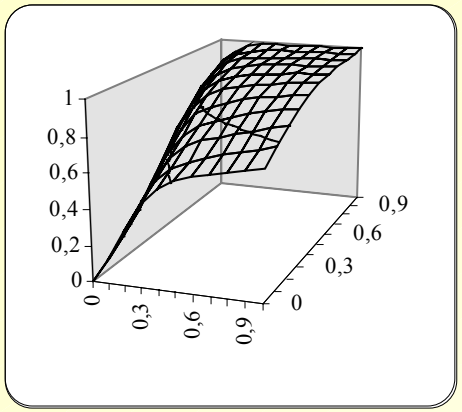
Cas linéaire



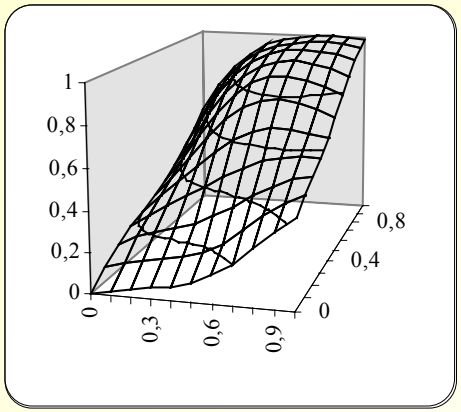
Cas non linéaire



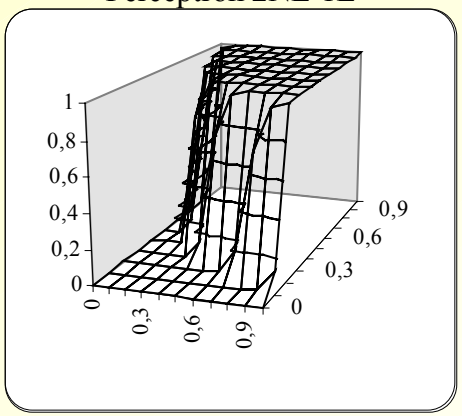
MLP : surface de réponse (même exemple)



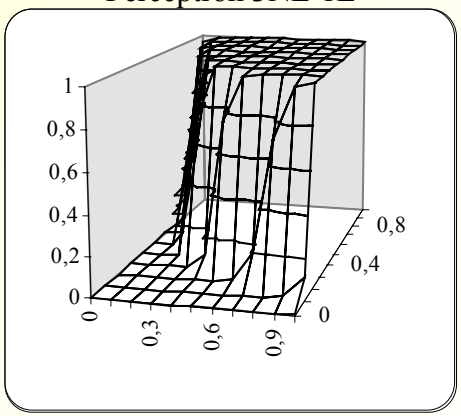
Perceptron 2NL-1L



Perceptron 3NL-1L



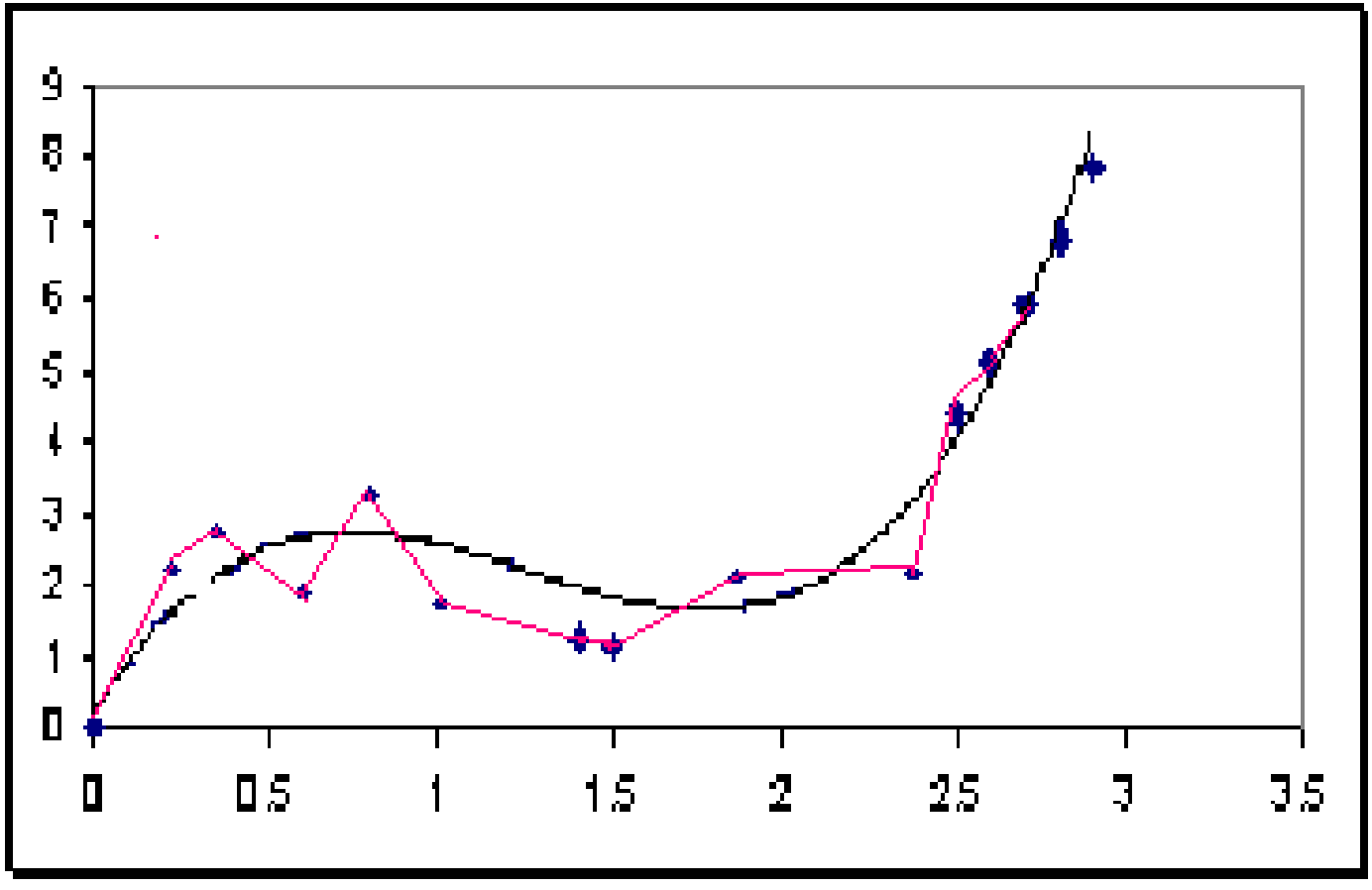
Perceptron 2NL-1NL (var. : 0.08)



Perceptron 3NL-1NL (var. : 0.22)

- Ce modèle MLP est un modèle de régression non linéaire, pour lequel on peut utiliser toutes les techniques classiques
  - d'estimation
  - de choix de modèles
  - de réduction du nombre de paramètres
- L'idée conductrice est d'essayer de faire mieux qu'un modèle linéaire, même en tenant compte des problèmes de sur-paramétrage, de minima locaux, etc.
- Il faut éviter le sur-apprentissage, et donc stopper l'estimation

Sur-apprentissage



Trois problèmes

- PROBLEME 1 : METHODE D'ESTIMATION = APPRENTISSAGE ?
- PROBLEME 2 : COMMENT COMPARER DEUX MODELES ?
- PROBLEME 3 : ELIMINATION DES PARAMETRES NON SIGNIFICATIFS ?

Estimation

L'estimation consiste à
minimiser la somme des carrés résiduels

$$E(W) = \sum (Y_i - f_W(X_i))^2$$

où (X_i, Y_i) sont T observations indépendantes

- Rq : Si les unités sont linéaires, le modèle est équivalent à un modèle linéaire
- Rq : Si les unités cachées non linéaires travaillent au voisinage de 0, elles sont approximativement linéaires et le modèle linéaire apparaît comme un sous-modèle approché du modèle considéré

MLP: algorithme de rétro-propagation

- Il s'agit de l'algorithme du gradient stochastique
- Pas de fonction d'erreur dans les couches cachées
  -> rétro-propagation de l'erreur (forme de Hebb)

avec $$\nabla w_{ki} = -\varepsilon \frac{\partial E}{\partial w_{ki}} = \varepsilon \ \delta_i \ \ x_k$$

E est l'erreur instantanée

$$\delta_i = \left(y_i - \hat{y}_i\right) \sigma'(u_i) \text{ pour les cellules de sortie}$$

$$\delta_i = \left(\sum_{j=1}^{J} \delta_j \ \ w_{ij} \right) \ \ \sigma'(u_i) \text{ pour les cellules cachées}$$

Minimisation

- On utilise de préférence une méthode du second ordre, du genre quasi-Newton (BFGS ou LEVENBERG-MARQUARDT), particulièrement adaptée à la minimisation d'une somme de carrés)
- Cette méthode fournit une approximation de l'inverse de la matrice Hessienne au minimum

$$\left(\nabla^2 E\left(\hat{W}\right)\right)^{-1}$$

- *Inconvénient* : une itération consiste à considérer l'ensemble des *T* observations.
- Si on a besoin de faire un apprentissage en continu, on utilise la méthode du gradient stochastique (la *rétro-propagation du gradient*). Alors on peut approcher la Hessienne par

$$\sum_i \left(\nabla f_{\hat{W}}\left(X_i\right)\right)\left(\nabla f_{\hat{W}}\left(X_i\right)\right)'$$

Critère de qualité d'un modèle

- Nombreuses méthodes (cross validation, ensemble d'apprentissage, de test, de validation; bootstrap, etc.)
- Utilisation de critère avec pénalité

$$\mathrm{AIC} = \ln\frac{E(W)}{T} + \frac{2M}{T},\ \mathrm{BIC} = \ln\frac{E(W)}{T} + \frac{M\ln T}{T},\ \mathrm{BIC}^* = \frac{E(W)}{T} + cste\frac{M\ln T}{T}$$

où *T* est le nombre d'observations
M est le nombre de paramètres
E(W) est le terme d'erreur

Ces critères contiennent un terme de pénalité, ils permettent de comparer deux modèles quelconques, non emboîtés

Ce critère est meilleur que le critère de *généralisation* , une bonne généralisation montre seulement que les données de test et les données d'apprentissage obéissent à la même loi, qu'il n'y a pas eu de changement de modèle.

Loi des estimateurs

- On peut étendre les résultats du modèle linéaire au cas non linéaire et on a
- Quand T tend vers l'infini,

$$\sqrt{T}\left(\hat{W} - W\right) \rightarrow N_M\left(0, \sigma^2 \Sigma^{-1}\right)$$

où σ^2 est la variance, estimée par $\dfrac{E(\hat{W})}{T}$

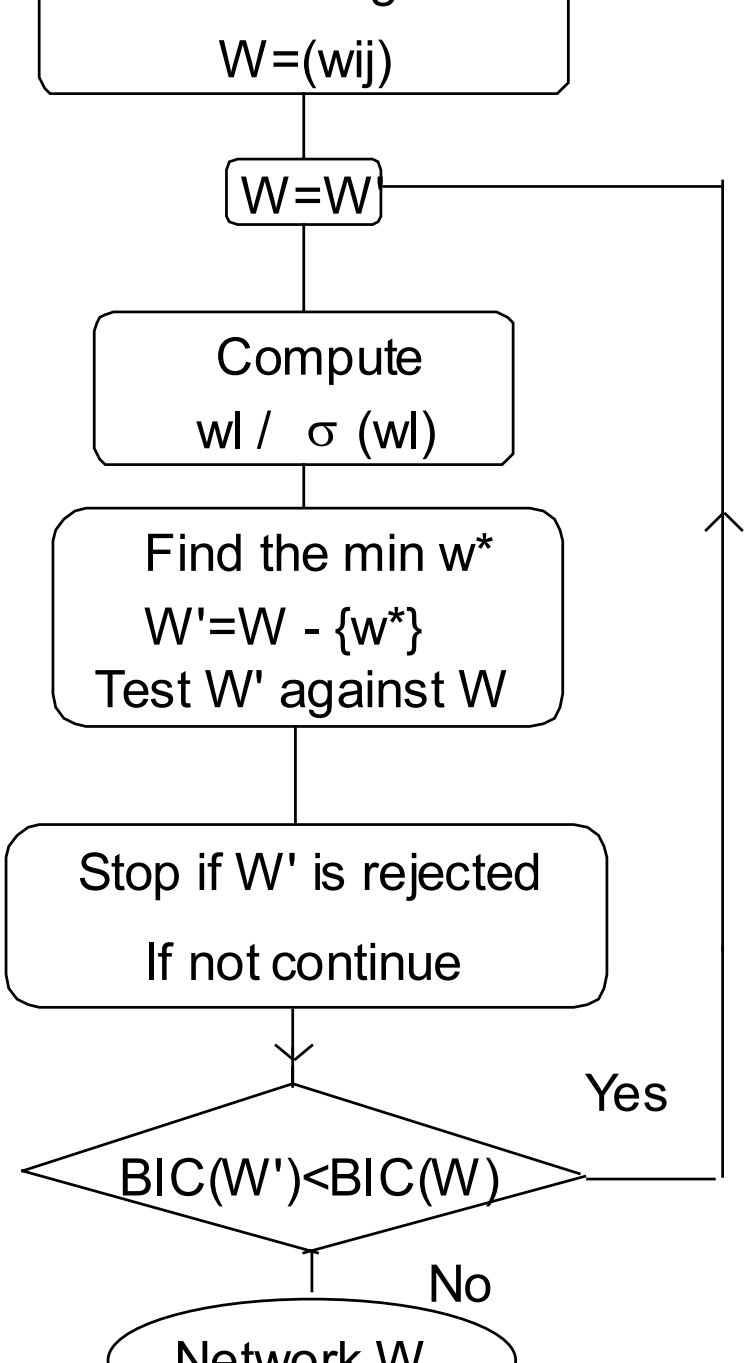
et où Σ est estimée par $\dfrac{1}{2T}\nabla^2 E(\hat{W})$

- Eliminer le poids w_l est équivalent à accepter l'hypothèse nulle "$w_l = 0$"
  contre la contre-hypothèse $w_l \neq 0$

Méthode SSM
Statistical Stepwise Method

- 1) Pour une architecture donnée (confortable), l'apprentissage donne un réseau W, de poids $\bar{w}_l, l = 1, \ldots, M$.
- 2) On calcule tous les quotients $Q(l) = \bar{w}_l / \bar{\sigma}(\bar{w}_l)$
  et on a ainsi toutes les statistiques de Student associées aux tests de "$w_l = 0$"
- 3) On définit l'indice l_1, pour lequel $Q(l_1)$ est minimum
- 4) On teste le modèle W_{l1} contre le modèle W, (par exemple on élimine $w^l{}_1$ ssi $Q(l_1)$ est plus petit que 1 ou 2
- 5) Si aucun poids ne peut être éliminé, stop.
  Si le poids w_{l1} est éliminé, on reprend l'apprentissage et on répète le point 2

- Le critère d'arrêt est naturel et objectif. Il n'est pas nécessaire d'examiner les performances du modèle
- L'algorithme d'élagage s'arrête quand tous les poids sont statistiquement significatifs
- Mais il faut beaucoup de données


W=(wij)
W=W'
Compute
wl / σ (wl)
Find the min w*
W'=W - {w*}
Test W' against W
Stop if W' is rejected
If not continue
Yes
BIC(W')<BIC(W)
No

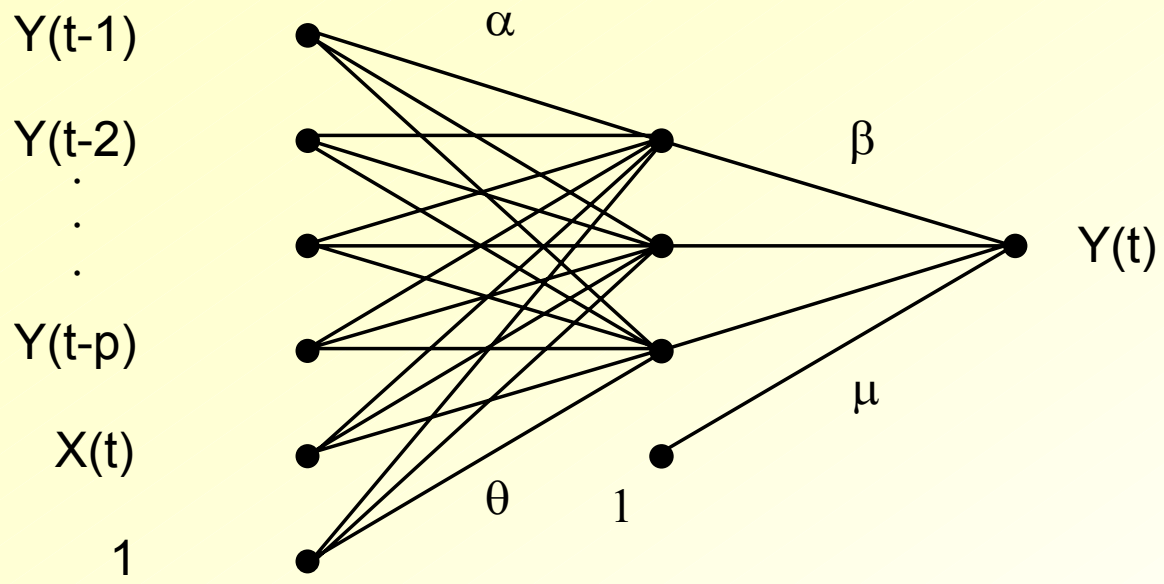
MLP: résumé des propriétés

- Approximateur universel !
- 1 seule couche cachée, mais nombre de neurones inconnu

- Difficulté de mise en oeuvre
- Difficile d'interpréter les valeurs internes
- Choix de la méthode de minimisation
  - Vitesse de convergence (très) lente pour la rétro-propagation
  - Critère d'arrêt
  - Minima locaux

- Choix de l'architecture et élagage des paramètres et unités superflus
- Choix d'un critère de qualité

Application aux séries temporelles



- On a un modèle ARMAX généralisé :

$$Y_t = f(Y_{t-1}, Y_{t-2}, \Lambda, Y_{t-p}, X_t, W) + \varepsilon_t$$
$$= f_W(Y_{t-1}, Y_{t-2}, \Lambda, Y_{t-p}, X_t)$$
$$= \mu + \sum_{i=1}^{k} \beta_j \phi\left(\sum_{i=1}^{p} \alpha_{ji} Y_{t-i} + \theta_i\right) + \sum \beta \ \phi(\alpha X_t + \theta)$$

Résultats théoriques (Mangeas, Yao, Rynkiewicz)

- On a les mêmes résultats asymptotiques sur la loi de l'estimateur de W (Mangeas, 1995)
- Les retards choisis en entrée sont ceux qui interviennent dans l'analyse linéaire
- On peut aussi étendre la méthode dans le cas où on réinjecte en entrée des résidus (écarts entre la sortie calculée et la sortie désirée)

MODELISATION DU TRAFIC INTERIEUR ROUTIER DE MARCHANDISES

- Les données sont mesurées en millions de tonnes-km

- On cherche un modèle pour la variable *Y* (*trm)*, qui représente le trafic intérieur routier de marchandises
- Le modèle de départ est inspiré d'un modèle ARIMAX, avec variables exogènes, et réintroduction des résidus
  - choix de la différentiation pour stationariser la série (ici *Y(t) - Y(t - 12)*) et toutes les variables exogènes
  - choix des variables explicatives. Ici trois variables :
    - *X1(ip9)*: indice de production industrielle sauf bâtiment et génie civil
    - *X2(iprmtk)* : indice de prix du transport sncf
    - *X3*(*iprout)* : indice du prix du transport routier

TRANSPORT ROUTIER

- Choix des retards pertinents
  - *trm(t-1), trm(t-2), trm(t-3)*
  - *ip9(t), ip9(t-1)*
  - *iptrmk(t), iprmtk(t-1)*
  - *iprout(t), iprout(t-1)*
- Choix des retards des résidus à considérer
  - *e(t-12)*

- On connaît l'écart-type résiduel du modèle ARIMAX, et on cherche à faire mieux
- Le modèle retenu préalablement est:

$$(I - B12)(I - 0.3B^3)(Y(t) - 92.8\ X1(t) - 127\ X3(t)) =$$
$$(I - 0.69\ B^{12})\ e(t) - 2.34$$

- Son écart -type est 3.72, le BIC est 80.1

DETAILS DE LA METHODE

- On part d'un perceptron avec un seul neurone caché, et on rajoute des unités cachées jusqu'il n'y ait plus d'amélioration en variance, ou trop de problème de colinéarité

- A partir d'une architecture sur-dimensionnée, on peut pratiquer la méthode SSM, en n'enlevant aucune constante, ni dans un premier temps les connexions entre unités cachées et la sortie

- On peut également rajouter ou ne pas enlever des connexions qui paraissent avoir un sens pour l'interprétation (priorité à la suppression des retards des exogènes, par exemple)

- L'initialisation se fait avec la constante de sortie égale à la moyenne et les autres valeurs aléatoires petites

RESULTATS

- On obtient un modèle neuronal presque aussi bon que le meilleur ARIMAX

  avec un écart-type de 3.71, et un BIC de 80.4

- Dans cet exemple, le modèle linéaire est suffisamment bon pour que le non linéaire n'apporte pas d'amélioration vraiment significative

- Cet exemple sert plutôt à tester la méthode

- Dans l'exemple des données du Sunspot, qui est mal modélisé par un modèle linéaire, le modèle neuronal apporte une amélioration sensible

Perceptrons multicouches
Innombrables applications

- De très nombreuses applications
  - machine à prononcer l'anglais
  - analyse de courbes, classification de formes (EEG, diagrammes, analyse financière
  - aide à la prise de décision
  - reconnaissance de lettres, de paroles, etc..., etc...
- Exemple de la modélisation du transport, mais en étudiant simultanément le transport par rail et par route
- Identification de séries temporelles et prévision
- Consommation d'eau, d'électricité

Introduction

Les premiers modèles

Les réseaux

Modèle de Hopfield

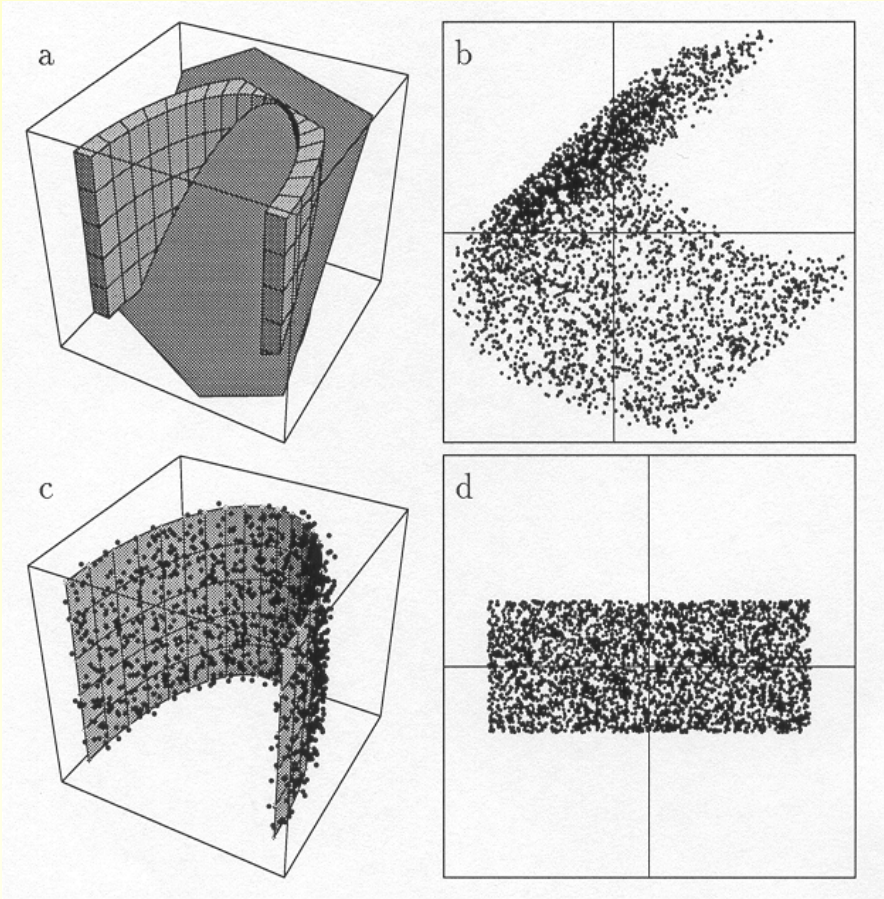
Le perceptron multicouches

Algorithme de Kohonen, autres modèles (en 7 transparents)

Conclusion

Cartes auto-organisatrices de Kohonen

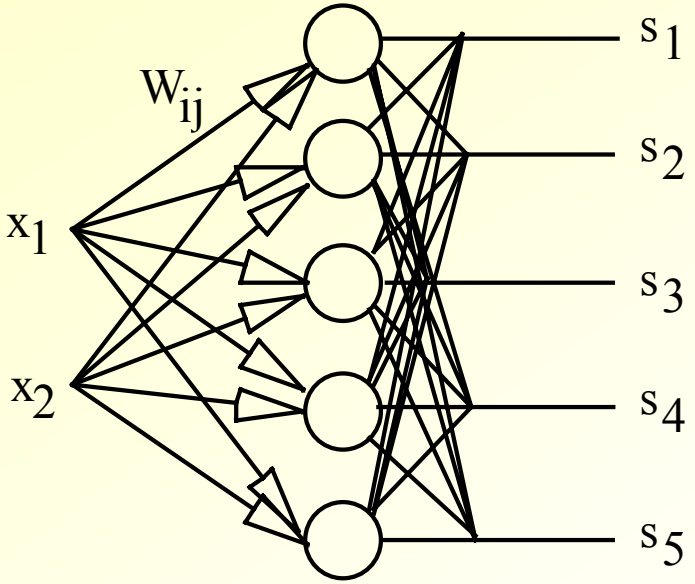
- Linéarité de l'ACP



Dessins de Demartines et Verleysen

Cartes auto-organisatrices de Kohonen

- Structure du réseau



- Choix du gagnant :
- Adaptation des poids :
  - Unité gagnante

$$\left\| x - C_{i_0} \right\| \leq \left\| x - C_i \right\|, \quad \forall i \in I$$

  - Mise à jour

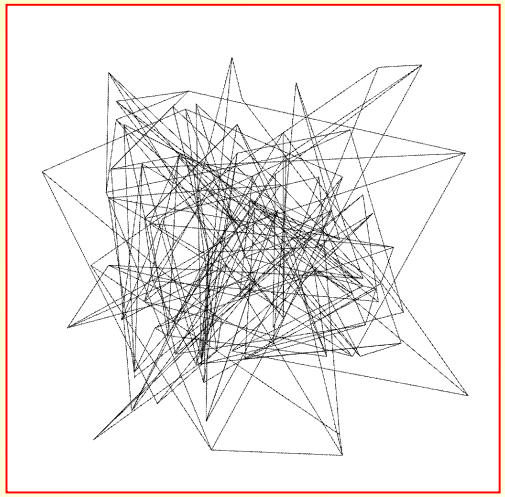
$$C_i(t+1) = C_i(t) + \varepsilon(t+1)\sigma(i, i_0)(x(t+1) - C_i(t))$$

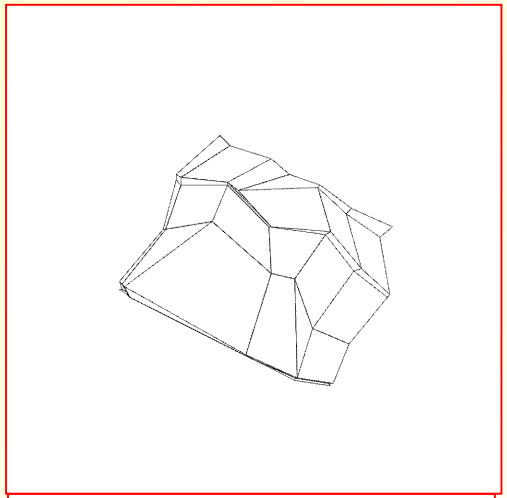
Kohonen: utilisation et limites

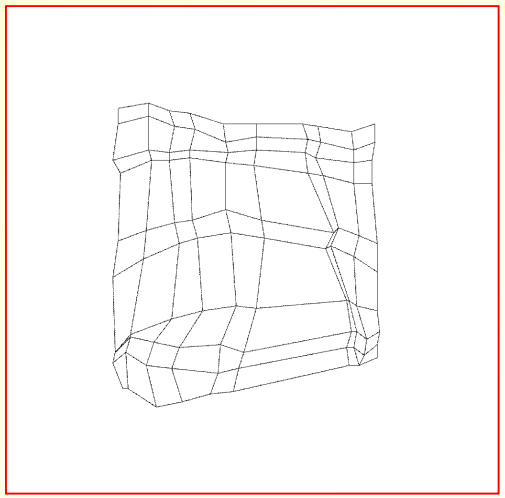
- Projection d'un espace quelconque dans un espace de dimension 1 ou 2
- Propriété d'organisation, ou respect de la topologie
- Réalise en même temps une quantification (avantage ou inconvénient…)
- (Très) facile à manipuler par rapport à d'autres méthodes non-linéaires de projection
- Effet papillon (et/ou minima locaux)
- Très difficile à étudier mathématiquement
- Si 0 voisins: quantification vectorielle pure ("competitive learning")
- Peut être étendu à de l'apprentissage supervisé
- Peut être implémenté en version batch

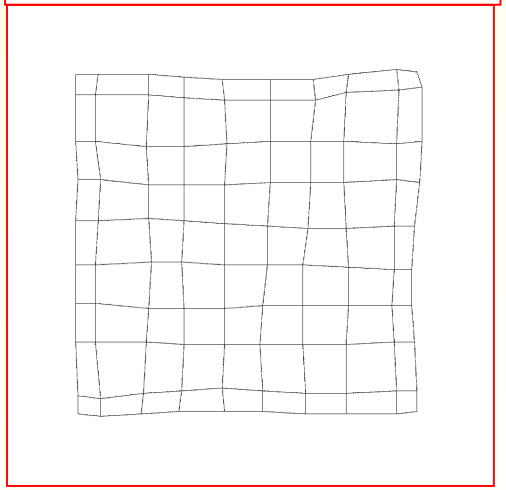
Convergence du réseau de Kohonen

- dimension d'entrée = dimension de la carte = 2





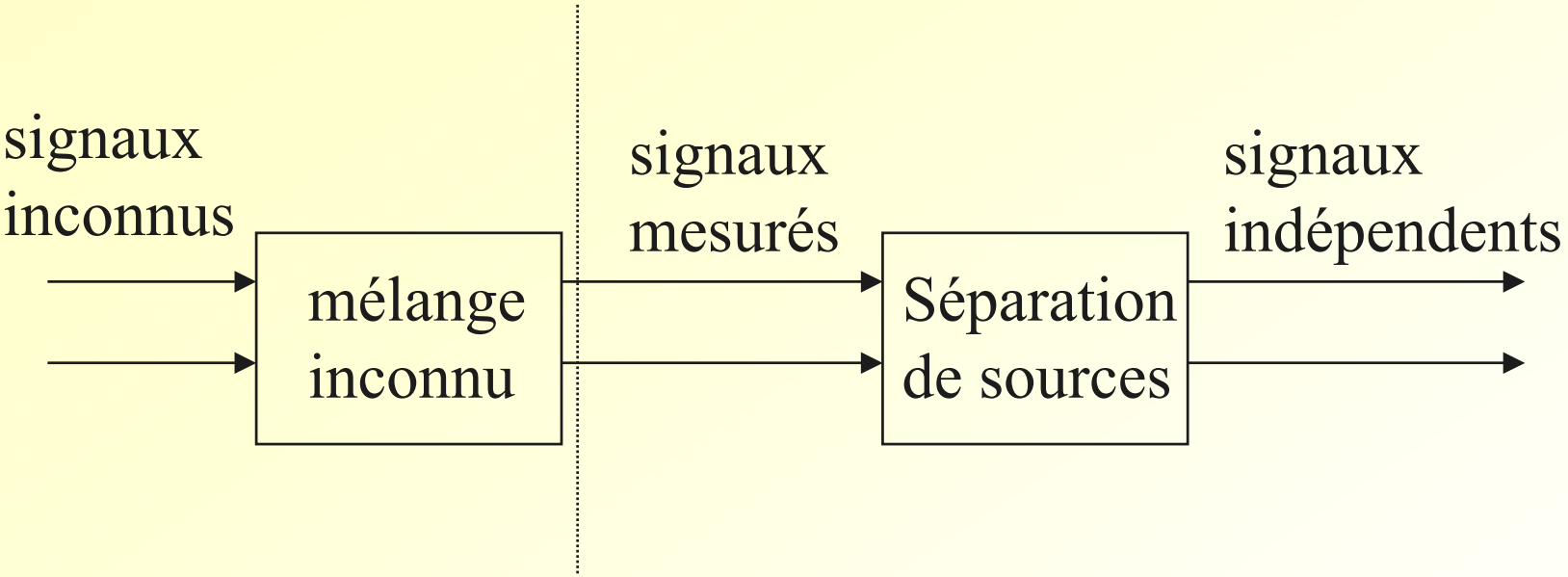




Autres modèles

- Réseaux récurents
- Séparation de sources
- Modèles de renforcement
- Etc…

Séparation de sources - cocktail party



- Le but est de retrouver des signaux indépendants
- Il faut au moins autant de mesures que de signaux inconnus
- On les retrouve à une constante et une permutation près !

Exemple (Verleysen)

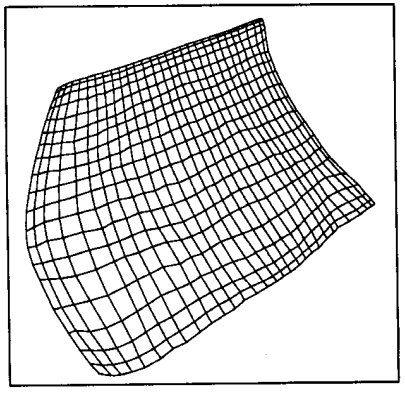
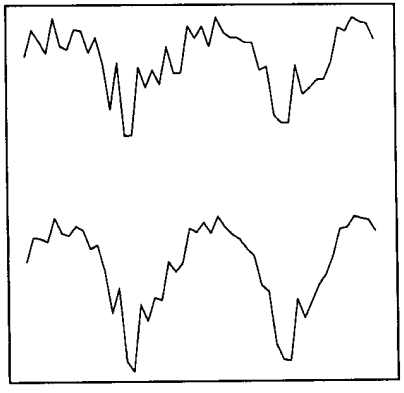


Fig. 1: Left: the converged map.



Right: mixed sources.

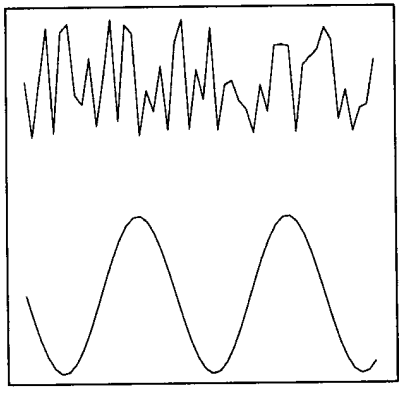
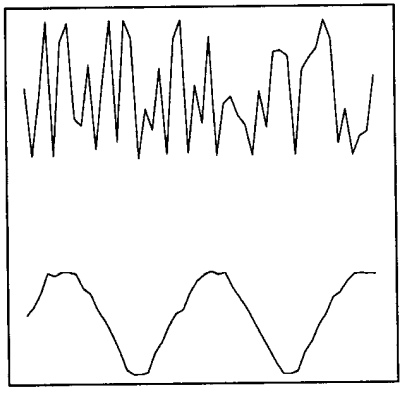


Fig. 2: Left: original source signals.

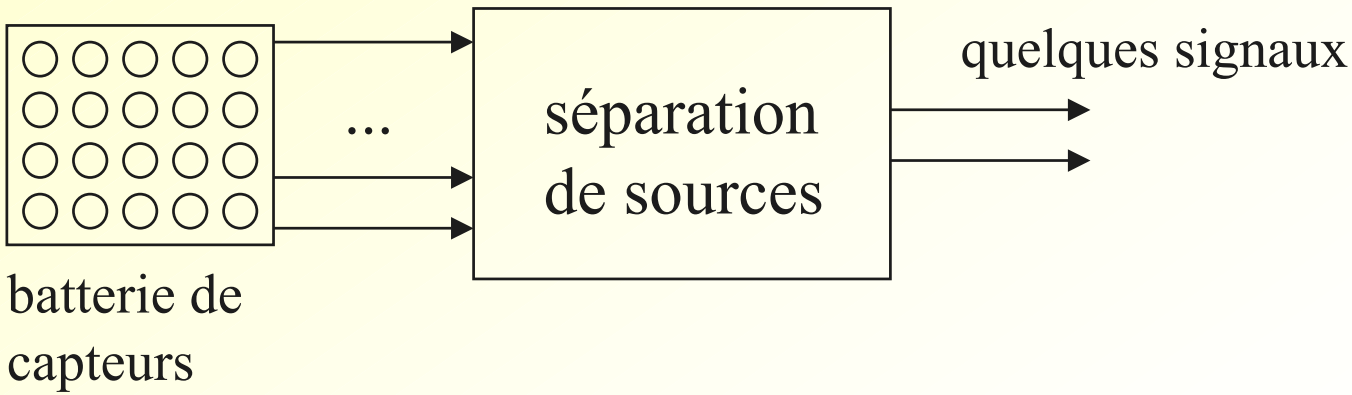


Right: separated source signals.

Séparation de sources

- 2 méthodes
  - Utilisation des cartes auto-organisatrices: capturer la “relation” qui existe entre les sources
  - Utilisation de la “CCA” (Curvilinear component analysis): test d’indépendance entre les signaux de sortie

- Utilisation possible :



Introduction

Les premiers modèles

Les réseaux

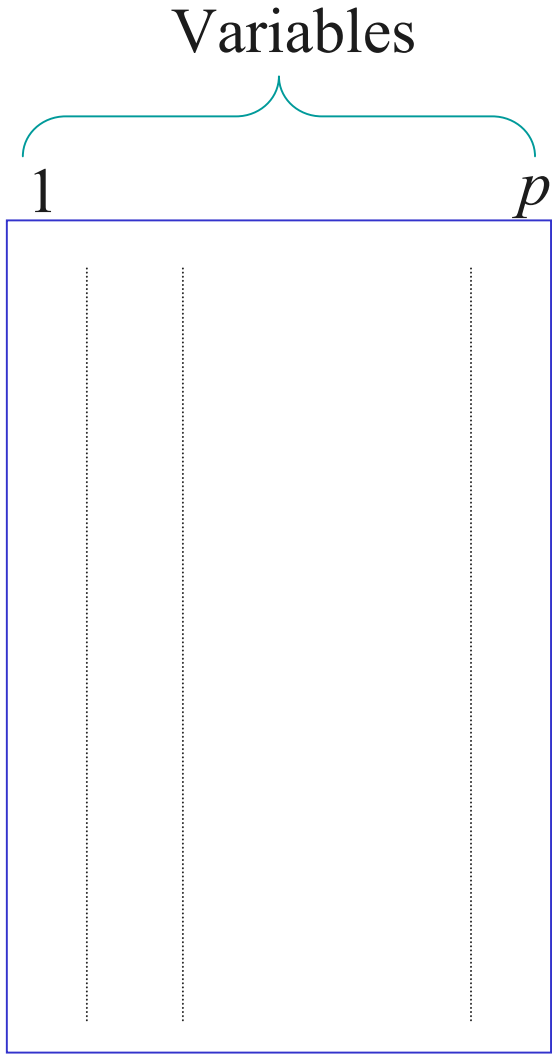
Modèle de Hopfield

Le perceptron multicouches

Algorithme de Kohonen (plus long)

Conclusion

Analyse de données, data mining



Comment extraire de l'information ?

N et p sont grands

Extraction d'individus types : Quantification Vectorielle

- K : espace des données, dimension p
- f : densité des données
- $x_1, x_2, \ldots, x_N$, : les données
- n : nombre de classes
- $C_1, C_2, \ldots, C_n$: quantifieurs ou vecteurs codes ou centres
- $A_1, A_2, \ldots, A_n$: classes

BUT : Minimiser la **distorsion quadratique** (l'erreur)
(= **Somme des carrés intra**)

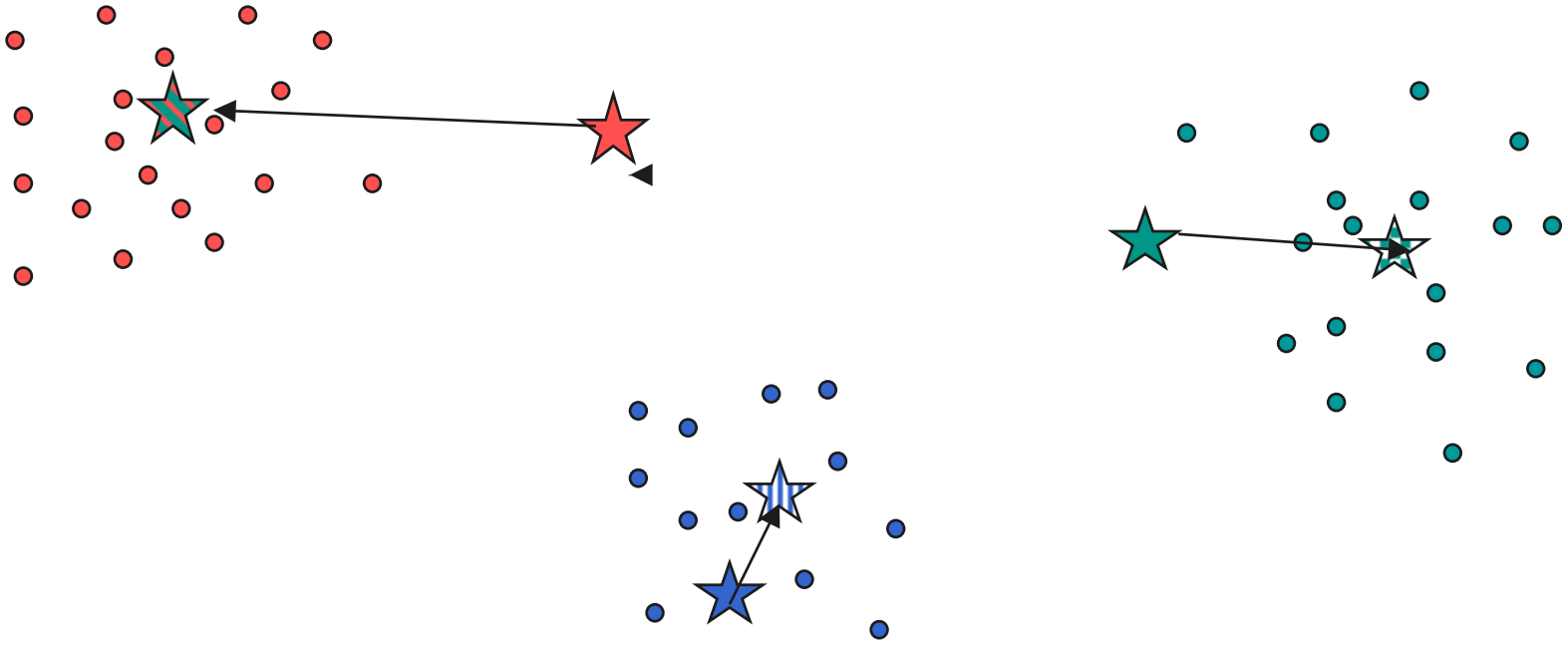
$$D_0(f, C_1, C_2, \Lambda, C_n) = \sum_{i=1}^{n} \int_{A_i} \|x - C_i\|^2 f(x)\, dx \qquad (1)$$

Estimée par

$$\hat{D}_o(f, C_1, C_2, \Lambda, C_n) = \frac{1}{N} \sum_{i=1}^{n} \sum_{x_j \in A_i} \|x_j - C_i\|^2 \qquad (2)$$

Algorithme Déterministe : Centres mobiles (FORGY, LLOYDS, LBG)

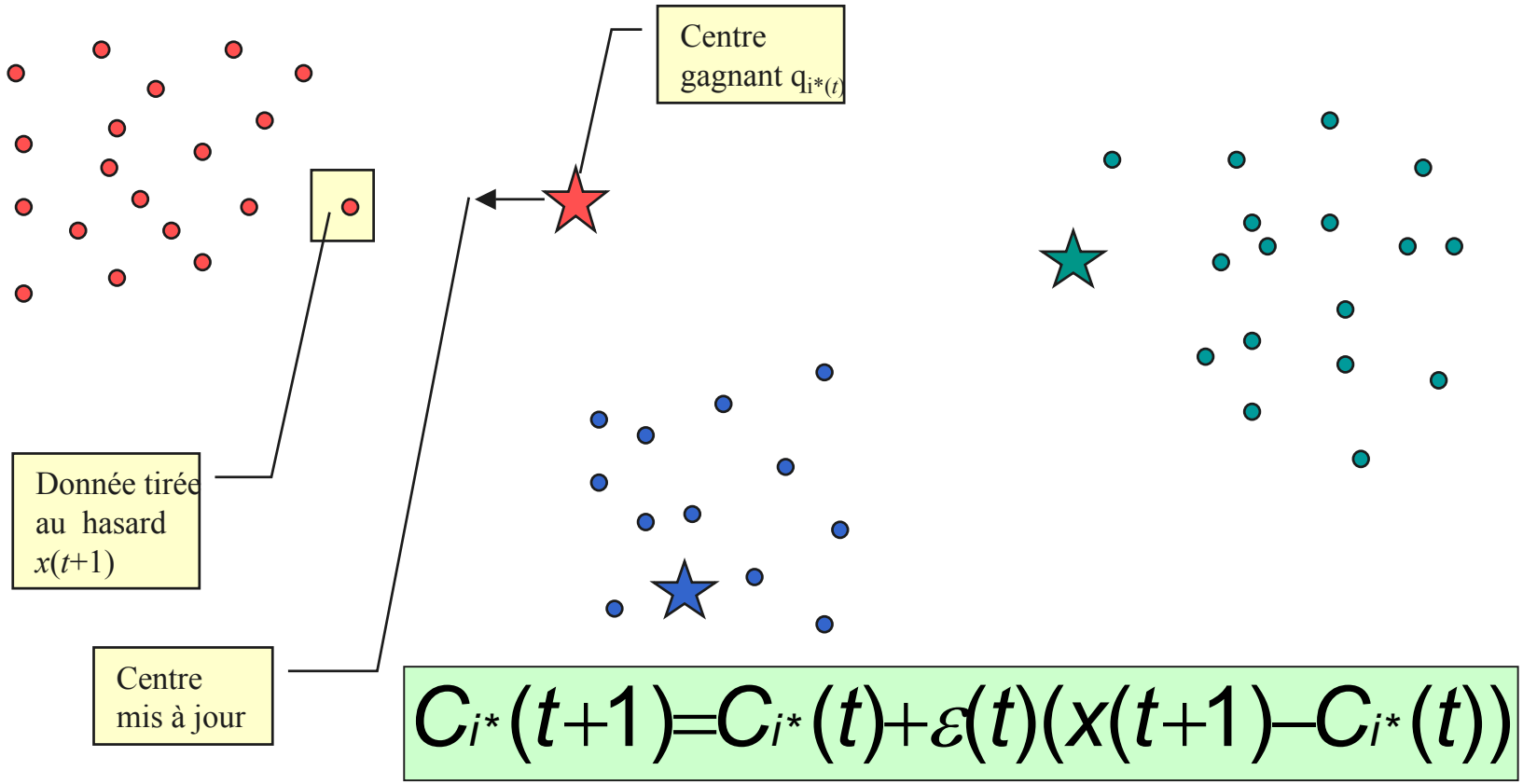
- A chaque étape, les classes sont définies (par les plus proches voisins), et les vecteurs codes sont re-calculés comme les centres de gravité des classes, etc.



- (On part de vecteurs codes aléatoires, on détermine les classes, puis les centres, puis les classes, etc.)

Algorithme Probabiliste associé (SCL)

- On déplace seulement le gagnant



- ***Avec l'algorithme de Kohonen, on déplace le vecteur code gagnant, mais aussi ses voisins.***

Algorithme SCL (0 voisin)

- L'algorithme SCL est la version stochastique de l'algorithme de Forgy
- L'algorithme de Forgy minimise la distorsion et converge vers un minimum local
- L'algorithme SCL converge ***en moyenne vers un minimum local***
- La solution dépend de l'initialisation

Algorithme de Kohonen (SOM)

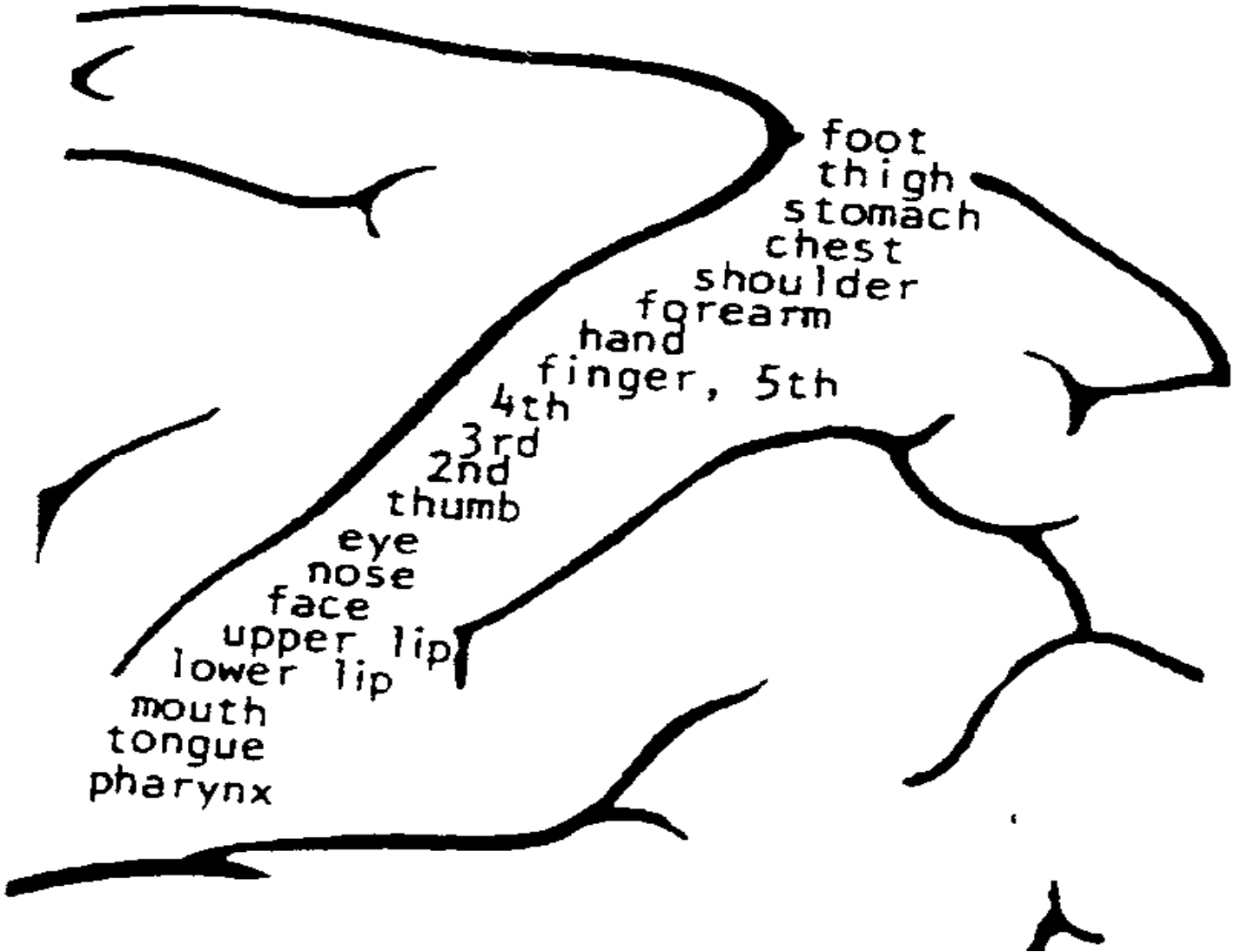
- Apprentissage non supervisé
- Les réponses associées à des entrées voisines sont voisines
- On parle d'auto-organisation, de respect de la topologie

- Les associations
  - rétine - cortex visuel
  - fréquences des sons - cortex auditif
  - peau - cortex sensoriel

respectent la notion de voisinage

- Nombreuses applications en représentation de données de grande dimension sur des réseaux de dimension 1 ou 2, ou classification où la notion de classes voisines a un sens

Cortex sensoriel (Kohonen)



Cortex sensoriel

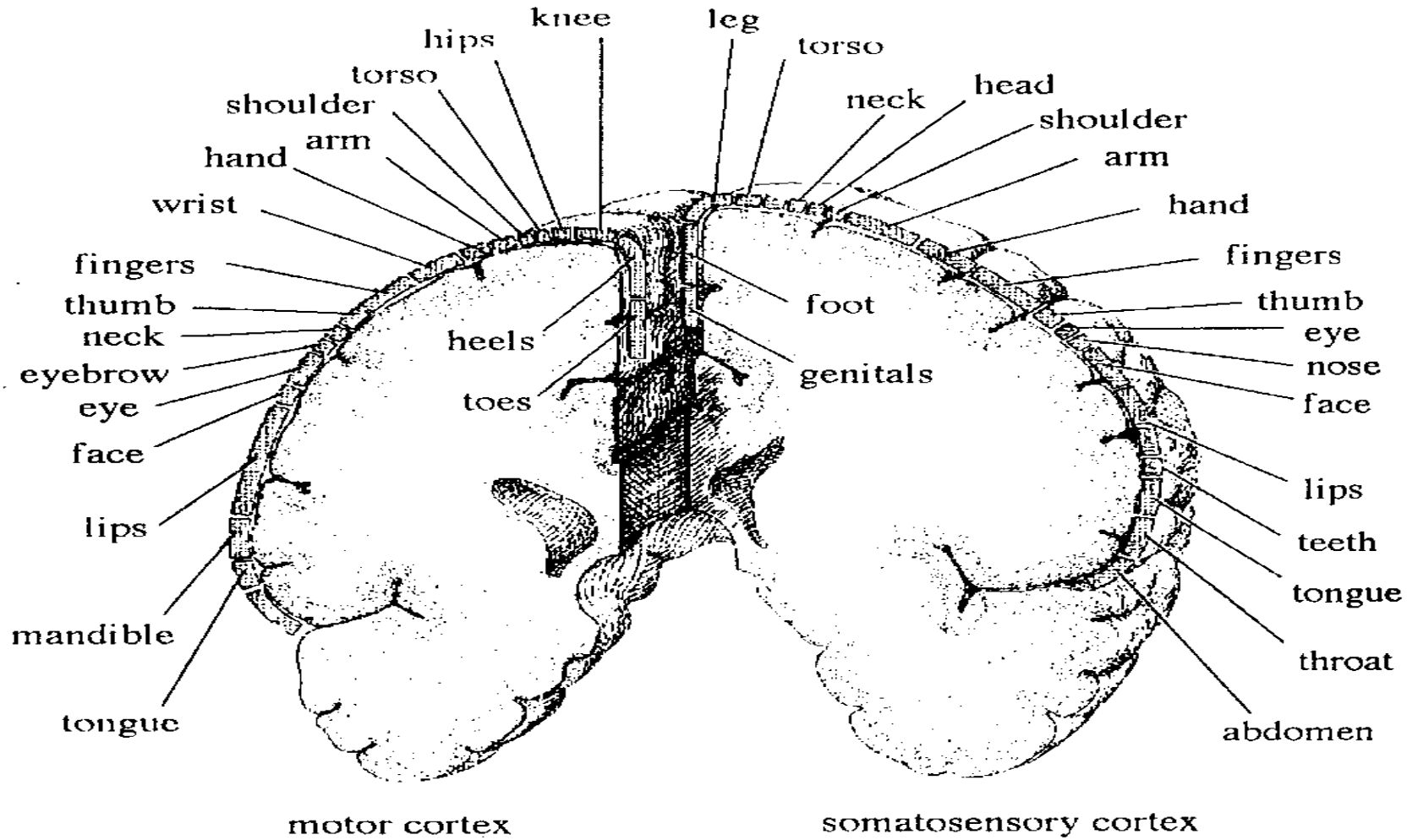
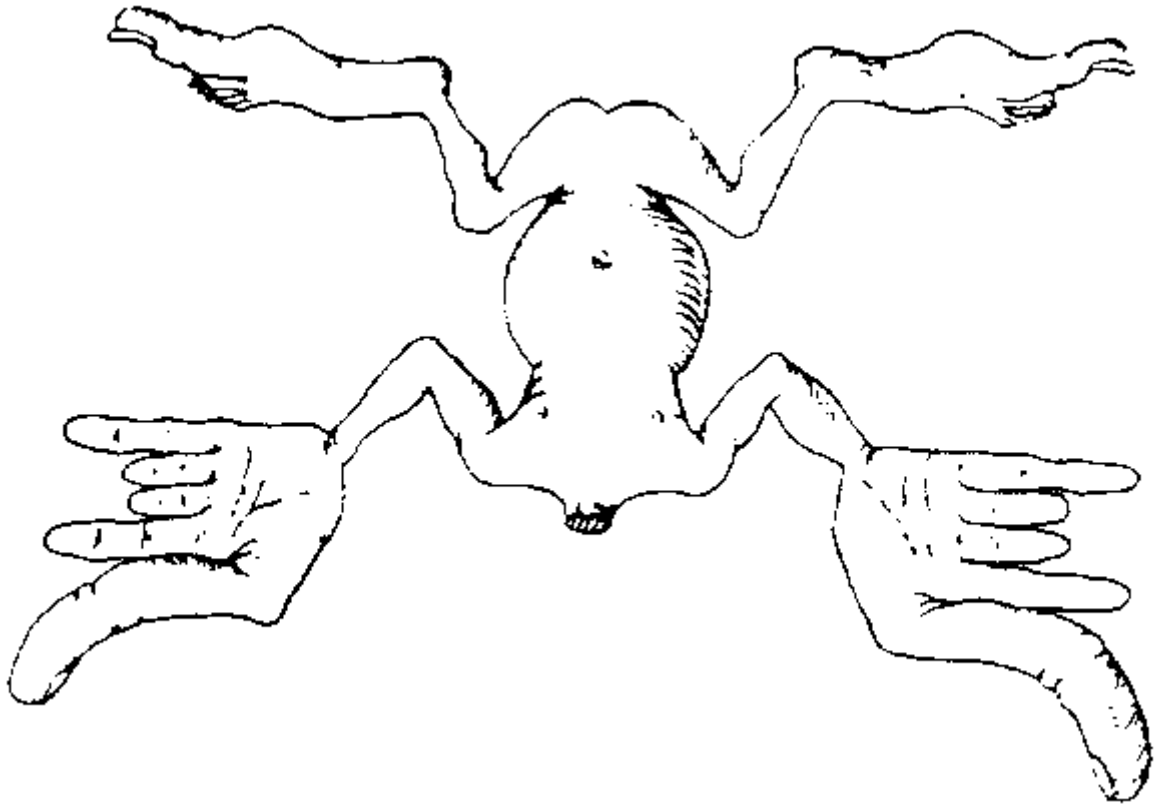
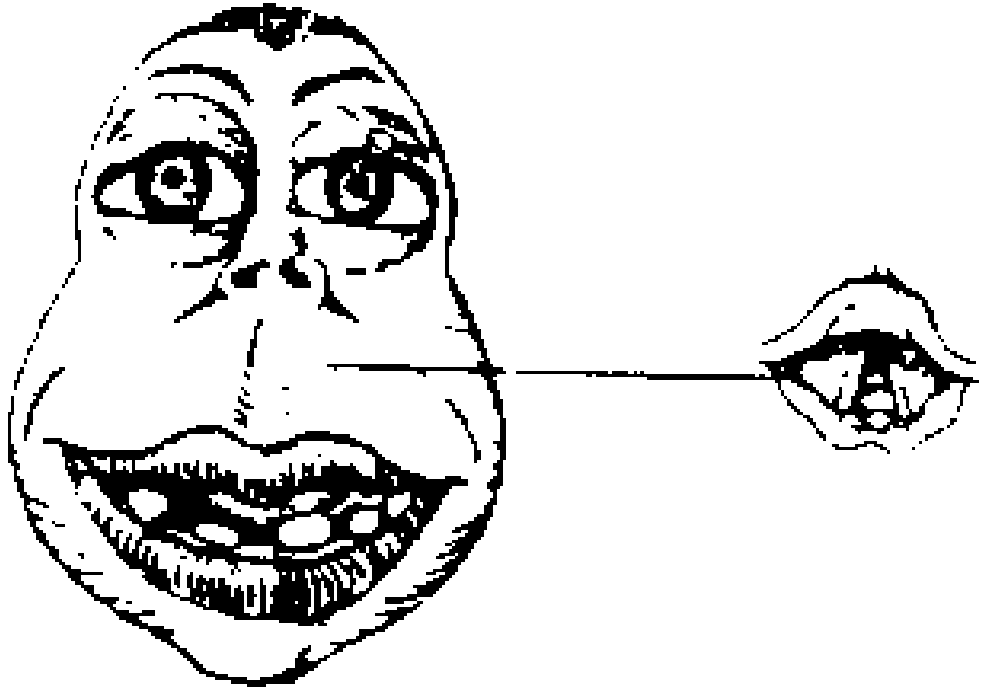


Fig. 15.3. The somatosensory and motor cortex

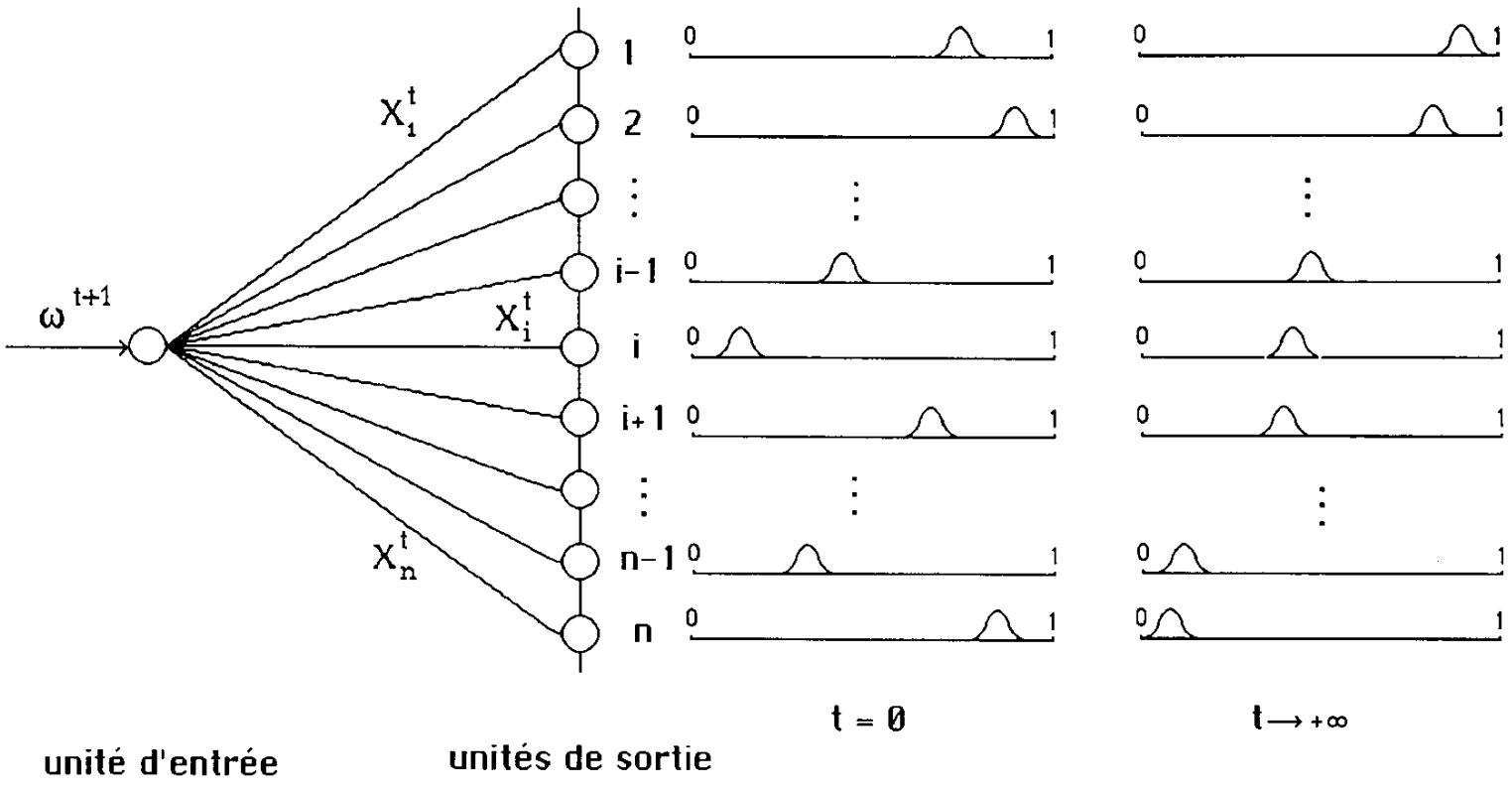
Homonculus (Anderson, Penfield and Boldrey)

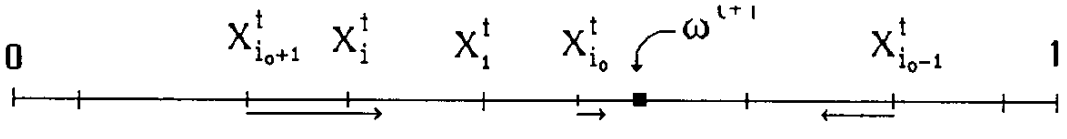


Tête d’homonculus (Anderson, Penfield and Boldrey))



Cortex auditif (Pagès et Fort)





représentation des poids dans l'espace des stimuli (d=1)

L'algorithme

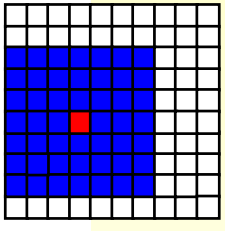
- Il s'agit d'un algorithme original de classification qui a été défini par Teuvo Kohonen, dans les années 80.
- L'algorithme regroupe les observations en classes, en respectant la topologie de l'espace des observations. Cela veut dire qu'on définit a priori une **notion de voisinage entre classes** et que des observations voisines dans l'espace des variables (de dimension p) appartiennent (après classement) à la même classe ou à des classes voisines.

- Les voisinages entre classes peuvent être choisis de manière variée, mais en général on suppose que les classes sont disposées sur une grille rectangulaire qui définit naturellement les voisins de chaque classe.

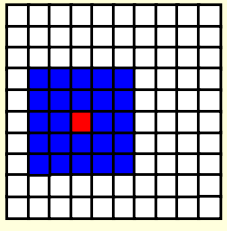
- Mais on peut choisir une autre topologie

Structure en grille ou en ficelle

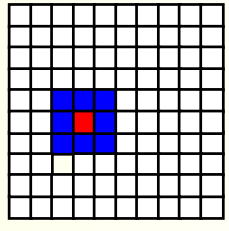
- Les grilles ne sont pas nécessairement carrées



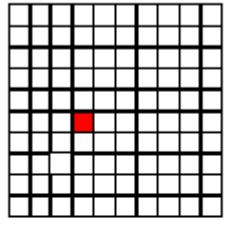
Voisinage de 49



Voisinage de 25



Voisinage de 9



Voisinage de 1

Voisinage de 7

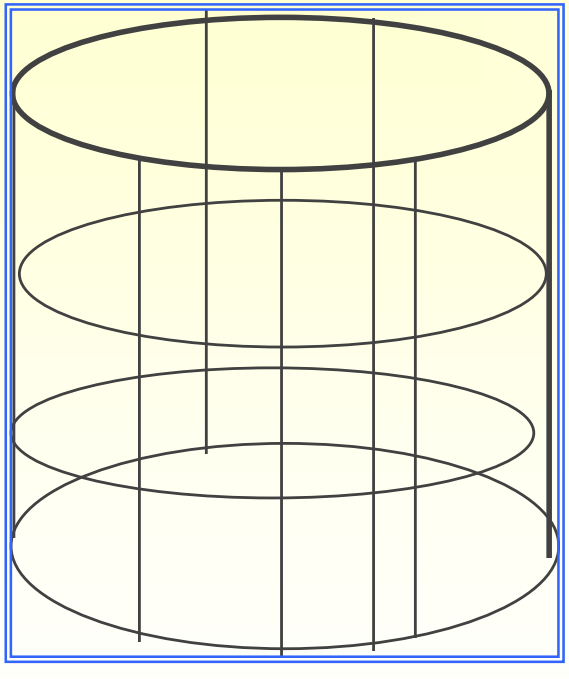
Voisinage de 5

Voisinage de 3

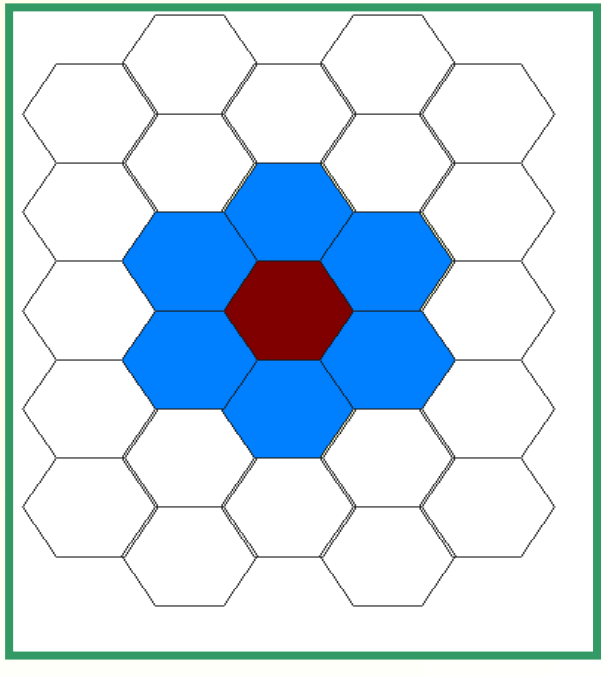
Voisinage de 1

Structure en cylindre, ou en pavage hexagonal

CYLINDRE



MAILLAGE HEXAGONAL



L'algorithme

Principe de l'algorithme de Kohonen

- L'algorithme de ***classement*** est **itératif**.
- L'initialisation : associer à chaque classe un vecteur code dans l'espace des observations choisi de manière aléatoire.
- Ensuite, à chaque étape, on choisit une observation au hasard, on la compare à tous les vecteurs codes, et on détermine la **classe gagnante**, c'est-à-dire celle dont le vecteur code est le plus proche au sens d'une distance donnée a priori.
- **On rapproche alors de l'observation les codes de la classe gagnante et des classes voisines**.
- Cet algorithme est analogue à **l'algorithme SCL**, pour lequel on ne modifie à chaque étape que le code de la classe gagnante.
- C 'est aussi un ***algorithme compétitif***

Notations (Kohonen, ou SOM)

- Espace des entrées K dans R^p
- n unités, rangées en réseau de dimension 1 ou 2, pour lesquelles est défini un système de voisinage
- A chaque unité i (i=1,...,n), est associé un ***vecteur code*** C_i de p composantes

- La réponse d'une unité i à l'entrée x est mesurée par la proximité de x avec le vecteur code C_i

- Initialisation aléatoire des vecteurs codes
- A l'étape t,
  - **on présente une entrée x**
  - **on cherche l'unité gagnante $i_0(x)$**
  - **on rapproche C_{i0} et les C_i voisins, de l'entrée x**

Définition de l'algorithme on-line

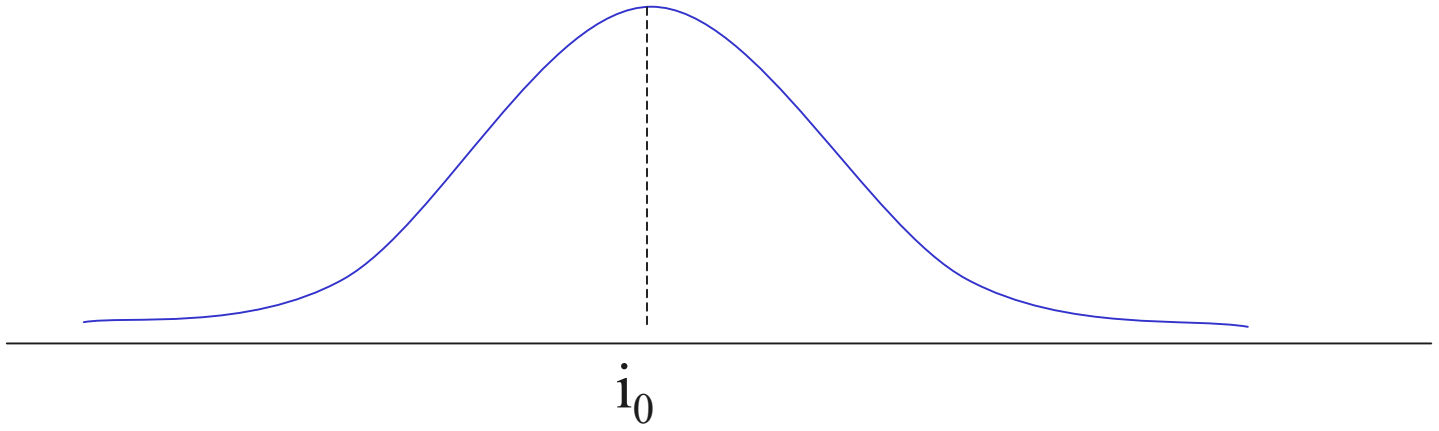
- Les $\{C_i(0)\}$ sont les vecteurs codes initiaux de dimension p
- $\varepsilon(t)$ est le **paramètre d'adaptation**, positif, <1, constant ou lentement décroissant
- La **fonction de voisinage** $\sigma(i,j)$=1 ssi i et j sont voisins, =0 sinon, la taille du voisinage décroît aussi lentement au cours du temps
- Deux étapes : au temps t+1, on présente $x(t+1)$, (tirages indépendants)
  - On détermine l'unité gagnante

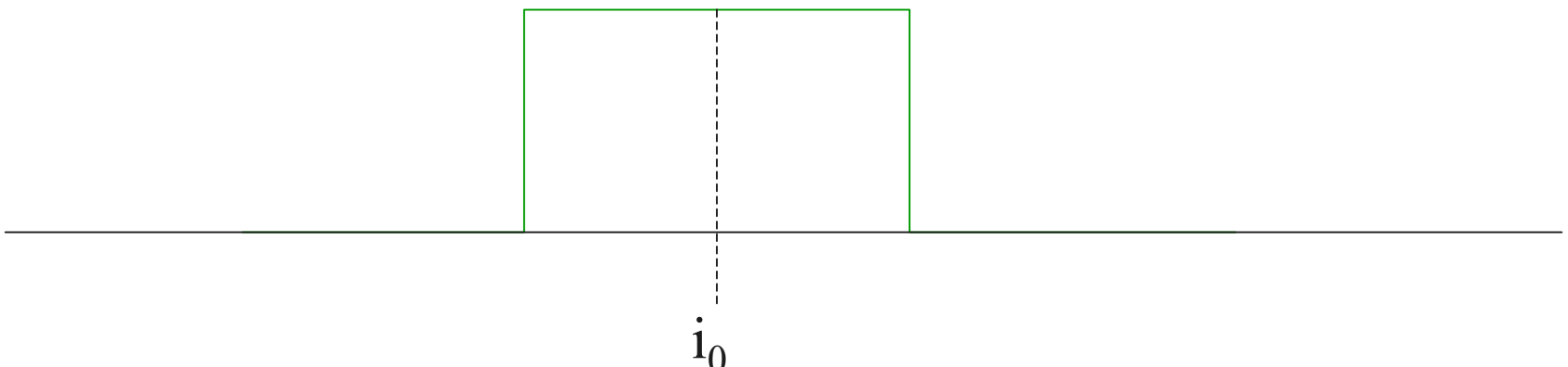
$$i_0(t+1)=\arg\min_i \left\| x(t+1)-C_i(t) \right\|$$

  - On met à jour les vecteurs codes

$$C_i(t+1)=C_i(t)+\varepsilon(t+1)\,\sigma(i_0(t+1),i)\,(x(t+1)-C_i(t))$$

Fonctions de voisinage σ

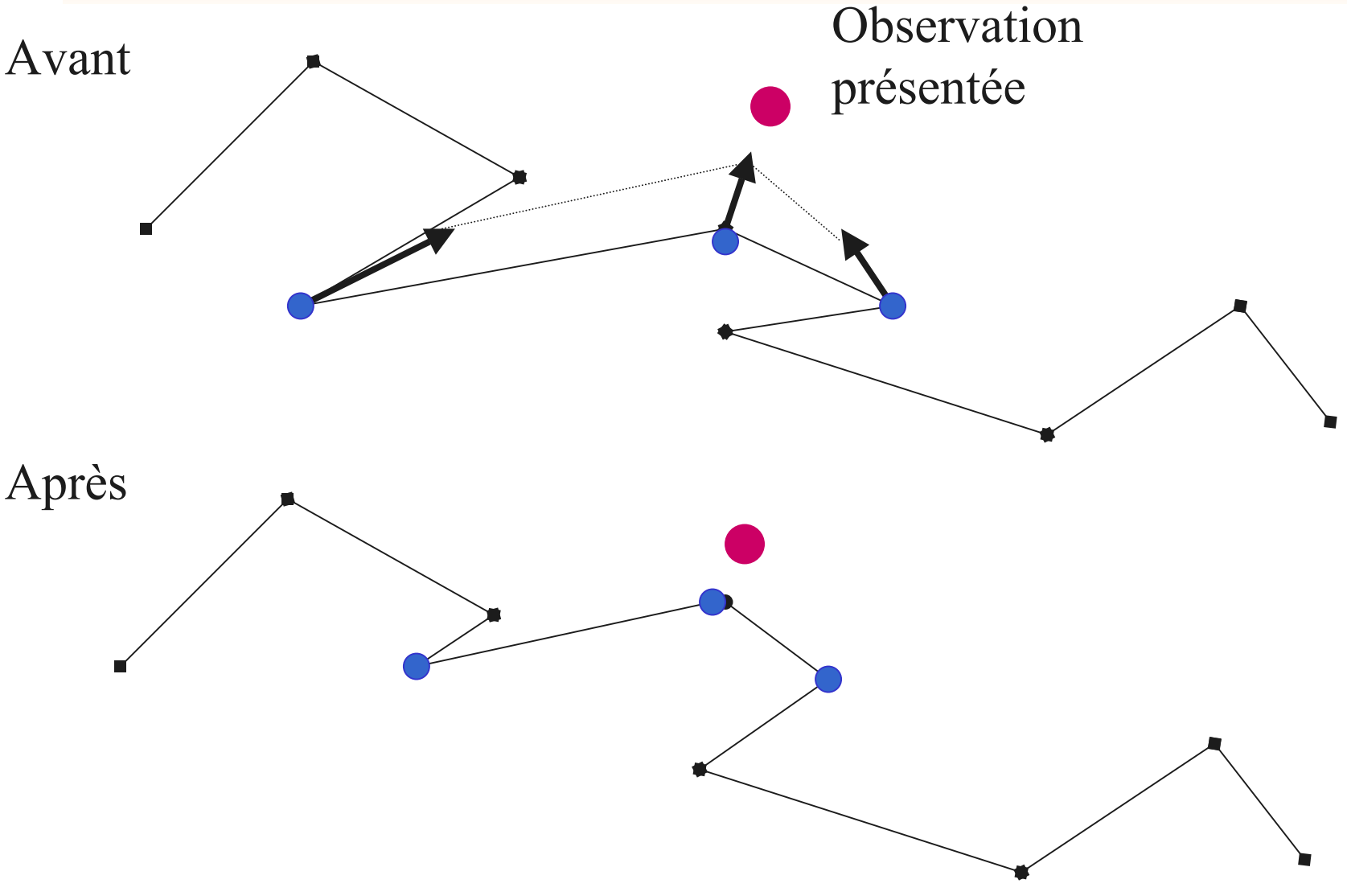




Kohonen / SCL

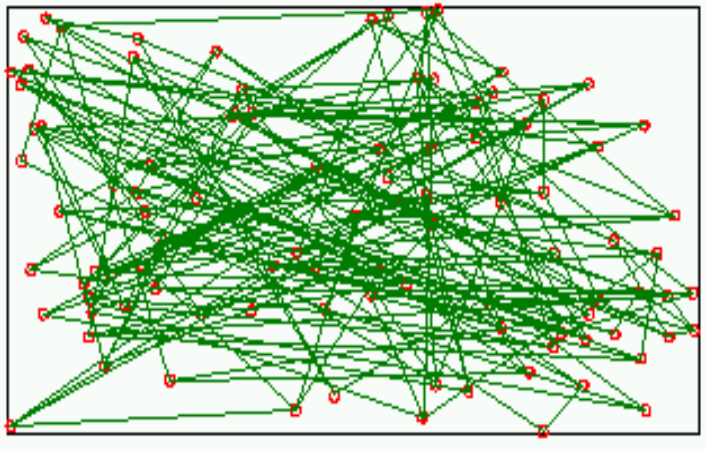
- En fait l'algorithme de Kohonen est une extension de la version stochastique de l'algorithme des centres mobiles
- Issu du domaine de la quantification vectorielle, de la théorie du signal
- Applications où les données sont très nombreuses, disponibles on-line,
- Pas besoin de les stocker

Exemple : une étape

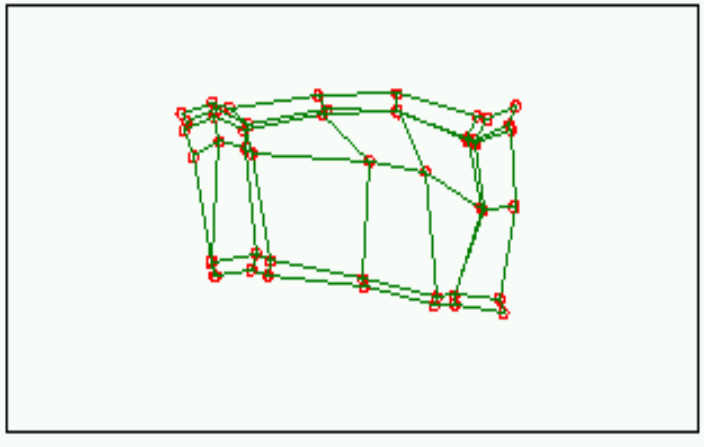


Démo en dimension 2

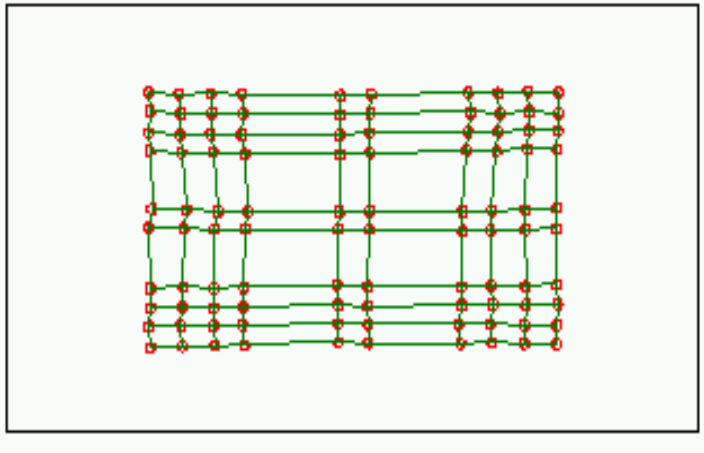
Grille:10x10 Etape= 0/1000000 Rayon= 0



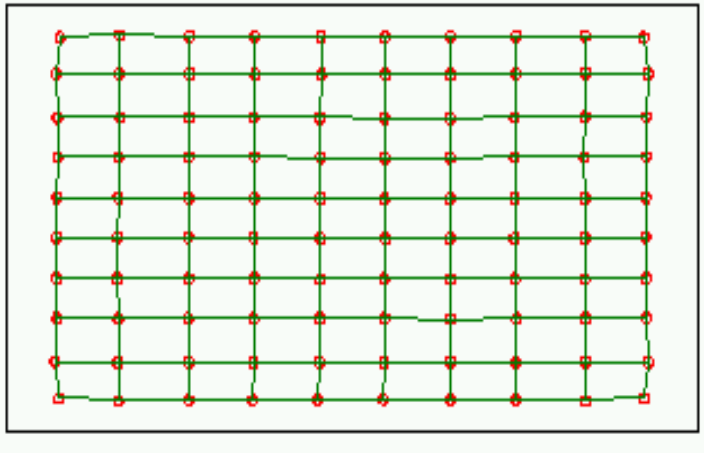
Grille:10x10 Etape= 1000/1000000 Rayon= 5



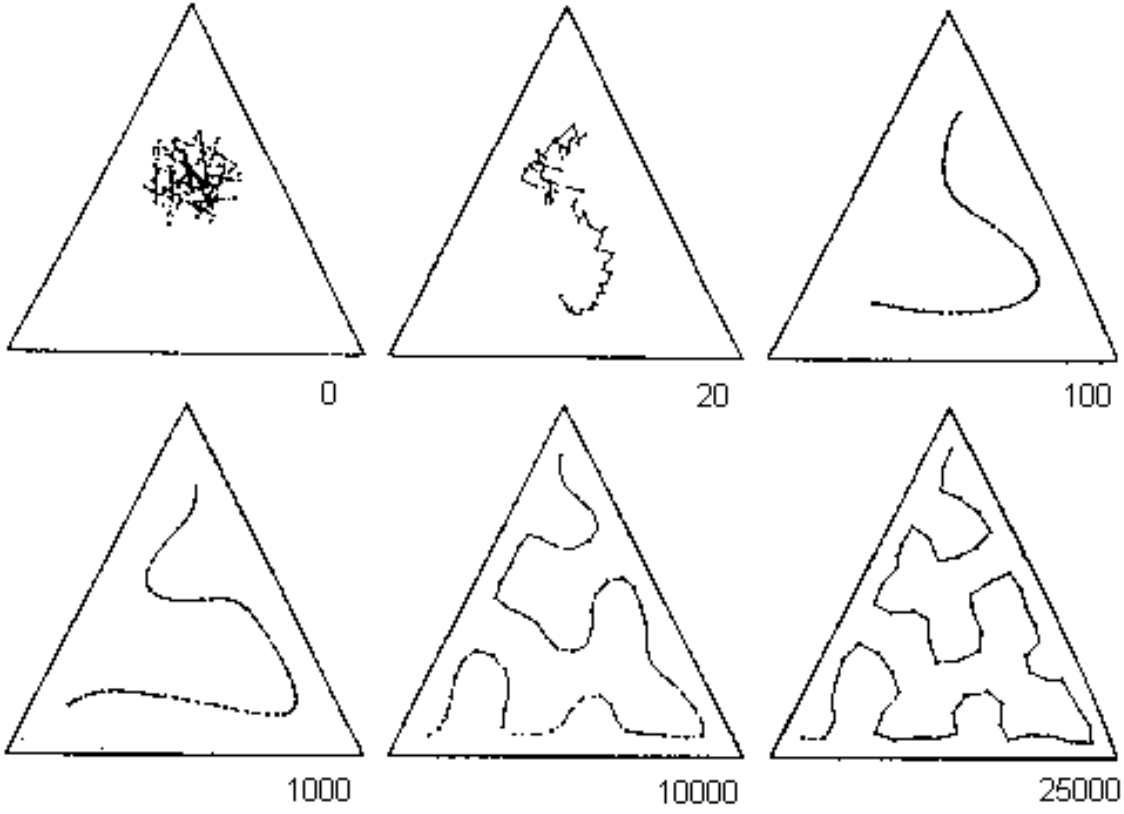
Grille:10x10 Etape= 100000/1000000 Rayon= 3



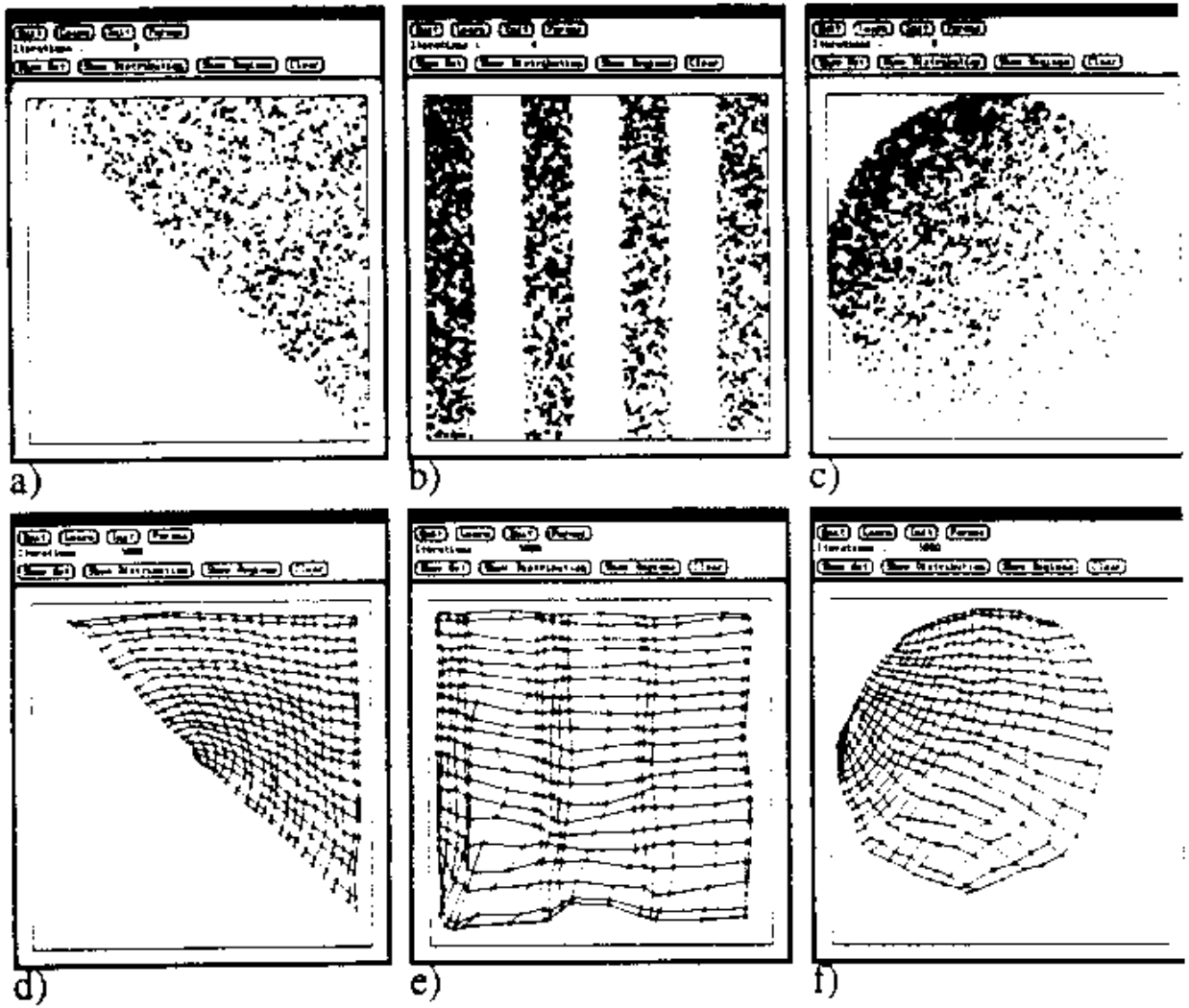
Grille:10x10 Etape= 1000000/1000000 Rayon= 0



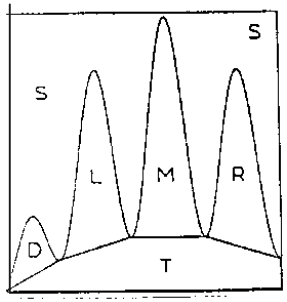
Exemples de simulations (Kohonen)



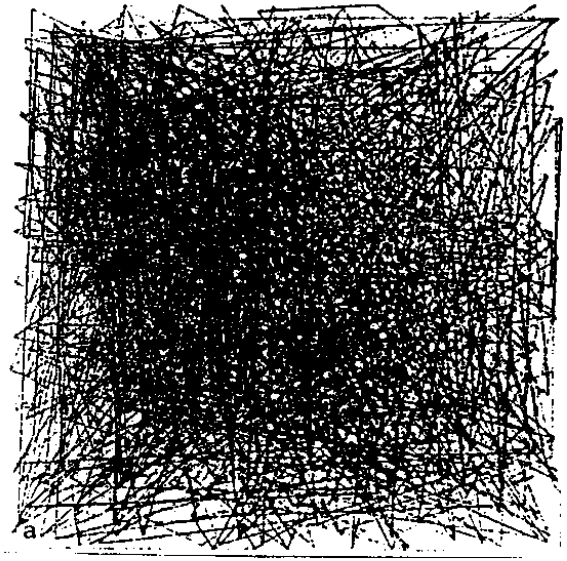
Exemples de simulations (EPFL)



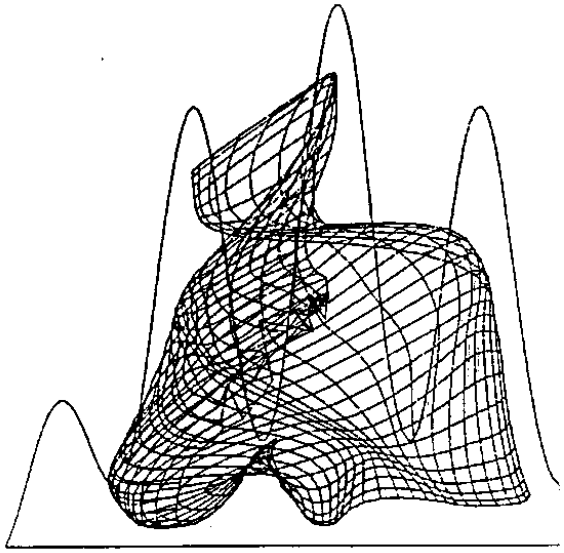
Adaptatif (Ritter et Schulten)



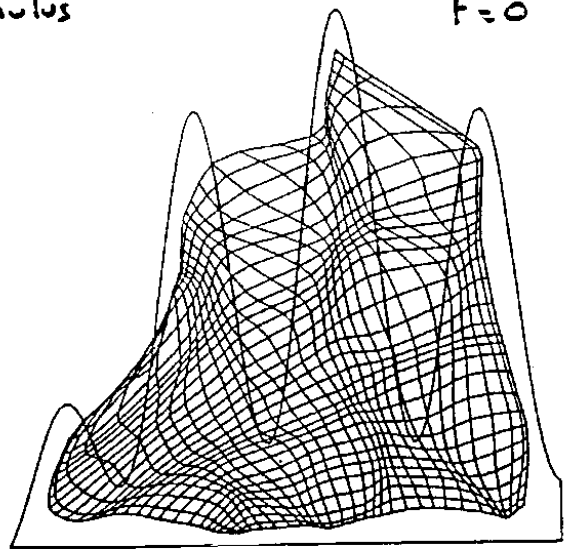
Ensemble de Présentation du Stimulus

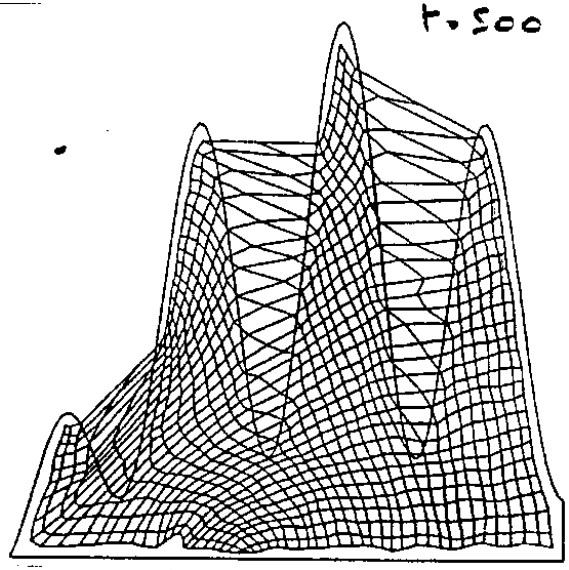


t=0

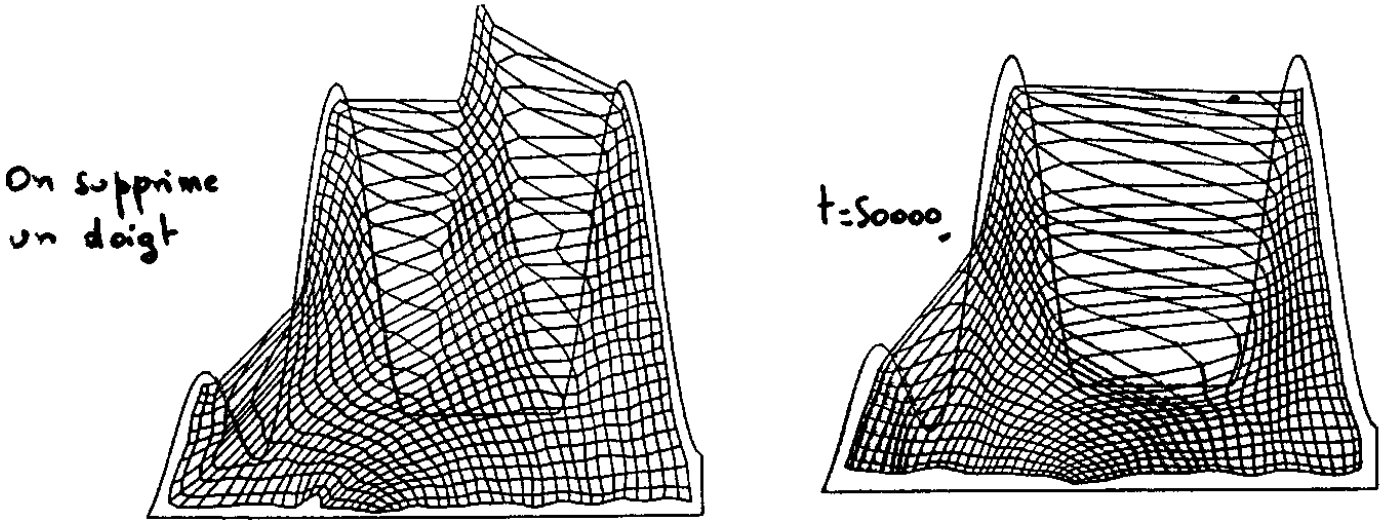


t=500





Adaptatif (Ritter et Schulten)



Étude Théorique

- On peut écrire

$$C(t+1) = C(t) + \varepsilon H(x(t+1), C(t))$$

- Algorithme dont la forme fait penser à un algorithme de gradient
- Mais en général (si la distribution des entrées est continue), **H ne dérive pas d'un potentiel** (Erwinn). L'algorithme on-line SOM n'est pas un algorithme de gradient.
- **On se restreint ici au cas où les entrées sont listées en nombre fini. Alors, il existe une fonction potentiel** qui est (cf Ritter et al. 92) la somme des carrés intra classes étendue
- **Dans ce cas, l'algorithme minimise la somme des carrés des écarts de chaque observation non seulement à son vecteur code, mais aussi aux vecteurs codes voisins (dans la structure fixée)**

Somme des carrés intra (rappel)

- L'algorithme SCL (0-voisin) est exactement l 'algorithme de gradient stochastique associé à la distorsion quadratique (ou somme des carrés intra)

$$D(C) = \sum_{i \in I} \int_{A_{i(x)}} \left\| x - C_i \right\|^2 f(x) dx$$

- estimée par

$$\hat{D}(C) = \frac{1}{N} \sum_{i=1}^{n} \sum_{x \in A_i} \left\| x - C_i \right\|^2$$

Somme des carrés intra-classes étendue aux classes voisines

- **Extension de la notion de somme des carrés intra-classes, qui est étendue aux classes voisines**

$$D_{SOM}(C) = \sum_{i=1}^{n} \sum_{\substack{x \text{ t.q. } i = i_0(x) \\ \text{ou } i \text{ voisin de } i_0(x)}} \|x - C_i\|^2$$

- En fait cette fonction a de nombreux minima locaux
- L'algorithme converge, moyennant les hypothèses classiques (Robbins-Monro) sur les ε, qui doivent décroître ni trop, ni trop peu
- ***La démonstration mathématique complète n'est faite que pour des données de dimension 1 et pour une structure de voisinage en ficelle***
- Pour accélérer la convergence, on prend au début une taille de voisinage assez grande et on la fait décroître

Équation différentielle ordinaire associée ODE

- L'ODE s'écrit

$$\frac{dC(i,u)}{du} = -\sum_{j \in I} \sigma(i,j) \int_{A_j(C(.,u))} (C(i,u) - x)\, \mu(dx$$

- où $C(i,t)$ remplace $C_i(t)$
- $C(.,t)$ est pour $(C_i(t), i \in I)$
- μ est la distribution des données x
- $A_i(C) = \{x \ / \ ||C_i - x|| = \min_j ||C_i - x|| \}$ est la i-ème classe formée des données pour lesquelles C_i est le gagnant. Ces classes dépendent des valeurs courantes de tous les vecteurs-codes. Elles forment une mosaïque de Voronoï.

Points fixes de l'ODE

- Si l'algorithme converge, ce sera vers un point d'équilibre de l'ODE, tel que

$$\forall i \in I, \sum_j \sigma(i,j) \int_{A_j(x^*)} (C^*(i) - x)\mu(dx) = 0$$

- i.e.

$$C^*(i) = \frac{\sum_j \sigma(i,j) \int_{A_j(C^*)} x \ \mu d(x)}{\sum_j \sigma(i,j) \mu(A_j(C^*))} \quad (1)$$

- Pour chaque i, $C^*(i)$ est le centre de gravité de toutes les classes, pondérés par les valeurs de la fonction $\sigma(i,j)$, $j \in I$

L'algorithme Batch

- On définit un processus déterministe pour calculer les solutions C^*
- On choisit C^0 et on pose pour chaque composante i

$$C^{k+1}(i) = \frac{\sum_j \sigma(i,j) \int_{A_j(C^k)} x \; \mu(dx)}{\sum_j \sigma(i,j) \, \mu(A_j(C^k))}$$

- Quand la mesure μ ne charge qu'un nombre fini N de données, le processus déterministe s'écrit

$$C_N^{k+1}(i) = \frac{\sum_j \sigma(i,j) \sum_{l=1}^{N} x_l \mathbb{1}_{A_j(C^k)}(x_l)}{\sum_j \sigma(i,j) \sum_{l=1}^{N} \mathbb{1}_{A_j(C^k)}(x_l)} \qquad (2)$$

L'algorithme Batch

- Si $N \to \infty$, en posant

$$\mu_N = \frac{1}{N} \sum_{I=1}^{N} \delta_{x_I}$$

- si μ_N converge faiblement vers μ, on a

$$\lim_{N \to \infty} \lim_{k \to \infty} C_N^{k+1}(i) = C^*(i)$$

où C^* est solution de (1)

L'algorithme (2) est l'algorithme de Kohonen Batch (KBATCH). C'est une simple extension de l'algorithme de Forgy. A chaque étape la mise à jour consiste à calculer les centres de gravité des classes pondérées par la fonction de voisinage.

Algorithme Quasi-Newtonien

- Même si *D* n'est pas partout différentiable et ne permet pas de prouver rigoureusement la convergence de l'algorithme on-line, il est intéressant de la contrôler au cours des itérations.
- On sait que l'algorithme de Forgy (algorithme batch dans le cas de 0 voisin) minimise la distorsion classique $D(x)$
- Dans le cas général (Algorithme KBATCH), on a

$$C_N^{k+1} = C_N^k - diag\nabla^2 D_{SOM}(C_N^k))^{-1}\nabla D_{SOM}(C_N^k$$

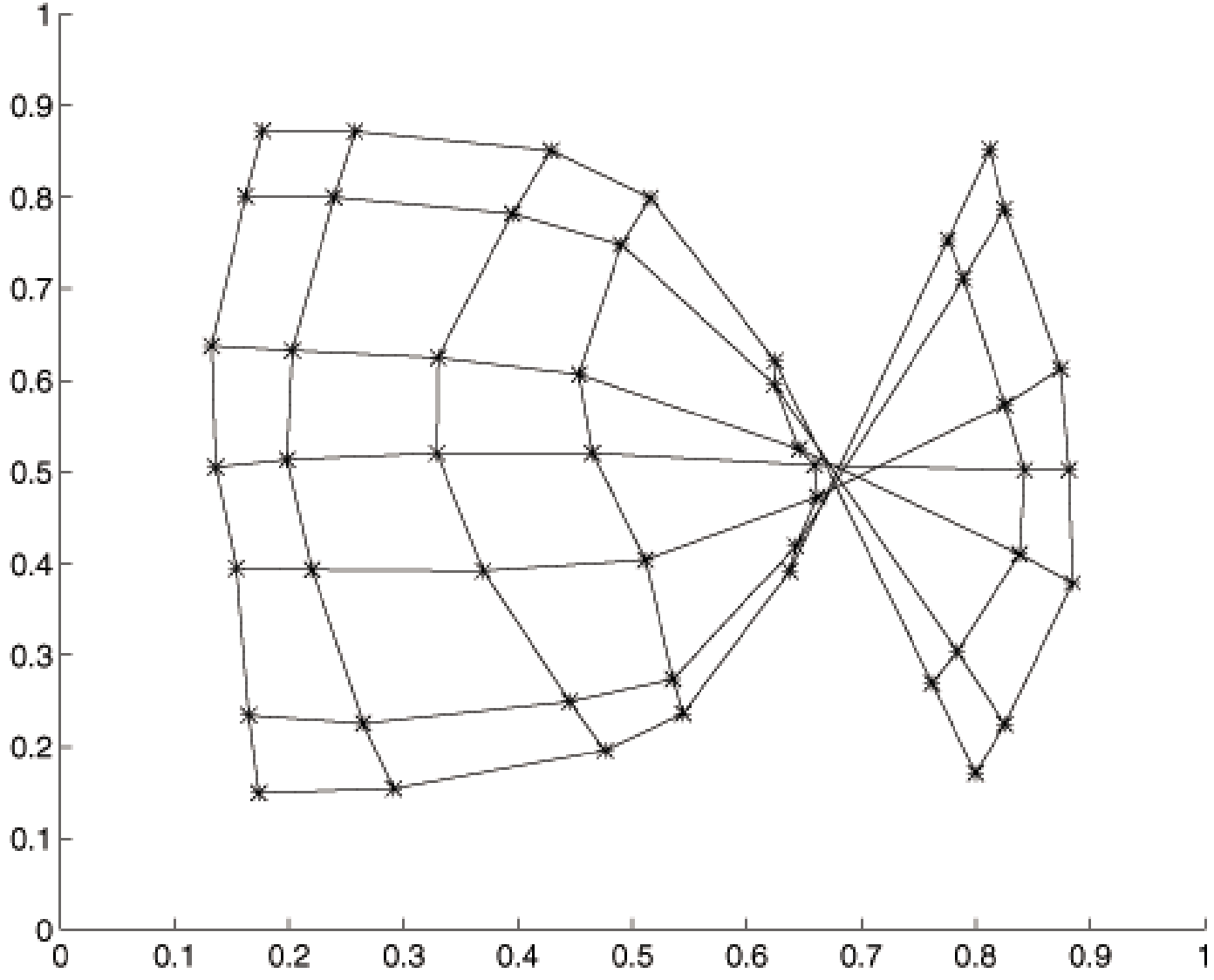
C'est-à-dire que l'algorithme Batch est un algorithme quasi-Newtonien associé à la distorsion étendue (si et seulement s'il n'y a pas de données sur les bords des classes).

Comparaison sur données simulées en 2D

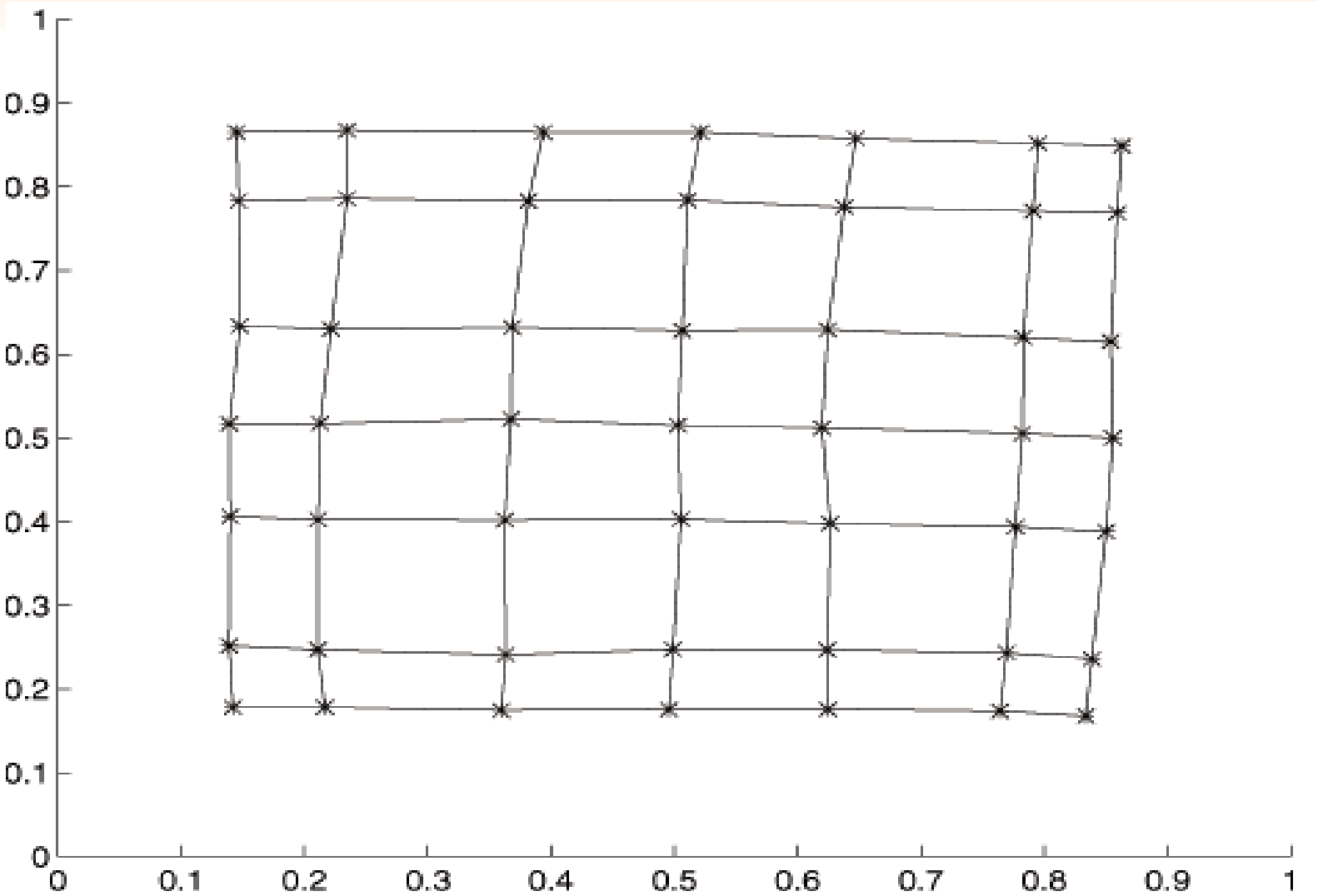
- On prend une grille 7 par 7, et une autre 10 par 10 (avec un système de voisinages fixe de 9 voisins) pour étudier
  - l'algorithme Kohonen batch, avec 100 itérations
  - l'algorithme on-line SOM, avec 50 000 itérations (i.e. équivalent)
- Les données sont uniformément distribuées dans un carré
- On choisit les mêmes valeurs initiales pour les deux algorithmes
- On observe que l'algorithme SOM trouve de meilleures solutions

(Dessins de Jean-Claude Fort)

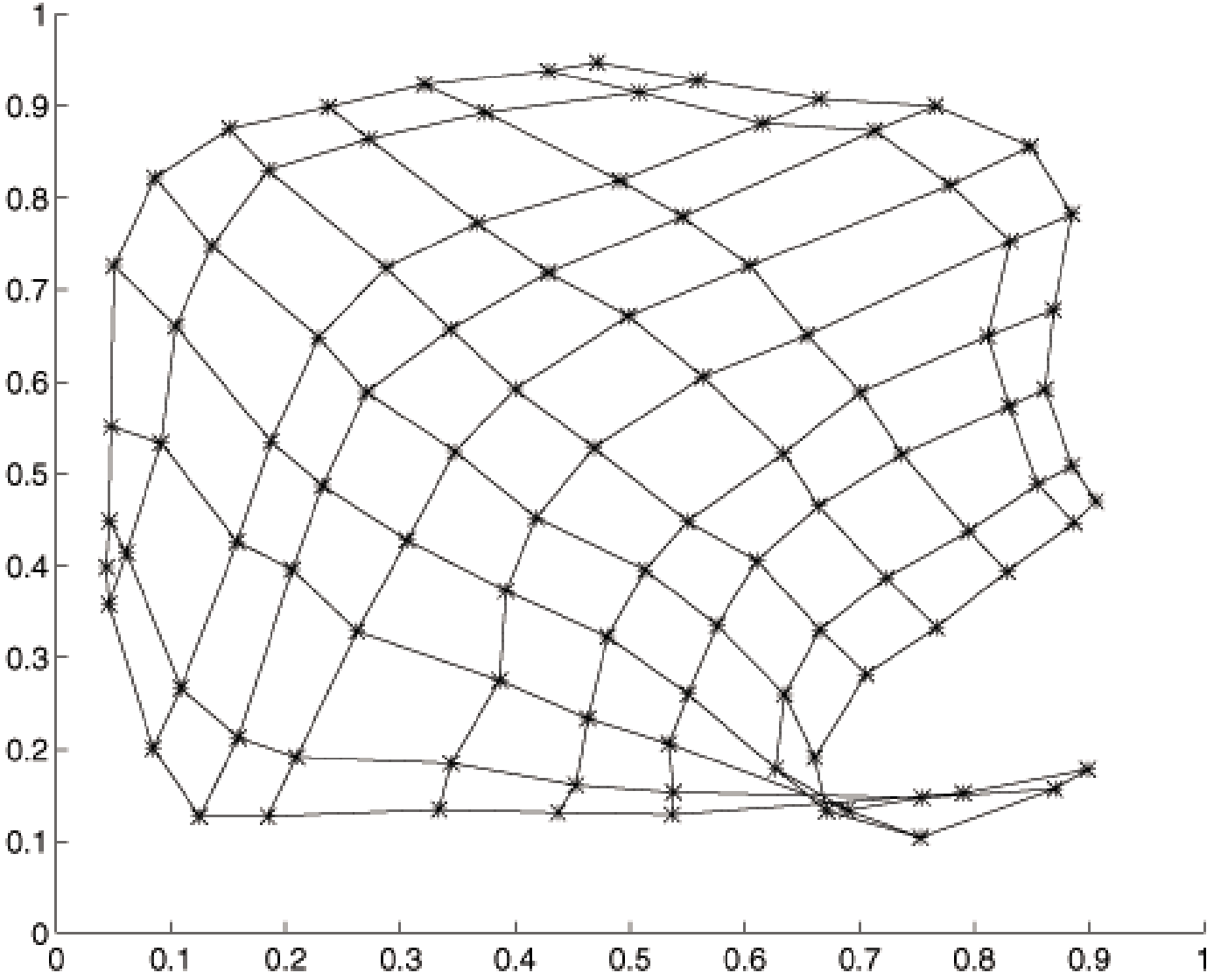
Algorithme batch pour des données uniformes sur une grille 7×7



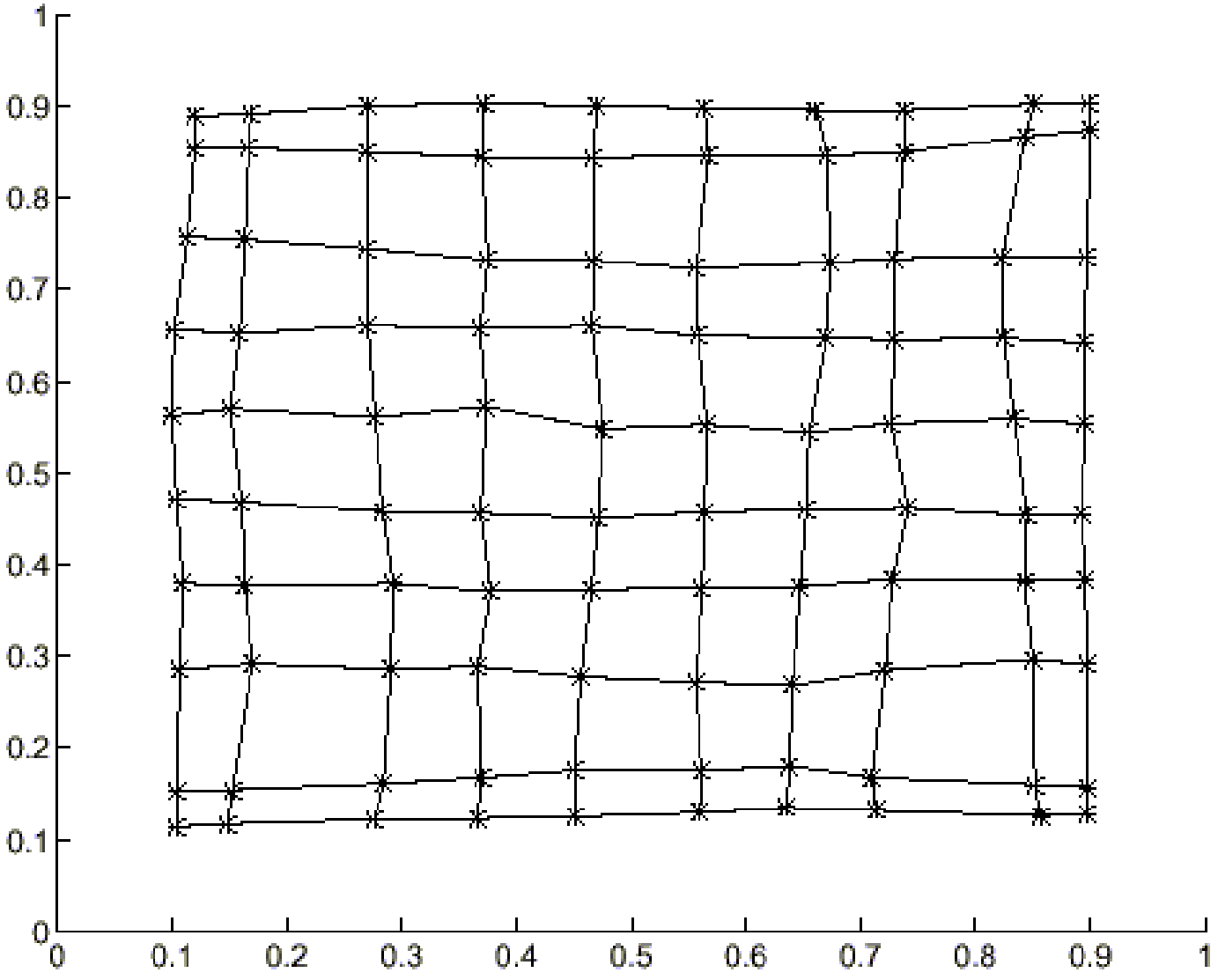
Algorithme on-line SOM pour des données uniformes sur une grille 7×7



Algorithme batch pour des données uniformes sur une grille 10×10



Algorithme on-line SOM pour des données uniformes sur une grille 10×10



Algorithme Batch (compléments)

- Autres exemples sur le site du SAMOS
- Voir Conférence NNSP Falmouth 2001, ESANN 2002, SFC 02, etc…

Algorithme de Kohonen
Applications

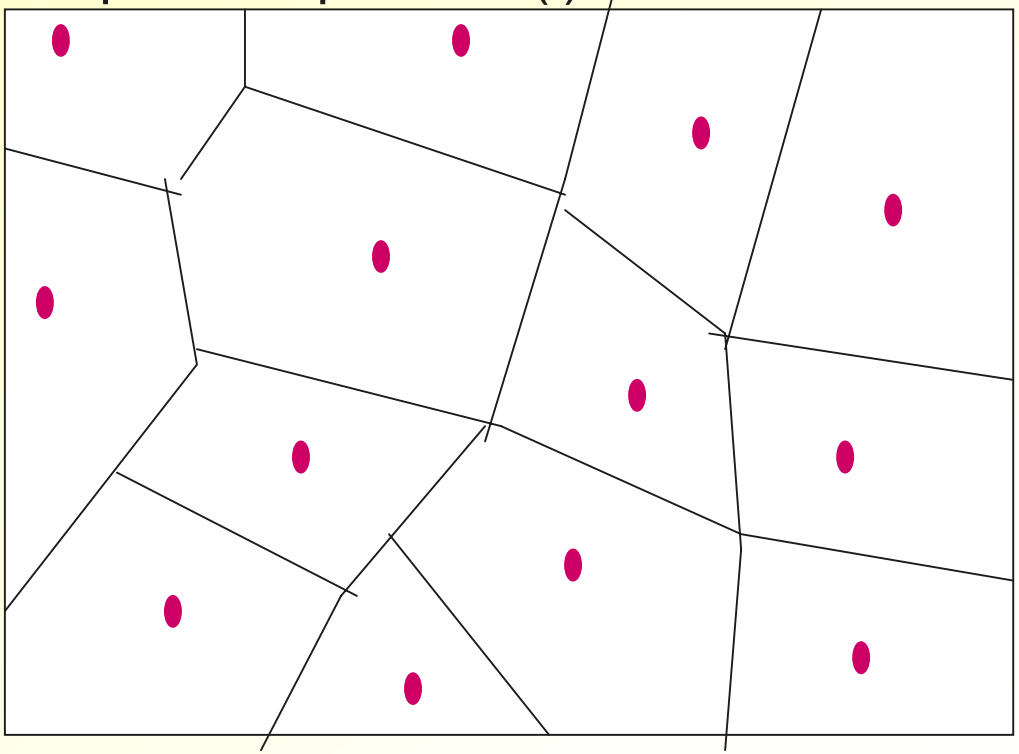
- CES LIMITATIONS N'EMPECHENT PAS LA RICHESSE DES APPLICATIONS :
- CARTOGRAPHIE (cf ACP)
- ANALYSE DES TABLEAUX DE CONTINGENCE (AFC)
- ETUDE DE DONNEES QUALITATIVES
- CALCUL NUMERIQUE D'INTEGRALE (Pagès)
- MAILLAGE
- CLASSIFICATION DE COURBES OU DE FORMES
- etc...etc...

Nombreuses applications (plusieurs milliers sur le site de Kohonen)

- Représentation des pays, (Blayo et Letremy)
- Communes d'Ile-de France, (Ibbou, Tutin)
- Courbes de consommation, prévision, (Rousset)
- Consommation au Canada, (Gaubert, Gardes, Rousset)
- Segmentation du marché du travail (Gaubert)
- Démographie et composition sociale dans la vallée du Rhône, (Letremy, P.A.R.I.S)
- Etude sur le du leasing en Belgique, (de Bodt, Ibbou)
- Profils de chocs de taux d'intérêts, (de Bodt)
- Chômeurs récurrents, (Gaubert)
- Niveau de vie des ménages (Ponthieux)
- Dépenses de formations en entreprise (Perraudin, Petit, Lémière), ...
- Géopolitique européenne (Perraudin)

Les classes : Mosaïque de Voronoï

- *Dans l 'espace des entrées, les classes forment une partition, ou mosaïque de Voronoï, dépendant des C.*
- $A_i(C) = \{x \ / \ ||C_i - x|| = \min_j || C_j - x || \}$: *i*-ème classe formée des données pour lesquelles $C(i)$ est le vecteur code gagnant.



x appartient à $A_i \Leftrightarrow$ l'unité i gagne quand on présente x

Cartes de Kohonen : Classification

- Pour représenter des données au moyen de l'algorithme de Kohonen, on prend comme entrées les lignes de la matrice des données
- Après apprentissage, chaque individu (ligne) correspond à une unité du réseau (celle qui gagne quand on présente cet individu)
- ***On classe une observation dans la classe A_i définie par l'unité gagnante qui lui correspond ($i=i_0(x)$)***
- On obtient donc une classification des individus, avec respect des voisinages
- La carte ainsi obtenue fournit une représentation plane
- Ici l'existence de proximités entre classes qui se ressemblent est essentielle

Représentation (KACP)

- Dans chaque classe on peut représenter le vecteur code
  - en donnant ses *P* composantes
  - en dessinant une courbe à *P* points

- Dans chaque classe, on peut
  - faire la liste des observations de cette classe
  - représenter en superposition les observations de la classe

- Ceci fournit une **représentation plane**, analogue à l'analyse en composantes principales (mais une seule carte et pas de projection orthogonale)

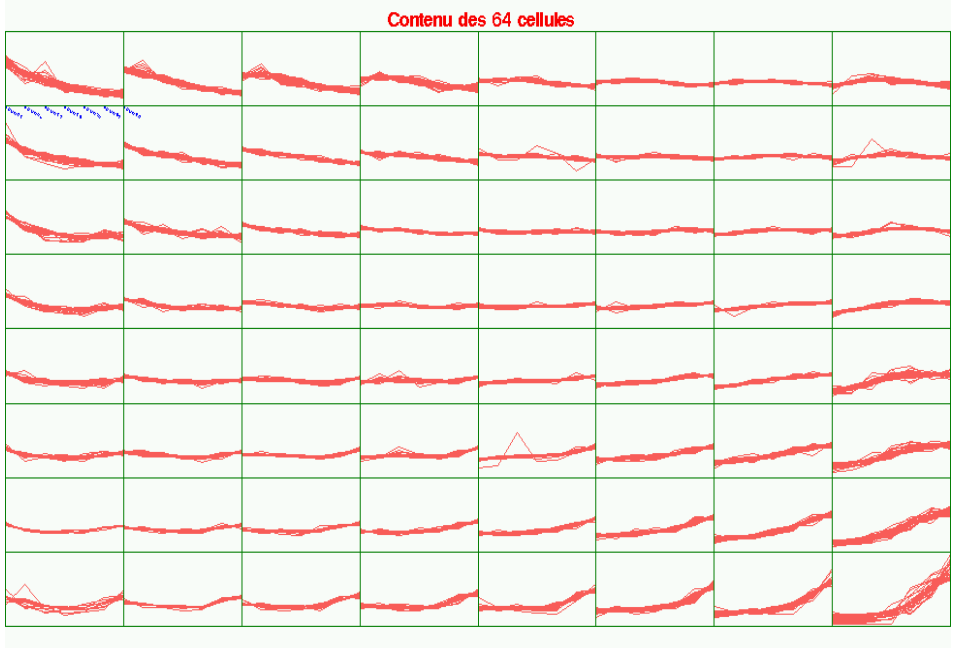
Classes et distances

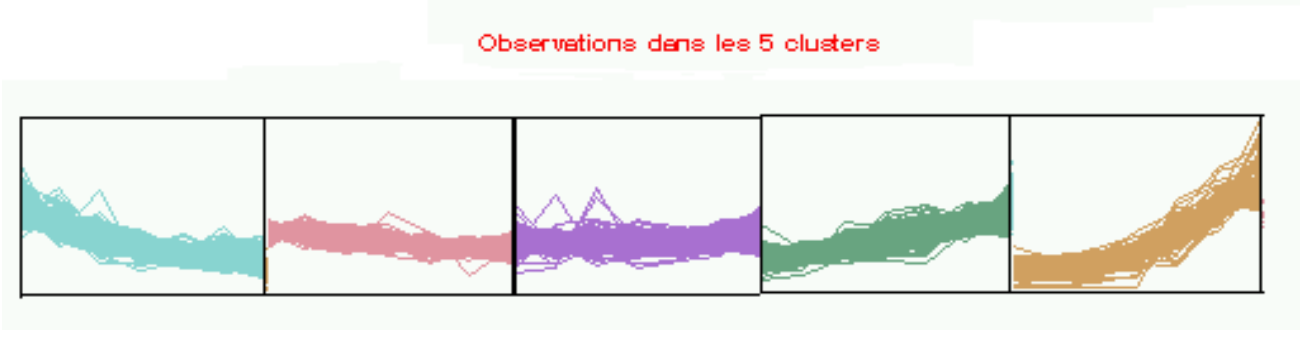
- Comme le nombre de classes est fixé a priori assez grand, il est utile de procéder à une regroupement
- On fait une classification hiérarchique sur les vecteurs codes, ce qui définit des super-classes
- On ***colorie ces super-classes*** (cf. classification mixte)

- On peut visualiser les distances entre les classes de Kohonen, car la disposition sur la grille donne une impression fausse d'équidistance

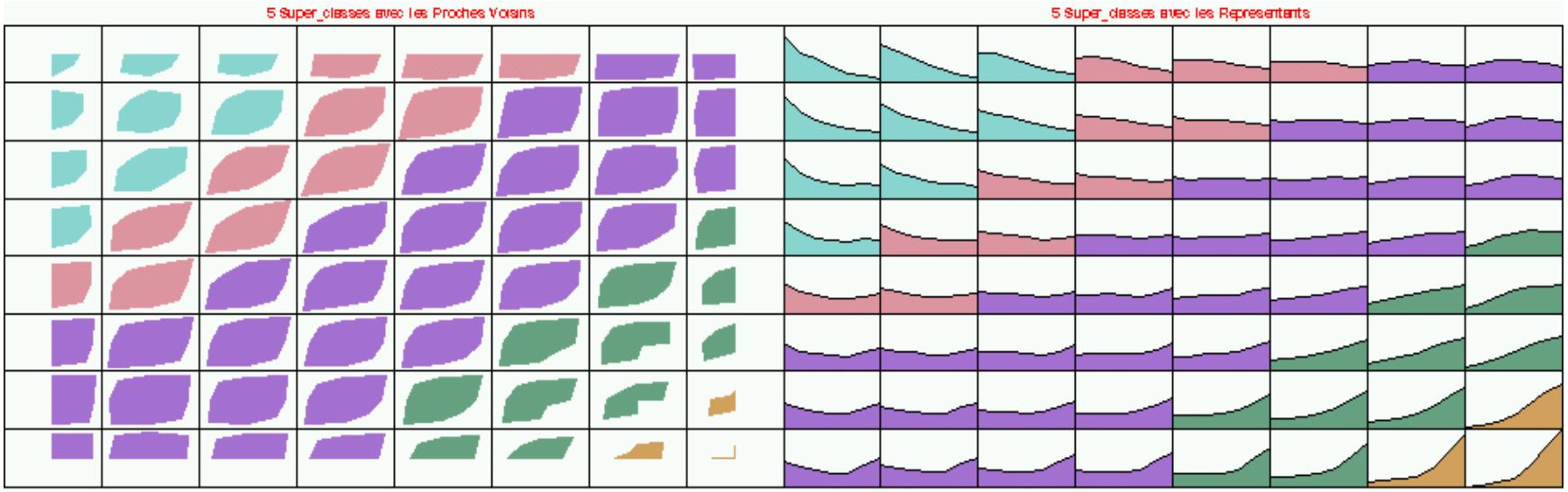
- Plus il y a du blanc entre deux classes (dans les 8 directions), plus la distance est grande

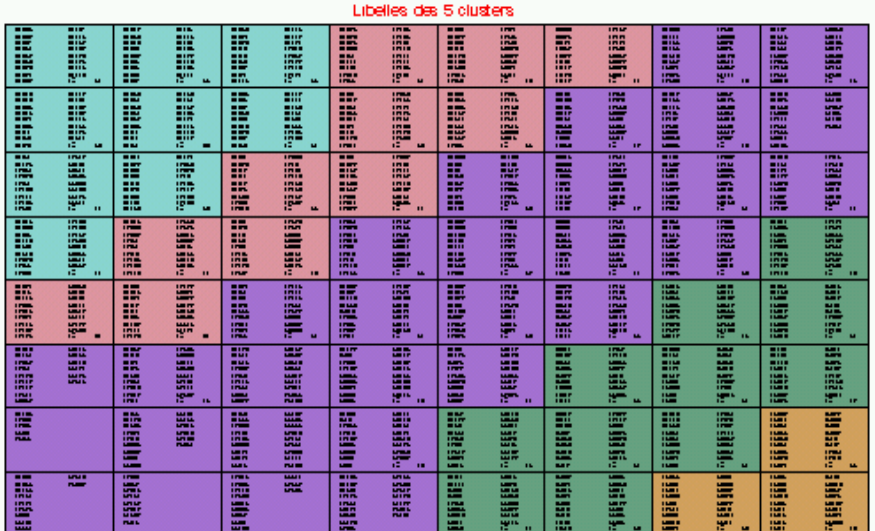
Variables quantitatives (communes de la vallée du Rhône, 7 recensements)
Contenu des classes et des super-classes



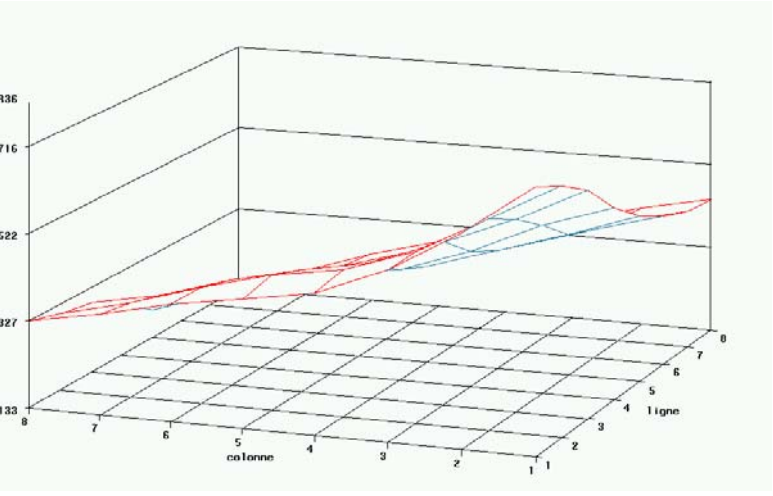


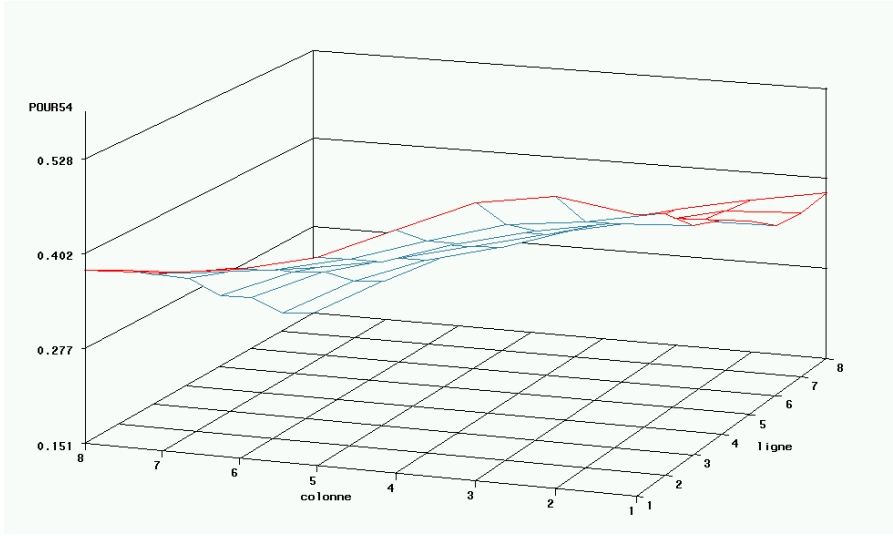
Variables quantitatives (communes de la vallée du Rhône)
Classes, super-classes et distances

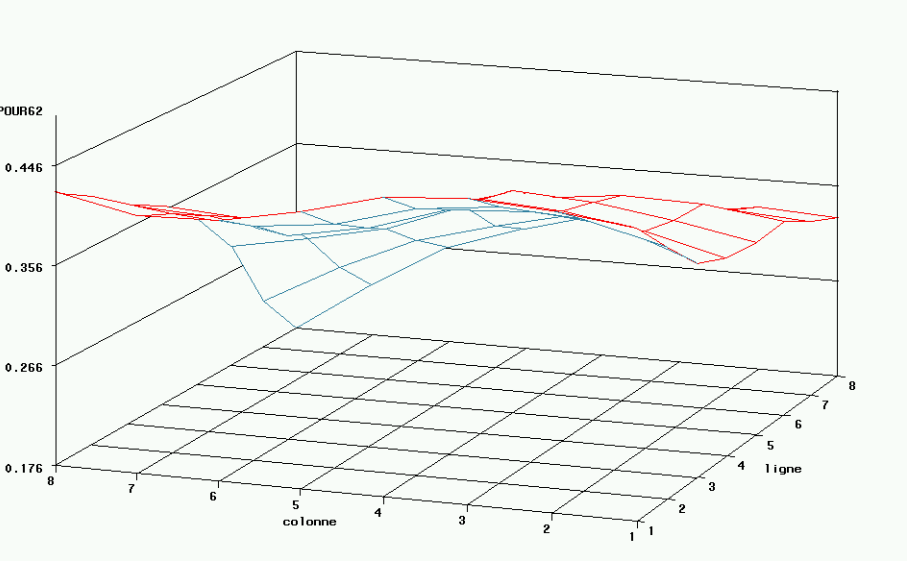


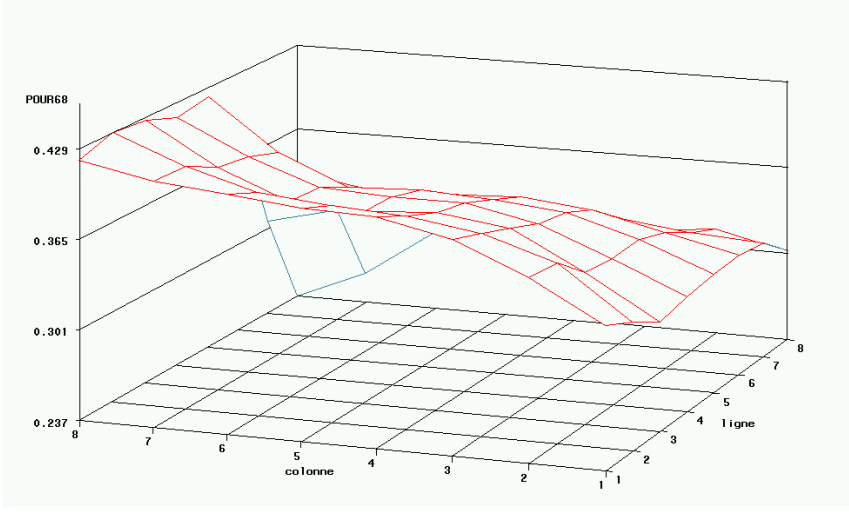


Les quatre premiers recensements

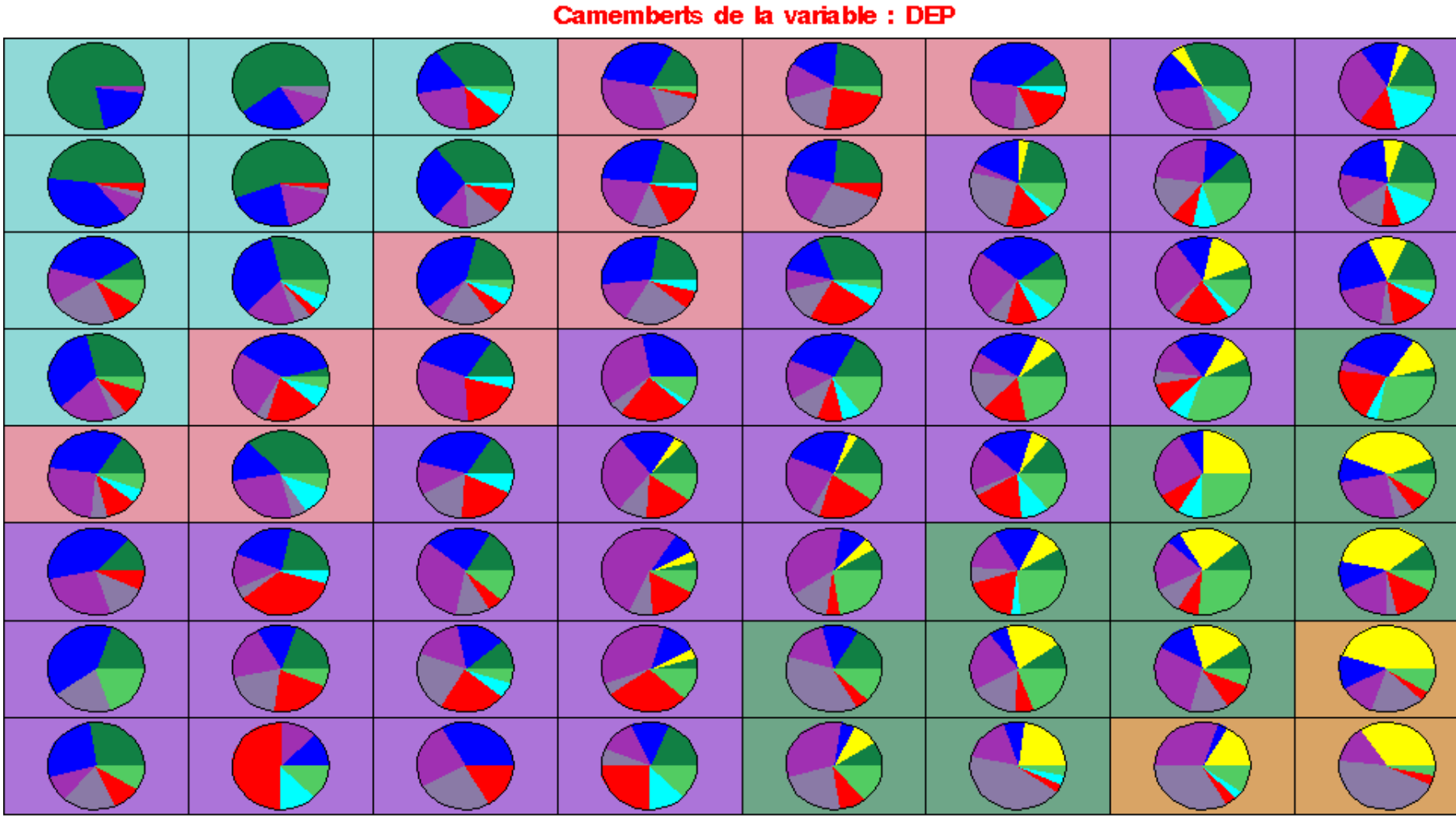








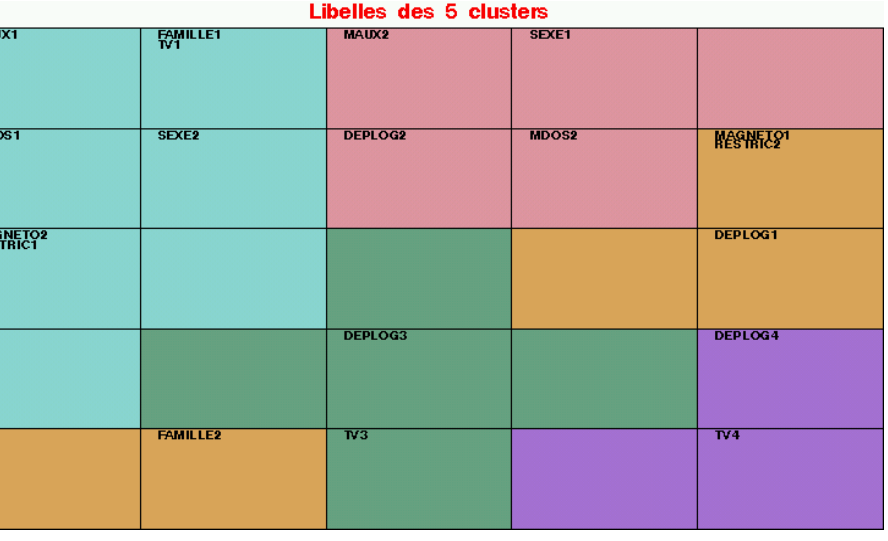
Par département

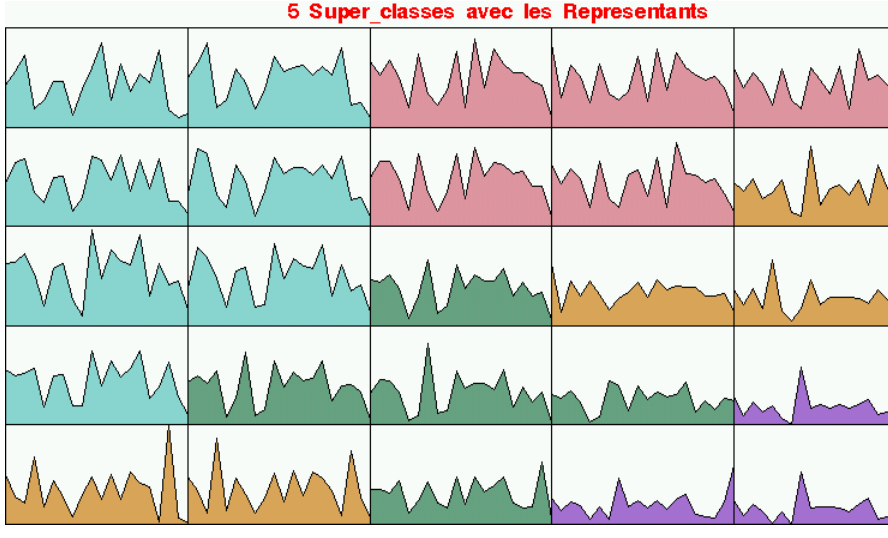


Départements par super-classes

- Ardèche (07)
- Bouches-du-Rhône (13)
- Drôme (26)
- Gard (30)
- Hérault (34)
- Isère (38)
- Haute-Loire (42)
- Vaucluse (84)

Variables qualitatives (exemple de la famille)
Classes et super classes pour les modalités





Conclusion

- C'est un très bon outil
  - de classification (accélération des méthodes type centres mobiles)
  - de visualisation en raison de la conservation des voisinages
  - de complément des méthodes factorielles classiques
- On peut combiner méthodes classiques et l'algorithme de Kohonen :
  - **KACP sur les coordonnées obtenues après une ACM**
  - **ACM (ou KACM, ou KDISJ) sur des variables qualitatives en y rajoutant une variable de classe obtenue par un KACP**
- On obtient directement des **scores** si on classe sur une ficelle
- On peut s'en servir en **prévision** en segmentant l'espace et en utilisant un modèle par segment (pré-traitement avant l'usage d'un perceptron ou d'un modèle auto-régressif)
- Outil de **prévision de courbes**, avec la même précision en chaque point de la courbe (au contraire des méthodes usuelles)

Conclusion

- Facilité de travail avec des **données manquantes** (cf thèse de Smaïl Ibbou) : les distances sont calculées sur les composantes présentes dans les observations
- Les données manquantes peuvent être **estimées** par les composantes correspondantes du vecteur code de la classe de l ’observation
- Utile pour une **initialisation accélérée** de toute autre méthode de quantification
- Application développée par T.Kohonen : aide à la recherche de mots clés dans de grands textes (WEB)

Références

- Beaucoup d'applications
- Les programmes de Patrick Letrémy

sont disponibles sur le site du SAMOS

http://samos.univ-paris1.fr