

IFT3395/6390

Fondements de l'apprentissage machine

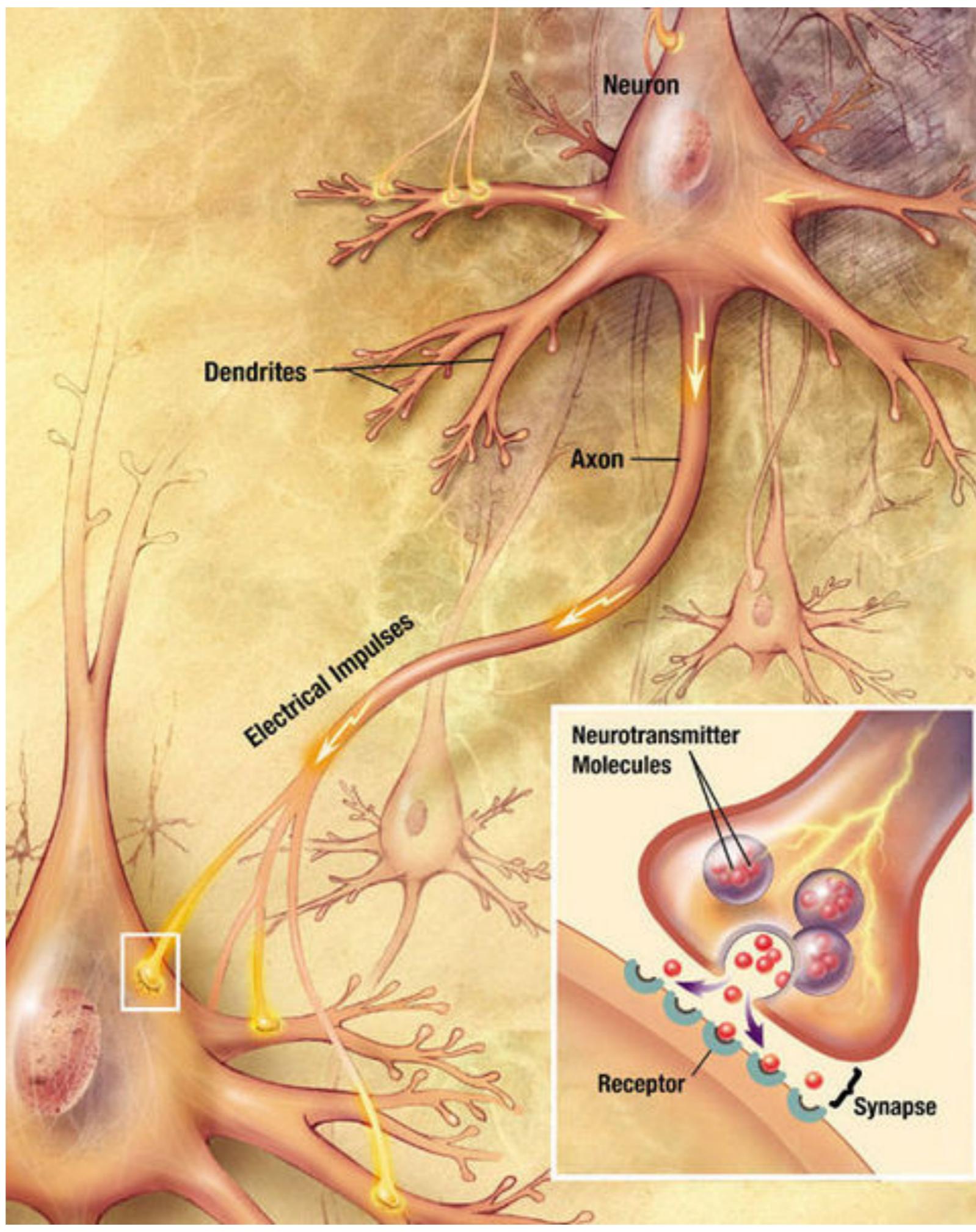
Réseaus de neurones artificiels

Professeur: Pascal Vincent

Le cerveau humain



- 100 milliard de neurones
- interconnectés en un réseau complexe
- chaque neurone connecté à plusieurs milliers d'autres



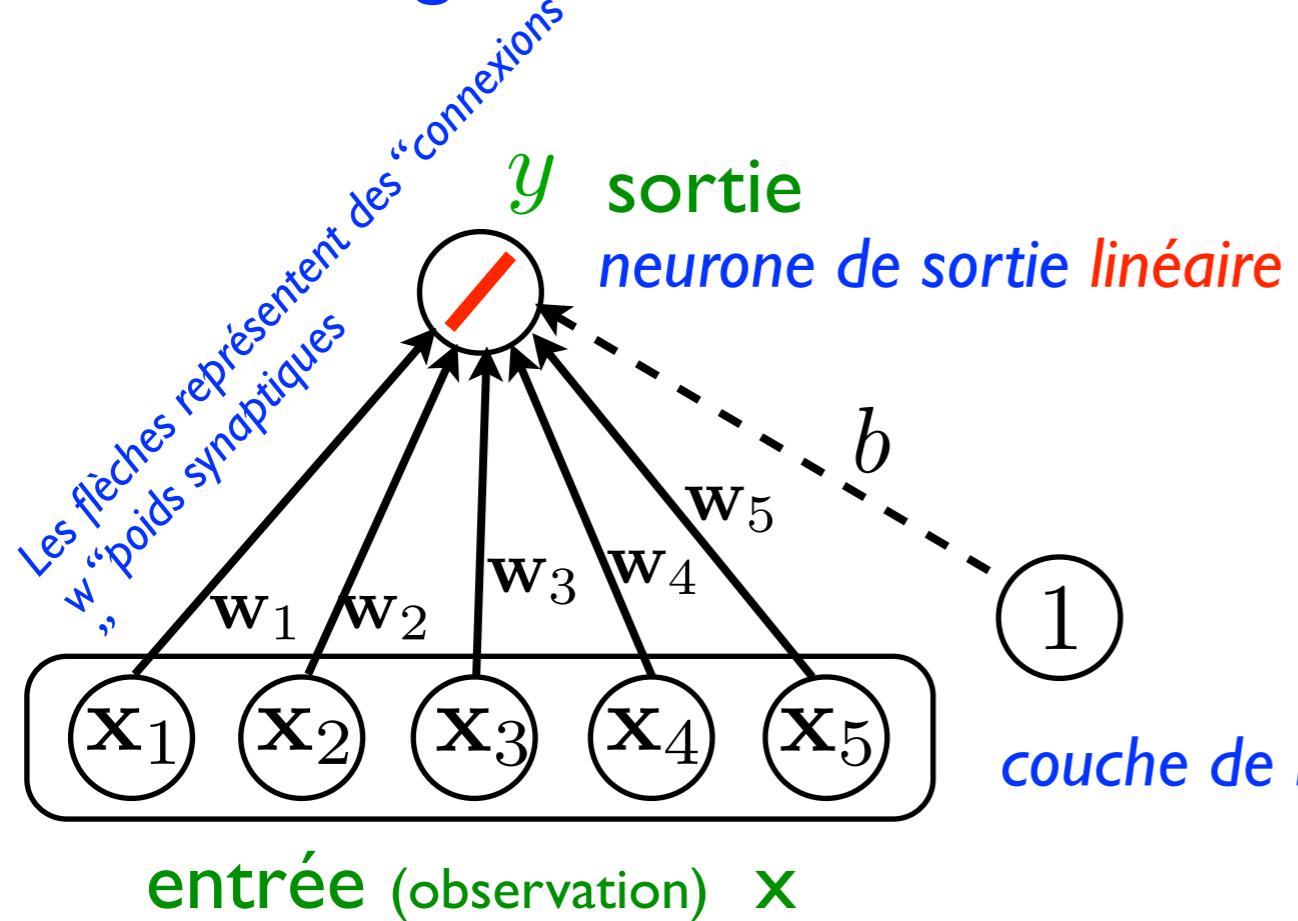
Terminologie neuronale

Neurone linéaire

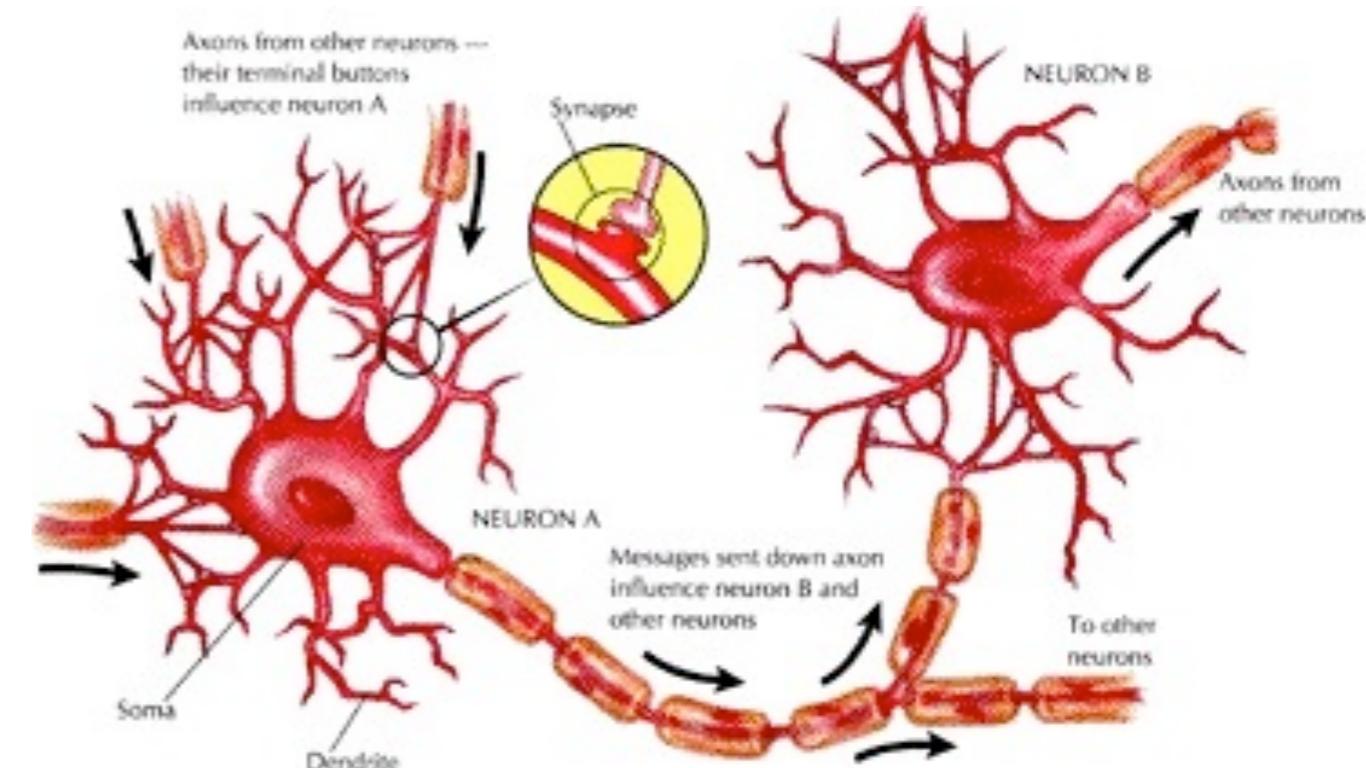
Compréhension intuitive du produit scalaire
chaque composante de x a une influence pondérée sur la sortie y

$$y = f_{\theta}(\mathbf{x}) = \mathbf{w}_1 \mathbf{x}_1 + \mathbf{w}_2 \mathbf{x}_2 + \dots + \mathbf{w}_d \mathbf{x}_d + b$$

Terminologie neuronale



couche de neurones d'entrée

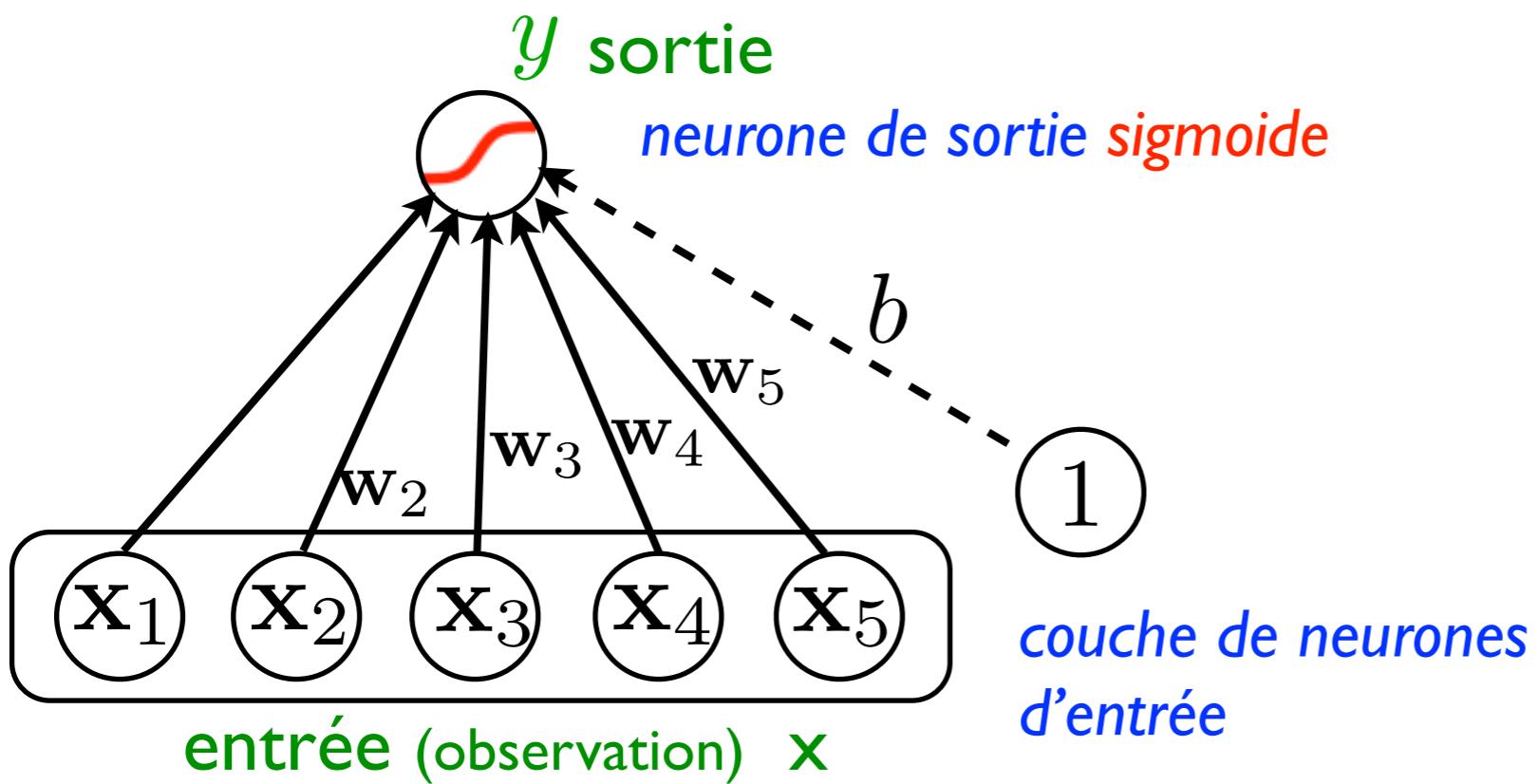
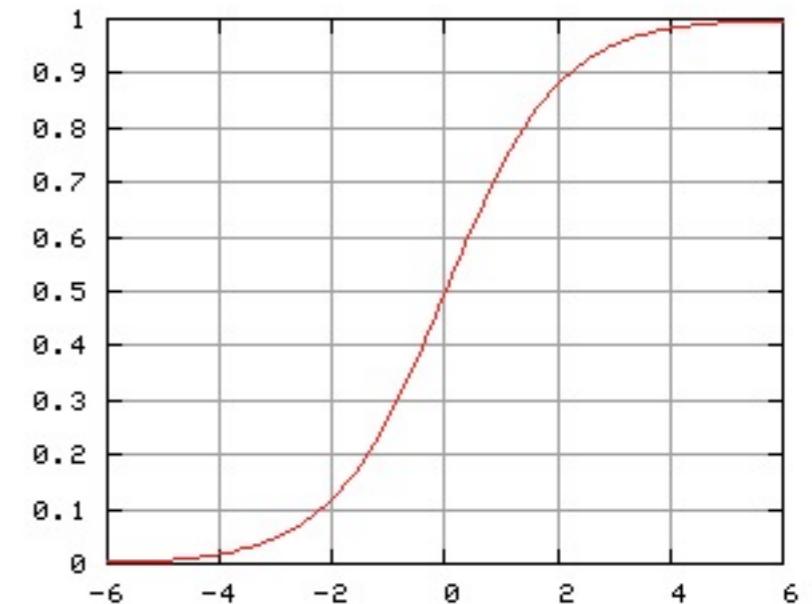


Neurone à fonction d'activation sigmoïde

$$f_{\theta}(\mathbf{x}) = f_{\mathbf{w}, b}(\mathbf{x}) = \text{sigmoid}(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

non-linéarité, fonction de transfert ou d'activation

$$\text{logistic sigmoid}(x) = \frac{1}{1 + e^{-x}}$$



Sigmoïde peut être vue comme

- Une alternative différentiable «douce» à la fonction escalier (décision binaire selon un seuil)
- Une approximation de la réponse en “taux d’impulsion” dans les neurones biologiques.

Autres fonctions d'activation courantes

Soit $a = w^T x + b$ l'activation linéaire d'un neurone.

Voici quelques fonctions d'activation (non linéarités) souvent employées:

- neurones binaires "durs": $I_{\{a>0\}}$
- $\text{sigmoid}(a) = \frac{1}{1+e^{-a}}$ (courbe en S: $\mathbb{R} \rightarrow [0, 1]$)
- $\tanh(a)$ (courbe en S: $\mathbb{R} \rightarrow [-1, 1]$)
- $\text{rectifier}(a) = [a]_+ = xI_{\{a>0\}} = (a \text{ si } a > 0, 0 \text{ sinon})$
- $\text{softplus}(a) = \log(1 + \exp(a))$ (version "douce" du rectifier)
- valeur absolue, carré, ...
- Neurones de type "RBF" (Radial Basis Function):
 $\exp(-\beta \|x - w\|^2)$

Rappel: la Régression Logistique (cas de deux classes)

Attention: il s'agit d'un algo de classification! (en dépit du nom)

Pour une tâche de **classification binaire**:

$$t \in \{0, 1\}$$

On veut estimer la **probabilité conditionnelle**:

$$y \simeq P(t = 1 | \mathbf{x})$$

On choisit

$$y \in [0, 1]$$

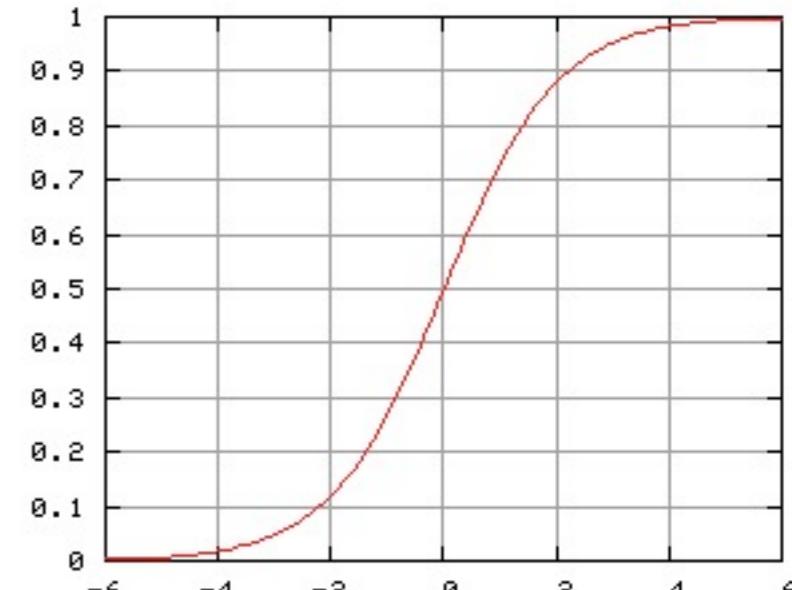
Une fonction de prédiction **non-linéaire** donnant une **probabilité**

$$f_{\theta}(\mathbf{x}) = f_{\mathbf{w}, b}(\mathbf{x}) = \underbrace{\text{sigmoid}(\langle \mathbf{w}, \mathbf{x} \rangle + b)}$$

non-linéarité, fonction de transfert ou d'activation

$$\text{logistic sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

La sigmoïde logistique est l'inverse de la fonction de lien *logit* dans la terminologie des Modèles Linéaires Généralisés (GLMs).



Perte d'entropie croisée («cross-entropy»)

$$L(y, t) = -(t \ln(y) + (1 - t) \ln(1 - y))$$

Ici, pas de solution analytique mais l'optimisation est **convexe**

Minimization du risque empirique

On doit spécifier:

- Une forme paramétrée pour la fonction f_θ
- Une fonction de coût (ou perte) spécifique $L(y, t)$

On définit alors le **risque empirique** comme:

$$\hat{R}(f_\theta, D_n) = \sum_{i=1}^n L(f_\theta(\mathbf{x}^{(i)}), t^{(i)})$$

c.a.d. perte totale sur l'ensemble d'entraînement

L'apprentissage revient à trouver la valeur optimale des paramètres:

$$\theta^\star = \arg \min_{\theta} \hat{R}(f_\theta, D_n)$$

C'est le principe de minimisation du risque empirique

Optimisation par descente de gradient

$$D_n = \{(x^{(1)}, t^{(1)}), \dots, (x^{(n)}, t^{(n)})\} \quad t^{(i)} \in \{0, 1\}$$

$$\hat{R}_\lambda(f_\theta, D_n) = \left(\sum_{i=1}^n L(f_\theta(\mathbf{x}^{(i)}), t^{(i)}) \right) + \underbrace{\lambda \Omega(\theta)}_{\text{terme de régularisation}}$$

- On initialise les paramètres aléatoirement
- On les mets à jour itérativement en suivant le gradient

Soit **descente de gradient batch** (par lot):

BOUCLE: $\theta \leftarrow \theta - \eta \frac{\partial \hat{R}_\lambda}{\partial \theta}$

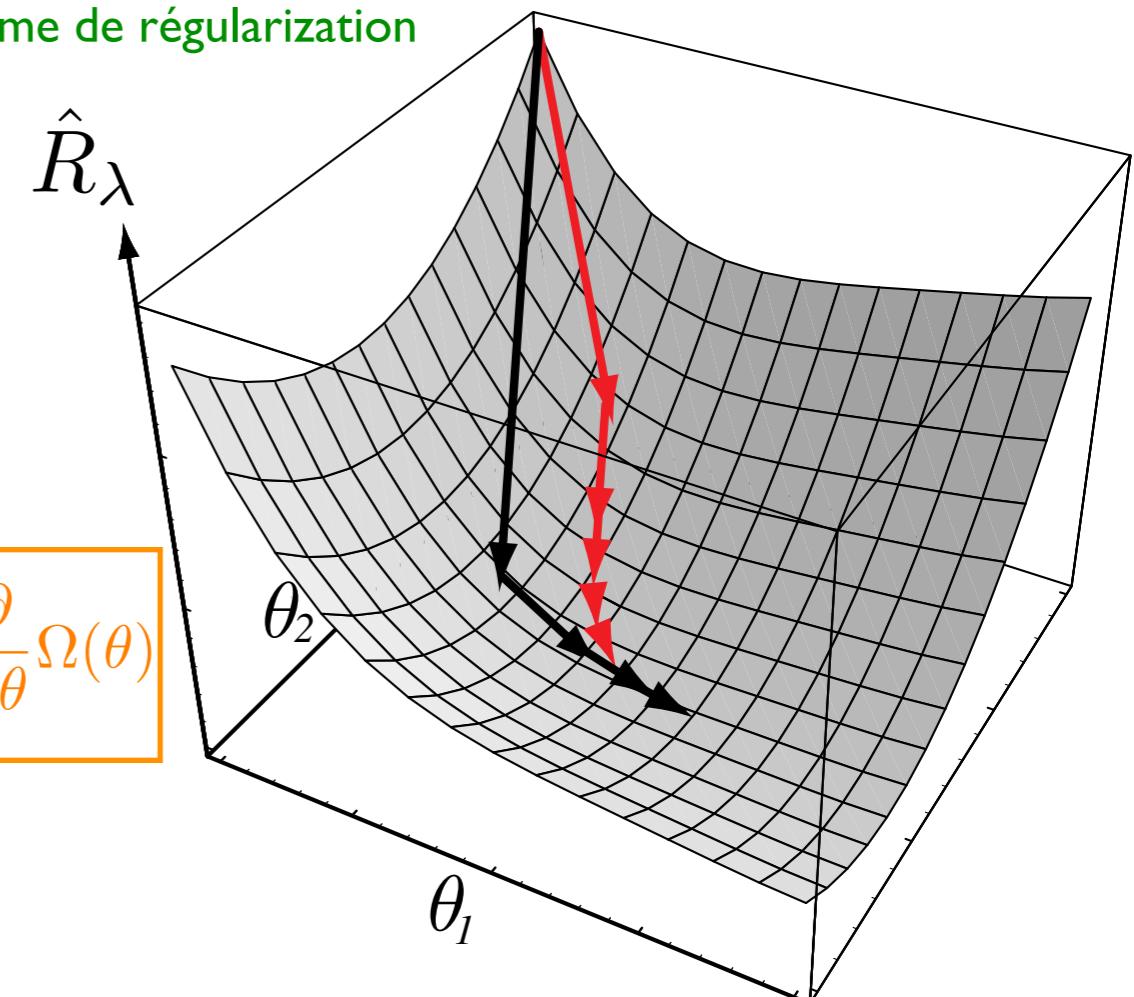
$$= \boxed{\left(\sum_{i=1}^n \frac{\partial}{\partial \theta} L(f_\theta(\mathbf{x}^{(i)}), t^{(i)}) \right) + \lambda \frac{\partial}{\partial \theta} \Omega(\theta)}$$

Ou **descente de gradient stochastique**:

BOUCLE:
 Choisir i dans 1...n

$$\theta \leftarrow \theta - \eta \frac{\partial}{\partial \theta} \left(L(f_\theta(\mathbf{x}^{(i)}), t^{(i)}) + \frac{\lambda}{n} \Omega(\theta) \right)$$

Ou **d'autres méthodes de descente de gradient**
(gradient conjugué, Newton, gradient naturel, ...)



Limitations de la Régression Logistique: ça demeure un classifieur linéaire!

On décide classe 1 si $P(t = 1|x) > 0.5$ (classe 0 sinon).

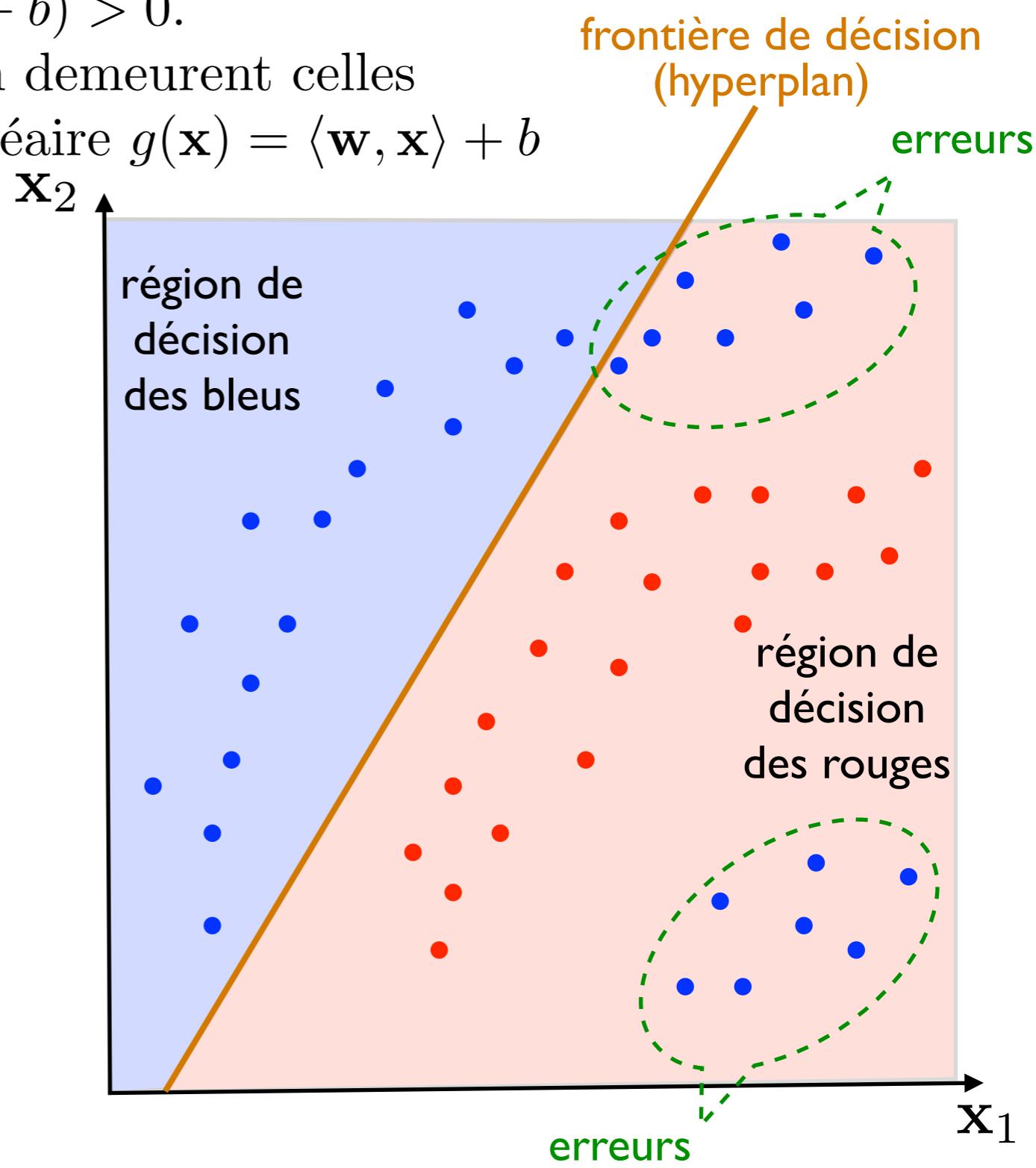
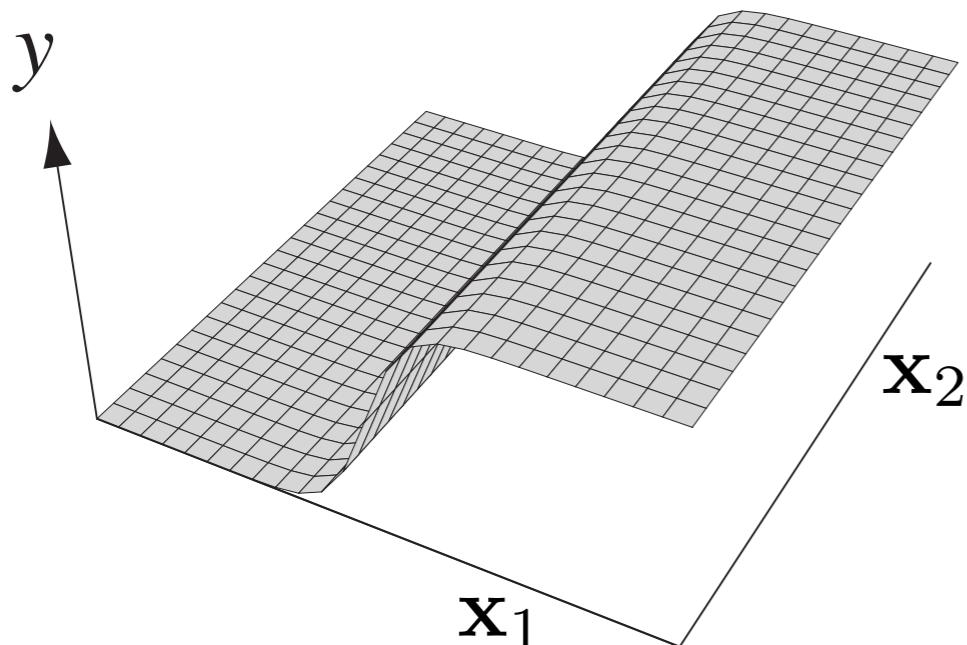
Or $\text{sigmoid}(\langle \mathbf{w}, \mathbf{x} \rangle + b) > 0.5 \equiv \langle \mathbf{w}, \mathbf{x} \rangle + b > 0$.

Donc les régions et frontière de décision demeurent celles associées à la fonction discriminante linéaire $g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$

La frontière de décision est un hyperplan.

→ inappropriate si les classes ne sont pas linéairement séparables

(ex: voir figure)

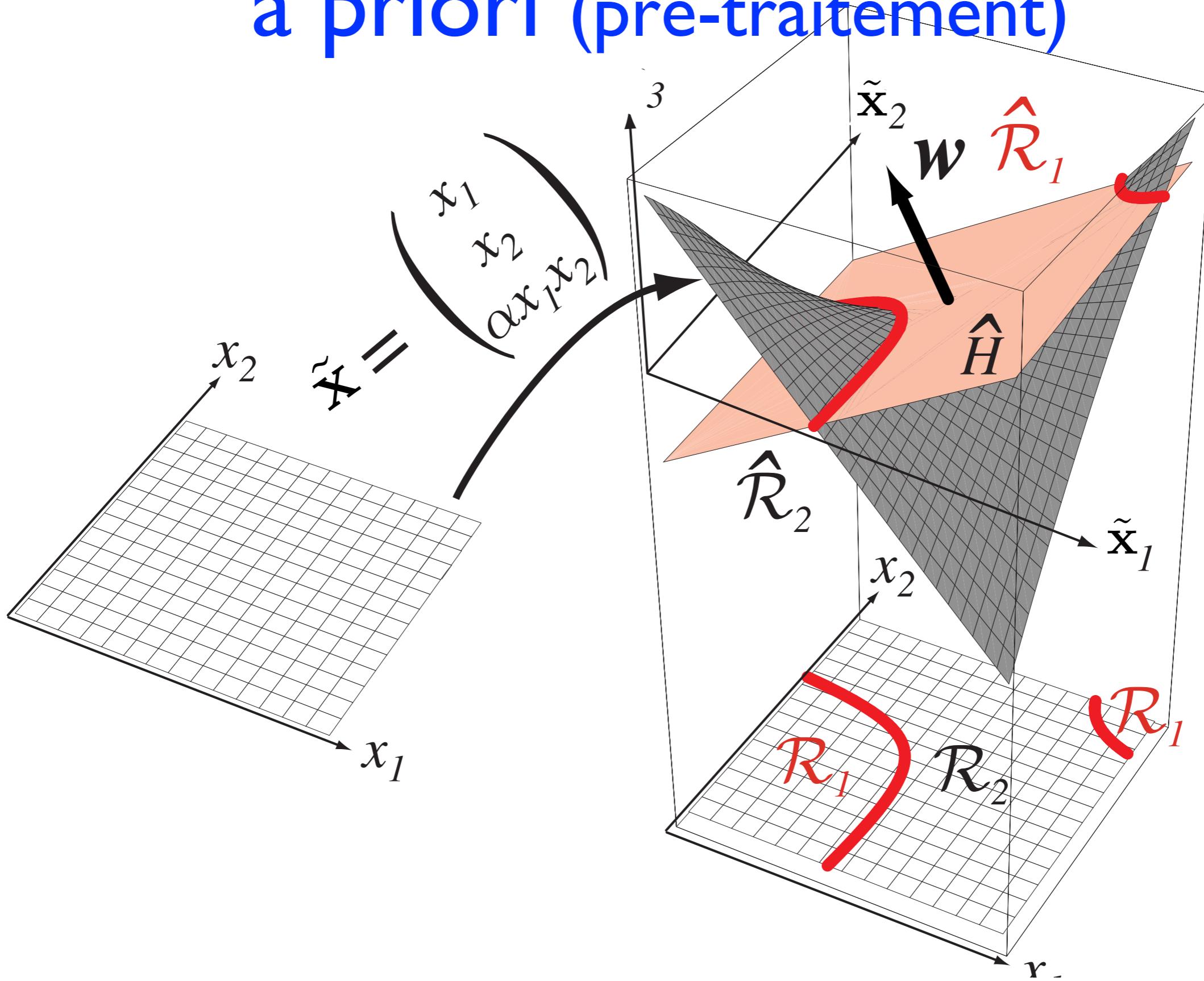


Comment obtenir un classifieur non linéaire à partir d'un classifieur linéaire?

Une vieille technique...

- D'abord appliquer une transformation non-linéaire à \mathbf{x} (projection dans un nouvel espace de traits caractéristiques)
$$\tilde{\mathbf{x}} = \phi(\mathbf{x})$$
- Apprendre une fonction discriminante linéaire dans ce nouvel espace (avec l'une des nombreuses techniques de classifieur linéaire)
- L'hyperplan séparateur dans ce nouvel espace correspond à une **frontière de décision non linéaire dans l'espace d'origine.**
- Et voilà! \Rightarrow **Classifieur non-linéaire de plus forte capacité.**

Ex. transformation non-linéaire à priori (pré-traitement)



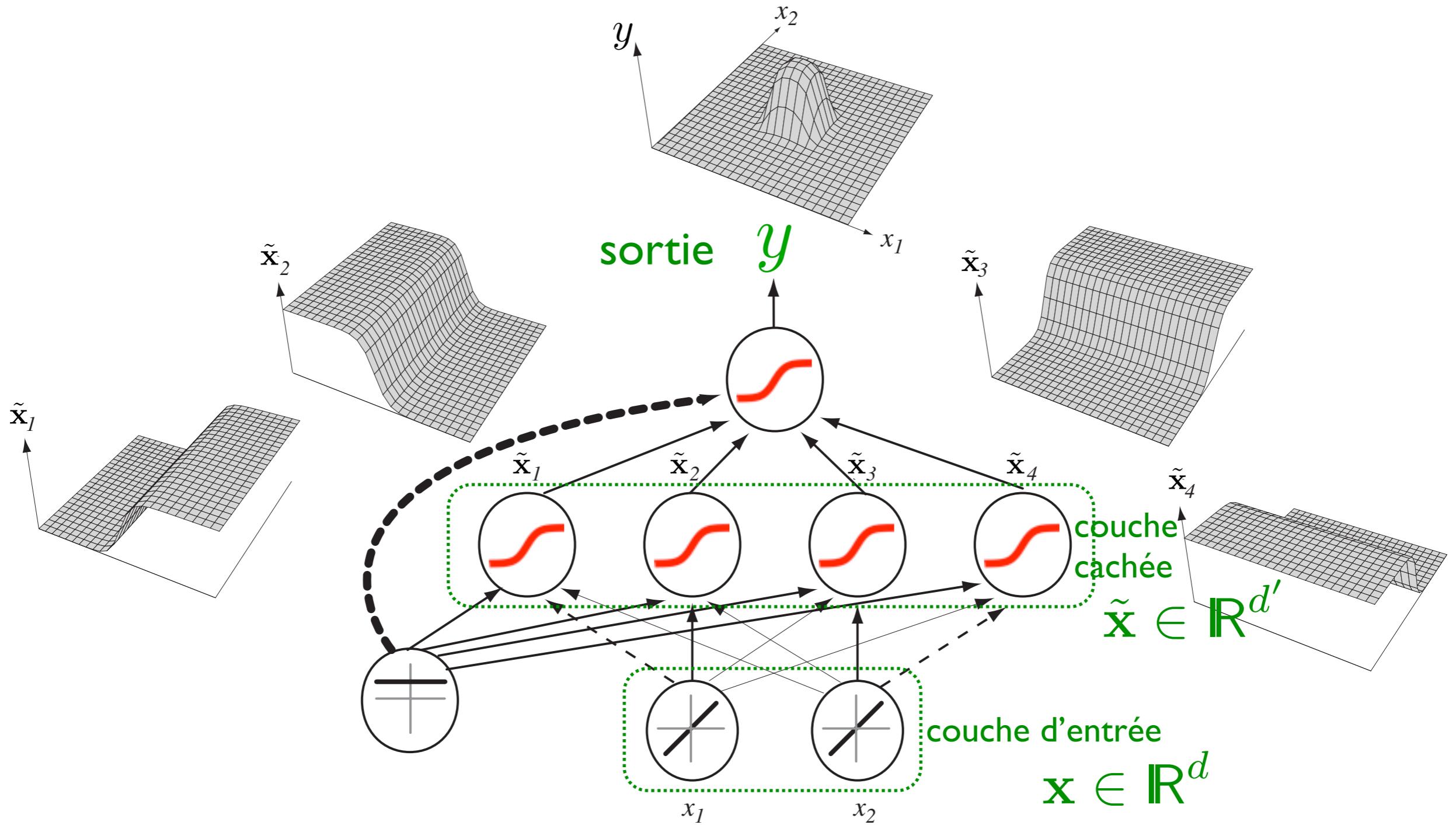
Mais quelle transformation?

Trois manières d'avoir une transformation $\tilde{\mathbf{x}} = \phi(\mathbf{x})$

- Utiliser une transformation fixée à priori explicitement
 - ➡ Exemple précédent
- Utiliser une transformation fixée à priori implicitement
 - ➡ Méthodes à noyau (SVM à noyau, Régression Logistique à noyau, ...)
- Apprendre les paramètres d'une transformation paramétrée:
 - ➡ Réseaux de neurones de type Perceptron Multicouche (M_{ulti}L_{ayer}P_{erceptron})

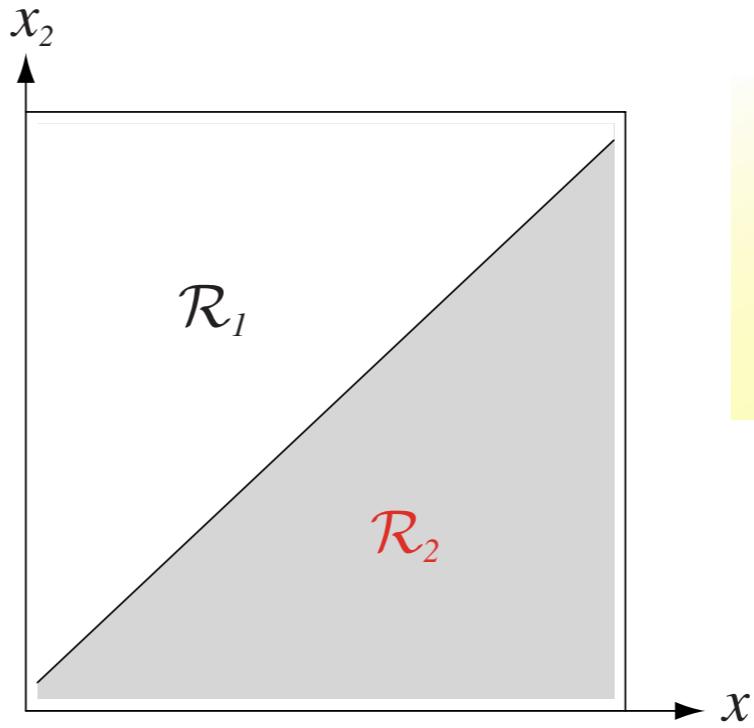
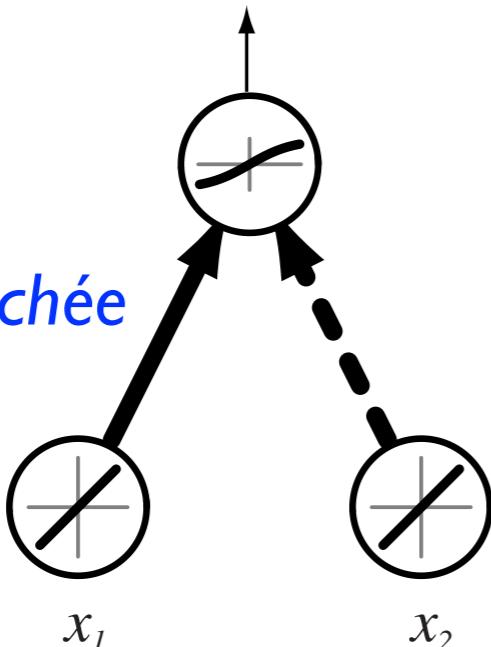
Réseau de Neurones:

Perceptron Multicouche (M_{ulti}L_{ayer}P_{erceptron})
avec une couche cachée de taille 4 neurones



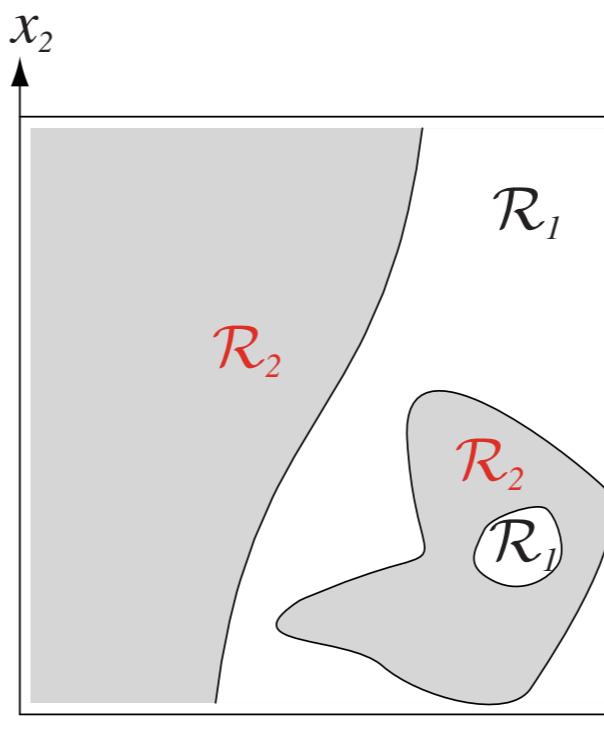
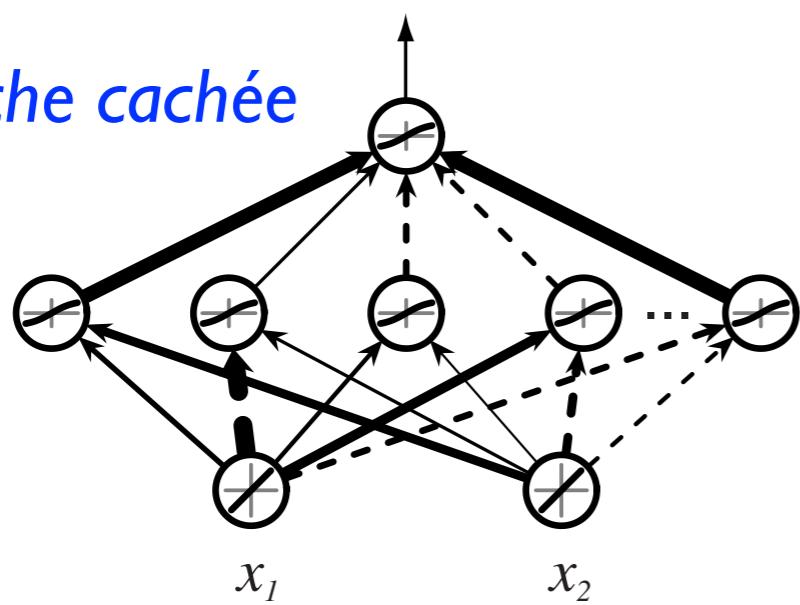
Capacité expressive de réseaux de neurones MLP à une couche cachée.

Sans couche cachée



== Régression logistique
limité à une frontière de
décision du type
hyperplan

Une couche cachée



Propriété
d'approximateur
universel

Toute fonction continue
peut être approximée
(aussi près que l'on veut avec
un nombre d'unités cachées
croissant)

Réseau de neurones (MLP)

avec une couche cachée de d' neurones
et un neurone de sortie sigmoid.

Forme de la fonction:

$$y = f_{\theta}(\mathbf{x}) = \text{sigmoid}(\langle \mathbf{w}, \tilde{\mathbf{x}} \rangle + b)$$

$$\tilde{\mathbf{x}} = \text{sigmoid}(\underbrace{\mathbf{W}^{\text{hidden}}}_{d' \times d} \mathbf{x} + \underbrace{\mathbf{b}^{\text{hidden}}}_{d' \times 1})$$

Paramètres:

$$\theta = \{\mathbf{W}^{\text{hidden}}, \mathbf{b}^{\text{hidden}}, \mathbf{w}, b\}$$

Optimisation des paramètres sur l'ensemble d'entraînement
(*entraînement du réseau*):

$$\theta^* = \arg \min_{\theta} \hat{R}_{\lambda}(f_{\theta}, D_n)$$

$\left(\sum_{i=1}^n L(f_{\theta}(\mathbf{x}^{(i)}), t^{(i)}) \right) + \underbrace{\lambda \Omega(\theta)}_{\text{terme de regularisation (weight decay)}}$

risque empirique

Réseau de neurones (MLP)

avec une couche cachée de d' neurones
et m neurones de sortie sigmoid

Forme de la fonction:

$$y = f_{\theta}(\mathbf{x}) = \text{sigmoid}(\underbrace{\mathbf{W}^{\text{out}}}_{m \times d'} \underbrace{\tilde{\mathbf{x}}}_{d'} + \mathbf{b}^{\text{out}})$$
$$\tilde{\mathbf{x}} = \text{sigmoid}(\underbrace{\mathbf{W}^{\text{hidden}}}_{d' \times d} \underbrace{\mathbf{x}}_d + \mathbf{b}^{\text{hidden}})$$

Paramètres:

$$\theta = \{\mathbf{W}^{\text{hidden}}, \mathbf{b}^{\text{hidden}}, \mathbf{W}^{\text{out}}, \mathbf{b}^{\text{out}}\}$$

Optimisation des paramètres sur l'ensemble d'entraînement
(*entraînement du réseau*):

$$\theta^* = \arg \min_{\theta} \hat{R}_{\lambda}(f_{\theta}, D_n)$$
$$\underbrace{\left(\sum_{i=1}^n L(f_{\theta}(\mathbf{x}^{(i)}), t^{(i)}) \right)}_{\text{risque empirique}} + \underbrace{\lambda \Omega(\theta)}_{\text{terme de regularisation (weight decay)}}$$

Quelle perte pour des neurones de sortie sigmoid?

Perte (erreur) quadratique

$$L(y, t) = \|y - t\|^2 = \sum_{k=1}^m (y_k - t_k)^2$$

OU Perte d'entropie croisée («cross-entropy»)

$$L(y, t) = - \sum_{k=1}^m t_k \ln(y_k) + (1 - t_k) \ln(1 - y_k)$$

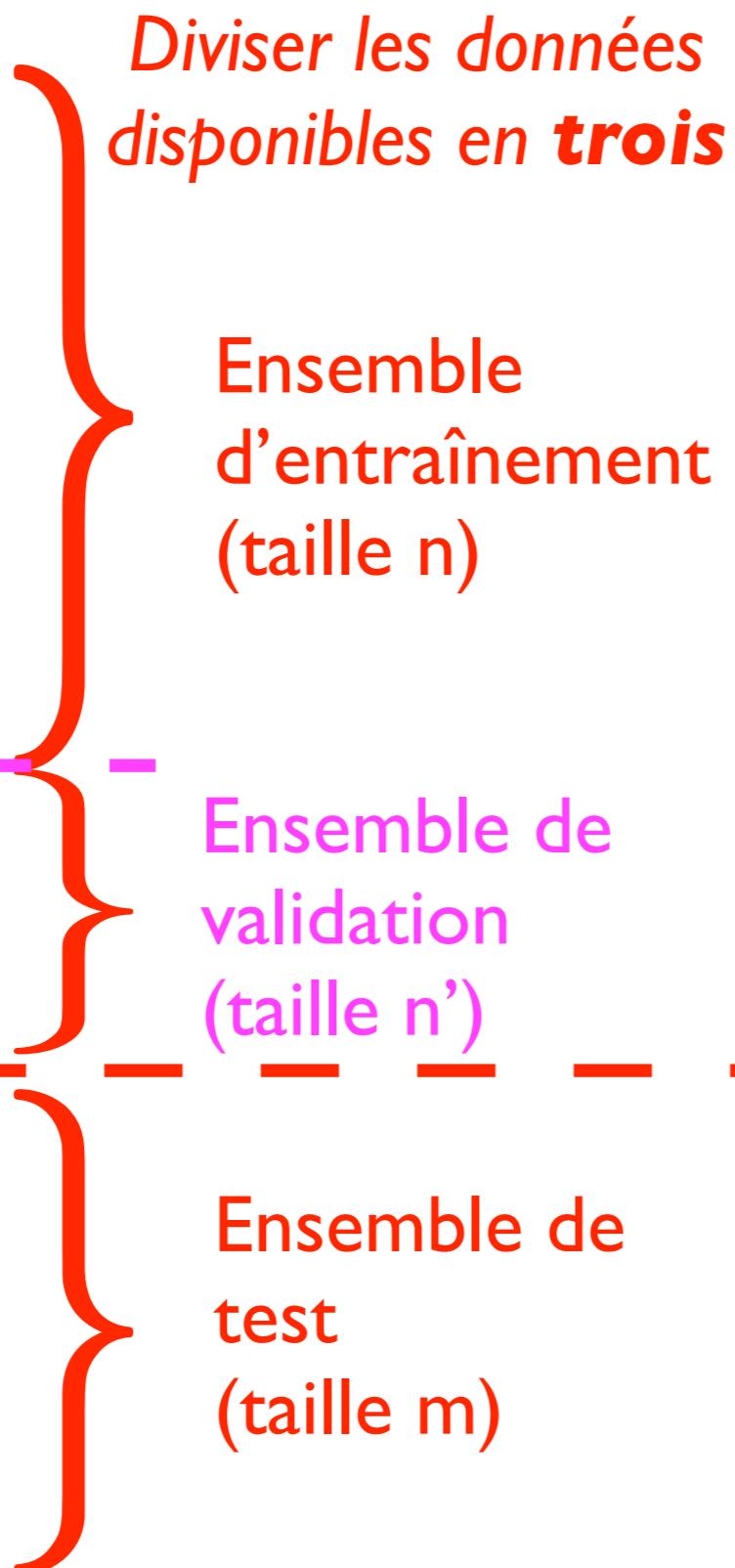
Hyper-paramètres contrôlant la capacité

- * Le réseau a un ensemble de *paramètres*: θ
 - optimisés sur l'ensemble d'entraînement par descente de gradient.
- * Il faut aussi ajuster des *hyper-paramètres* qui contrôlent la “capacité” du réseau
 - nombre de neurones cachés d
 - contrôle de régularisation λ (weight decay)
 - arrêt prématué de l'optimisation (nombre d'époques d'entraînement)
et d'autres qui sont nécessaires à la technique d'optimisation
ex: pas de gradient ou taux d'apprentissage (*learning rate*)
- choisis par une procédure de sélection de modèle **sur un ensemble de validation** disjoint de l'ensemble d'entraînement.

Ajustement des hyper-paramètres

$D =$

$(\mathbf{x}^{(1)}, t^{(1)})$
$(\mathbf{x}^{(2)}, t^{(2)})$
\vdots
$(\mathbf{x}^{(N)}, t^{(N)})$



Si on a trop peu d'exemples, on peut utiliser la validation croisée en k blocs ou leave-one-out (“jack-knife”)

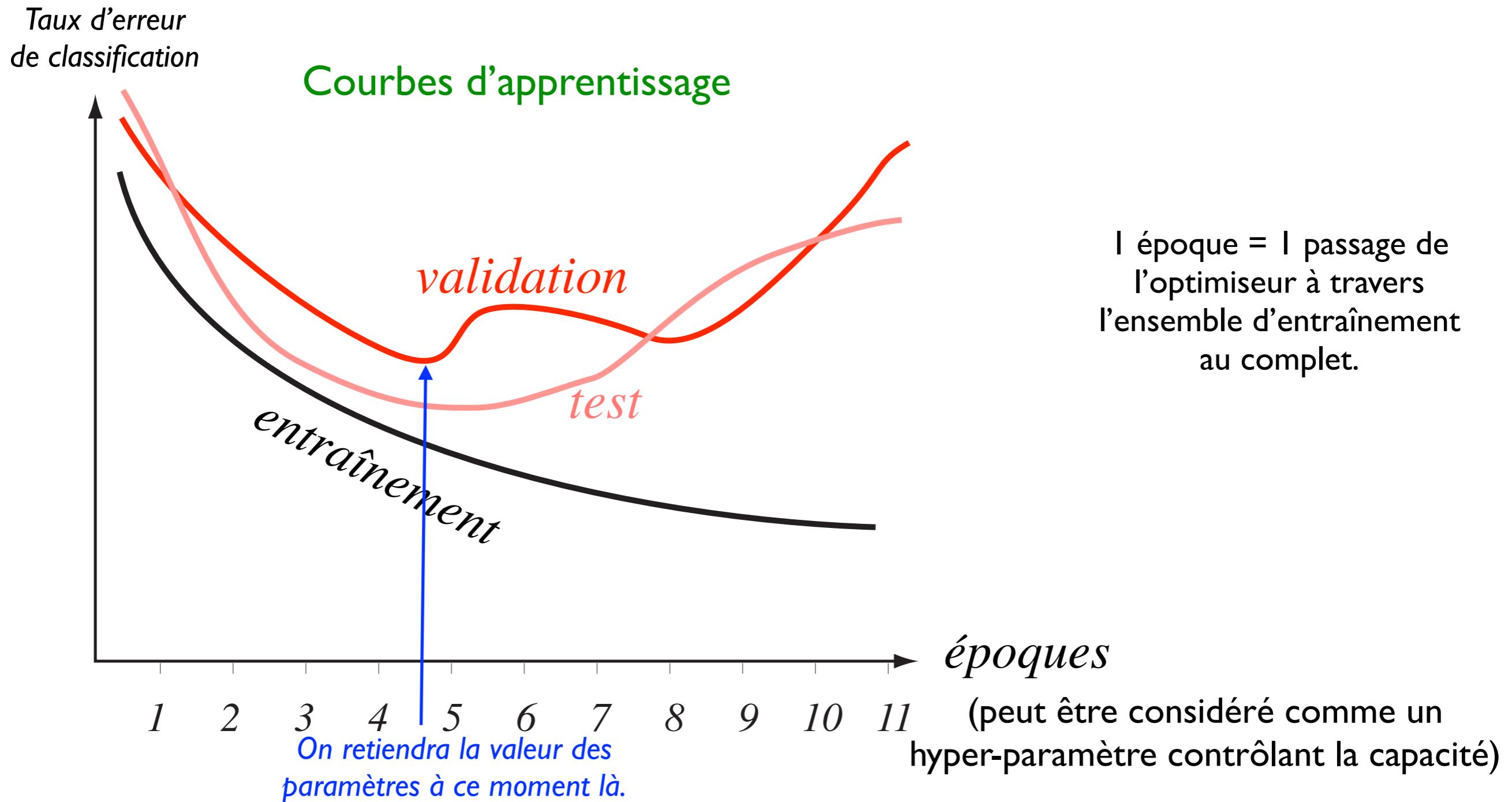
Pour chaque combinaison de valeurs d'hyper-paramètres considérée:

- 1) *Entraîner le réseau*, c.a.d. trouver la valeur des paramètres qui minimisent le risque empirique regularisé *sur l'ensemble d'entraînement* par descente de gradient.
- 2) *Évaluer la performance sur l'ensemble de validation* en se basant sur le critère qui nous intéresse vraiment (ex. erreur de classification)

Retenir la valeur des hyper-paramètres et le réseau entraîné qui a donné la meilleure performance de validation.
(Optionnellement réentraîner sur l'union des ensembles d'entraînement et de validation).

Évaluer la performance de généralisation sur l'ensemble de test, qui n'a jamais été utilisé ni pour trouver les paramètres ni les hyper-paramètres (pour une évaluation non-biaisée de la perf. de généralisation).

Ex: arrêt prématué de l'optimisation



En résumé

- Les réseaux de neurones de type *feed-forward* (comme les MLPs) sont des **fonctions non-linéaires paramétrées**.
- ... entraînés par **descente de gradient** à minimiser un objectif sur un ensemble d'entraînement.
- Les nombreux détails architecturaux (ex: taille et type des couches cachées) et autres hyper-paramètres permettant de **contrôler la capacité**, doivent être ajustés avec une technique de sélection de modèle rigoureuse
(ex: validation croisée avec ensemble de validation séparé).

Note: il y a beaucoup d'autres types de Réseaux de Neurones artificiels...

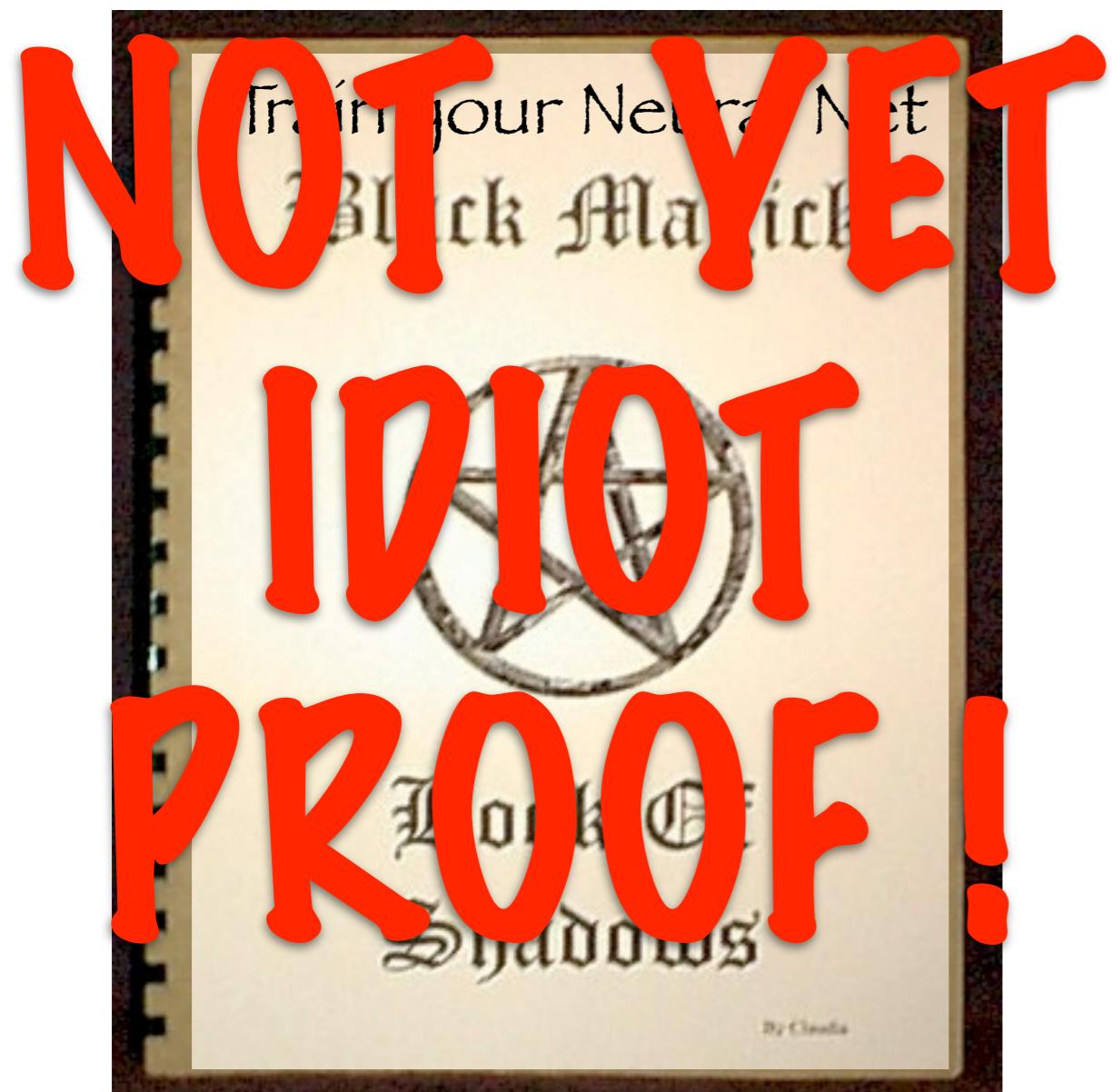
Détails pratiques importants

- Le calcul efficace du gradient se fait par la technique de **rétropropagation des gradients** (utilisation intelligente de la règle de dérivation en chaîne qui évite de refaire inutilement des calculs)
- Importance d'avoir une représentation des données appropriée au départ (traits catégoriques en représentation one-hot, traits continus standardisés)
- Importance de l'initialisation aléatoire des poids dans un intervalle approprié.
- Sensibilité aux hyper-paramètres de la technique de descente de gradient (learning rate, ...)

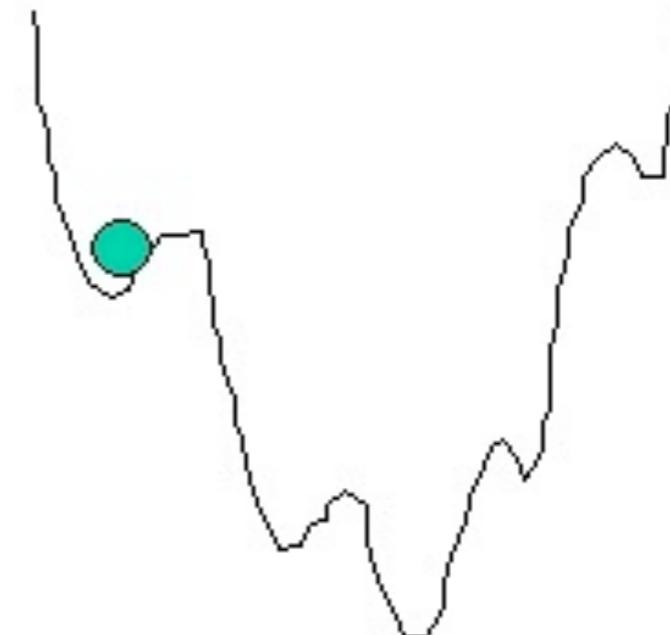
=> Détails à voir dans les prochaines démos et devoirs.

Réseaux de neurones: de nouveau «à la mode»

- Difficiles à entraîner
(beaucoup d'hyper-parameters à ajuster)



- Optimisation non-convexe
 - ➡ minima locaux: solution depend d'où on part...

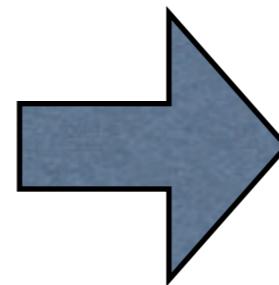


Convexité peut être trop restrictive

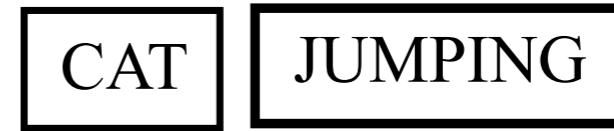
Les optimisations convexes sont mathématiquement plus sympathiques et plus simples, mais les problèmes complexes du monde réel peuvent nécessiter des modèles non-convexes.

Réseaux multi-couches profonds

La notion de niveau de représentation



very high level representation:



... etc ...

slightly higher level representation

raw input vector representation:

$$\mathcal{X} = \begin{bmatrix} 23 & 19 & 20 & \cdots & 18 \end{bmatrix}$$

$x_1 \quad x_2 \quad x_3 \quad \vdots \quad x_n$

