



# Apprentissage Supervisé

Stephan Cléménçon,  
[stephan.clemencon@telecom-paristech.fr](mailto:stephan.clemencon@telecom-paristech.fr)

Telecom ParisTech, Paris, France



# Sommaire

# Intervenants

## • Stephan Cléménçon (Telecom ParisTech - STA)

- Contact : [stephan.clemencon@telecom-paristech.fr](mailto:stephan.clemencon@telecom-paristech.fr)
- Profil : Enseignement/Recherche/Conseil/Industrie
- Web : <http://www.tsi.enst.fr/~clemenco/>
- Mots-clés : processus stochastiques (markoviens, empiriques, etc.), apprentissage statistique, applications : finance, high tech, biosciences

# Session 2 - Apprentissage Supervisé

- **Algorithmes de Classification**

Moyennes locales, arbres de classification

Perceptron, SVM et réseaux de neurones

Méthodes ensemblistes : bagging, boosting et forêts aléatoires

Classification multi-classe

- **Algorithmes de Régression**

Adapter les algorithmes de classification à la régression  
(arbres de régression, etc.)

- **Autres Problèmes Supervisés**

Régression ordinaire, ranking

# Cadre générique - apprentissage supervisé

- Couple de v.a. =  $(X, Y) \sim P$  inconnue
- $X$  = vecteur d'entrée à valeurs dans  $\textcolor{blue}{X}(\mathbb{R}^d)$ , ici  $d \gg 1$
- $Y$  = label/étiquette dans  $\textcolor{blue}{Y} \subset \mathbb{R}$
- A priori,  $X$  modélise une information utile pour prédire  $Y$   
Règle prédictive :  $g : \textcolor{blue}{X} \rightarrow \textcolor{blue}{Y}$  choisie dans une classe  $\mathcal{G}$   
(e.g. prédicteur linéaire  $g(x) = {}^t \beta x + \alpha$ )
- Fonction de perte :  $\ell : \textcolor{blue}{Y} \times \textcolor{blue}{Y} \rightarrow \mathbb{R}^+$
- Risque (inconnu !) = Erreur de généralisation

$$L(g) = \mathbb{E}(\ell(Y, g(X)))$$

à minimiser sur  $g \in \mathcal{G}$ .

- Données =  $D_n = \{(X_1, Y_1), \dots, (X_n, Y_n)\} \stackrel{i.i.d.}{\sim} P$

# Sommaire

# Classification binaire

- Exemples : Prédition de l'état d'un système (normal vs anormal), ciblage commercial, diagnostic médical, etc.
- $Y = \{-1, +1\}$
- Fonction de perte :

$$\ell(y, z) = \mathbb{I}\{y \neq z\}$$

- Risque d'erreur :

$$L(g) = \mathbb{P}\{Y \neq g(X)\}$$

$$= \mathbb{P}\{Y \cdot g(X) < 0\} = \mathbb{E}(\mathbb{I}\{-Y \cdot g(X) > 0\})$$

# Approches Paramétriques

-

## Rappels

# Régression logistique

- Modélisation explicite de

$$\eta(x) = \mathbb{P}(Y = +1 \mid X = x) \in ]0, 1[$$

- Transformée **logistique** :  $f(x) = \text{logit } \eta(x) = \log\left(\frac{\eta(x)}{1-\eta(x)}\right)$

- Transformée inverse :  $\eta(x) = \frac{e^{f(x)}}{1+e^{f(x)}}$

- Supposons  $f \in \mathcal{F} = \{f_\theta(x); \theta \in \Theta\}$  avec  $\Theta \subset \mathbb{R}^d$

$$\eta_\theta(x) = \frac{e^{f_\theta(x)}}{1 + e^{f_\theta(x)}}$$

- Ex : régression logistique **linéaire**

$$f(x) = \alpha + {}^t \beta \cdot x, \quad \theta = (\alpha, \beta)$$

- Maximiser la **log-vraisemblance**

$$l_n(\theta) = \sum_{i=1}^n \left\{ \frac{1+y_i}{2} \log(\eta_\theta(x_i)) + \frac{1-y_i}{2} \log(1 - \eta_\theta(x_i)) \right\}$$

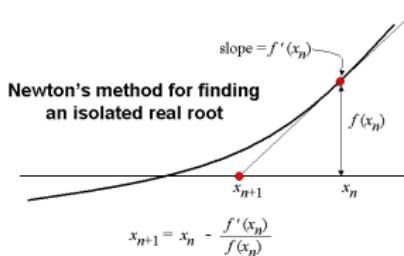
# Régression logistique

- Même dans le cas linéaire, l'équation de score

$$\nabla_{\theta} I_n(\theta) = 0$$

ne peut être résolue explicitement !

- Implémenter une méthode de Newton-Raphson



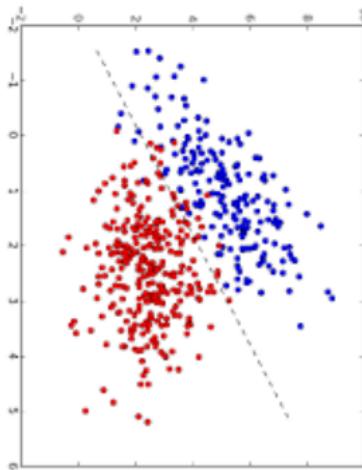
- Alternative : modèle probit  $\Phi^{-1}(\eta(X)) = \alpha + {}^t \beta X$

with  $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp(-\frac{t^2}{2}) dt.$

# Régression logistique linéaire

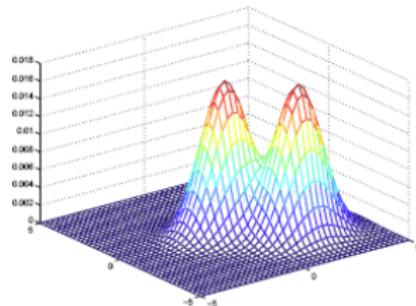
- Classifieur linéaire :  $\eta_\theta(x) \geq 1/2 \Leftrightarrow f_\theta(x) \geq 0$

$$\text{'Plug-in' } g(x) = \operatorname{sgn} \left( \hat{\alpha} + {}^t \hat{\beta} \cdot x \right),$$



# Analyse Discriminante Linéaire

- **Hypothèse** : les lois conditionnelles de  $X$  sachant  $Y = +1$ , sachant  $Y = -1$  sont **Gaussiennes** de même matrice de covariance  $\Gamma$  mais des moyennes distinctes  $\mu_+$  et  $\mu_-$ . Soit  $p = \mathbb{P}\{Y = +1\}$ .
- Estimer les moments d'ordre 1 et 2, puis le rapport de vraisemblance : **le label prédit est le label le plus probable sachant  $X$ .**



# Analyse Discriminante Linéaire

Au point  $X = (X^{(1)}, \dots, X^{(d)})$ , on prédit  $Y = +1$  si

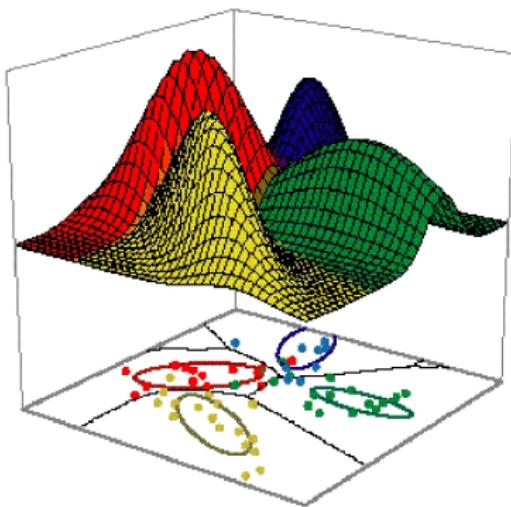
$$\log \left( \frac{\mathbb{P}\{Y = +1 | X\}}{\mathbb{P}\{Y = -1 | X\}} \right) > 0 \Leftrightarrow$$

$$\log\left(\frac{p}{1-p}\right) - \frac{1}{2}(\mu_+ - \mu_-)^t \Gamma^{-1}(\mu_+ - \mu_-) + x^t \Gamma^{-1}(\mu_+ - \mu_-) > 0$$

- On remplace  $\mu_+$ ,  $\mu_-$  et  $\Gamma$  par leurs versions statistiques
- Un classifieur '**plug-in**' linéaire
- $\neq$  régression logistique linéaire sauf si  $p = 1/2$

# Analyse Discriminante Linéaire - Extensions

- **Naïve Bayes** : on suppose que, sachant  $Y$ , les variables prédictives  $X^{(1)}, \dots, X^{(d)}$  sont indépendantes
- Extensions **non linéaires** : analyse discriminante quadratique (QDA), mélange de Gaussiennes
- L'extension au **cadre multiclasse** est immédiate



# Le Perceptron Monocouche

- L'espace d'entrée est divisé en deux régions par un **hyperplan affine**

$$g(x) = \operatorname{sgn}({}^t w \cdot X + \theta)$$

- L'algorithme de **Rosenblatt (1962)** pour minimiser

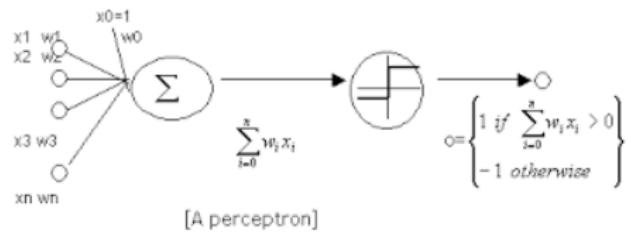
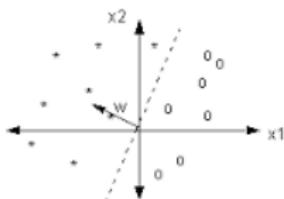
$$-\sum_i y_i ({}^t w \cdot x_i + \theta)$$

- Choisir au hasard un point mal classé par la règle courante  $(x_i, y_i)$
- Effectuer une descente de gradient à la vitesse  $\rho$

$$\begin{pmatrix} w \\ \theta \end{pmatrix} \leftarrow \begin{pmatrix} w \\ \theta \end{pmatrix} + \rho \begin{pmatrix} y_i x_i \\ y_i \end{pmatrix}$$

- Convergence si les données sont **linéairement séparables**

# Le Perceptron Monocouche



# Sommaire

# Approches Non Paramétriques

-  
Moyennes Locales et  
Arbres de Classification

## Une méthode nonparamétrique simple : les *K*-plus proches voisins

- Soit  $K \geq 1$ . On considère une **distance**  $d$  sur  $\mathbb{R}^D$ , (ex : distance euclidienne)
- En tout point  $x$ , soit  $\sigma = \sigma_x$  la permutation de  $\{1, \dots, n\}$  telle que

$$d(x, x_{\sigma(1)}) \leq \dots \leq d(x, x_{\sigma(n)})$$

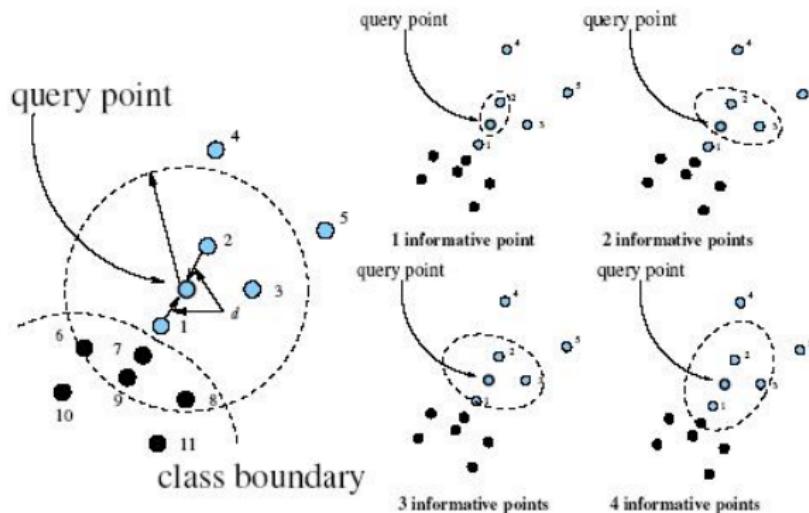
- Extraire les ***K*-plus proches voisins** de  $x$

$$\{x_{\sigma(1)}, \dots, x_{\sigma(K)}\}$$

- Vote à la majorité** :  $N_y = \text{Card}\{k \in \{1, \dots, K\}; y_{\sigma(k)} = y\}$ ,  $y \in \{-1, 1\}$

$$C(x) = \arg \max_{y \in \{-1, +1\}} N_y,$$

# Une méthode nonparamétrique simple : les $K$ -plus proches voisins



# Les $K$ -plus proches voisins

Consistance universelle (Stone '77)

Si  $k = k_n \rightarrow \infty$  et  $k_n = o(n)$ , la classifieur des  $K$ -plus proches voisins est consistant

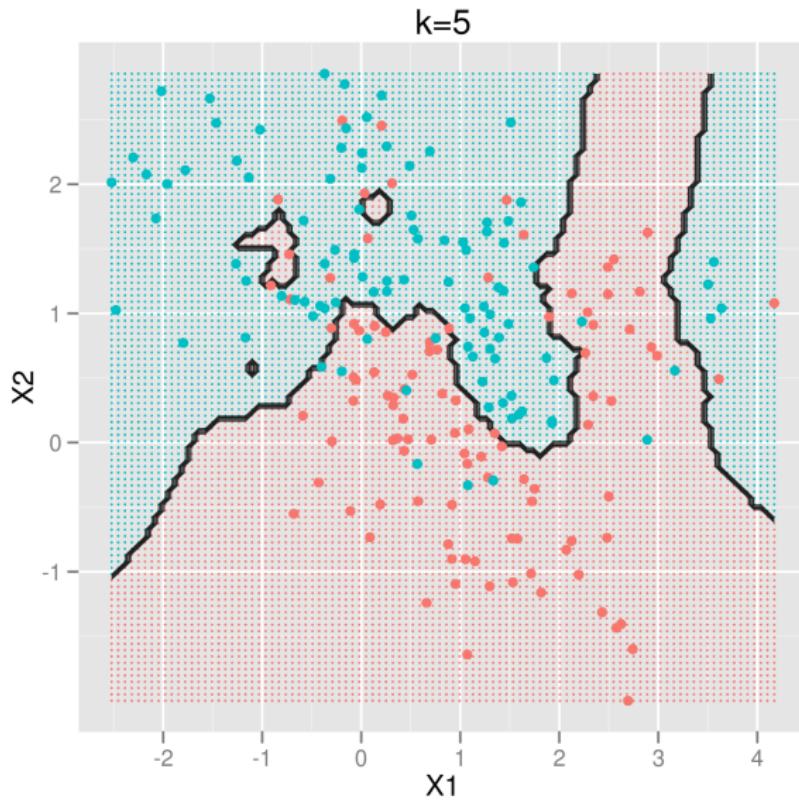
$$L(C_{K-NN}) - L^* \rightarrow 0, \text{ as } n \rightarrow \infty$$

Mais...

- La vitesse est **arbitrairement lente**
- Fléau de la dimension : ordonner les données est **coûteux en calcul**
- **Instabilité** : choix de  $K$  ? de la métrique  $D$  ?
- **Metric learning** (e.g. distance Mahalanobis distance)
- Variantes avec des **poids**



# Les $K$ -plus proches voisins Une méthode trop flexible ?



## Histogrammes - Moyennes Locales

- Les limites des  $K$ -plus proches voisins : le voisin le plus proche peut être très loin de  $X$  !
- Considérer une **partition** de l'espace d'entrée :

$$C_1 \bigcup \cdots \bigcup C_K = \mathcal{X}$$

- Appliquer la **règle majoritaire** : si  $X$  tombe dans  $C_k$ ,
  - ➊ Compter le nombre d'exemples d'apprentissage avec label positif dans  $C_k$
  - ➋ Si  $\sum_{i: X_i \in C_k} \mathbb{I}\{Y_i = +1\} > \sum_{i: X_i \in C_k} \mathbb{I}\{Y_i = -1\}$ , prédire  $Y = +1$ . Prédire  $Y = -1$  sinon.
- Cette règle correspond au **classifieur "plug-in"**  $2\mathbb{I}\{\hat{\eta}(x)\} - 1$ , où

$$\hat{\eta}(x) = \sum_{k=1}^K \mathbb{I}\{x \in C_k\} \frac{\sum_{i=1}^n \mathbb{I}\{Y_i = +1, X_i \in C_k\}}{\sum_{i=1}^n \mathbb{I}\{X_i \in C_k\}}$$

est l'**estimateur de Nadaraya-Watson estimator** de la probabilité a posteriori.

# Histogrammes - Moyennes Locales

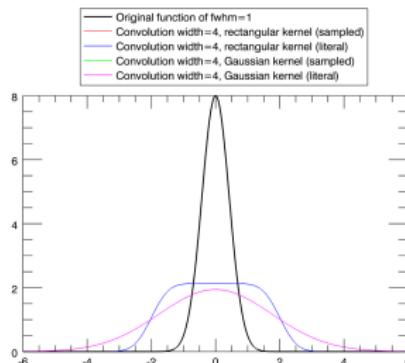
- **Lisser** l'estimateur, la région de décision !
- Remplacer la fonction indicatrice par un **noyau de convolution** :

$$K : \mathbb{R}^d \rightarrow \mathbb{R}_+, \quad K \geq 0, \text{ symétrique et } \int K(x)dx = 1$$

- Fenêtre  $h > 0$  et **mise à l'échelle**

$$K_h(x) = \frac{1}{h} K(x/h)$$

- Exemples : noyau Gaussian, de Novikov, de Haar, etc.



## Méthodes à noyaux - Moyennes Locales

- Si  $\sum_{i=1}^n \mathbb{I}\{Y_i = +1\} K_h(x - X_i) > \sum_{i=1}^n \mathbb{I}\{Y_i = -1\} K_h(x - X_i)$ , prédire  $Y = +1$ . Prédire  $Y = -1$  sinon.
- Cette règle correspond au **classifieur "plug-in"**  $2\mathbb{I}\{\tilde{\eta}(x)\} - 1$ , où

$$\tilde{\eta}(x) = \frac{\sum_{i=1}^n \mathbb{I}\{Y_i = +1\} K_h(x - X_i)}{\sum_{i=1}^n K_h(x - X_i)}$$

est l'**estimateur de Nadaraya-Watson estimator** de la probabilité a posteriori.

- **Argument statistique** : Si  $\eta$  est une fonction "régulière",  $\tilde{\eta}$  peut être un meilleur estimateur que  $\hat{\eta}$  (de plus faible variance mais... biaisé)

# Arbres de classification : l'algorithme CART

- Si la partition est donnée à l'avance (avant d'observer les données)...

## Arbres de classification : l'algorithme CART

- Si la partition est donnée à l'avance (avant d'observer les données)...  
de nombreuses cellules peuvent être vides !

# Arbres de classification : l'algorithme CART

- Si la partition est donnée à l'avance (avant d'observer les données)...  
de nombreuses cellules peuvent être vides !
- Choisir la partition **en fonction des données d'apprentissage** !

# Arbres de classification : l'algorithme CART

- Si la partition est donnée à l'avance (avant d'observer les données)...  
de nombreuses cellules peuvent être vides !
- Choisir la partition **en fonction des données d'apprentissage !**
- The CART Book - Breiman, Friedman, Olshen & Stone (1986)
- Un algorithme de partitionnement récursif **glouton** :  
 $X = (X^{(1)}, \dots, X^{(d)}) \in \mathbb{R}^d$

## Arbres de classification : l'algorithme CART

- Données d'apprentissage  $(X_1, Y_1), \dots, (X_n, Y_n)$
- Pour toute région  $R \subset \mathbb{X}$ , considérer le **label majoritaire** :  $\bar{Y}_R$ , où

$$\bar{Y}_R = +1 \text{ if } \sum_{i=1}^n \mathbb{I}\{Y_i = +1, X_i \in R\} > \frac{1}{2} \sum_{i=1}^n \mathbb{I}\{X_i \in R\}$$

et  $\bar{Y}_R = -1$  sinon

- On part du noeud racine  $R = \mathbb{X} = C_{0,0}$  et du classifieur constant  $\bar{Y}_{C_{0,0}}$ . Le but est de scinder la cellule  $C_{0,0}$

$$C_{0,0} = C_{1,0} \bigcup C_{1,1}$$

de façon à raffiner le classifieur courant et obtenir

$$g_1(x) = \bar{Y}_{C_{1,0}} \mathbb{I}\{x \in C_{1,0}\} + \bar{Y}_{C_{1,1}} \mathbb{I}\{x \in C_{1,1}\}.$$

## "Faire pousser l'arbre"

- La scission de  $C_{0,0} = \textcolor{blue}{X}$  est effectuée de manière à minimiser  $\widehat{L}_N(g_1)$ , ou de façon équivalente la *mesure d'impureté*

$$\sum_{i=1}^N \mathbb{I}\{X_i \in C_{1,0}, Y_i \neq \bar{Y}_{C_{1,0}}\} + \mathbb{I}\{X_i \in C_{1,1}, Y_i \neq \bar{Y}_{C_{1,1}}\}$$

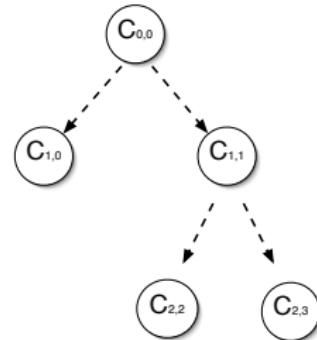
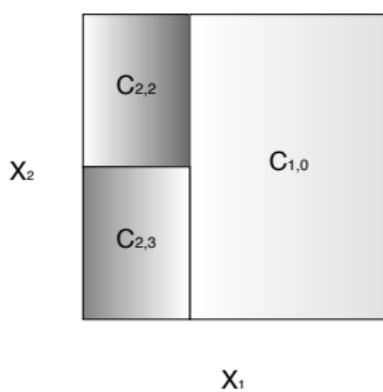
- On considère des régions de la forme

$$C_{1,0} = C_{0,0} \cap \{X^{(j)} \leq s\},$$

$$C_{1,1} = C_{0,0} \cap \{X^{(j)} > s\}.$$

- Il est suffisant de choisir les meilleurs seuils de scission parmi les valeurs  $X_i^{(j)}$  !

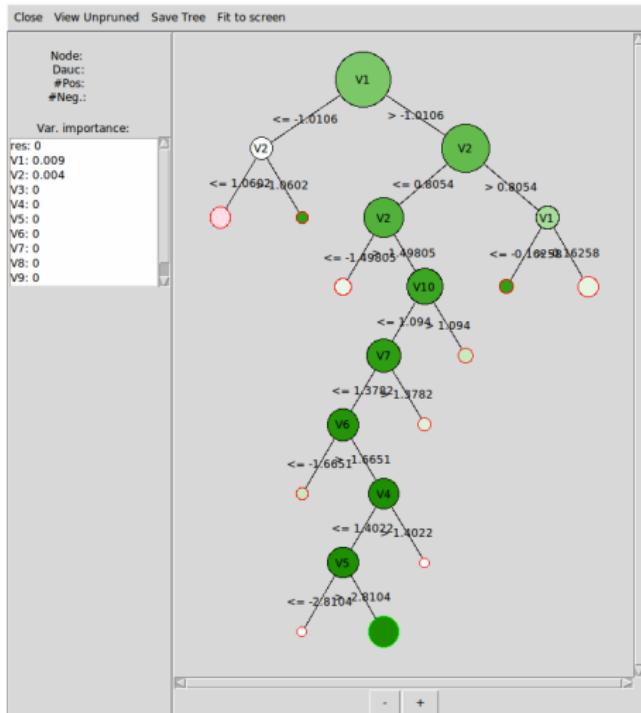
# "Faire pousser l'arbre"



## "Faire pousser l'arbre"

- Afin de scinder la cellule  $C_{j,k}$ , si elle n'est pas pure et contient au moins  $n_{\min}$  données d'apprentissage, itére rla double boucle :
  - ➊ De  $j = 1$  à  $d$ , trouver  $s$  (meilleur seuil de scission pour  $X^{(j)}$ ) de manière à minimiser l'impureté des régions
$$C_{j,k} \cap \{X_j > s\} \text{ and } C_{j,k} \cap \{X_j \leq s\}$$
  - ➋ Trouver la meilleure variable de scission  $X^{(j)}$
- Mesures d'**impureté** :
  - erreur de classification
  - indice de Gini
  - entropie

# Arbres de classification : l'algorithme CART



# Arbres de classification : l'algorithme CART

- Interprétabilité, visualisation
- Variables qualitatives
- Données incomplètes
- Quantification de l'importance relative des variables prédictives
- Randomisation
- Scissions diagonales
- Asymétrisation de l'erreur/impureté
- Extension au cadre multiclasse, à la régression
- Sélection de modèle : "meilleur sous-arbre, " élagage" rapide
- Algorithme alternatif : C4.5 (Ross Quinlan)

# Arbres de classification : l'algorithme CART

- Interprétabilité, visualisation
- Variables qualitatives
- Données incomplètes
- Quantification de l'importance relative des variables prédictives
- Randomisation
- Scissions diagonales
- Asymétrisation de l'erreur/impureté
- Extension au cadre multiclasse, à la régression
- Sélection de modèle : "meilleur sous-arbre, " élagage" rapide
- Algorithme alternatif : C4.5 (Ross Quinlan)
- Mais... **performance prédictive moyenne et grande instabilité**

# Sommaire

# **Ensemble Learning**

---

**Bagging, Boosting et  
Forêts Aléatoires**



## En bref

- Ensemble Learning - Méthodes de Consensus
- Bagging - accroître la stabilité
- Boosting - "La meilleure technique sur l'étagère"
- "Le hasard fait bien les choses !" - les Forêts Aléatoires

## Méthodes de Consensus

- Au lieu d'ajuster un unique classifieur, combiner les prédictions d'un **ensemble** de classifieurs

$$C_1(X), \dots, C_M(X).$$

Amit et Geman (1997)

- Vote majoritaire :

$$\text{sign} \left( \sum_{m=1}^M C_m(X) \right)$$

- Variante - vote majoritaire pondéré :  $\alpha_i \geq 0$ ,  $\sum_i \alpha_i = 1$

$$\text{sign} \left( \sum_{m=1}^M \alpha_m C_m(X) \right)$$

- Extension au cadre multiclasse, à la régression
- Un vieux challenge : "ranking" et consensus

# Bagging - Agrégation Bootstrap

- Bootstrap **aggregating** technique - Breiman (1996)
- Appliquable à tout algorithme  $\mathcal{L}$
- A partir des données d'apprentissage  $\mathcal{D}_n$  :
  - Générer indépendamment  $B \geq 1$  échantillons bootstrap  $\mathcal{D}_n^{*(b)}$  (par tirage avec remise dans  $\mathcal{D}_n$ )
  - Pour  $b : 1$  à  $B$ , mettre en oeuvre l'algorithme  $\mathcal{L}$  à partir de  $\mathcal{D}_n^{*(b)}$ , produisant le classifieur  $C^{*(b)}$
  - Agréger les prédictions bootstrap en calculant le vote majoritaire :

$$C_{bag}(X) = sign \left( \sum_{b=1}^B C^{*(b)}(X) \right)$$

- Variante : si  $C^{*(b)}(X) = sign(f^{*(b)}(X))$ ,

$$\tilde{C}_{bag} = sign \left( \sum_{b=1}^B f^{*(b)}(X) \right)$$

## Bagging - Commentaires

- Le bagging peut **réduire significativement la variance** de procédures instables (ex : arbres de décision)
- La réduction de variance peut conduire à une erreur test moindre
- En régression :  $f_{bag}(x) = \mathbb{E}[f^*(x)]$  (espérance prise sur  $\mathcal{D}_n$ )

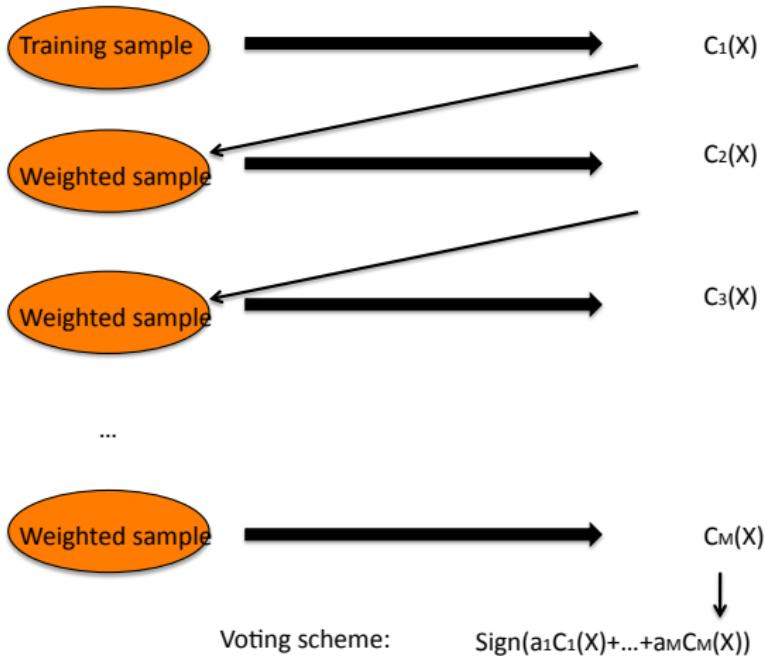
$$\begin{aligned}\mathbb{E} \left[ (Y - f^*(x))^2 \right] &= \mathbb{E} \left[ (Y - f_{bag}(x))^2 \right] \\ &\quad + \mathbb{E} \left[ (f_{bag}(x) - f^*(x))^2 \right] \geq \mathbb{E} \left[ (Y - f_{bag}(x))^2 \right]\end{aligned}$$

- En classification :  
L'agrégation bootstrap d'un bon classifieur l'améliore, mais ...  
celle d'un mauvais classifieur peut le détériorer encore !

# Boosting

- AdaBoost - Freund & Schapire (1995)
- L'ingrédient pour un "apprentissage lent", résistant au surapprentissage : une méthode de classification "faible"  $\mathcal{L}$
- Heuristique :
  - appliquer  $\mathcal{L}$  à des versions pondérées de l'échantillon original
  - accroître le poids des observations mal classées par la règle prédictive courante
  - agréger les classifieurs de façon non uniforme  
(un bon prédicteur ne devrait pas être construit à partir de quelques données aberrantes)
- AdaBoost surpassé ses concurrents sur la plupart des bases de données de référence
- Interprétation statistique : cinq ans plus tard...

# Boosting - Schéma général



# L'algorithme "Adaptive Boosting"

- Initialisation : poids uniformes,  $\omega_i = 1/n$  affectés à chaque exemple  $(X_i, Y_i)$ ,  $1 \leq i \leq n$
- De  $m : 1$  à  $M$ ,
  - Au moyen de l'algorithme  $\mathcal{L}$ , ajuster un classifieur faible  $C_m$  à partir de l'échantillon pondéré  $\{(X_i, Y_i, \omega_i) : 1 \leq i \leq n\}$
  - Calculer l'erreur de classification pondérée

$$err_m = \sum_{i=1}^n \omega_i \mathbb{I}\{Y_i \neq C_m(X_i)\}$$

- et  $a_m = \log((1 - err_m)/err_m)$
- Mettre à jour les poids :
    - $\omega_i \leftarrow \omega_i \exp(a_m \mathbb{I}\{Y_i \neq C_m(X_i)\})$
    - $\omega_i \leftarrow \omega_i / \sum_{j=1}^n \omega_j$
  - Sortie :  $C_{Boost}(X) = sign \left( \sum_{m=1}^M a_m C_m(X) \right)$

# AdaBoost résiste au surapprentissage !

- Classifieur faible typique : stumps (arbres de profondeur 1)
- Lorsque  $M$  croît, l'erreur test décroît et se stabilise



## Aspects pratiques

- Comment mettre en oeuvre  $\mathcal{L}$  à partir d'un échantillon pondéré ?
  - modifier le critère explicitement (ex : CART, SVM, k-NN, etc.)
  - tirer un échantillon d'apprentissage avec la distribution  $\sum_i \omega_i \delta_{(X_i, Y_i)}$
- Quand faut-il stopper les intéractions ?
  - tracer l'erreur test en fonction de  $M$
  - on stoppe lorsque l'erreur test se stabilise

# Une interprétation statistique du Boosting

- Friedman, Hastie & Tibshirani (2000)
- Stagewise forward additive modelling
- Perte exponentielle :  $C(X) = \text{sign}(f(X))$

$$L_e(f) = \mathbb{E}[\exp(-Yf(X))]$$

- Solution optimale :

$$f^*(X) = \frac{1}{2} \log \left( \frac{\eta(X)}{1 - \eta(X)} \right)$$

## Forward stagewise additive modelling

- Heuristique : raffiner la règle prédictive courante  $f_{m-1}(x)$  en ajoutant  $\alpha_m C_m(x)$ , avec  $\alpha_m \in \mathbb{R}$  et  $C_m(x) \in \{-1, +1\}$
- Comment choisir  $\alpha_m$  et  $C_m(x)$  de façon à minimiser le risque empirique exponentiel ?

$$\arg \min_{\alpha, C} \sum_{i=1}^n \exp(-Y_i(f_{m-1}(X_i) + \alpha C(X_i))) = ?$$

- Posons  $\omega_i = \exp(-Y_i f_{m-1}(X_i))$ , le risque empirique s'écrit alors :

$$\sum_{i=1}^n \omega_i \exp(-Y_i \alpha C(X_i))$$

- Quel que soit  $\alpha > 0$ , le classifieur de risque minimum est celui qui minimise le risque pondéré :

$$\sum_{i=1}^n \omega_i \mathbb{I}\{Y_i \neq C(X_i)\}$$

## Forward stagewise additive modelling

- Soit  $C_m(X)$  la solution de ce problème de classification pondérée :

$$err_m = \sum_{i=1}^n \omega_i \mathbb{I}\{Y_i \neq C_m(X_i)\}$$

- Il reste enfin à minimiser en  $\alpha$  :

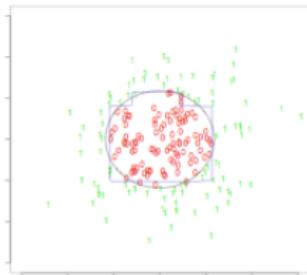
$$e^\alpha err_m + e^{-\alpha}(1 - err_m),$$

et obtenir  $\alpha_m = (1/2) \cdot \log((1 - err_m)/err_m)$

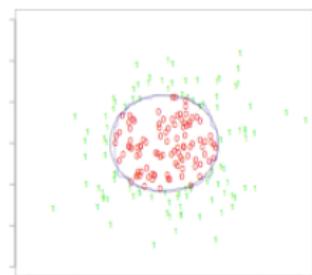
- Très nombreuses variantes : autres fonctions de perte, seuillage des poids, etc.

# L'agrégation produit des régions "régulières"

Decision Boundary: Tree



Decision Boundary: Boosting



When the nested spheres are in  $R^{10}$ , CART™ produces a rather noisy and inaccurate rule  $\hat{C}(X)$ , with error rates around 40%.

Bagging and Boosting average many trees, and produce smoother decision boundaries.

# Forêts Aléatoires

- Ingrédients : bagging + randomisation
- Randomiser la collection de variables prédictives (*i.e.* les composantes de  $X$ ) : avant de scinder chaque noeud d'un arbre de décision bootstrap
- Classifieur faible typique : arbre de faible profondeur, sans élagage
- L'agrégation préserve la consistance...  
mais aucune explication théorique de la performance observée !
- Heuristique : la randomisation "enrichit" la règle
- Randomisation des données d'apprentissage lorsqu'elles sont massives

# Sommaire

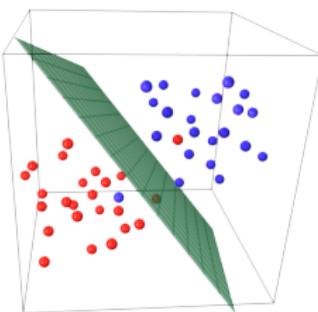
# Séparateur linéaire

## Définition

Soit  $\mathbf{x} \in \mathbb{R}^P$

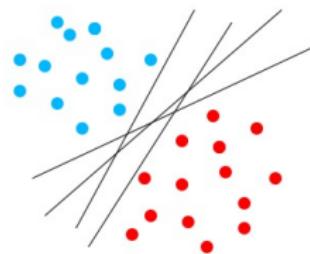
$$f(\mathbf{x}) = \text{signe}(\mathbf{w}^T \mathbf{x} + b)$$

L'équation :  $\mathbf{w}^T \mathbf{x} + b = 0$  définit un hyperplan dans l'espace euclidien  $\mathbb{R}^P$



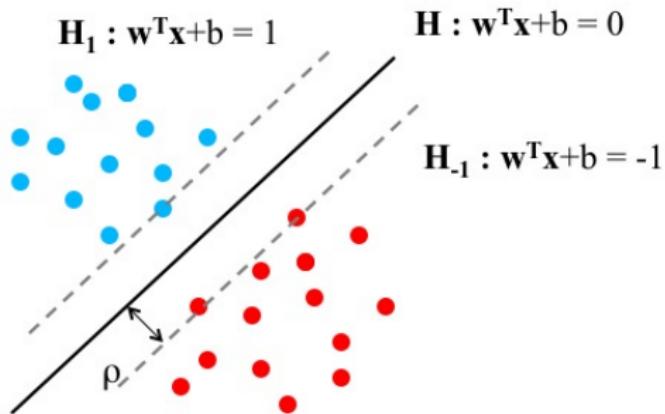
Exemple : données d'apprentissage en 3D et séparateur linéaire

# Cas de données linéairement séparables



Exemple en 2D : quelle droite choisir ?

# Critère de marge



# Critère de marge

## Notion de marge géométrique

- Pour séparer les données, on considère un triplet d'hyperplans :
  - $H : \mathbf{w}^T \mathbf{x} + b = 0$ ,  $H_1 : \mathbf{w}^T \mathbf{x} + b = 1$ ,  $H_{-1} : \mathbf{w}^T \mathbf{x} + b = -1$
- On appelle *marge géométrique*,  $\rho(\mathbf{w})$  la plus petite distance entre les données et l'hyperplan  $H$ , ici donc la moitié de la distance entre  $H_1$  et  $H_{-1}$
- Un calcul simple donne :  $\rho(\mathbf{w}) = \frac{1}{\|\mathbf{w}\|}$ .

# Nouvelle fonction de coût à optimiser

Comment déterminer  $\mathbf{w}$  et  $b$  ?

- Maximiser la marge  $\rho(\mathbf{w})$  tout en séparant les données de part et d'autre de  $H_1$  et  $H_{-1}$
- Séparer les données bleues ( $y_i = 1$ ) :  $\mathbf{w}^T \mathbf{x}_i + b \geq 1$
- Séparer les données rouges ( $y_i = -1$ ) :  $\mathbf{w}^T \mathbf{x}_i + b \leq -1$

# SVM linéaire : cas séparable

## Optimisation dans l'espace primal

$$\begin{array}{ll}\text{minimiser}_{\mathbf{w}, b} & \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{sous la contrainte} & y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \quad i = 1, \dots, n.\end{array}$$

## Référence

Boser, B. E.; Guyon, I. M.; Vapnik, V. N. (1992). "A training algorithm for optimal margin classifiers". Proceedings of the fifth annual workshop on Computational learning theory - COLT '92. p. 144.

# Programmation quadratique sous contraintes inégalités

Problème du type (attention les notations changent !)

- un problème d'optimisation ( $\mathcal{P}$ ) est défini par

$$\begin{aligned} & \text{minimiser sur } \mathbb{R}^n \quad J(\mathbf{x}) \\ & \text{avec} \quad h_i(\mathbf{x}) = 0, 1 \leq i \leq p \\ & \quad g_j(\mathbf{x}) \leq 0, 1 \leq j \leq q \end{aligned}$$

- rappel de vocabulaire :

- les  $h_i$  sont les **contraintes d'égalité** (notées  $\mathbf{h}(\mathbf{x}) = 0$ )
- les  $g_j$  sont les **contraintes d'inégalité** (notées  $\mathbf{g}(\mathbf{x}) \leq 0$ )
- l'**ensemble des contraintes** est

$$\mathcal{C} = \{\mathbf{x} \in \mathbb{R}^n \mid h_i(\mathbf{x}) = 0, 1 \leq i \leq p \text{ et } g_j(\mathbf{x}) \leq 0, 1 \leq j \leq q\}$$

**ensemble des points admissibles ou réalisables**

# Programmation quadratique sous contraintes inégalités

Problème du type :

$$\begin{aligned} & \min_x f(x) \\ & \text{s.c. } g(x) \leq 0 \end{aligned}$$

- Ici,  $g(x)$  linéaire
  - $f$  strictement convexe
- 
- ① Lagrangien :  $J(x, \lambda) = f(x) + \lambda g(x)$ ,  $\lambda \geq 0$

# Programmation quadratique sous contraintes inégalités

## Lagrangien

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 + \sum_i \alpha_i (1 - y_i (\mathbf{w}^T \mathbf{x}_i + b))$$
$$\forall i, \alpha_i \geq 0$$

# Conditions de Karush-Kuhn-Tucker

En l'extremum, on a

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \mathbf{w} - \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i = 0$$

$$\nabla_b \mathcal{L}(b) = - \sum_{i=1}^n \alpha_i y_i = 0$$

$$\forall i, \alpha_i \geq 0$$

$$\forall i, \alpha_i [1 - y_i (\mathbf{w}^T \mathbf{x}_i + b)] = 0$$

# Obtention des $\alpha_i$ : résolution dans l'espace dual

$$\mathcal{L}(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^T \mathbf{x}_j)$$

- Maximiser  $\mathcal{L}$  sous les contraintes  $\alpha_i \geq 0$  et  $\sum_i \alpha_i y_i = 0, \forall i = 1, \dots, n$
- Faire appel à un solveur quadratique

# SVM linéaires ou Optimal Margin Hyperplan

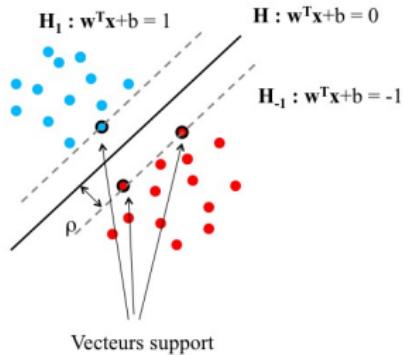
Supposons que les multiplicateurs de Lagrange  $\alpha_i$  soient déterminés :

## Equation d'un SVM linéaire

$$f(\mathbf{x}) = \text{signe}\left(\sum_{i=1}^n \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b\right)$$

Pour classer une donnée  $\mathbf{x}$ , ce classifier combine linéairement les valeurs de classe  $y_i$  des données support avec des poids du type  $\alpha_i \mathbf{x}_i^T \mathbf{x}$  dépendant de la ressemblance entre  $\mathbf{x}$  et les données support au sens du produit scalaire.

# Vecteurs "supports"



Les données d'apprentissage  $\mathbf{x}_i$  telles que  $\alpha_i \neq 0$  sont sur l'un ou l'autre des hyperplans  $H_1$  ou  $H_{-1}$ . Seules ces données dites *vecteur de support* comptent dans la définition de  $\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$

NB :  $b$  est obtenu en choisissant une donnée support ( $\alpha_i \neq 0$ )

# Cas réaliste : SVM linéaire dans le cas données non séparables

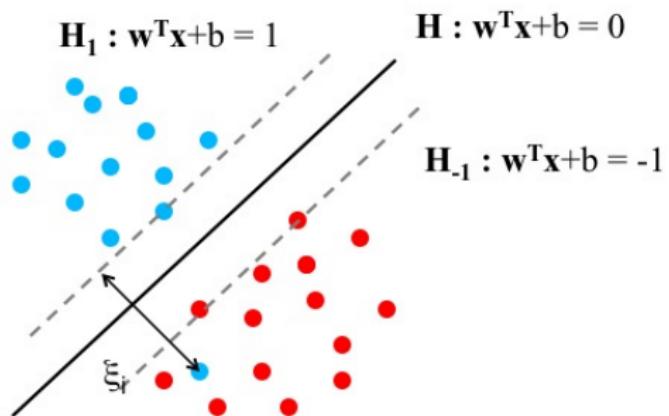
Introduire une variable d'écart  $\xi_i$  pour chaque donnée :

Problème dans le primal

$$\min_{\mathbf{w}, b, \xi} \quad \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i$$

sous les contraintes  $y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i \quad i = 1, \dots, n.$   
 $\xi_i \geq 0 \quad i = 1, \dots, n.$

# Cas réaliste : SVM linéaire dans le cas données non séparables



# Cas réaliste : SVM linéaire dans le cas données non séparables

## Problème dans le dual

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

sous les contraintes  $0 \leq \alpha_i \leq C \quad i = 1, \dots, n.$

$$\sum_i \alpha_i y_i \quad i = 1, \dots, n.$$

# Conditions de Karush-Kuhn-Tucker (KKT)

Soit  $\alpha^*$  la solution du problème dual :

$$\forall i, [y_i f_{w^*, b^*}(x_i) - 1 + \xi_i^*] \leq 0 \quad (1)$$

$$\forall i, \alpha_i^* \geq 0 \quad (2)$$

$$\forall i, \alpha_i^* [y_i f_{w^*, b^*}(x_i) - 1 + \xi_i^*] = 0 \quad (3)$$

$$\forall i, \mu_i^* \geq 0 \quad (4)$$

$$\forall i, \mu_i^* \xi_i^* = 0 \quad (5)$$

$$\forall i, \alpha_i^* + \mu_i^* = C \quad (6)$$

$$\forall i, \xi_i^* \geq 0 \quad (7)$$

$$\mathbf{w}^* = \sum_i \alpha_i^* y_i \mathbf{x}_i \quad (8)$$

$$\sum_i \alpha_i^* y_i = 0 \quad (9)$$

$$(10)$$

# Différents cas de figure

Soit  $\alpha^*$  la solution du problème dual :

- si  $\alpha_i^* = 0$ , alors  $\mu_i^* = C > 0$  et donc,  $\xi_i^* = 0$  :  $x_i$  est bien classé
- si  $0 < \alpha_i^* < C$  alors  $\mu_i^* > 0$  et donc,  $\xi_i^* = 0$  :  $x_i$  est tel que :  
 $y_i f(x_i) = 1$
- si  $\alpha_i^* = C$ , alors  $\mu_i^* = 0$ ,  $\xi_i^* = 1 - y_i f_{w^*, b^*}(x_i)$

NB : on calcule  $b^*$  en utilisant un  $i$  tel que  $0 < \alpha_i^* < C$

# Cas réaliste : SVM linéaire dans le cas données non séparables

## Quelques remarques

- certaines données support peuvent donc être de l'autre côté des hyperplans  $H_1$  ou  $H_{-1}$
- C est un hyperparamètre qui contrôle le compromis entre la complexité du modèle et le nombre d'erreurs de classification du modèle.

# SVM : approche par régularisation

## Optimisation dans l'espace primal

$$\min_{\mathbf{w}, b} \quad \sum_{i=1}^n (1 - y_i(\mathbf{w}^T \mathbf{x}_i + b))_+ + \lambda \frac{1}{2} \|\mathbf{w}\|^2$$

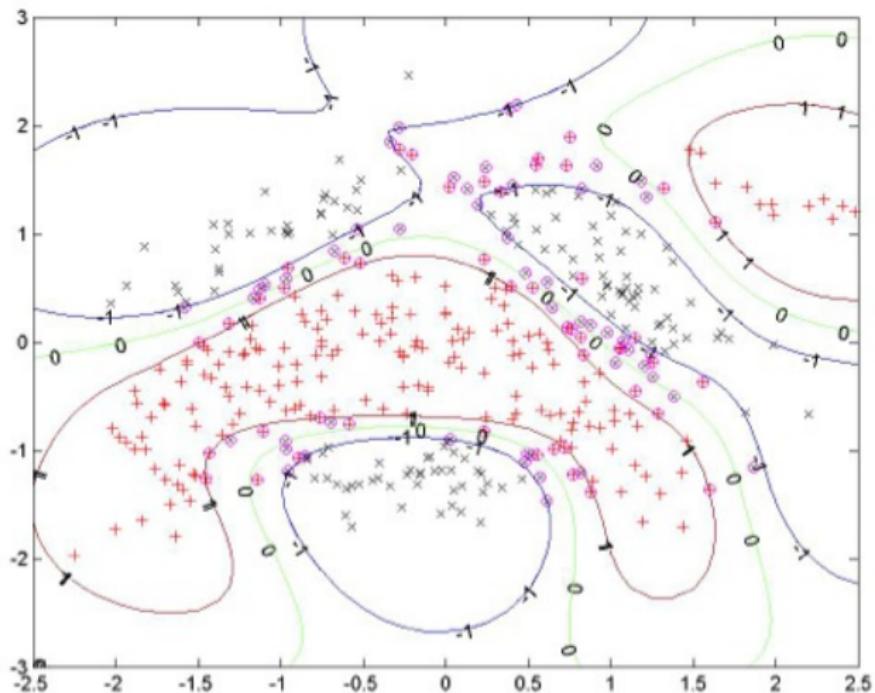
Avec :  $(z)_+ = \max(0, z)$

$f(\mathbf{x}) = \text{signe}(h(\mathbf{x}))$

Fonction de coût :  $L(\mathbf{x}, y, h(\mathbf{x})) = (1 - yh(\mathbf{x}))_+$

$yh(\mathbf{x})$  est appelée marge du classifieur

# Support Vector Machine : le cas non linéaire



## Remarque

Le problème de l'hyperplan de marge optimale ne fait intervenir les données d'apprentissage qu'à travers de produits scalaires.

$$\max_{\alpha} \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

sous les contraintes  $0 \leq \alpha_i \leq C \quad i = 1, \dots, n.$

$$\sum_i \alpha_i y_i \quad i = 1, \dots, n.$$

## Remarque 1 : apprentissage

Si je transforme les données à l'aide d'une fonction  $\varphi$  (non linéaire) et si je sais calculer les produits scalaires  $\varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j)$ , je peux apprendre une fonction de séparation non linéaire.

$$\max_{\alpha} \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j)$$

sous les contraintes  $0 \leq \alpha_i \leq C \quad i = 1, \dots, n.$

$$\sum_i \alpha_i y_i \quad i = 1, \dots, n.$$

Pour classer une nouvelle donnée  $\mathbf{x}$ , je n'ai besoin que de savoir calculer  $\varphi(\mathbf{x})^T \varphi(\mathbf{x}_i)$ .

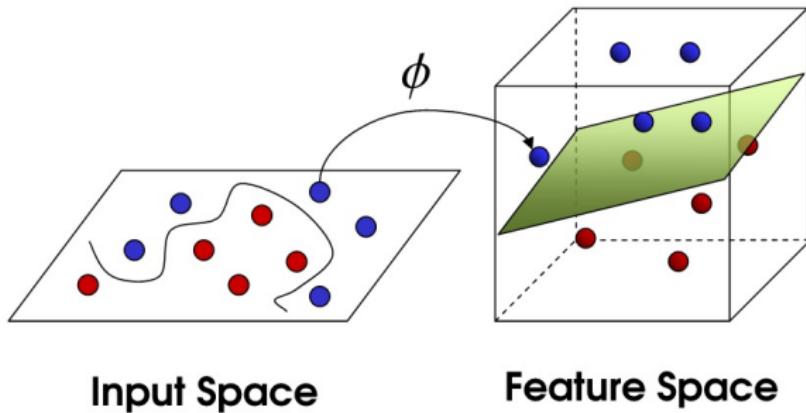
## Astuce du noyau

Si on remplace  $\mathbf{x}_i^T \mathbf{x}_j$  par l'image par une fonction  $k : k(\mathbf{x}_i, \mathbf{x}_j)$  telle qu'il existe un espace de caractérisques  $\mathcal{F}$  et une fonction de caractéristique (feature map)  $\varphi : \mathcal{X} \rightarrow \mathcal{F}$  et  $\forall (\mathbf{x}, \mathbf{x}') \in \mathcal{X}, k(\mathbf{x}, \mathbf{x}') = \varphi(\mathbf{x})^T \varphi(\mathbf{x}')$ , alors on peut appliquer le même algorithme d'optimisation (résolution dans le dual) et obtenir :

$$f(\mathbf{x}) = \text{signe}\left(\sum_{i=1}^n \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b\right)$$

Des telles fonctions existent et sont appelées *noyaux*.

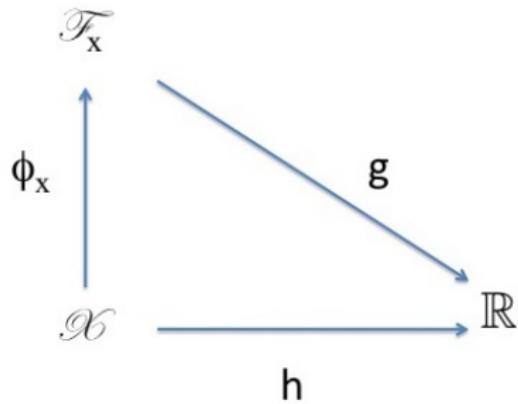
# Astuce du noyau et feature map 1/2



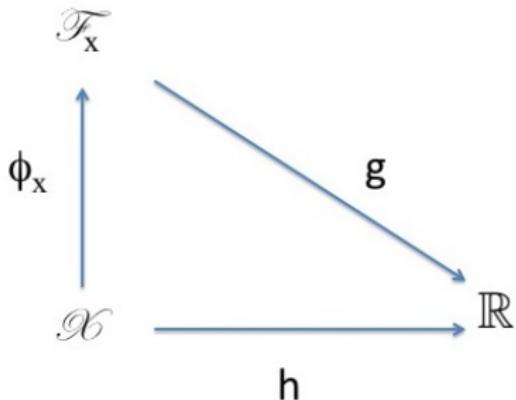
Input Space

Feature Space

# Astuce du noyau et feature map 2/2



## Astuce du noyau et feature map 2/2



Fonction  $h$  du type :

$$h(\mathbf{x}) = \sum_{i=1}^n \beta_i \varphi(\mathbf{x})^T \varphi(\mathbf{x}_i) = \sum_{i=1}^n \beta_i k(\mathbf{x}, \mathbf{x}_i),$$

avec  $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  un noyau positif défini.

# Noyaux

## Définition

Soit  $\mathcal{X}$  un ensemble. Soit  $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ , une fonction symétrique. La fonction  $k$  est appelée *noyau* positif défini si et seulement si quel que soit le sous-ensemble fini  $\{\mathbf{x}_1, \dots, \mathbf{x}_m\}$  de  $\mathcal{X}$  et le vecteur colonne  $\mathbf{c}$  de  $\mathbb{R}^m$ ,

$$\mathbf{c}^T K \mathbf{c} = \sum_{i,j=1}^m c_i c_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

# Théorème de Moore-Aronzajn

## Théorème de Moore-Aronzajn

Soit  $K$  un noyau positif défini. Alors, il existe un unique espace de Hilbert  $\mathcal{F}$  pour lequel  $k$  est un noyau reproduisant :

$$\forall x \in \mathcal{X}, \langle f(\cdot), k(\cdot, x) \rangle_{\mathcal{F}} = f(x)$$

$$\text{On a en particulier : } \langle k(\cdot, x), k(\cdot, x') \rangle_{\mathcal{F}} = k(x, x')$$

# Théorème de Moore-Aronzajn

## Théorème de Moore-Aronzajn

Soit  $K$  un noyau positif défini. Alors, il existe un unique espace de Hilbert  $\mathcal{F}$  pour lequel  $k$  est un noyau reproduisant :

$$\forall x \in \mathcal{X}, \langle f(\cdot), k(\cdot, x) \rangle_{\mathcal{F}} = f(x)$$

$$\text{On a en particulier : } \langle k(\cdot, x), k(\cdot, x') \rangle_{\mathcal{F}} = k(x, x')$$

**NB** : Cela veut dire qu'on peut toujours choisir  $\varphi(x) = k(\cdot, x)$

*Important : un noyau peut admettre plusieurs fonctions de caractérisques et espaces correspondants mais un seul est RKHS (espace de Hilbert à noyau reproduisant).*

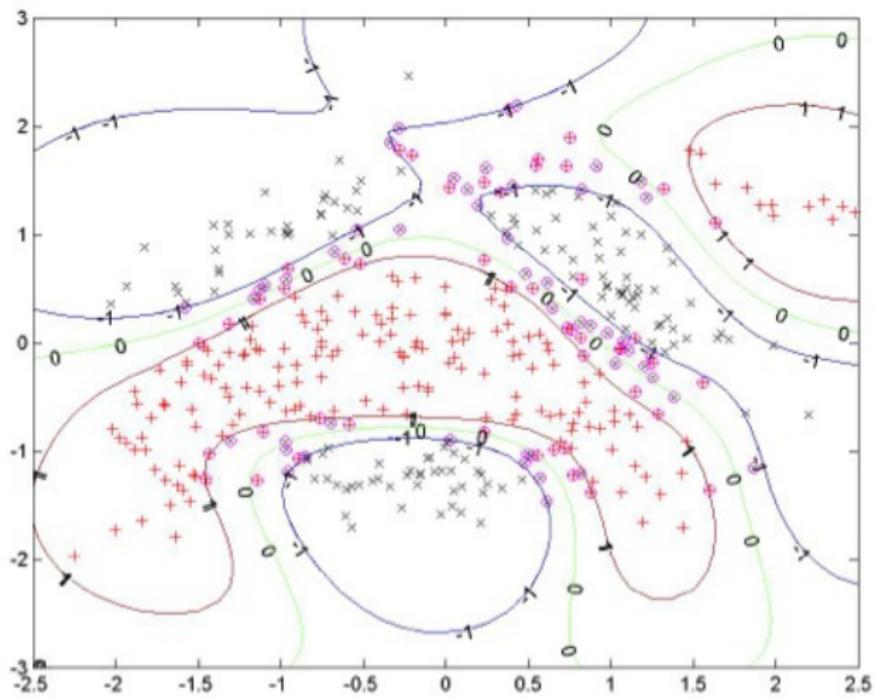
# Noyaux

## Noyaux entre vecteurs

$\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$

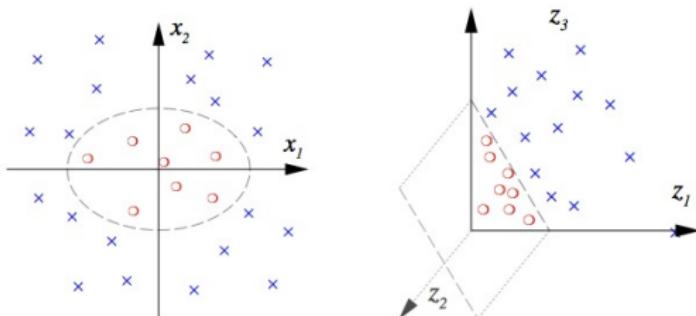
- Noyau linéaire :  $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$
- Noyau polynomial :  $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^d$
- Noyau gaussien :  $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma ||\mathbf{x} - \mathbf{x}'||^2)$

# Support Vector Machine : séparateur non linéaire par noyau gaussien



# Exemple : noyau polynomial

$$\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$



# Exemple : noyau polynomial

## Astuce du noyau

On remarque que  $\varphi(\mathbf{x}_1)^T \varphi(\mathbf{x}')$  peut se calculer sans travailler dans  $\mathbb{R}^3$

$$\text{Je peux définir } k(\mathbf{x}, \mathbf{x}') = \varphi(\mathbf{x})^T \varphi(\mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^2$$

# Construction d'un noyau

- Combiner des noyaux connus
- Des noyaux spécifiques à certains types de données :
  - **Objets structurés** : ensembles, graphes, arbres, séquences, ...
  - Données non structurées avec une structure sous-jacente : textes, images, documents, signaux, objets biologiques
- **Sélection d'un noyau :**
  - Hyperparameter learning : Chapelle et al. 2002
  - Multiple Kernel Learning : étant donnés  $k_1, \dots, k_m$ , apprendre une combinaison convexe  $\sum_i \beta_i k_i$  (see SimpleMKL Rakotomamonjy et al. 2008, unifying view in Kloft et al. 2010)

# Régression

## Cadre probabiliste et statistique

Soit  $X$  un vecteur aléatoire de  $\mathcal{X} = \mathbb{R}^p$

$Y$  une variable aléatoire continue  $\mathcal{Y} = \mathbb{R}$

Soit  $P$  la loi de probabilité jointe de  $(X, Y)$ , loi fixée mais inconnue

Supposons que  $S_{app} = \{(x_i, y_i), i = 1, \dots, n\}$  soit un échantillon i.i.d. tiré de la loi  $P$

# Régression

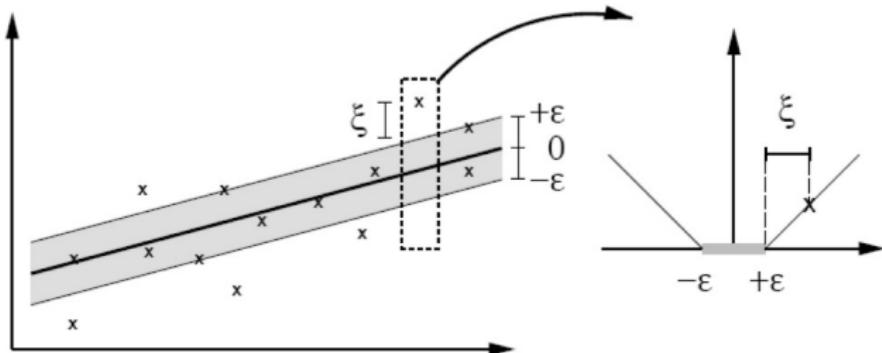
## Cadre probabiliste et statistique

- A partir de  $S_{app}$ , déterminer la fonction  $f \in \mathcal{F}$  qui minimise  $R(f) = \mathbb{E}_P[\ell(X, Y, f(X))]$
- $\ell$  étant une fonction de coût local qui mesure à quel point la vraie cible et la prédiction par le classifieur sont différentes

Pb : la loi jointe n'est pas connue : on ne peut pas calculer  $R(f)$

# Support Vector Regression

- Etendre l'idée de la marge maximal soft à la régression
  - Imposer un  $\varepsilon$ -tube : perte  $\varepsilon$ -insensible
- $$|y' - y|_\varepsilon = \max(0, |y' - y| - \varepsilon)$$



# Support Vector Regression

## SVR dans l'espace primal

Etant donnés  $C$  and  $\varepsilon$

$$\min_{w,b,\xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i (\xi_i + \xi_i^*)$$

s.c.

$$\forall i = 1, \dots, n, y_i - f(x_i) \leq \varepsilon + \xi_i$$

$$\forall i = 1, \dots, n, f(x_i) - y_i \leq \varepsilon + \xi_i^*$$

$$\forall i = 1, \xi_i \geq 0, \xi_i^* \geq 0$$

$$\text{avec } f(x) = w^T \varphi(x) + b$$

Cas général :  $\varphi$  feature map associée à un noyau défini positif  $k$ .

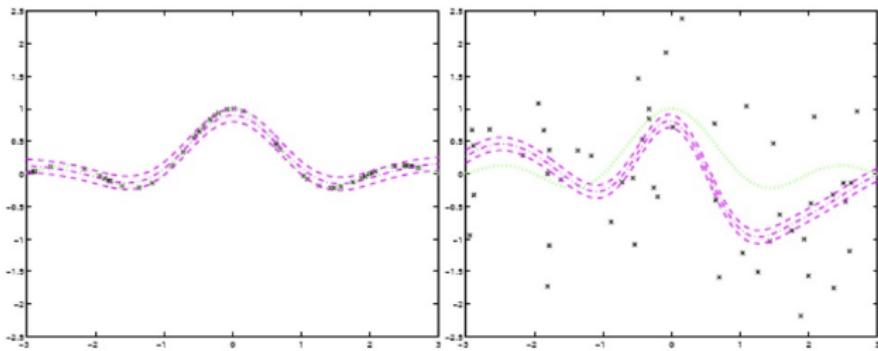
# Solution dans le dual

$$\begin{aligned} & \min_{\alpha, \alpha^*} \sum_{i,j} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) k(x_i, x_j) + \varepsilon \sum_i (\alpha_i + \alpha_i^*) - \\ & \sum_i y_i (\alpha_i - \alpha_i^*) \\ \text{s.c. } & \sum_i (\alpha_i - \alpha_i^*) = 0 \text{ et } 0 \leq \alpha_i \leq C \text{ et } 0 \leq \alpha_i^* \leq C \\ w &= \sum_{i=1}^n (\alpha_i - \alpha_i^*) \varphi(x_i) \end{aligned}$$

## Solution

$$f(x) = \sum_{i=1}^n (\alpha_i - \alpha_i^*) k(x_i, x) + b$$

# Support Vector Regression : example in 1D

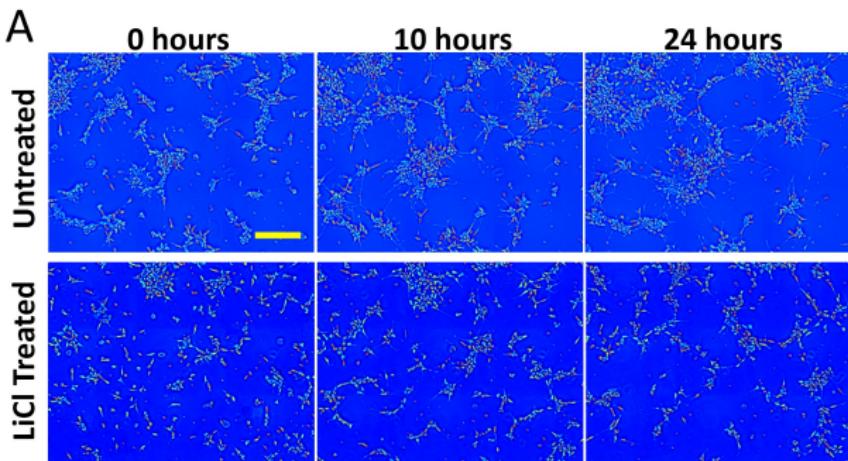


*Identical* machine parameters ( $\varepsilon = 0.2$ ), but different amounts of noise in the data.

B. Schölkopf, Canberra, February 2002

# Sommaire

# Neuron network growth over 24 hours

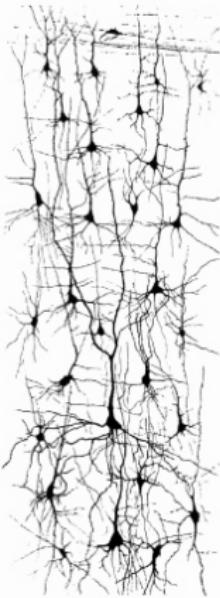


In 2014, the group of Gabriel Popescu at Illinois U. visualized a growing net of baby neurons using spatial light interference microscopy (SLIM). Ref : [http://light.ece.illinois.edu/  
wp-content/uploads/2014/03/Mir\\_SRep\\_2014.pdf](http://light.ece.illinois.edu/wp-content/uploads/2014/03/Mir_SRep_2014.pdf)  
Video : [https://youtu.be/KjKsU\\_4s0nE](https://youtu.be/KjKsU_4s0nE)

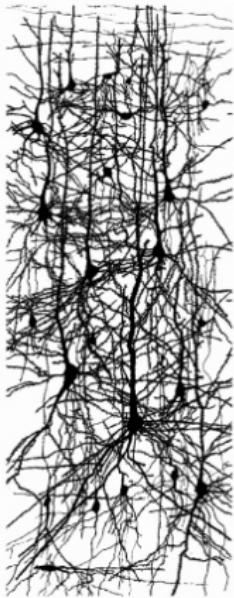
# Développement des réseaux de neurones chez l'enfant



nouveau-né



3 mois après la naissance

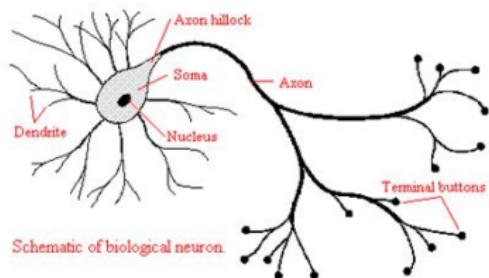


à l'âge de 2 ans

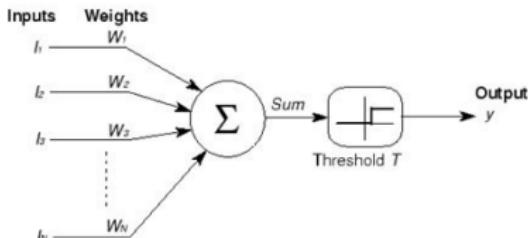
Développement des réseaux de connections entre les neurones chez l'enfant.

# Neurone

## Le neurone

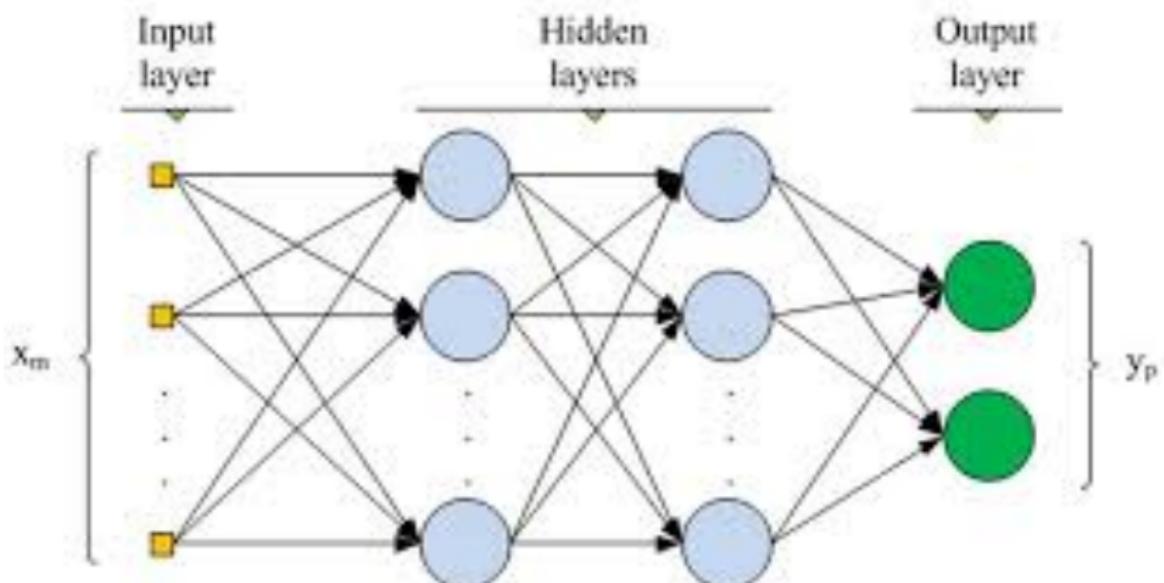


Neurone biologique



Neurone artificiel

# Réseau de neurones formels (perceptron multi-couches)



# Du neurone formel aux réseaux de neurones formels 1/2

- Neurone formel : Mc Cullogh et Pitts, 1943
- Règle d'apprentissage du perceptron, Rosenblatt, 1957
- Minsky et Papert : capacité limitée du perceptron, 1959
- Apprentissage d'un perceptron multi-couches par rétropropagation du gradient, Y. Le Cun, 1985, Hinton et Sejnowski, 1986.
- Perceptron multi-couches = approximateur universel, Hornik et al. 1991
- Convolutional networks, 1995, Y. Le Cun et Y. Bengio
- Entre 1995 et 2008, peu d'expansion du domaine (pbs fonctions non convexes, temps d'apprentissage, pas de théorie)

# Du neurone formel aux réseaux de neurones formels 1/2

- Généralisation des GPU (processeurs graphiques) 2005
- Très large ensemble d'images : Imagenet, Fei-Fei et al. 2008 (maintenant 11 millions d'images)
- Réseaux de neurones de plus en plus profonds appris avec de gigantesques bases de données
- Apprentissage initial non supervisé (avec auto encodeur)
- Word2vec (Mikolov et al. 2013)
- Dropout (Srivastava et al. 2014)

# Agenda

- Données non structurées
- Rappel : neurone formel ou perceptron
- Perceptron multi-couches
- Autoencodeurs
- Un mot sur les réseaux convolutionnels

# Définition du neurone formel

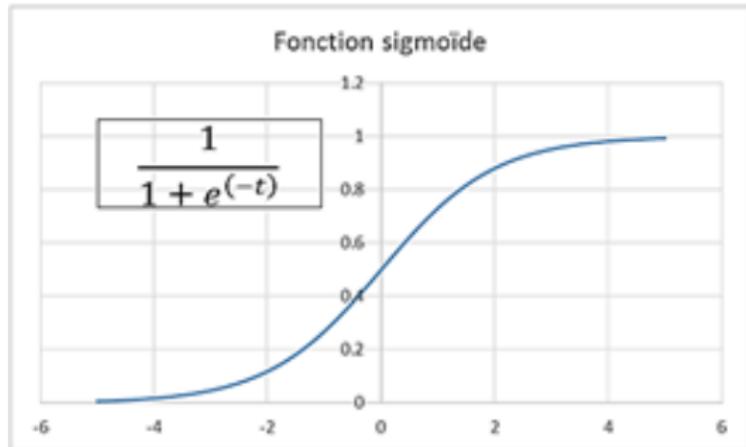
- une fonction d'activation
- un vecteur de poids et un biais (intercept)

$$f(x) = g(w^T x + b) \quad (11)$$

On choisit  $g$  différentiable

# Fonctions d'activation du neurone formel

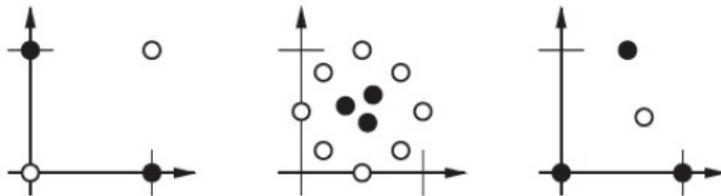
par exemple :



Mais aussi tanh (sortie entre -1 et 1).

# Limitation du neurone formel

Limité aux données linéairement séparables :



# Ajouter une couche de traitement intermédiaire

$$\begin{aligned}\Phi(x)_1 &= \text{AND}(\bar{x}_1, x_2) \\ \Phi(x)_2 &= \text{AND}(x_1, \bar{x}_2)\end{aligned}$$

Maintenant, calculer :

$$f(x) = g(\Phi(x)^T w + b)$$

On parle de fonction de redescription (feature map) ou de représentation interne.

**Grand avantage des réseaux de neurones à plus d'une couche** : apprentissage de la fonction  $\Phi$ .

# Approximateur universel

En 1991, Hornik et al. démontrent que la famille des perceptrons à une couche cachée et à  $p + 1$  entrées est dense dans l'ensemble des fonctions continues de  $\mathbb{R}^p$  dans  $\mathbb{R}$ . Un MLP à une couche cachée est un **approximateur universel**.

# Approximateur universel

En 1991, Hornik et al. démontrent que la famille des perceptrons à une couche cachée et à  $p + 1$  entrées est dense dans l'ensemble des fonctions continues de  $\mathbb{R}^p$  dans  $\mathbb{R}$ . Un MLP à une couche cachée est un **approximateur universel**.

D'autres exemples d'approximateurs universels que vous connaissez :

- Régresseur linéaire : **NON**
- SVM avec noyau universel tel que le noyau Gaussien : **OUI**
- Forêts aléatoires : **OUI**
- Boosting de stumps : **OUI**

# Exemple d'un réseau de neurones multi-couches "feedforward"

Prenons comme exemple un MLP à une couche de sortie de taille  $K=1$ , une couche cachée de taille  $M + 1$ , un vecteur d'entrée de taille  $p + 1$  pour la régression

Famille de fonctions  $\mathcal{H}_{mlp} = \{h_{mlp} : \mathcal{R}^{p+1} \rightarrow \mathcal{Y}\}$

$$h_{MLP}(x) = \sum_{j=0}^M w_j^{(2)} z_j \quad (12)$$

$$z_j = \tanh(a_j) \quad (13)$$

$$a_j = \sum_{i=0}^p w_{ji}^{(1)} x_i \quad (14)$$

## Remarque sur la fonction de saturation

La tangente hyperbolique est choisie comme fonction de saturation, dérivable.

$$h(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \quad (15)$$

$$h'(a) = 1 - h(a)^2 \quad (16)$$

En termes de calculs, cette fonction est très avantageuse car la dérivée se définit directement en terme de  $h(a)$ . Nous avons une propriété similaire pour la fonction sigmoïde :  $g(a) = \frac{1}{1+\exp(-\frac{1}{2}a)}$ .

# Architecture d'un réseau de neurones multi-couches "feedforward"

- Nous avons choisi : la sortie unique du régresseur MLP fournit une valeur réelle
- Pour un problème de classification à  $K$  classes, nous aurions choisi  $K$  sorties avec la fonction sigmoïde ou mieux softmax
- Pour un problème de régression à  $K$  sorties, nous aurions plutôt choisi,  $K$  sorties linéaires (ici  $K = 1$ )

## Exemple d'un réseau de neurones multi-couches "feedforward"

Prenons comme exemple un MLP à une couche de sortie de taille  $K=1$ , une couche cachée de taille  $M + 1$ , un vecteur d'entrée de taille  $p + 1$  pour la régression

$$\text{Famille de fonctions } \mathcal{H}_{mlp} = \{h_{mlp} : \mathcal{R}^{p+1} \rightarrow \mathcal{Y}\}$$

$$h_c(x) = g\left(\sum_{j=0}^M w_{jc}^{(2)} z_j\right) \quad (17)$$

$$z_j = g(a_j) \quad (18)$$

$$a_j = \sum_{i=0}^p w_{ji}^{(1)} x_i \quad (19)$$

$$\text{avec } g(t) = \frac{1}{1+\exp(-1/2t)}.$$

# Apprentissage à partir de données

$$\mathcal{L}(W; \mathcal{S}) = \sum_{n=1}^N \ell(h(x_n), y_n))$$

Régression :

$$\ell(h(x_n), y_n) = (h(x_n) - y_n)^2$$

Classification (maximiser la vraisemblance) : On interprète  $f_c(x) = p(y = c|x)$  (plusieurs sorties : on peut utiliser la fonction softmax )

$$\ell(h(x), y) = -\log f_y(x)$$

Importante remarque :  $\mathcal{L}$  est non convexe et possède de nombreux minima locaux

- Le mieux que nous puissions faire, c'est trouver un bon minimum local
- C'est principalement pour cette raison que les MLP ont été pendant longtemps abandonnés en faveur des SVM/SVR plus faciles à optimiser

# Algorithme d'optimisation

## La rétropropagation du gradient

- L'idée est d'appliquer un algorithme de descente du gradient : on rétropropage une erreur à travers chacune des couches, en débutant par la dernière couche,
- On utilise la règle de dérivation en chaîne :  
$$\frac{\partial L(W)}{\partial w_{ji}^{(1)}} = \frac{\partial L(W)}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}^{(1)}}$$
 pour pouvoir corriger les poids de la couche cachée.
- Une fois toutes les corrections calculées, on met à jour les poids du réseau.
- L'algorithme peut s'appliquer globalement ou localement (nous allons voir ce que cela signifie)

# La rétropropagation du gradient

## Références :

Y. LeCun : Une procédure d'apprentissage pour réseau à seuil asymétrique (a Learning Scheme for Asymmetric Threshold Networks), Proceedings of Cognitiva 85, 599-604, Paris, France, 1985.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986)  
Learning representations by back-propagating errors. Nature, 323, 533–536.

# Rappelons la descente de gradient ordinaire

Soit une fonction  $\mathcal{C}(\theta)$  dépendant de  $\theta$  :

- Les valeurs  $\theta$  telles que  $\frac{\partial \mathcal{C}(\theta)}{\partial \theta} = 0$  correspondent à des minima ou des maxima de cette fonction.
- Lorsque  $\mathcal{C}$  est strictement convexe en  $\theta$ , l'algorithme que nous allons présenter s'approche de la solution aussi prèsque possible.
- Cependant, même quand  $\mathcal{C}$  n'est pas s. convexe, nous pouvons toujours essayer de trouver un "bon" minimum local.

**Idée** : corriger  $\theta$  itérativement par :  $\theta^{t+1} \leftarrow \theta^t - \eta_t \frac{\partial \mathcal{C}(\theta)}{\partial \theta}$

Après chaque mise à jour, le gradient est ré-évalué pour le nouveau vecteur de paramètre et la correction est à nouveau calculée

# Rappelons la descente de gradient globale

$$\mathcal{C}(\theta) = \sum_{n=1}^N c_n(\theta)$$

- ①  $E = 1000$  ;
- ②  $\varepsilon = \text{petite valeur}$
- ③  $\theta^0$  valeur initiale ;  $t = 0$  ;
- ④ Tant que ( $E > \varepsilon$ )

- $\theta^{t+1} \leftarrow \theta^t - \eta_t \sum_{n=1}^N \frac{\partial c_n(\theta^t)}{\partial \theta^t}$
- calculer  $E = L(\theta^{t+1})$

- ⑤ Fournir  $\theta$  courant

# Choix de $\eta_k$

## Théorème :

Si la série  $(\sum_k \eta_k)$  diverge et si  $(\sum_k \eta_k^2)$  converge alors l'algorithme de gradient converge vers un minimum local.

# Descente de gradient stochastique et locale

$$\mathcal{C}(\theta) = \sum_{n=1}^N c_n(\theta)$$

- ➊  $E = 1000$  ;
- ➋  $\varepsilon = \text{petite valeur}$
- ➌  $\theta^0$  valeur initiale ;  $t = 0$  ;
- ➍  $nb_{cycle} = 0$
- ➎ Tant que ( $E \geq \varepsilon$ ) et ( $nb_{cycle} < 500$ )
  - $nb_{cycle} = nb_{cycle} + 1$
  - pour  $\ell = 1$  à  $N$ 
    - Tirer uniformément un indice  $n \in \{1, \dots, N\}$
    - $\theta^{t+1} \leftarrow \theta^t - \eta_t \frac{\partial c_n(\theta^t)}{\partial \theta^t}$
  - calculer  $E = L(\theta^{t+1})$

# Descente de gradient stochastique avec minibatch de taille constante

$$\mathcal{C}(\theta) = \sum_{n=1}^N c_n(\theta)$$

- ➊  $E = 1000$  ;
- ➋  $\varepsilon = \text{petite valeur}$
- ➌  $\theta^0$  valeur initiale ;  $t = 0$  ;
- ➍  $nb_{cycle} = 0$
- ➎ Tant que ( $E \geq \varepsilon$ ) et ( $nb_{cycle} < 500$ )
  - ➏  $nb_{cycle} = nb_{cycle} + 1$ 
    - ➐ Tirer uniformément  $M$  fois un indice  $n \in \{1, \dots, N\}$
    - ➑  $\theta^{t+1} \leftarrow \theta^t - \eta_t \frac{\partial c_M(\theta^t)}{\partial \theta^t}$
    - ➒ calculer  $E = L(\theta^{t+1})$

# Rétropropagation du gradient (1/4)

On souhaite appliquer la règle de descente de gradient aux poids de la couche 1 et de la couche 2 (ici réduite à une unité de sortie).

Soit  $\ell = \frac{1}{2}(h(x) - y)^2$

**Calcul pour une donnée, algorithme local**

Gradient par rapport aux poids de sortie :

$$\frac{\partial \ell}{\partial w_j^{(2)}} = \frac{\partial \ell}{\partial h(x)} \frac{\partial h(x)}{\partial w_j^{(2)}} \quad (20)$$

Gradient par rapport aux poids de la couche cachée :

$$\frac{\partial \ell}{\partial w_{ji}^{(1)}} = \frac{\partial \ell}{\partial h(x)} \frac{\partial h(x)}{\partial w_{ji}^{(1)}} \quad (21)$$

# Rétropropagation du gradient local (2/4) : les calculs

## Calcul pour une donnée, algorithme local

Gradient pour les poids de sortie :

$$\frac{\partial \ell}{\partial w_j^{(2)}} = \frac{\partial \ell}{\partial h(x)} \frac{\partial h(x)}{\partial w_j^{(2)}} \quad (22)$$

$$\frac{\partial \ell}{\partial h(x)} = h(x) - y \quad (23)$$

$$\frac{\partial h(x)}{\partial w_j^{(2)}} = \frac{\partial g(w_j^{(2)}z_j + \sum_{k \neq j} w_k^{(2)}z_k)}{\partial w_j^{(2)}} \quad (24)$$

(25)

# Rétropropagation du gradient local (3/4) : les calculs

## Calcul pour une donnée, algorithme local

Gradient pour les poids de la couche cachée :

$$\frac{\partial \ell}{\partial w_{ji}^{(1)}} = \frac{\partial \ell}{\partial h(x)} \frac{\partial h(x)}{\partial w_{ji}^{(1)}} \quad (26)$$

$$\frac{\partial h(x)}{\partial w_{ji}^{(1)}} = w_j^{(2)} \frac{\partial g(\sum_k w_{jk} x_k)}{\partial w_{ji}^{(1)}} \quad (27)$$

$$\frac{\partial h(x)}{\partial w_{ji}^{(1)}} = (1 - g(\sum_k w_{jk} x_k)^2) w_j^{(2)} x_i \quad (28)$$

# Rétropropagation du gradient (4/4) : l'algorithme

Pour une descente locale pour un exemple  $x_n$  tiré uniformément :

- ➊ Calculer pour  $x_n$ ,  $h(x_n)$
- ➋ Calculer les gradients :  $\frac{\partial \ell_n}{\partial w_j^{(2),t}}$  puis  $\frac{\partial \ell_n}{\partial w_{ji}^{(1),t}}$
- ➌ Corriger tous les poids avec les gradients préalablement calculés :
  - Corriger la couche (1) :
  - Pour tout  $j=0$  à  $M$  :
    - $w_j^{(1),t+1} \leftarrow w_j^{(1),t} - \eta_t \frac{\partial \ell_n}{\partial w_j^{(1),t}}$
  - Corriger la couche (2) : ici unique neurone de sortie
    - $w^{(2),t+1} \leftarrow w^{(2),t} - \eta_t \frac{\partial \ell_n}{\partial w^{(2),t}}$

# Régularisation et early stopping

## ● Early Stopping

- Une première méthode de régularisation a été proposée dans les années 90 : il s'agit d'arrêter *a priori* l'apprentissage prématulement avant le sur-apprentissage : on évite de se rapprocher trop près d'un minimum !

## ● Régularisation

- On peut plus rigoureusement définir :  
$$\mathcal{L}(W, \mathcal{S}_{app}) = \sum_n \ell(h(x_n), y_n) + \lambda_2 \|w^{(2),*}\|^2 + \lambda_1 \|w^{(1),*}\|^2$$
- On évitera de régulariser  $w_0^{(2)}$ ,  $w_{j0}^{(1)}$  et  $w_{0i}^{(2)}$ . Le \* signifie qu'on ne considère pas ces coordonnées-là.

En pratique, on note que :  $\|w^{(1),*}\|^2 = \sum_{ji, j \neq 0, i \neq 0} (w_{ji}^{(1)})^2$ .

# Sélection de modèles

Le MLP a plusieurs hyperparamètres :

- Nb de couches cachées
- Tailles des couches cachées
- paramètre  $\lambda$
- $nb_{cycle}, \varepsilon$
- $\eta_t = \frac{\gamma}{1+t}$

La plupart sont trouvés par VALIDATION CROISEE.

- **Pour**

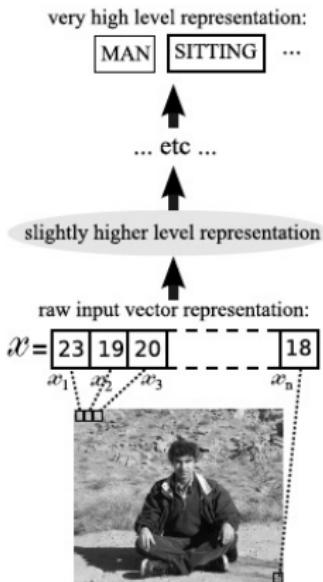
- Flexibilité au niveau des sorties : plusieurs classes, etc..
- Technique éprouvée depuis 1985
- Algorithme de gradient stochastique se plie bien aux besoins du BIG DATA
- Bénéficie des architectures GPU
- PLUG and PLAY : on peut enchaîner différents traitements dans un même paradigme

- **Contre**

- Fonction de perte non convexe : pas de minimum global
- Descente de gradient nécessite souvent de nombreux ajustements
- Pas de cadre théorique
- Beaucoup de développements *ad hoc*

# Réseaux dits profonds - Deep Learning

Image Y. Bengio



# Apprentissage des réseaux dits profonds

A partir de 3 couches cachées, on parle de "deep learning", ce type de réseau est a priori intéressant pour traiter des données complexes comme des images ou du texte.

## Pourquoi utiliser plusieurs couches cachées ?

Même si un réseau à une couche cachée est en principe un approximateur universel, cela ne veut pas dire qu'un réseau à une couche cachée fournit la meilleure représentation et les meilleures performances.

# Apprentissage des réseaux dits profonds

Malgré le risque de surapprentissage,

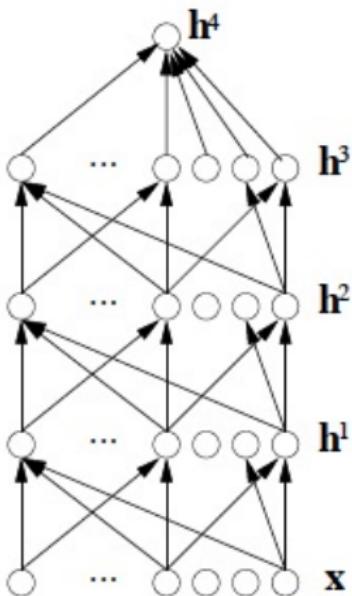
deux bonnes raisons de s'intéresser aux réseaux profonds

- l'amélioration des capacités de calcul et de mémoire (GPU)
- la disponibilité de gigantesques bases de données (Imagenet, Fei-Fei, 2008)

Apprendre un réseau profond par simple rétropropagation du gradient ne fonctionne pas si bien que cela (Bengio et al. 2007 ; Erhan et al. 2009).

Le réseau tombe dans des minima locaux très mauvais sans une bonne initialisation.

# Apprentissage des réseaux dits profonds



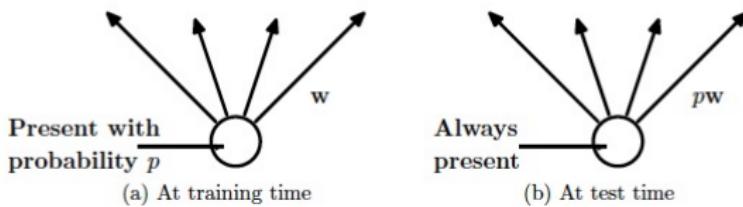
# Apprentissage des réseaux dits profonds

Deux améliorations notables

- Dropout
- Auto-encodeurs

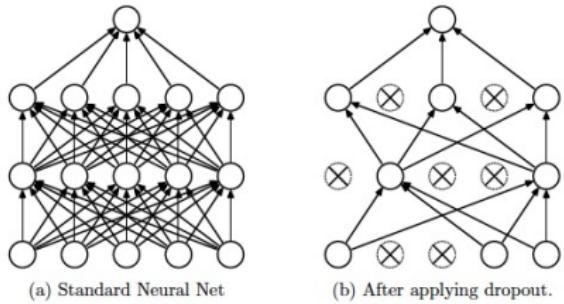
# Eviter l'overfitting des réseaux profonds par dropout 1/3

Pour les réseaux profonds ( $>>2$  couches) :



- Durant l'apprentissage, à chaque étape de gradient : chaque unité(neurone) est présente avec une probabilité  $p$ , ce qui veut dire que certains neurones ne sont pas présents et donc ne sont pas corrigés systématiquement.
- Pendant la prédiction (en test), chaque unité est présente et un facteur  $p$  est appliqué à ses poids.

# Eviter l'overfitting des réseaux profonds par dropout 2/3



## Une interprétation :

Si on dispose de  $m$  neurones au total, c'est comme si on apprenait avec  $2^m$  réseaux clairsemés et au moment du test, on superpose tous ces réseaux en un seul avec lequel on prédit.

Les neurones ne peuvent s'adapter les uns aux autres

# Eviter l'overfitting des réseaux profonds par dropout 3/3

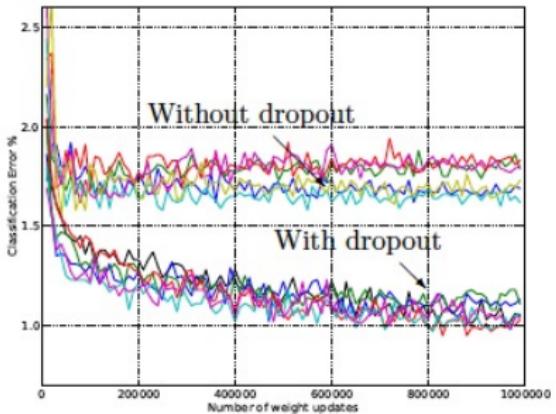


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

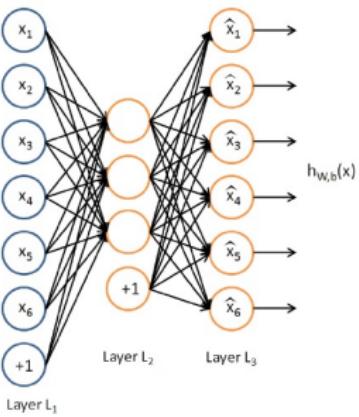
# Apprentissage des réseaux dits profonds

Les réseaux à plusieurs couches sont aujourd'hui appris en étant initialisés par un apprentissage non supervisé souvent à l'aide d'autoencodeurs ou de Machines de Boltzman restreintes (RBM). Nous allons voir comment...

# Autoencodeurs

## Autoencodeurs

Un autoencodeur (aussi appelé *réseau diabolo*) est un réseau à une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie. Ce type de réseau cherche à construire une représentation interne (la couche du milieu) en apprenant à prédire l'entrée à partir de celle-ci :  $x \approx g(x)$ .



# Apprentissage d'autoencodeurs

- Un auto-encodeur s'apprend par rétropropagation du gradient. Dans le cas des autoencodeurs parcimonieux, la fonction de perte utilisée est en général le critère quadratique pénalisé par un terme de régularisation qui contraint l'activité (moyenne) de chaque unité de la couche cachée à rester limitée.
- L'autoencodeur n'a d'autre intérêt que d'apprendre des représentations internes dans le cas de données complexes

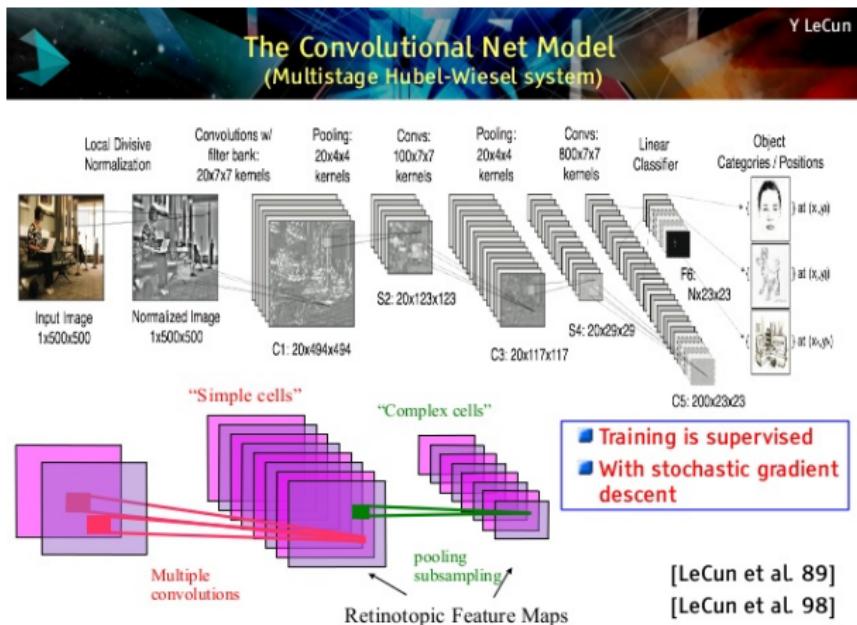
# Autoencodeur et apprentissage d'un réseau "feed-forward" profond (Erhan et al.)

Pour chaque couche cachée en démarrant par la plus proche de l'entrée  $x$ , on définit ses poids en les extrayant de la première couche d'un autoencodeur appris sur :

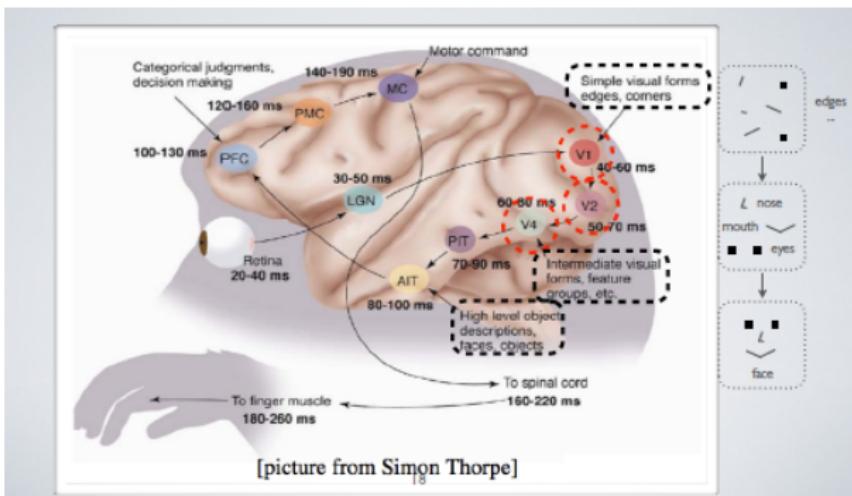
- Poids de la couche 2 : on apprend un autoencodeur  $x \approx h(x)$ . On initialise les poids de la couche 2 par ceux de la couche 2 de l'autoencodeur
- Poids de la couche 3 : on apprend un autoencodeur  $f_1(x) \approx h(f_1(x))$ . On initialise les poids de la couche 3 par ceux de la couche 2 de ce nouvel autoencodeur
- etc...
- Ensuite, on apprend de manière supervisée à partir de cette initialisation

# Réseaux convolutionnels pour les images

Y. Le Cun.

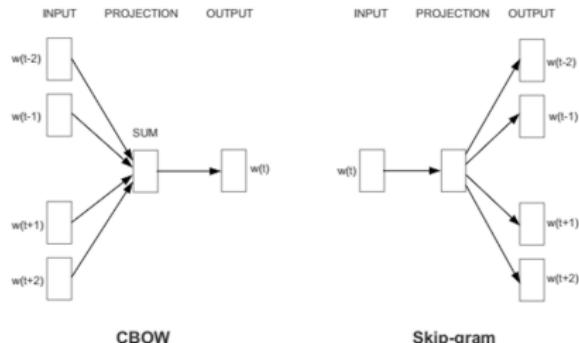


# Cortex visuel



# Skip-gram modèle pour codage de mots

Mikolov et al. 2013.



NB : représentations continues des mots déjà proposées en 2002/2003.

# Sommaire

# Références Ensemble Learning

- Y. Amit, D. Geman, and K. Wilder, Joint induction of shape features and tree classifiers, IEEE Trans. Pattern Anal. Mach. Intell., 19, 1300-1305, 1997.
- Breiman, L., Bagging predictors. *Mach Learn* (1996) 24 : 123.
- Y. Freund, R. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting. In Computational Learning Theory, 1995.
- J. Friedman, T. Hastie and R. Tibshirani, Additive logistic regression : a statistical view of boosting. *Ann. Statist.* Vol. 28, No. 2 (2000), 337-407.
- Breiman, L., Random Forests. *Mach. Learn.* (2001), Vol. 45, No. 1, pp 5–32
- Tutorial : Ensemble Methods in Machine Learning. T.G. Dietterich, available at :  
<http://web.engr.oregonstate.edu/~tgd/publications/mcs-ensembles.pdf>

# Références - Réseaux de Neurones

- Le cours de Hugo Larochelle (youtube)
- Notes de cours IT6266, Université de Montréal, Équipe de Yoshua Bengio.
- Learning Deep Architectures for AI, Yoshua Bengio, Foundations Trends in Machine Learning, 2009
- Dropout : A simple way to prevent overfitting, Srivastava et al. JMLR 2014
- Pattern Recognition and Machine Learning, C. Bishop, Springer, 2006.
- <http://deeplearning.net/tutorial/> : pour tout document y compris implémentations...

# Sommaire

## Références SVM

- BOSER, Bernhard E., Isabelle M. GUYON, and Vladimir N. VAPNIK, 1992. A training algorithm for optimal margin classifiers. In : COLT '92 : Proceedings of the Fifth Annual Workshop on Computational Learning Theory. New York, NY, USA : ACM Press, pp. 144-152.
- CORTES, Corinna, and Vladimir VAPNIK, 1995. Support-vector networks. Machine Learning, 20(3), 273–297.
- A tutorial review of RKHS methods in Machine Learning, Hoffman, Schölkopf, Smola, 2005 ([https://www.researchgate.net/publication/228827159\\_A\\_Tutorial\\_Review\\_of\\_RKHS\\_Methods\\_in\\_Machine\\_Learning](https://www.researchgate.net/publication/228827159_A_Tutorial_Review_of_RKHS_Methods_in_Machine_Learning))

# Références - Réseaux de Neurones

- Le cours de Hugo Larochelle (youtube)
- Notes de cours IT6266, Université de Montréal, Équipe de Yoshua Bengio.
- Learning Deep Architectures for AI, Yoshua Bengio, Foundations Trends in Machine Learning, 2009
- Dropout : A simple way to prevent overfitting, Srivastava et al. JMLR 2014
- Pattern Recognition and Machine Learning, C. Bishop, Springer, 2006.
- <http://deeplearning.net/tutorial/> : pour tout document y compris implémentations...



Questions ?

Merci !