



# Apprentissage automatique

Rym Guibadj

LISIC, EILCO



## Contenu

- Introduction
- Apprentissage supervisé
- Apprentissage non supervisé
- Apprentissage par renforcement

## Volume horaire

- 5 x 2h Cours
- 8 x 2h TD machine

## Evaluation

- TP
- Examen
- Note module =  $0.3 \times \text{TP} + 0.7 \times \text{Examen}$

# Références



- Apprentissage artificiel : Concepts et algorithmes, Antoine Cornuéjols et Laurent Miclet
- Intelligence Artificielle, Stuart Russel et Perter Norving, édition Pearson
- Notes du cours de Hugo Larochelle (Université de Sherbrooke)
- Notes du cours de Fabien Tytaud (Université du Littoral Côte d'Opale)



# Apprentissage automatique

## Quelques domaines de l'IA

- Représentation des connaissances
- Résolution de problèmes
- Reconnaissance de la parole
- Reconnaissance de l'écriture
- Reconnaissance des visages
- Robotique
- **Apprentissage automatique (artificiel)**

# Apprentissage automatique

## Définitions

- L'**apprentissage automatique** fait référence au développement, à l'analyse et à l'implémentation de méthodes qui permettent à une machine d'**évoluer** et ainsi de remplir des tâches qu'il est difficile ou impossible de remplir par des moyens algorithmiques plus classiques.
- "L'**apprentissage** dénote des **changements** dans un système qui ... lui permet de faire la même tâche **plus efficacement la prochaine fois**". *Herbert Simon*.



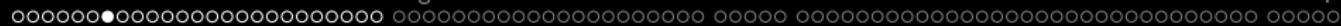
# Apprentissage automatique

- On dira qu'une machine **apprend** dès lors qu'elle change sa structure, son programme ou ses données en fonction de données en entrée ou de réponses à son environnement de sorte à ce que **ses performance futures deviennent meilleures**
- L'objectif de l'apprentissage automatique est de **concevoir des programmes pouvant s'améliorer automatiquement avec l'expérience**



# Pourquoi l'apprentissage automatique ?

- Certaines tâches ne sont bien définies que via un ensemble d'exemples
- Pour découvrir des relations importantes dans des données (fouille de données)
- Les machines peuvent ne pas fonctionner sur tous les environnements
- La quantité de connaissances disponibles à propos de certaines situations sont telles que le cerveau humain ne puisse les expliciter
- L'environnement change constamment



## Des applications diverses :

- Reconnaissance de la parole.
- Diagnostique médical.
- Moteurs de recherche.
- Les jeux.
- Conduite autonome.
- ...

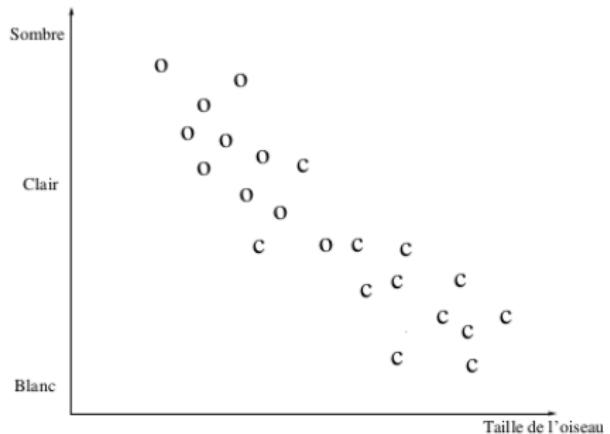
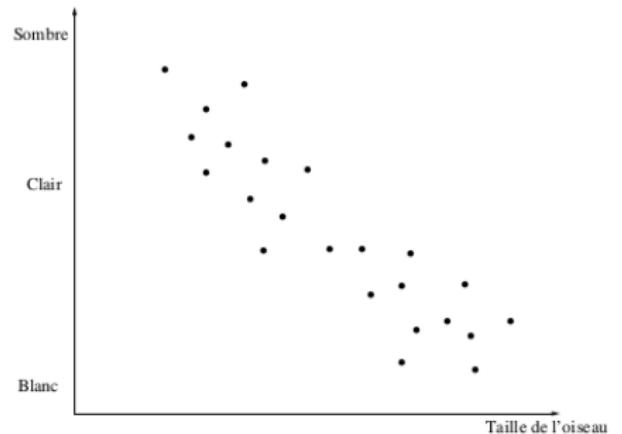
## Exemple d'apprentissage :

- Imaginons un étang sur lequel nagent des oies et des cygnes (nous admettons qu'il n'y a pas d'autres oiseaux dans cette région).
- Deux personnes arrivent dont l'un est expert et l'autre débutant.
- Le débutant souhaite apprendre à distinguer une oie d'un cygne. Il doit se contenter de mesurer ce qui lui paraît caractéristique : le niveau de gris du plumage et la taille de la bête.



## Exemple d'apprentissage :

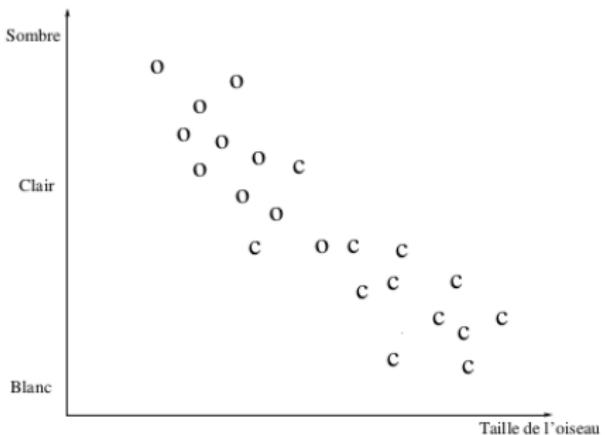
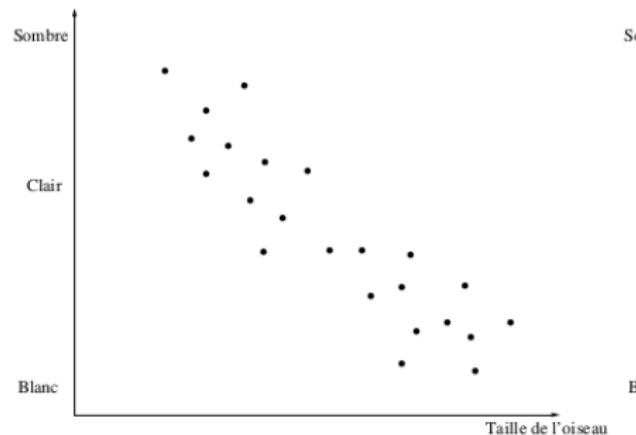
- Le premier graphique représente les mesures de chaque oiseau prise par l'amateur
- Le second graphique représente les mêmes oiseaux étiquetés par l'expert





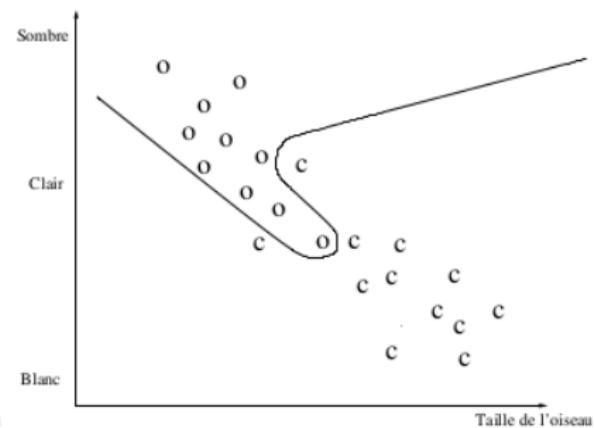
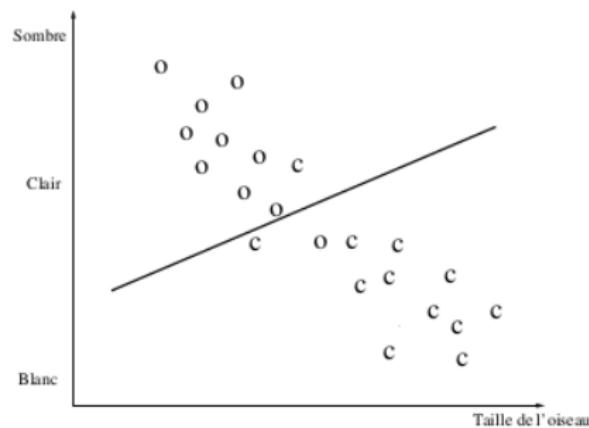
## Exemple d'apprentissage :

- Il faut maintenant que l'amateur trouve une règle permettant de séparer les exemples en minimisant l'erreur.



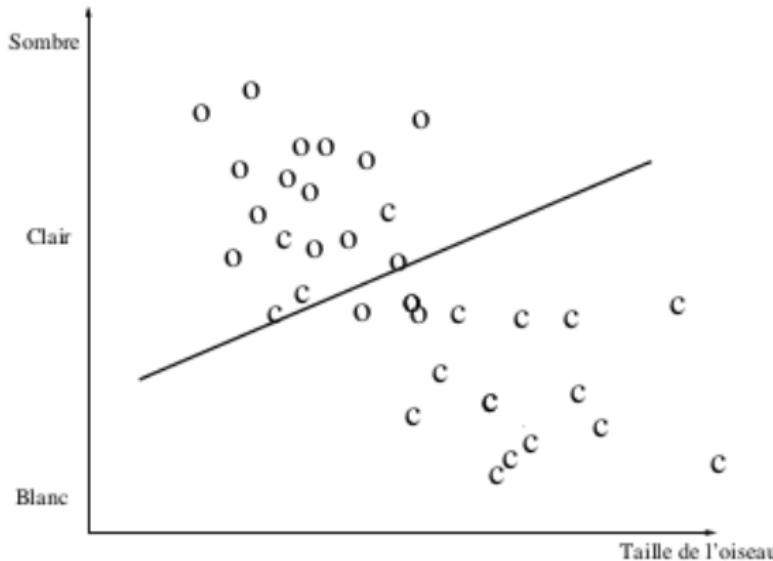
## Exemple d'apprentissage :

- Une règle de décision simple et une règle de décision complexe pour séparer les oies des cygnes.



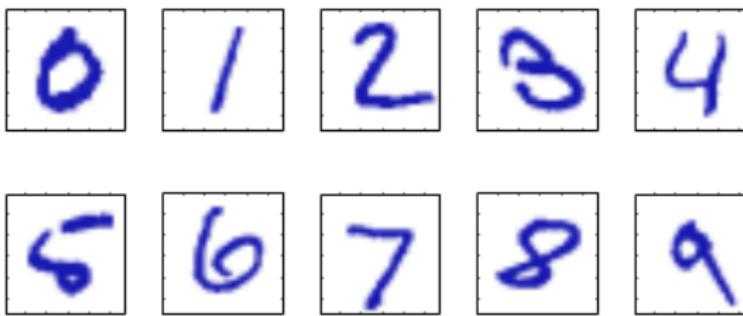
## Exemple d'apprentissage :

- Le test de la règle simple sur d'autres oiseaux.



# Exemple d'apprentissage :

Comment reconnaître des caractères manuscrits

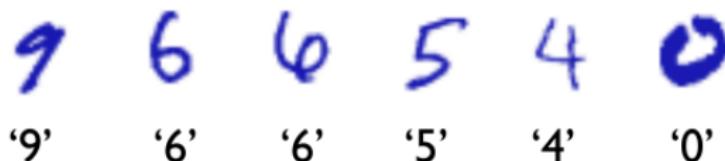


- **Par énumération de règles** : trop fastidieux, difficile de couvrir tous les cas.
- **En donnant à l'ordinateur la capacité d'apprendre** : laisser l'ordinateur faire des essais et apprendre de ces erreurs.

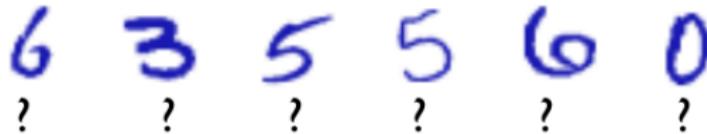


## Données d'entraînement et généralisation

On fournit à l'algorithme d'apprentissage des **données d'entraînements**



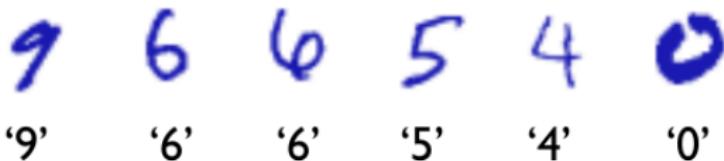
L'algorithme retourne un programme capable de **généraliser** à de nouvelles données



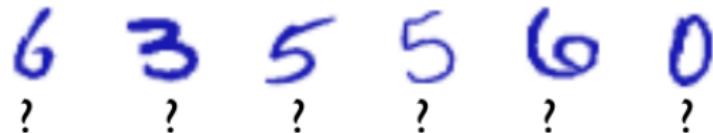
## Données d'entraînement, généralisation et modèle

On note l'**ensemble d'entraînement** :

$\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$ . On appelle  $x_i$  une **entrée** et  $y_i$  la **cible**.



Le «programme» généré par l'algorithme d'apprentissage  $f(x)$  est appelé **un modèle** capable de généraliser à de nouvelles données.





## Ensemble de tests

On utilise un ensemble de test  $D_{test}$  pour mesurer la performance de généralisation de notre modèle  $f(x)$

6 3 5 5 6 0  
'6' '3' '5' '5' '6' '0'



## Types d'apprentissage :

- **Apprentissage supervisé** : il ya une cible à prédire

$$\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$$

- Chaque exemple est associé à une étiquette
- Objectif : prédire l'étiquette de chaque donnée
- Le système apprend à classer les données

- **Apprentissage non supervisé** : cible n'est pas fournie

$$\mathcal{D} = \{x_1, \dots, x_N\}$$

- Les exemples ne sont pas étiquetés
- Objectif : trouver une structure aux données
- Le système apprend une classification des données



## Types d'apprentissage :

### • Apprentissage par renforcement

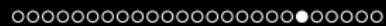
- Les exemples sont (parfois) associés à une récompense ou une punition
- Objectif : trouver les actions qui maximisent les récompenses
- Le système apprend une politique de décision



## Types d'apprentissage :

**Apprentissage supervisé** : il ya une cible à prédire

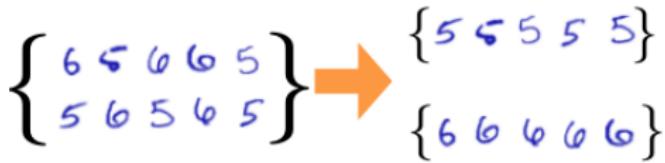
- **classification** : la cible est une classe  $y \in \{1, \dots, K\}$ .  
Exemple : reconnaissance de caractères
  - $x$  : vecteur des intensités de tous les pixels de l'image
  - $y$  : identité du caractère
- **régession** : la cible est un nombre réel  $y \in R$ . Exemple : prédiction de la valeur d'une action à la bourse
  - $x$  : vecteur contenant l'information sur l'activité économique de la journée
  - $y$  : valeur d'une action à la bourse le lendemain



## Types d'apprentissage :

**Apprentissage non supervisé** : cible n'est pas fournie.

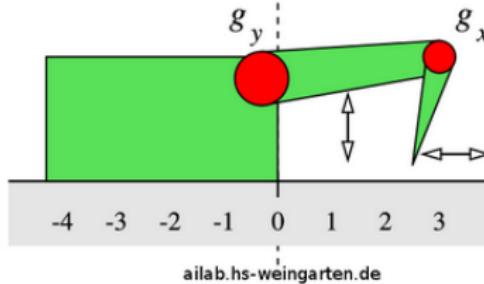
- Partitionnement / *clustering*
- Exemple : partitionnement des caractères manuscrits



# Types d'apprentissage :

## Apprentissage par renforcement

- Exemple : robot motorisé qui avancent sur différents types de sols
  - Observe les effets de ses actions
  - Déduit de ses observations la qualité de ses actions
  - Améliore ses actions futures



	1	2	3	4	5
1	0 0 0 1 2 0 0 -31 0 -50 0	0 0 0 0 0 0 11 -29 32 -65 -47 -5 -52	0 0 -1 0 0 0 4 0 43 0 0 -43	0 0 0 0 0 0 0 0 16 0 -47 0	0 0 0 0 0 0 0 0 0 0 -26 -4 0
2					
3					
4					



# Types d'apprentissage :

## Apprentissage par renforcement

- Exemple : configuration automatique de machines virtuelles d'un cluster de serveur linux dans un contexte de charge dynamique
  - 3 paramètres pertinents ont été choisis pour chacune des 3 VMs utilisées.
  - 3 actions sont possibles sur chaque paramètre : incrémentation, décrémentation ou invariance de celui-ci.
  - Les récompenses sont définies par le SLO (service level objectives) auquel on soustrait le temps de réponse.



# Apprentissage supervisé

- Aussi appelé **analyse discriminante**
- Les données d'apprentissage sont étiquetées
  - Un **expert** ou **oracle** doit préalablement étiqueter des exemples.
- Le processus se passe en deux phases :
  - La **phase d'apprentissage** (*hors ligne*) : déterminer un modèle de données étiquetées
  - La **phase de test** (*en ligne*) : prédire l'étiquette d'une nouvelle donnée, connaissant le modèle préalablement appris.



# Définition formelle

- Données d'apprentissage
  - $N$  couples entrée sortie  $(x_n, y_n)_{1 \leq n \leq N}$  avec  $x_n \in X$  et  $y_n \in Y$
  - On suppose que ces données sont tirées selon une loi (de probablilité) inconnue
- Objectif de l'apprentissage
  - déterminer une fonction de prédiction  $f : X \rightarrow Y$  qui soit en accord avec le données d'apprentissage

## Les différentes représentations

- Si  $f$  est une fonction continue on parle alors de **régession**.
- Si  $f$  est une fonction discrète on parle alors de **classification**.
- Si  $f$  est une fonction binaire on parle alors d'**apprentissage de concept**.



# Méthodes d'apprentissage supervisé

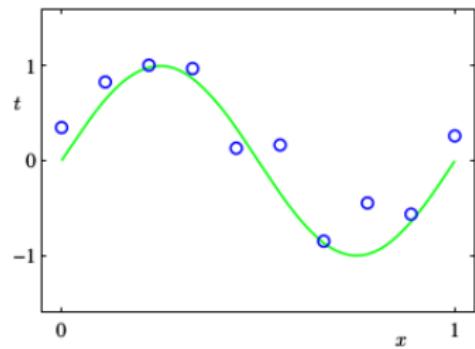
## Liste non exhaustive

- Régression polynomiale
- Les k plus proches voisins
- Arbre de décision
- Les réseaux de neurones
- Les machines à vecteur support
- etc.



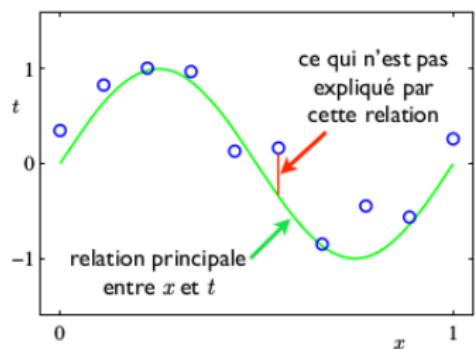
# Régression

- Exemple simple : régression en une dimension ( $1D$ )
  - entrée : scalaire  $x$
  - cible : scalaire  $y$
- Données d'entraînements  $\mathcal{D}$  contiennent :
  - $(x_1, \dots, x_N)^T$
  - $(y_1, \dots, y_N)^T$
- Objectif : faire une prédiction  $\hat{y}$  pour une nouvelle entrée  $\hat{x}$



# Régression

- Exemple simple : régression en une dimension ( $1D$ )
  - entrée : scalaire  $x$
  - cible : scalaire  $y$
- Données d'entraînements  $\mathcal{D}$  contiennent :
  - $(x_1, \dots, x_N)^T$
  - $(y_1, \dots, y_N)^T$
- Objectif : faire une prédiction  $\hat{y}$  pour une nouvelle entrée  $\hat{x}$

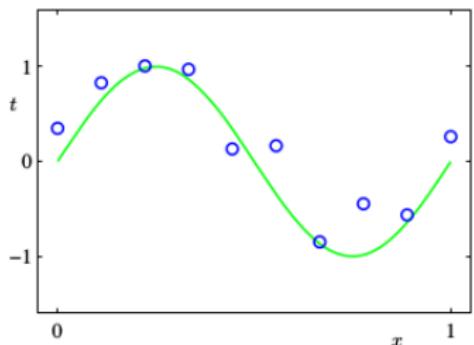


# Régression polynomiale, modèle

- On va supposer qu'une bonne prédiction aurait une forme polynomiale

$$f(x, w) = w_0 + w_1 x + w_2 x^2 + \dots + w_M x^M$$

$$f(x, w) = \sum_{j=0}^M w_j x^j$$



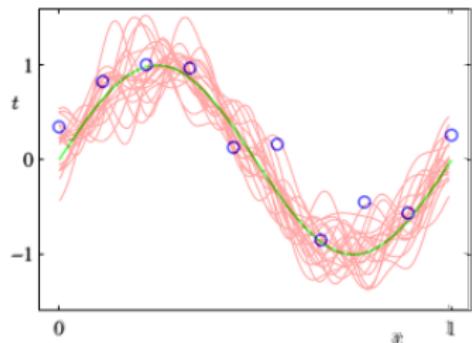
- $f(x, w)$  est notre modèle. Il représente nos hypothèses sur le problème à résoudre
  - La valeur de  $M$  représente le degré du polynôme.
  - Le vecteur  $w = (w_1, w_2, \dots, w_M)$  représente les paramètres du modèle qu'on doit trouver



# Régression : minimisation de perte (coût, erreur)

- Comment trouver  $w$  ? (problème d'apprentissage)
- On cherche le  $w^*$  qui minimise la somme de notre perte / erreur / coût sur l'ensemble d'entraînement

$$E(w) = \frac{1}{2} \sum_{n=1}^N \{f(x_n, w) - y_n\}^2$$

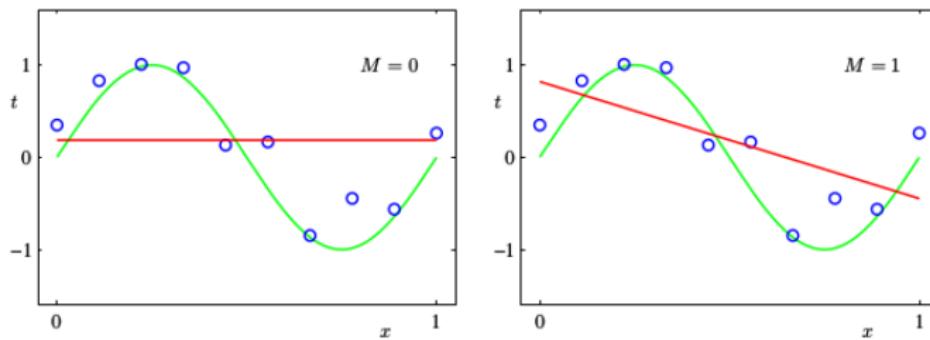


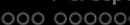
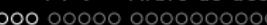
- Un algorithme d'apprentissage résoudrait ce problème : trouver  $w^*$  à partir des données



# Régression : sous-apprentissage

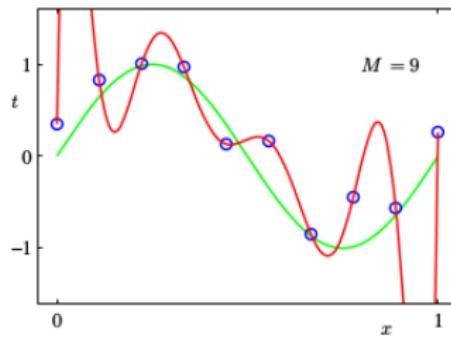
- Comment choisir  $M$  (le degré du polynôme) ?
  - De trop petites valeurs auront une grande perte sur l'ensemble d'entraînement : situation de **sous-apprentissage**

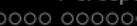




# Régression : sur-apprentissage

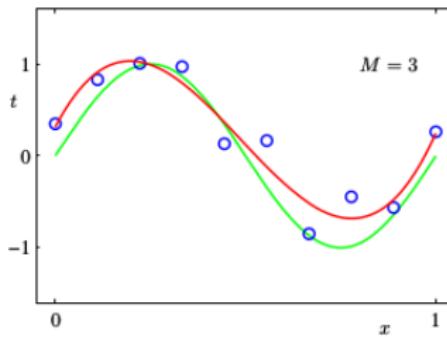
- Comment choisir  $M$  (le degré du polynôme) ?
  - De trop grandes valeurs vont seulement apprendre l'ensemble d'entraînement "par cœur" : situation de **sur-apprentissage**
  - Apprendre à prédire ce que n'est pas prévisible à partir de  $x$  seulement (ex. du bruit)





# Régression : sélection de modèle

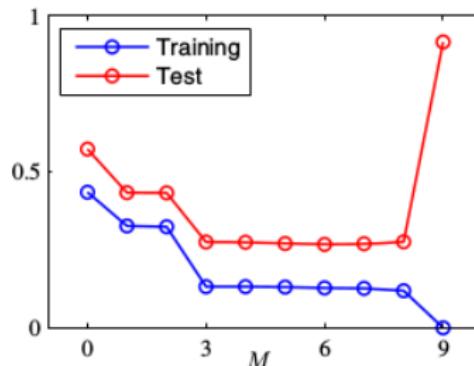
- Comment choisir  $M$  (le degré du polynôme) ?
  - On voudrait une valeur intermédiaire qui permet de retrouver la tendance générale de la relation entre  $x$  et  $y$ , sans le bruit
  - c'est ce qui va permettre de bien généraliser à de nouvelles entrées !
  - trouver cette meilleure valeur de  $M$  s'appelle de la **sélection de modèle**.
  - Il existe différentes techniques pour trouver le meilleur  $M$ .





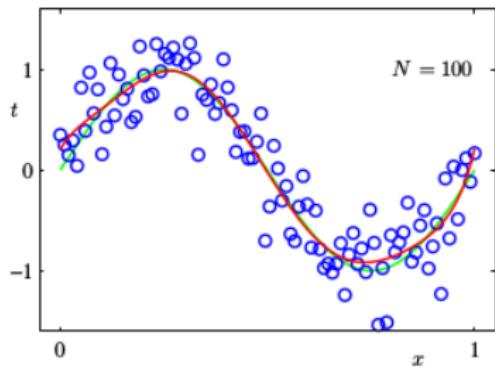
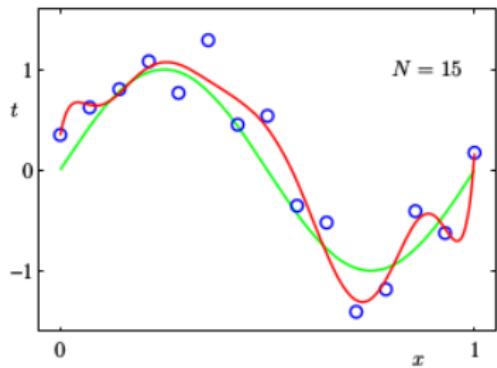
## Régression : capacité d'un modèle, performance

- La **capacité** d'un modèle est son aptitude à apprendre "par cœur". Exemple : plus  $M$  est grand, plus le modèle a de capacité.
- Plus la capacité est grande, plus la différence entre l'erreur d'entraînement et l'erreur de test augmente.
- En régression, l'erreur sur tout un ensemble est souvent mesurée par la racine de la moyenne des erreurs au carré  $E_{RMS} = \sqrt{2E(w^*)/N}$



# Régression : généralisation vs. quantité de données

- Plus la quantité de données d'entraînement augmente, plus le modèle entraîné va bien généraliser



# Régession : régularisation

- Comment utiliser un grand  $M$  avec peu de données
- Régularisation** : on réduit la capacité autrement
  - On pénalise la somme des carrés des paramètres

$$\tilde{E}(x) = \frac{1}{2} \sum_{n=1}^N \{f(x_n, w) - y_n\}^2 + \frac{\lambda}{2} \|w\|^2$$

- où  $\|w\|^2 = w^T w = w_0^2 + w_1^2 + \dots + w_M^2$



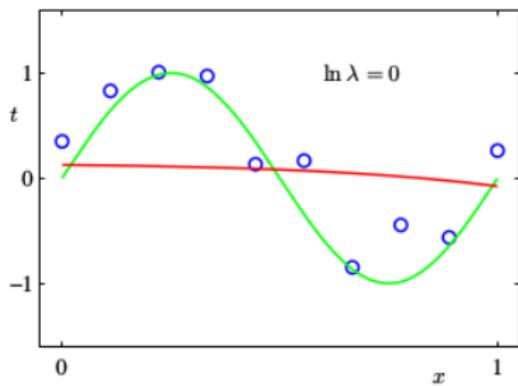
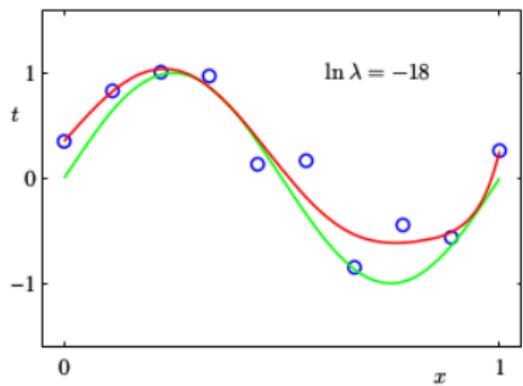
# Régression : régularisation

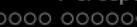
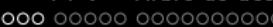
- Valeurs des paramètres  $w^*$  pour différents  $M$ , sans régularisation

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^*$	0.19	0.82	0.31	0.35
$w_1^*$		-1.27	7.99	232.37
$w_2^*$			-25.43	-5321.83
$w_3^*$			17.37	48568.31
$w_4^*$				-231639.30
$w_5^*$				-231639.30
$w_6^*$				640042.26
$w_7^*$				-1061800.52
$w_8^*$				1042400.18
$w_9^*$				-557682.99
$w_{10}^*$				125201.43

# Régression : régularisation

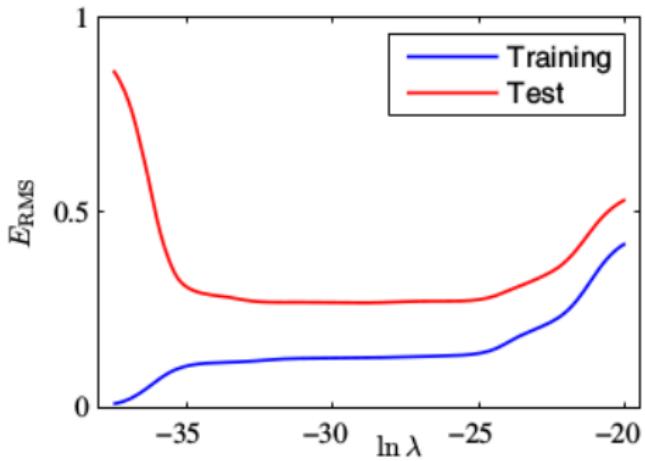
- Plus la régularisation est forte, moins le modèle sera flexible (donc il aura moins de capacité)

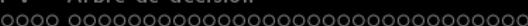




# Régression : régularisation

- Comme  $M$ , la force de la régularisation a une influence sur l'erreur d'entraînement et de test



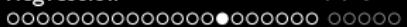


## Régession : sélection de modèle

- Soit l'algorithme d'apprentissage qui optimise

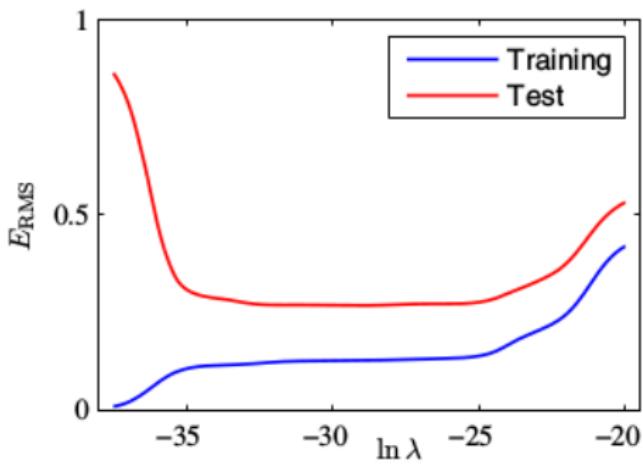
$$\tilde{E}(x) = \frac{1}{2} \sum_{n=1}^N \{f(w_n, w) - y_n\}^2 + \frac{\lambda}{2} \|w\|^2$$

- On appelle  $M$  et  $\lambda$  des **hyper-paramètres** : ils doivent être déterminés avant l'entraînement.
- La **sélection de modèle** est le choix de la valeur de ces **hyper-paramètres**.



## Régression : hyper-paramètres

- Le choix des hyper-paramètres va influencer la performance sur de nouvelles données (test)



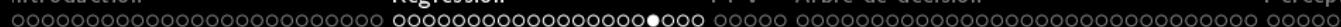


## Régression : ensemble de validation

- Solution I : on réserve des données d'entraînement pour comparer différentes valeurs
  - garde la majorité pour l'ensemble d'entraînement  $\mathcal{D}_{train}$  (ex. 80%)
  - le reste,  $\mathcal{D}_{valid}$  (ex. 20%), servira à comparer les hyper-paramètres
- On appelle  $\mathcal{D}_{valid}$  l'**ensemble de validation**

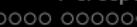
# Régession : algorithme de sélection de modèle

- Pour chaque valeur d'hyper-paramètres à comparer
  - obtenir un modèle entraîné à partir de  $\mathcal{D}_{train}$
  - évaluer la performance du modèle sur  $\mathcal{D}_{valid}$
- retourner le choix d'hyper-paramètres ayant donné le modèle avec la meilleure performance sur  $\mathcal{D}_{valid}$



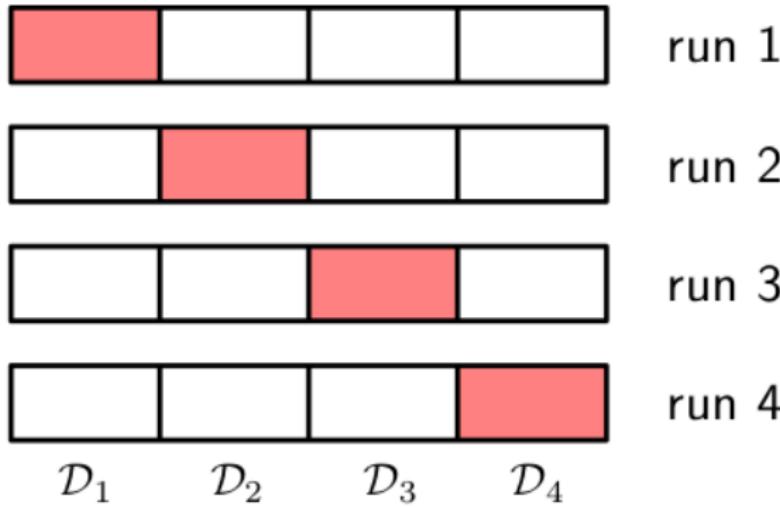
## Régression : S-fold cross-validation

- Lorsqu'on a peu de données, 20% est trop peu pour estimer la performance de généralisation
- On pourrait répéter la procédure de séparation *train/valid* plus d'une fois
- **S-fold cross-validation** : divise les données en  $S$  portions différentes. Chaque portion est utilisée une fois en tant que  $\mathcal{D}_{valid}$



## Régession : S-fold cross-validation

- Exemple :  $S = 4$



# Régression : S-fold cross-validation

## Sélection de modèle avec S-fold cross-validation

Pour  $s = 1 \dots S$

Pour chaque valeur d'hyper-paramètres à comparer

obtenir un modèle entraîné à partir de  $\mathcal{D}_{train} = \mathcal{D} \setminus \mathcal{D}_s$

évaluer la performance du modèle sur  $\mathcal{D}_{valid} = \mathcal{D}_s$

retourner la valeur des hyper-paramètres ayant la meilleure performance moyenne sur les ensembles  $\mathcal{D}_{valid}$

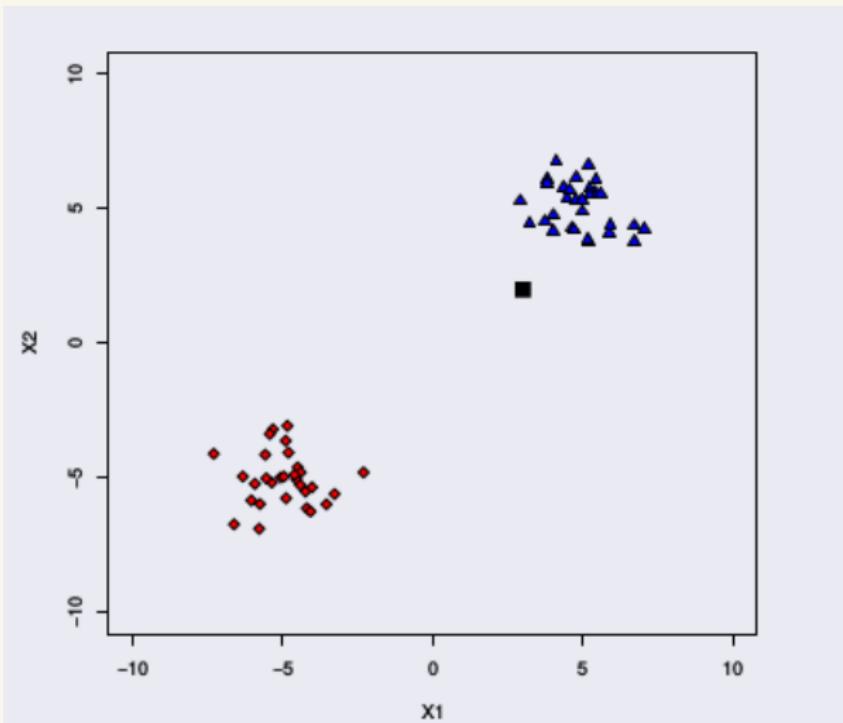
- Si  $S = N$ , on parle alors de méthode **leave-one-out**

# Régression : S-fold cross-validation

- Comment déterminer la liste des valeurs d'hyper-paramètres à comparer ?
- recherche sur une grille
  - détermine une liste de valeur pour chaque hyper-paramètre
  - construit la liste de toutes les combinaisons possibles

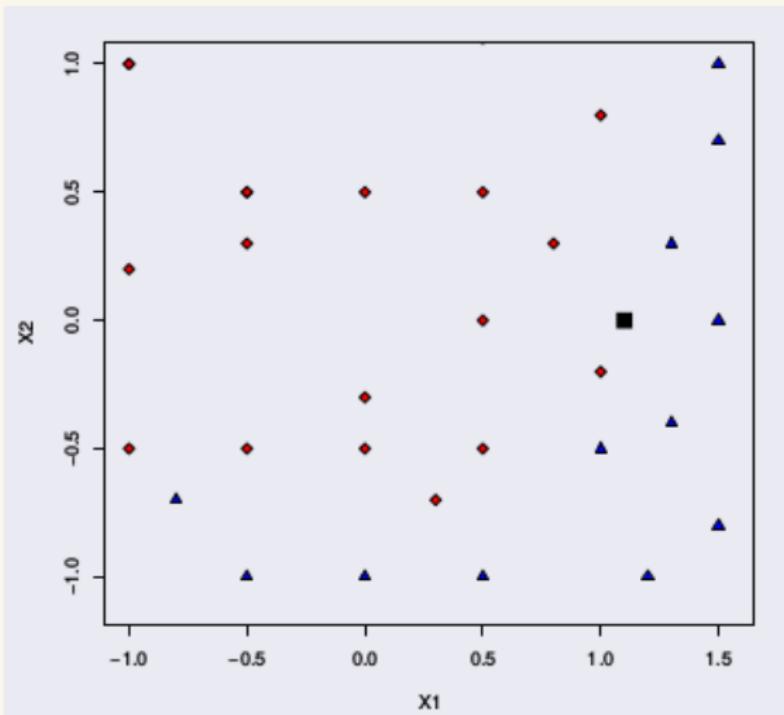
# Algorithme des $K$ plus proches voisins

Comment classer le carré noir ?



# Algorithme des $K$ plus proches voisins

Comment classer le carré noir ?



# Algorithme des $K$ plus proches voisins

## Comment classer le carré noir ?

- On cherche le plus proche voisin.
- On tag le nouveau point avec la classe correspondante.

## Cas des points aberrants

- Un point aberrant change fortement la frontière apprise.
- Comportement non souhaité.

## Une solution

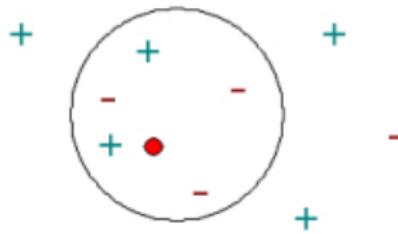
- Utiliser plus de 1 voisin.
- Compter les  $K$  plus proches voisins et prendre la classe dominante.



# Algorithme des $K$ plus proches voisins

Choix de  $K$  ?

- Pour le Plus Proche Voisin, le résultat est un plus
- Pour les 2 Plus Proches Voisins, le résultat est indéterminé
- Pour les 5 Plus Proches Voisins, le résultat est un moins



# Algorithme des $K$ plus proches voisins

- Données :

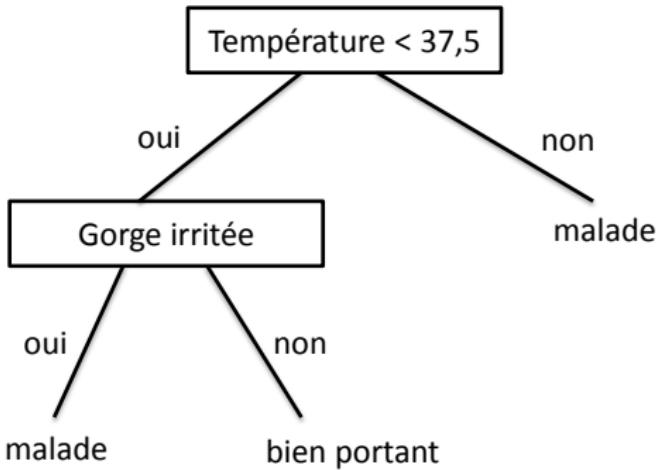
- L'ensemble des points d'apprentissage  $(x_i, y_i)$ .
- Le nouveau point à classer  $x$ .

- Algorithme :

- Toutes les distances  $D(x_i, x)$ .
- Sélection des  $K$  plus proches exemples  $x_1 \dots x_k$ .
- Choix de la classe la plus présente parmi les  $y_1 \dots y_k$  correspondants.

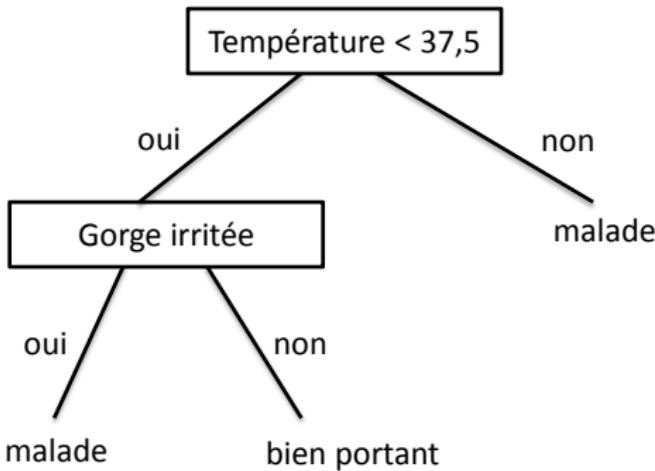
# Arbres de décision

- Exemple :

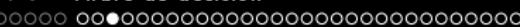


# Arbres de décision

- Exemple :



- Remarque : un arbre code en fait un ensemble de règles (conjonctions, disjonctions)
  - SI Température < 37.5 ET gorge irritée ALORS malade
  - SI Température < 37.5 ET gorge non irritée ALORS malade
  - SI Température <= 37.5 ALORS malade



# Arbres de décision

- Un **arbre de décision** est un arbre au sens informatique
- Les noeuds internes sont les **noeuds de décision**.
- Un noeud de décision est étiqueté par un **test** qui peut être appliqué à chaque description d'un individu d'une population.
- Chaque test examine la valeur d'un unique attribut.
- Dans les arbres de décision binaires on omet les labels des arcs.
- Les feuilles sont étiquetées par une classe.



# Arbres de décision

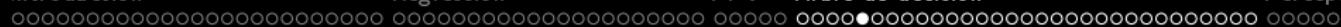
## Construction d'un arbre :

- Noeuds internes : **sélectionner** un attribut comme étiquette, les arcs sont étiquetés par les valeurs de l'attribut
- Feuilles : **couper** l'arbre avec une valeur de l'attribut cible

## On veut en général :

- Un taux d'erreur faible
- Une bonne généralisation
- Un arbre de petite taille compréhensible pour un non expert
- etc.

Nombreux algos : ID3, C4.5, CART, CHAID, algo. évo., etc.



## Exemple : Banque

client	M	A	R	E	I
1	moyen	moyen	village	oui	oui
2	élevé	moyen	bourg	non	non
3	faible	âgé	bourg	non	non
4	faible	moyen	bourg	oui	oui
5	moyen	jeune	ville	oui	oui
6	élevé	âgé	ville	oui	non
7	moyen	âgé	ville	oui	non
8	faible	moyen	village	non	non

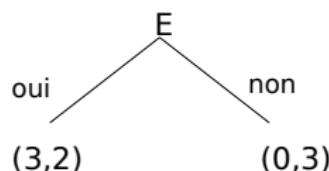
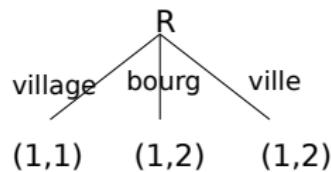
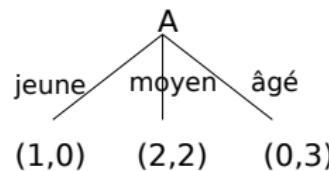
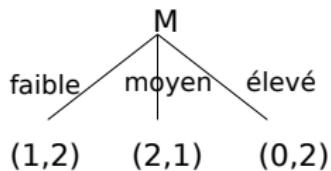
- M : La moyenne des montants sur le compte
- A : Tranche d'âge du client
- R : Localité de résidence du client
- E : Etudes supérieures
- I : Consultation du compte par Internet

On veut construire un arbre de décision pour savoir si un client consulte son compte par Internet.

## Exemple : Banque

- On construit l'arbre d'une façon **descendante**
- On choisit un test, on divise l'ensemble d'apprentissage  $S$  et on réapplique récursivement l'algorithme.
- On initialise avec l'arbre vide.
- Des 8 éléments de  $S$ , 3 sont de classe oui et 5 de classe non.
- $S$  est caractérisée par  $(3, 5)$ .
- Si le noeud n'est pas déjà **terminal** on choisit un test.
- Ici il y a 4 choix ( $M, A, R, E$ )

# Les 4 choix



- Quel test choisir ?



# Quel test choisir ?

- Un test est **intéressant** s'il permet une bonne discrimination.
- Une fonction qui mesure le degré de mélange des exemples doit :
  - prendre son maximum lorsque les exemples sont équirépartis (ici par exemple (4,4)).
  - prendre son minimum lorsque les exemples sont dans une même classe ((0,8) ou (8,0)).

# Mesure d'entropie

## Entropie $H$

Mesure de la quantité d'incertitude dans un ensemble (dispersion)

$$H(S) = - \sum_{x \in X} p(x) \log_2 p(x)$$

- $S$  : ensemble des données
- $X$  : ensemble des classes de  $S$
- $p(x)$  : proportion de la classe  $x \in X$  dans  $S$

Lorsque  $H(S) = 0$ ,  $S$  est parfaitement classé.



# Mesure d'entropie

Voir exemple de calcul au tableau

# Gain d'information

## Information gain (information mutuelle)

Mesure de la différence d'entropie entre avant et après le partitionnement selon un attribut

$$IG(S, T) = H(S) - \sum_{i=1}^{i=t} p(S_i)H(S_i)$$

- $T = \{S_1, \dots, S_t\}$  sous-ensembles du partitionnement de  $S$ ,  
 $S = \bigcup_{i=1}^{i=t} S_i$
- $p(S_i) = |S_i|/|S|$
- $H(S), H(S_i)$  : entropies de  $S$  et de  $S_i$

# Gain d'information

Voir exemple de calcul au tableau

# Pseudo code

**ID3(exemples, cible, attributs) :**

**si** tous les exemples sont positifs (resp. négatifs) **alors**

**retourner** une feuille avec l'étiquette positif (resp. négatif)

**si** attributs est vide **alors**

**retourner** une feuille avec l'étiquette la plus fréquente

**sinon**

$A \leftarrow$  attribut de plus grand gain d'information

**construire** un noeud avec l'étiquette  $A$

**pour** chaque valeurs  $v_i$  de  $A$

**ajouter** la branche  $v_i$  au noeud

**si** exemples( $A = v_i$ ) est vide **alors**

**ajouter** à la branche la feuille

          avec l'étiquette la plus fréquente

**sinon**

**ajouter** à la branche le sous-arbre

          ID3(exemples( $A = v_i$ ), cible, attributs  $-A$ )

# C4.5 (Ross Quinlan, 1993)

## Amélioration de ID3

- Le critère du gain avantage les attributs discrets ayant beaucoup de valeurs distinctes
- Utilisation du ratio de gain d'information au lieu de  $IG$  :

$$ratioIG(S, T) = IG(S, T) / SplitInfo(S, T)$$

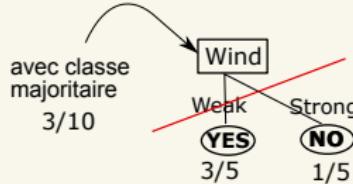
$$SplitInfo(S, T) = \sum_{i=1}^{i=k} -\frac{|S_i|}{|S|} * \log_2 \frac{|S_i|}{|S|}$$

- $S$  : ensemble d'exemples.
- $S_i$  : nombre d'exemples valant  $A_i$  pour l'attribut  $T$ .
- $k$  : nombre de valeurs de l'attribut

# C4.5 (Ross Quinlan, 1993)

## Amélioration de ID3

- Possibilité de valeur "null" :  
Exemple ignoré lors dans le calcul du noeud
- Prise en compte des attributs à valeur continue :  
Discrétisation par  $P(A < a_i)$   
pour toutes les valeurs possibles de  $A$ , calcul de  $IG$
- Elagage (pruning) pour réduire la taille de l'arbre :  
Technique bottom-up : branches finales élaguées  
lorsque taux d'erreur plus grand qu'en remplaçant par une feuille

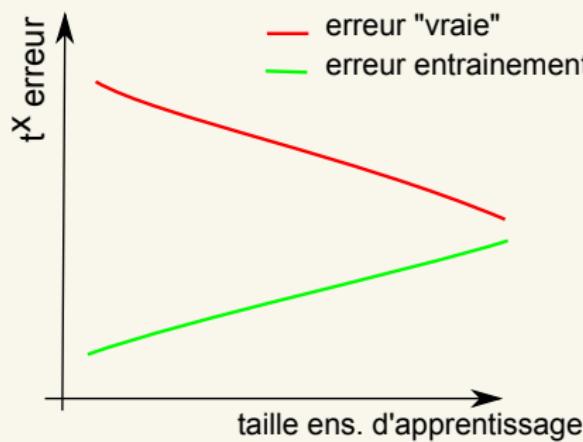




# Les erreurs

## Relation entre erreurs

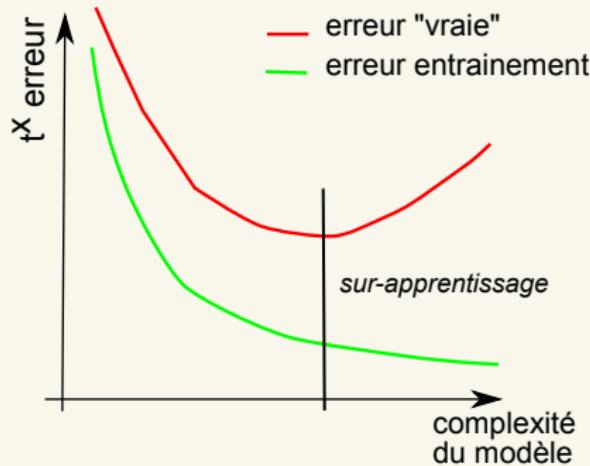
- Erreur d'apprentissage : taux d'erreur sur l'ensemble des exemples d'apprentissage
- Erreur "vraie" : erreur sur l'ensemble de tous les exemples possibles



# Sur-apprentissage

## Excès d'apprentissage

Sur-spécialisation du modèle sur l'ensemble d'entraînement  
⇒ Perte de capacité de généralisation  
≈ Apprentissage "par cœur"



Mesure de complexité d'un arbre de décision : nombre de feuilles

# Evaluation d'un modèle d'apprentissage

## Technique

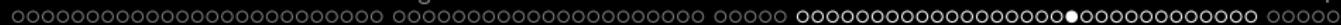
Partitionner l'ensemble des exemples en :

- un ensemble d'apprentissage ( $\approx 70\%$ )
- un ensemble *indépendant* de test ( $\approx 30\%$ )

Le taux d'erreur est estimé (sans biais) sur l'ensemble de test.

## Inconvénient

- Requiert un nombre important d'exemples
- Dilemme :
  - Plus on met d'exemples dans le test, plus l'estimation est précise
  - Plus on met d'exemples dans l'apprentissage, meilleur est le modèle (a priori)



# Elagage des arbres de décisions

## Généralités

- La poursuite de l'algorithme de construction jusqu'à son terme naturel fabrique un arbre avec des feuilles correspondant à des classes parfaitement homogènes.
- Il y a là un apprentissage par coeur et donc un risque de sous-estimation de la probabilité d'erreur par le taux d'erreur apparent (qui vaut 0 ).
- Le nombre de noeuds de l'arbre de décision est un critère de complexité simple et efficace
- Chercher la valeur "optimale" du nombre de noeuds revient donc à trouver une technique pour contrôler la taille de l'arbre

# Elagage des arbres de décisions

## Le pré-élagage

- On cesse de diviser un noeud lorsque la pureté des points est suffisantes (inférieure à un certain seuil).
- Utilisation de critères locaux (à une feuille)
- On peut manquer un très bon développement.

## Le post-élagage

- Elaguer l'arbre lorsqu'il est parfaitement développé.
- Utiliser un ensemble indépendant de l'ensemble d'apprentissage (ensemble de validation).
- Mesurer l'erreur commise sur cet ensemble.

# Elagage des arbres de décisions

- Soit  $T_0$  un arbre de décision à élaguer
- On construit l'ensemble  $T$  de tous les arbres de décision obtenus à partir de  $T_0$  en remplaçant certains noeuds internes par des feuilles :  $T = \{T_0, T_1, T_2, \dots, T_n\}$  tel que  $T_n$  est l'arbre constitué d'une seule feuille.
- L'idée est de comparer le coût de l'arbre élagué et de l'arbre non élagué.
- On s'arrête si le coût du premier est supérieur.



# Elagage des arbres de décisions

Choisir le noeud  $v$  qui minimise :

- Pour chaque noeud interne  $v$  d'un arbre  $T_k$  on calcule le coût suivant :

$$w(T_k, v) = \frac{\Delta R_{v,k}^S}{n(k) * (nt(v, k) - 1)}$$

- $\Delta R_{v,k}^S$  : le nombre d'erreurs supplémentaires que commet l'arbre  $T_k$  sur  $S$  lorsqu'on élague à la position  $v$
- $n(k)$  : le nombre de feuilles de  $T_k$
- $nt(v, k)$  : le nombre de feuilles du sous-arbre de  $T_k$  situées sous le noeud  $v$ . Il représente le nombre de feuilles supprimées

## Objectif

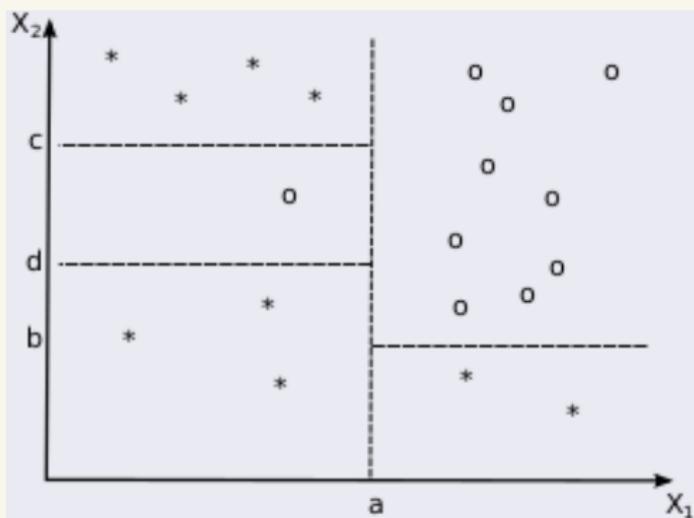
Trouver le meilleur compromis entre taille et taux d'erreur apparent.

# Elagage des arbres de décisions

## Processus itératif

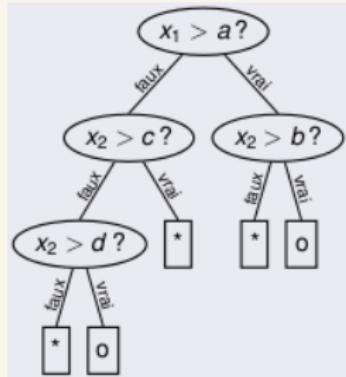
- $T_{k+1}$  obtenu à partir de  $T_k$  auquel on coupe la branche qui permet un  $w$  minimal.
- Soit  $\{T_0, \dots, T_k, \dots, T_n\}$  la suite obtenue où  $T_n$  est réduit à une feuille.
- Sélection de l'arbre  $T_k$  dont le nombre d'erreurs calculées sur ensemble de validation  $S_{val}$  est minimal

# Un exemple d'élagage



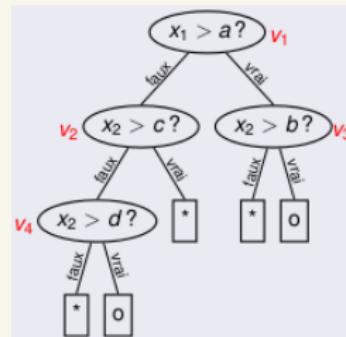
# Un exemple d'élagage

Arbre  $T_0$



# Un exemple d'élagage

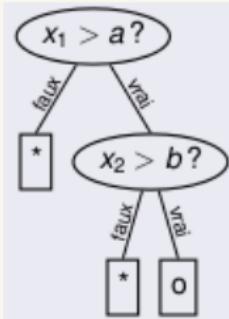
## Calcul des valeurs sur $T_0$



- $w(T_0, v_1) = \frac{\Delta R_{v_1,0}^S}{n(k)*(nt(v_1,0)-1)} = \frac{9}{5*(5-1)} = \frac{9}{20}$
- $w(T_0, v_2) = \frac{1}{5*(3-1)} = \frac{1}{10}$
- $w(T_0, v_3) = \frac{2}{5*(2-1)} = \frac{2}{5}$
- $w(T_0, v_4) = \frac{1}{5*(2-1)} = \frac{1}{5}$

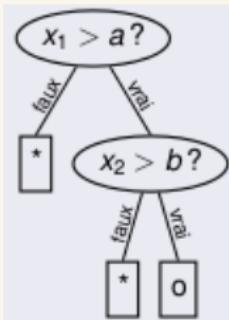
# Un exemple d'élagage

Arbre  $T_1$



# Un exemple d'élagage

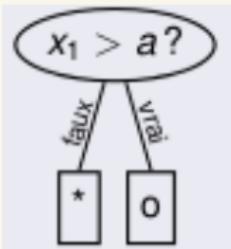
## Calcul des valeurs sur $T_1$



- $w(T_1, v_1) = \frac{9-1}{3*(3-1)} = \frac{4}{3}$
- $w(T_1, v_2) = \frac{2-1}{3*(2-1)} = \frac{1}{3}$

# Un exemple d'élagage

Arbre  $T_2$



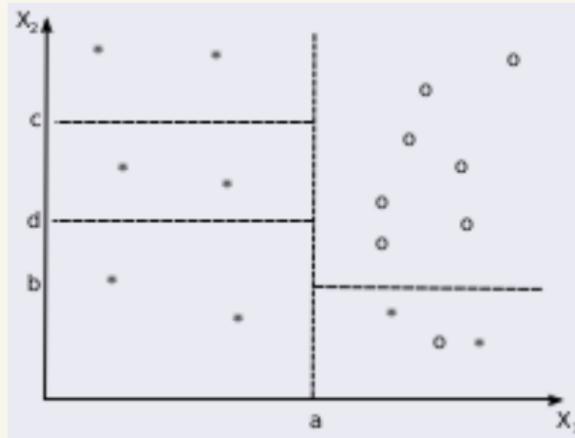
# Un exemple d'élagage

## Choix de l'arbre

- Le choix de l'arbre se fera sur un ensemble de validation parmi  $T_0$ ,  $T_1$  et  $T_2$ .
- L'arbre qui aura la meilleure estimation du taux d'erreur de classification sera choisi.

# Un exemple d'élagage

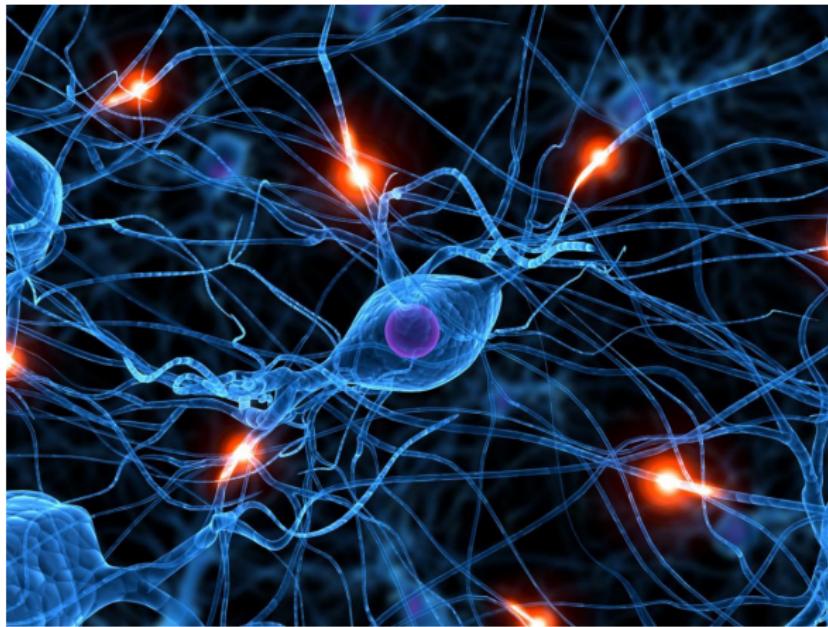
Par exemple, sur cet ensemble de validation



- Erreurs  $T_0$  :  $\frac{3}{16}$
- Erreurs  $T_1$  :  $\frac{1}{16} \Rightarrow$  choisir  $T_1$
- Erreurs  $T_2$  :  $\frac{2}{16}$

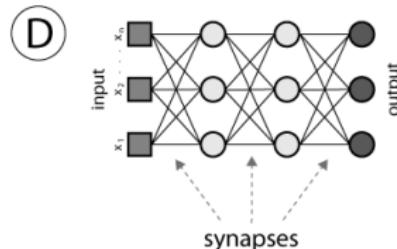
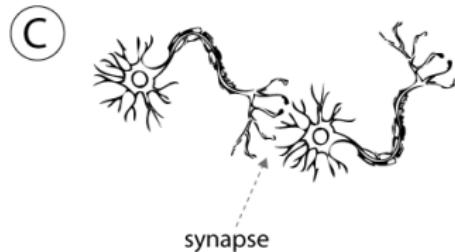
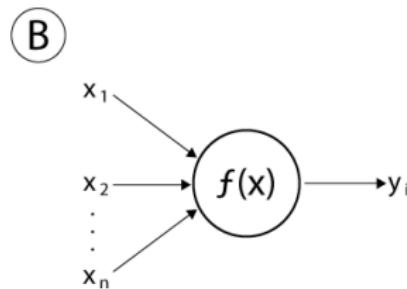
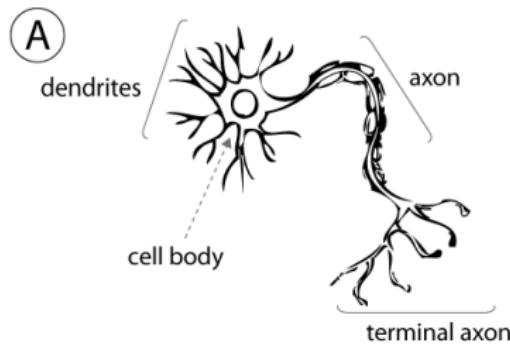
# Perceptron : introduction

Le **cerveau** : réseau très complexe ayant un grand nombre de cellules de base interconnectées. Il y a environ 100 milliards de neurones et  $10^{15}$  connexions.

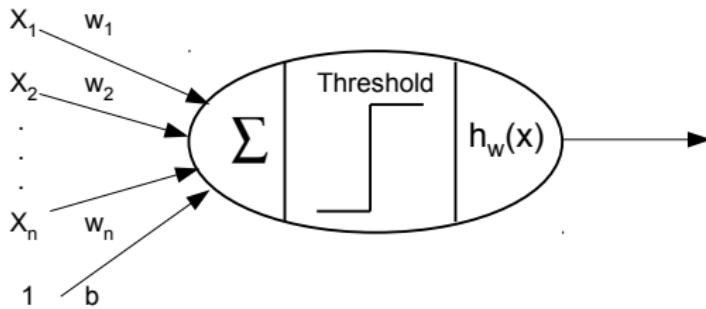


# Perceptron : introduction

Les réseaux de neurones : modèle très simple du cerveau où les unités de calcul élémentaires sont interconnectées.



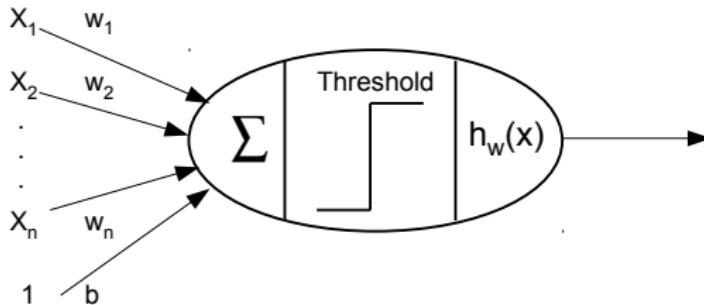
# Perceptron (Rosenblatt, 1957)



## Définition

- Le perceptron est un classifieur linéaire.
- Il a une seule sortie à laquelle toutes les entrées sont connectées.

# Perceptron (Rosenblatt, 1957)

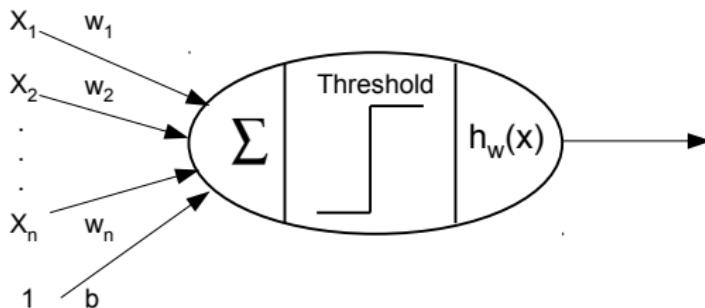


## Principe

- Le **vecteur de poids**  $W$  correspond aux **paramètres** du modèle
- On ajoute également un biais  $b$ , qui équivaut à ajouter une entrée  $x_{n+1} = 1$
- A partir de l'échantillon d'apprentissage, il faut trouver le vecteur de poids  $W$  et le biais  $b$ , tel que :

$$W \cdot X + \text{biais} \begin{cases} \geq 0 \\ \leq 0 \end{cases} \Rightarrow X \in \begin{cases} c_1 \\ c_2 \end{cases}$$

# Perceptron (Rosenblatt, 1957)



## Principe

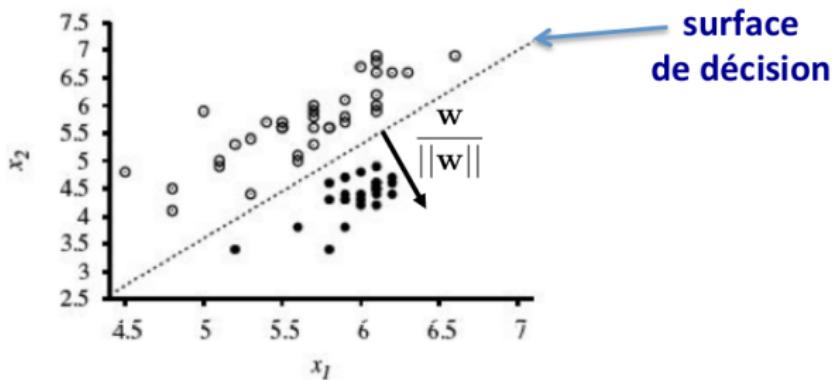
- Idée : modéliser la décision à l'aide d'une fonction linéaire, suivie d'un seuil :

$$h_W(x) = \text{Threshold}(W.X)$$

où  $\text{Threshold}(z) = 1$  si  $z \geq 0$ , sinon  $\text{Threshold}(z) = 0$

# Surface de séparation

- Le perceptron cherche un **séparateur linéaire** entre les deux classes



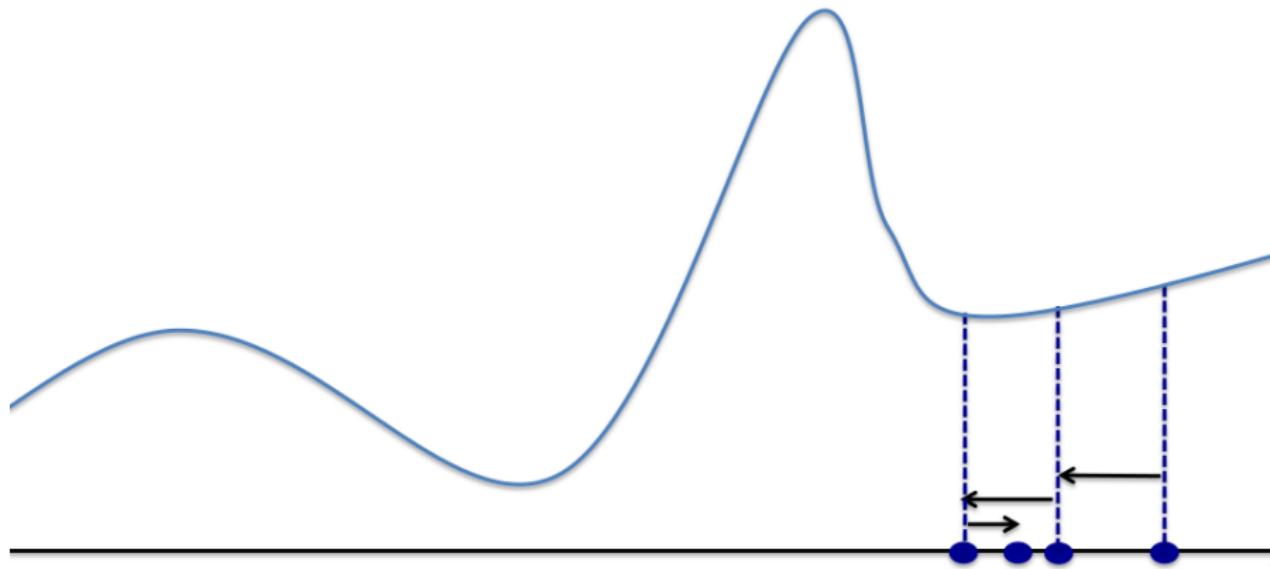
- La **surface de décision** d'un classifieur est la surface qui sépare les deux régions classifiées dans les deux classes différentes



# Apprentissage vue comme la minimisation d'une perte

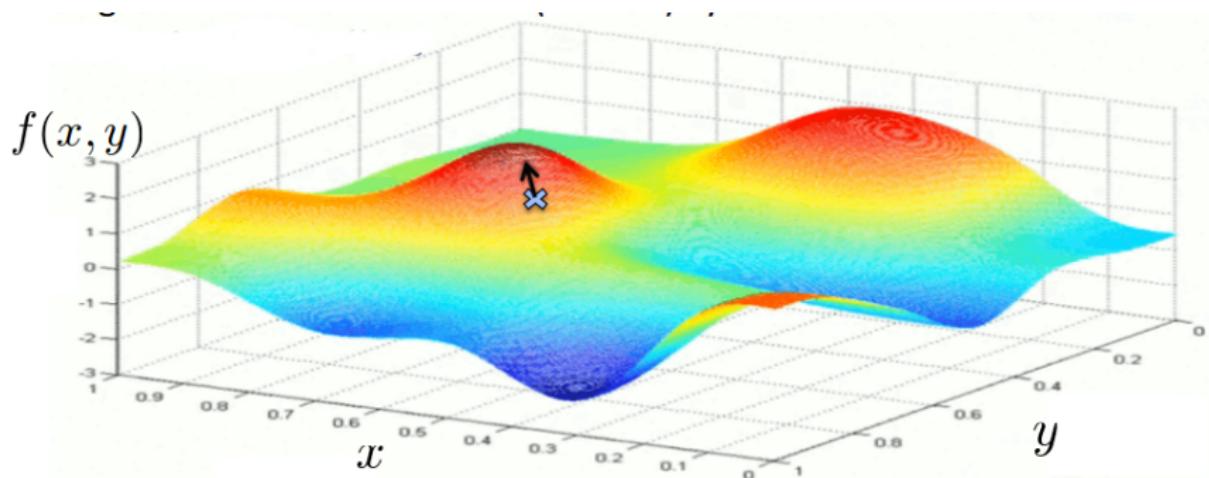
- Le problème de l'apprentissage peut être formulé comme un problème d'optimisation
  - pour chaque exemple d'entraînement, on souhaite minimiser une certaine distance  $Loss(y_t, h_w(x_t))$  entre la cible  $y_t$  et la prédiction  $h_w(x_t)$
  - on appelle cette distance une **perte** :  
$$Loss(y_t, h_w(x_t)) = (y_t - h_w(x_t))^2$$

# Algorithme de descente de gradient



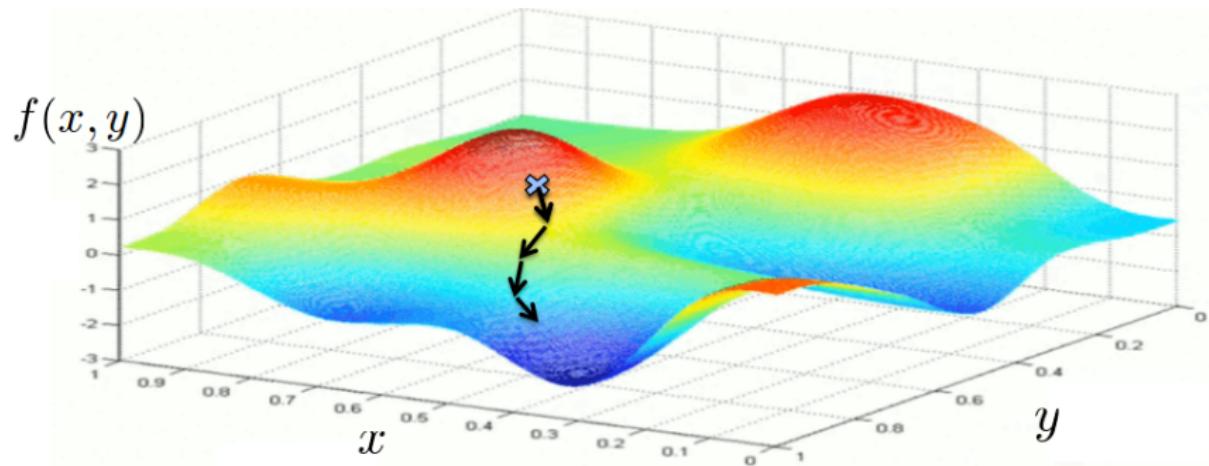
# Algorithme de descente de gradient

- Le gradient donne la direction (vecteur) ayant le taux d'accroissement de la fonction le plus élevé



# Algorithme de descente de gradient

- Le gradient donne la direction (vecteur) ayant le taux d'accroissement de la fonction le plus élevé



# Algorithme de descente de gradient

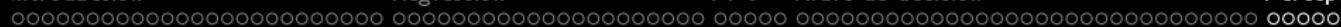
- En apprentissage automatique, on souhaite optimiser :

$$\frac{1}{|D|} \sum_{(x_t, y_t) \in D} Loss(y_t, h_w(x_t))$$

- Le gradient par rapport à la perte moyenne contient les dérivées partielles :

$$\frac{1}{|D|} \sum_{(x_t, y_t) \in D} \frac{\partial}{\partial w_i} Loss(y_t, h_w(x_t)) \quad \forall i$$

- Devrait calculer la moyenne des dérivées sur tous les exemples d'entraînement avant de faire une mise à jour des paramètres !



# Descente de gradient stochastique

- Descente de gradient stochastique : mettre à jour les paramètres à partir du gradient d'un seul exemple, choisi aléatoirement :

Initialiser  $w$  aléatoirement

Pour  $T$  itérations

Pour chaque exemple d'entraînement  $(x_t, y_t)$

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(y_t, h_w(x_t)) \quad \forall i$$

- Cette procédure est plus efficace lorsque l'ensemble d'entraînement est grand

## Retour sur le perceptron

- On utilise le gradient pour déterminer une direction de mise à jour des paramètres :

$$\frac{\partial}{\partial w_i} \text{Loss}(y_t, h_w(x_t)) = \frac{\partial}{\partial w_i} (y_t - h_w(x_t))^2 =$$

$$2(y_t - h_w(x_t)) * \frac{\partial}{\partial w_i} (y_t - h_w(x_t)) \simeq -2(y_t - h_w(x_t))x_{t,i}$$

- Pour mettre à jour les paramètres, on va dans la direction opposée de ce gradient :

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} \text{Loss}(y_t, h_w(x_t)) \quad \forall i$$

- on obtient la règle d'apprentissage du perceptron

$$w_i \leftarrow w_i + \alpha(y_t - h_w(x_t))x_{t,i} \quad \forall i$$

# Algorithme du perceptron

- 1 Pour chaque paire de l'ensemble d'apprentissage  $(x_t, y_t) \in D$ 
  - Calculer la réponse produite par le perceptron
$$h_w(x_t) = \text{Threshold}(w \cdot x_t)$$
  - Si  $y_t = h_w(x_t)$  alors l'exemple est bien classé : ne rien faire
  - Sinon ( $y_t \neq h_w(x_t)$ ) mettre à jour les poids et le biais selon la règle suivante :

$$w_i \leftarrow w_i + \alpha(y_t - h_w(x_t)) * x_{t,i} \quad \forall i$$

- 2 Retourner à 1 jusqu'à l'atteinte d'un critère d'arrêt : nombre maximal d'itérations atteint ou tous les exemples sont bien classés

La mise à jour des poids est appelée la **règle d'apprentissage du perceptron**. Le multiplicateur  $\alpha$  est appelé le **taux d'apprentissage**

# Exemple de déroulement

$x_t$	$y_t$
[2, 0]	1
[0, 3]	0
[3, 0]	0
[1, 1]	1

- Ensemble d'apprentissage  $D$
- On initialise les poids et le biais aléatoirement :  $w \leftarrow [0, 0]$  ,  $b = 0.5$
- On choisit le pas égal à 0.1 :  $\alpha = 0.1$
- On lit le prochain exemple.
- On calcule la sortie.
- On met à jour les poids si nécessaire  
 $w_i \leftarrow w_i + \alpha(y_t - h_w(x_t)) * x_{t,i}$
- On ré-itère tant qu'on n'a pas convergé et qu'on a du temps

# Exemple de déroulement

- Ensemble d'apprentissage  $D$

$x_t$	$y_t$
[2, 0]	1
[0, 3]	0
[3, 0]	0
[1, 1]	1

Iteration	Exemple	$b$	$w_1$	$w_2$	$\sum x_{t,i} * w_i$	$h_w(x_t)$	$y_t$
1	[2, 0]	0.5	0	0	0.5	1	1

- Puisque  $h(x_1) = y_1$ , on ne fait pas de mise à jour  $w$  et  $b$

# Exemple de déroulement

- Ensemble d'apprentissage  $D$

$x_t$	$y_t$
[2, 0]	1
[0, 3]	0
[3, 0]	0
[1, 1]	1

Iteration	Exemple	$b$	$w_1$	$w_2$	$\sum x_{t,i} * w_i$	$h_w(x_t)$	$y_t$
1	[2, 0]	0.5	0	0	0.5	1	1
2	[0, 3]	0.5	0	0	0.5	1	0

- Puisque  $h(x_2) \neq y_2$ , on met à jour  $w$  et  $b$  :
- $w \leftarrow w + \alpha * (y_2 - h(x_2)) * x_2 = [0, 0] + 0.1 * (0 - 1)[0, 3] = [0, -0.3]$
- $b \leftarrow b + \alpha * (y_2 - h(x_2)) = 0.5 + 0.1(0 - 1) = 0.4$

# Exemple de déroulement

- Ensemble d'apprentissage  $D$

$x_t$	$y_t$
[2, 0]	1
[0, 3]	0
[3, 0]	0
[1, 1]	1

Iteration	Exemple	$b$	$w_1$	$w_2$	$\sum x_{t,i} * w_i$	$h_w(x_t)$	$y_t$
1	[2, 0]	0.5	0	0	0.5	1	1
2	[0, 3]	0.5	0	0	0.5	1	0
3	[3, 0]	0.4	0	-0.3	0.4	1	0

## Calcul de la séparatrice

$$b + w_1 x_1 + w_2 x_2 = 0$$

$$x_2 = -\frac{w_1}{w_2} * x_1 - \frac{b}{w_2}$$

$$x_2 = -\frac{0}{-0.3} * x_1 - \frac{0.4}{-0.3}$$

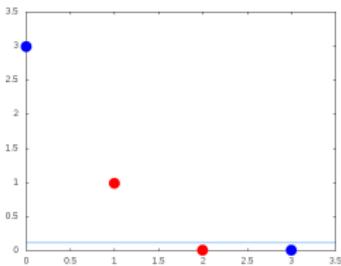
$$x_2 = 0.12$$

# Exemple de déroulement

- Ensemble d'apprentissage  $D$

$x_t$	$y_t$
[2, 0]	1
[0, 3]	0
[3, 0]	0
[1, 1]	1

Iteration	Exemple	$b$	$w_1$	$w_2$	$\sum x_{t,i} * w_i$	$h_w(x_t)$	$y_t$
1	[2, 0]	0.5	0	0	0.5	1	1
2	[0, 3]	0.5	0	0	0.5	1	0
3	[3, 0]	0.4	0	-0.3	0.4	1	0



# Exemple de déroulement

- Ensemble d'apprentissage  $D$

$x_t$	$y_t$
[2, 0]	1
[0, 3]	0
[3, 0]	0
[1, 1]	1

Iteration	Exemple	$b$	$w_1$	$w_2$	$\sum x_{t,i} * w_i$	$h_w(x_t)$	$y_t$
1	[2, 0]	0.5	0	0	0.5	1	1
2	[0, 3]	0.5	0	0	0.5	1	0
3	[3, 0]	0.4	0	-0.3	0.4	1	0

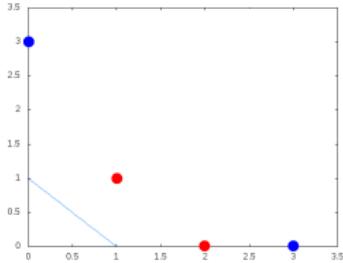
- Puisque  $h(x_3) \neq y_3$ , on met à jour  $w$  et  $b$  :
- $w \leftarrow w + \alpha * (y_3 - h(x_3)) * x_2 = [0, -0.3] + 0.1 * (0 - 1)[3, 0] = [-0.3, -0.3]$
- $b \leftarrow b + \alpha * (y_3 - h(x_3)) = 0.4 + 0.1(0 - 1) = 0.3$

# Exemple de déroulement

- Ensemble d'apprentissage  $D$

$x_t$	$y_t$
[2, 0]	1
[0, 3]	0
[3, 0]	0
[1, 1]	1

Iteration	Exemple	$b$	$w_1$	$w_2$	$\sum x_{t,i} * w_i$	$h_w(x_t)$	$y_t$
1	[2, 0]	0.5	0	0	0.5	1	1
2	[0, 3]	0.5	0	0	0.5	1	0
3	[3, 0]	0.4	0	-0.3	0.4	1	0
4	[1, 1]	0.3	-0.3	-0.3	-0.3	0	1



# Exemple de déroulement

- Ensemble d'apprentissage  $D$

$x_t$	$y_t$
[2, 0]	1
[0, 3]	0
[3, 0]	0
[1, 1]	1

Iteration	Exemple	$b$	$w_1$	$w_2$	$\sum x_{t,i} * w_i$	$h_w(x_t)$	$y_t$
1	[2, 0]	0.5	0	0	0.5	1	1
2	[0, 3]	0.5	0	0	0.5	1	0
3	[3, 0]	0.4	0	-0.3	0.4	1	0
4	[1, 1]	0.3	-0.3	-0.3	-0.3	0	1

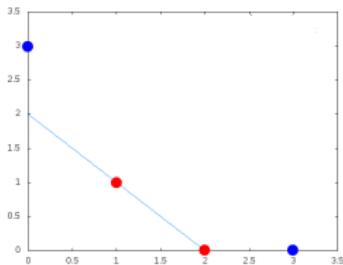
- Puisque  $h(x_4) \neq y_4$ , on met à jour  $w$  et  $b$  :
- $w \leftarrow w + \alpha * (y_4 - h(x_4)) * x_2 = [-0.3, -0.3] + 0.1 * (1 - 0)[1, 1] = [-0.2, -0.2]$
- $b \leftarrow b + \alpha * (y_4 - h(x_4)) = 0.3 + 0.1(1 - 0) = 0.4$

# Exemple de déroulement

- Ensemble d'apprentissage  $D$

$x_t$	$y_t$
[2, 0]	1
[0, 3]	0
[3, 0]	0
[1, 1]	1

Iteration	Exemple	$b$	$w_1$	$w_2$	$\sum x_{t,i} * w_i$	$h_w(x_t)$	$y_t$
1	[2, 0]	0.5	0	0	0.5	1	1
2	[0, 3]	0.5	0	0	0.5	1	0
3	[3, 0]	0.4	0	-0.3	0.4	1	0
4	[1, 1]	0.3	-0.3	-0.3	-0.3	0	1
5	[2, 0]	0.4	-0.2	-0.2	0	1	1



# Exemple de déroulement

$x_t$	$y_t$
[2, 0]	1
[0, 3]	0
[3, 0]	0
[1, 1]	1

- Ensemble d'apprentissage  $D$

Iteration	Exemple	$b$	$w_1$	$w_2$	$\sum x_{t,i} * w_i$	$h_w(x_t)$	$y_t$
1	[2, 0]	0.5	0	0	0.5	1	1
2	[0, 3]	0.5	0	0	0.5	1	0
3	[3, 0]	0.4	0	-0.3	0.4	1	0
4	[1, 1]	0.3	-0.3	-0.3	-0.3	0	1
5	[2, 0]	0.4	-0.2	-0.2	0	1	1

- Puisque  $h(x_1) = y_1$ , on ne met pas à jour  $w$  et  $b$
- Et ainsi de suite, jusqu'à l'atteinte d'un critère d'arrêt...

# Exemple de déroulement

- Ensemble d'apprentissage  $D$

$x_t$	$y_t$
[2, 0]	1
[0, 3]	0
[3, 0]	0
[1, 1]	1

Iteration	Exemple	$b$	$w_1$	$w_2$	$\sum x_{t,i} * w_i$	$h_w(x_t)$	$y_t$
1	[2, 0]	0.5	0	0	0.5	1	1
2	[0, 3]	0.5	0	0	0.5	1	0
3	[3, 0]	0.4	0	-0.3	0.4	1	0
4	[1, 1]	0.3	-0.3	-0.3	-0.3	0	1
5	[2, 0]	0.4	-0.2	-0.2	0	1	1
6	[0, 3]	0.4	-0.2	-0.2	-0.2	0	0
7	[3, 0]	0.4	-0.2	-0.2	-0.2	0	0
8	[1, 1]	0.4	-0.2	-0.2	0	1	1

- Arrêt de l'algorithme car tous les exemples sont bien classés.

## Exemple 2 de déroulement

- Apprendre la fonction **OU logique**

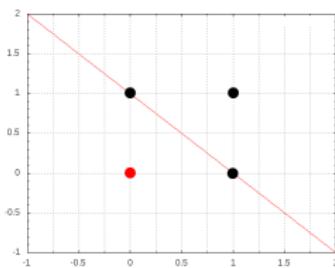
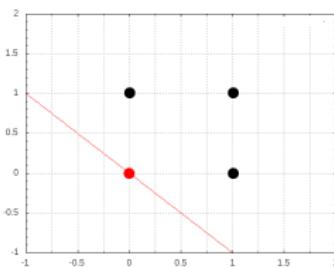
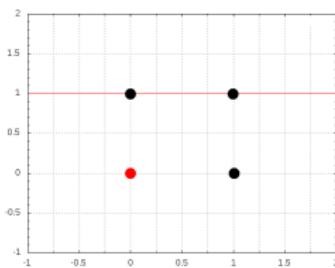
$x_t$	$y_t$
[0, 0]	0
[0, 1]	1
[1, 0]	1
[1, 1]	1

- Ensemble d'apprentissage  $D$

- On initialise les poids et le biais aléatoirement :  $w \leftarrow [0, 1]$  ,  $b = -1$
- On choisit le pas égal à 1 :  $\alpha = 1$

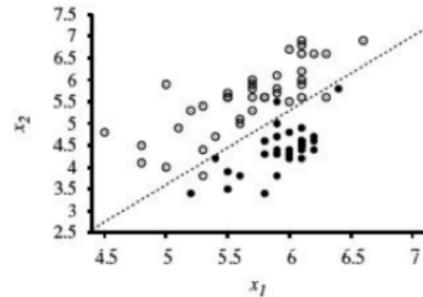
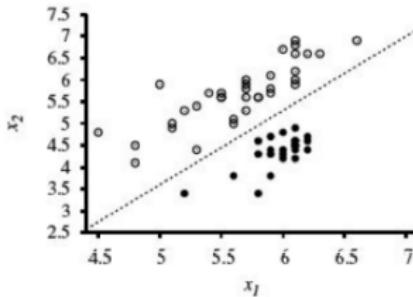
# Exemple 2 de déroulement

Iteration	Exemple	$b$	$w_1$	$w_2$	$\sum x_{t,i} * w_i$	$h_w(x_t)$	$y_t$
1	[0, 0]	-1	0	1	-1	0	0
2	[0, 1]	-1	0	1	0	1	1
3	[1, 0]	-1	0	1	-1	0	1
4	[1, 1]	0	1	1	2	1	1
5	[0, 0]	0	1	1	0	1	0
6	[0, 1]	-1	1	1	0	1	1
7	[1, 0]	-1	1	1	0	1	1
8	[1, 1]	-1	1	1	1	1	1
9	[0, 0]	-1	1	1	-1	0	0



# Convergence et séparabilité

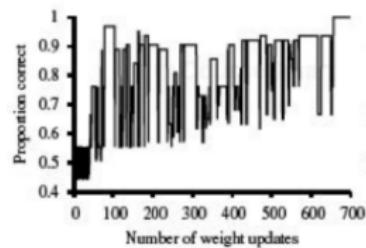
- Si les exemples sont **linéairement séparables**, le perceptron garanti de converger à **une solution avec une erreur nulle** sur l'ensemble d'entraînement, pour tout  $\alpha$



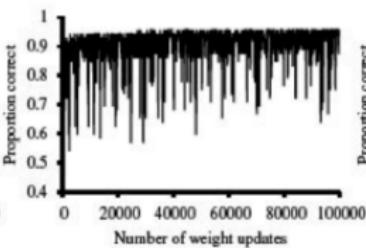
- Sinon, pour garantir la convergence à **une solution ayant la plus petite erreur possible en entraînement**, on doit décroître le taux d'apprentissage, par ex. selon  $\alpha_k = \frac{\alpha}{1+\beta k}$

# Courbe d'apprentissage

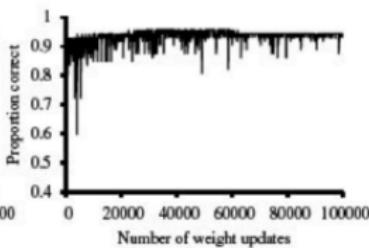
- La courbe d'apprentissage est la courbe du taux d'erreur (ou de succès) en fonction du nombre de mises à jour des paramètres
- Utile pour visualiser la progression de l'apprentissage



linéairement séparable



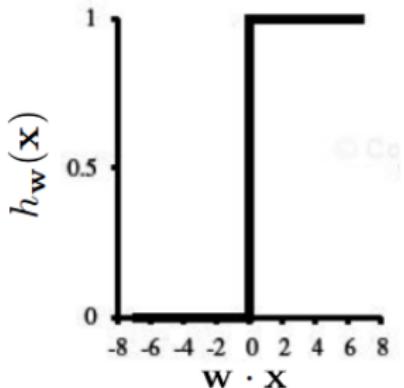
pas linéairement séparable



taux d'apprentissage décroissant

# Apprentissage vu comme la minimisation d'une perte

- La procédure de descente de gradient stochastique est applicable à n'importe quelle perte dérivable partout
- Dans le cas du perceptron, la dérivée de  $h_w(x)$  n'est pas continue lorsque  $w \cdot x = 0$



- L'utilisation de la fonction *Threshold* fait que la courbe d'entraînement peut être instable.

# Régression logistique (sigmoid)

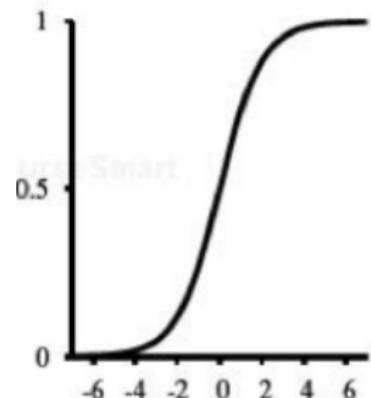
## Idée

Plutôt que de prédire une classe, prédire une probabilité d'appartenir à cette classe :

$$h_w(x) = p(y = \text{classe1}|x) = \text{Logistic}(w \cdot x) = \frac{1}{1+e^{-w \cdot x}}$$

Choisir la classe la plus probable selon le modèle

- Si  $h_w(x_t) \geq 0.5$  choisir la classe 1
- Sinon choisir la classe 2



# Régression logistique (sigmoid)

- Pour obtenir une règle d'apprentissage, on définit d'abord une perte :

$$\frac{\partial}{\partial w_i} \text{Loss}(y_t, h_w(x_t)) = \frac{\partial}{\partial w_i} (y_t - h_w(x_t))^2 = \frac{\partial}{\partial w_i} (y_t - \text{Logistic}(w \cdot x_t))^2$$

- On dérive la règle d'apprentissage :

$$\begin{aligned} \frac{\partial}{\partial w_i} \text{Loss}(y_t, h_w(x_t)) &= 2(y_t - \text{Logistic}(w \cdot x_t)) \frac{\partial}{\partial w_i} (y_t - \text{Logistic}(w \cdot x_t)) \\ &= -2(y_t - \text{Logistic}(w \cdot x_t)) * \text{Logistic}'(w \cdot x_t) * \frac{\partial}{\partial w_i} (w \cdot x_t) \\ &= -2(y_t - \text{Logistic}(w \cdot x_t)) * \text{Logistic}'(w \cdot x_t) * x_{t,i} \end{aligned}$$

- La dérivée de la fonction logistique vérifie :

$$\text{Logistic}'(z) = \text{Logistic}(z) * (1 - \text{Logistic}(z))$$

- On a donc

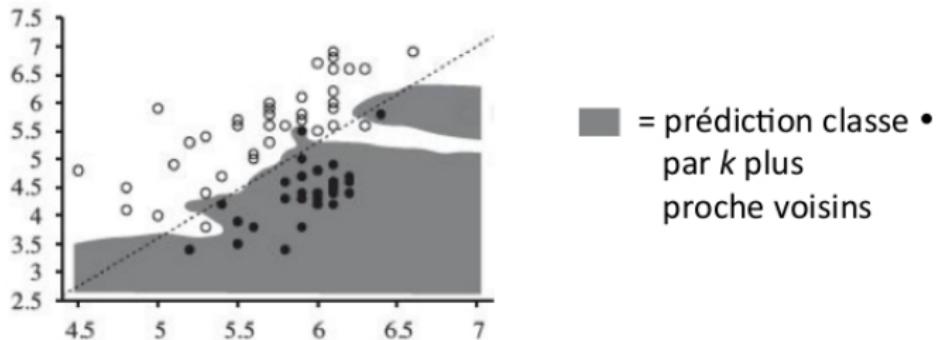
$$\begin{aligned} \frac{\partial}{\partial w_i} \text{Loss}(y_t, h_w(x_t)) &= -2(y_t - \text{Logistic}(w \cdot x_t)) * \text{Logistic}(w \cdot x_t) \\ &\quad * (1 - \text{Logistic}(w \cdot x_t)) * x_{t,i} \end{aligned}$$

- On obtient la règle d'apprentissage suivante

$$w_i \leftarrow w_i + \alpha (y_t - \text{Logistic}(w \cdot x_t)) * \text{Logistic}(w \cdot x_t) * (1 - \text{Logistic}(w \cdot x_t)) * x_{t,i} \quad \forall i$$

# Limitation des classifieurs linéaires

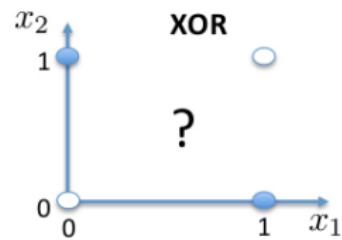
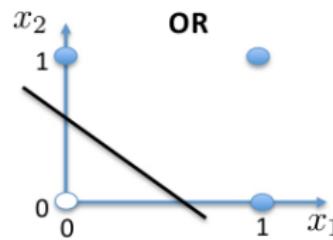
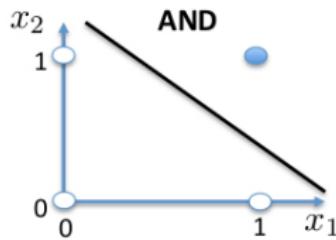
- Si les données d'entraînement sont linéairement séparables, le perceptron va trouver cette séparation



- L'algorithme des  $k$  plus proches voisins est non-linéaire, mais coûteux en mémoires et temps de calcul

# Limitation des classifieurs linéaires

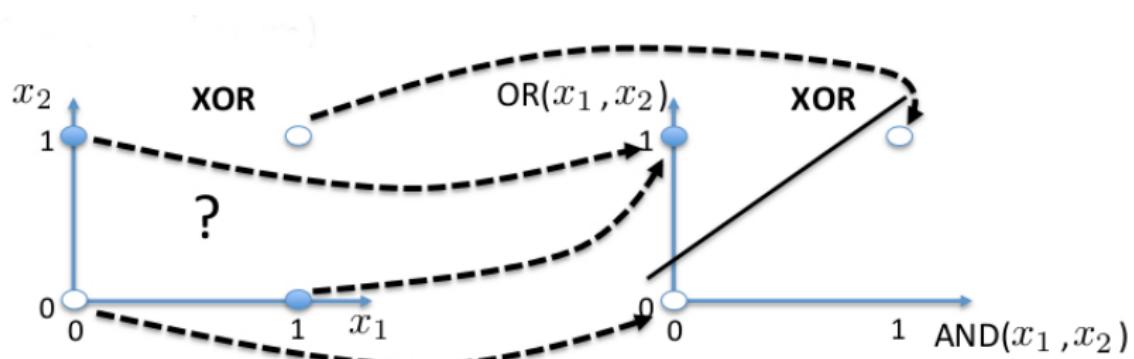
- La majorité des problèmes de classification ne sont pas linéaires



- Un classifieur linéaire ne peut même pas apprendre la fonction XOR!

# Limitation des classificateurs linéaires

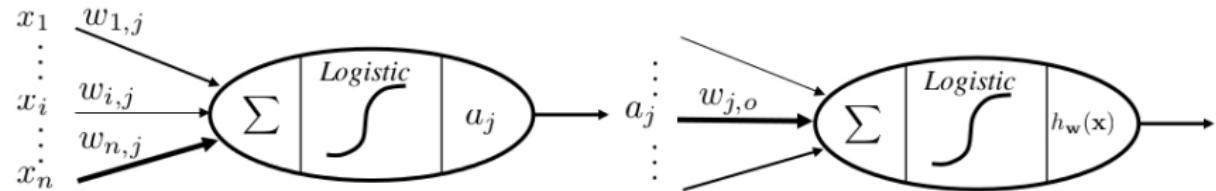
- On pourrait transformer l'entrée de façon à rendre le problème linéairement séparable.
- Dans le cas de XOR, on pourrait remplacer
  - $x_1$  par  $AND(x_1, x_2)$
  - $x_2$  par  $OR(x_1, x_2)$



# Réseau de neurones artificiel

## Principe

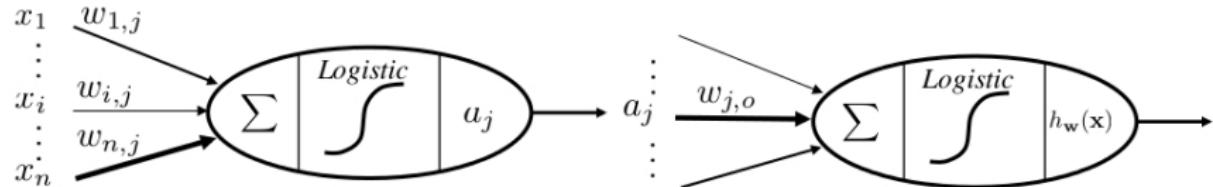
Apprentissage des poids afin d'entraîner le réseau à effectuer des tâches de classification ou de régression



# Réseau de neurones artificiel

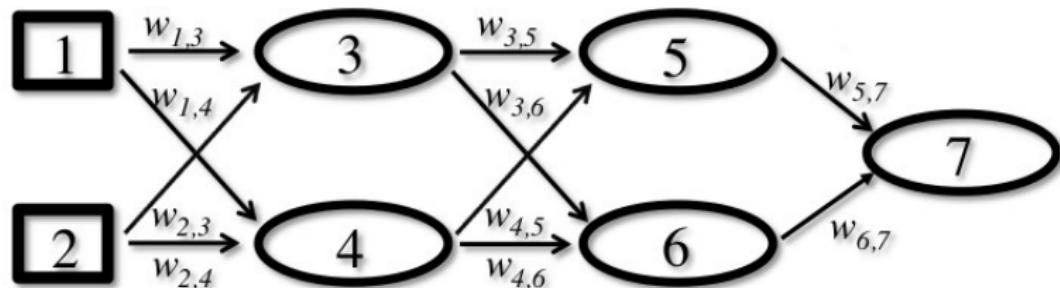
## Principe

- Un **neurone d'entrée** est une unité correspondant à une donnée d'entrée.
- Un **neurone de sortie** est une unité qui fournit une hypothèse d'apprentissage.
- Un **neurone caché** est une unité interne du réseau, qui n'est ni une entrée ni une sortie.



# Réseau de neurones artificiel

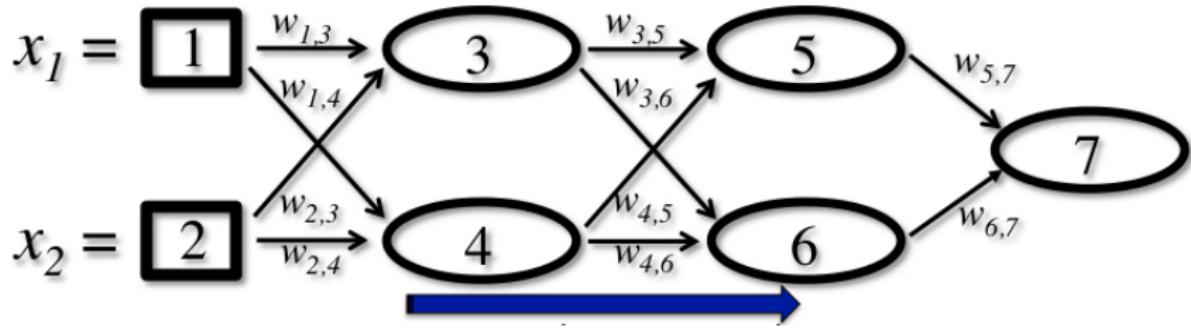
Réseau de neurones avec 4 couches dont deux couches cachées



- On note  $a_j$  l'activité du  $j^{eme}$  "neurone", incluant les neurones d'entrée et de sortie
- Pour les neurones d'entrée on aura  $a_j = x_j$
- Pour les autres neurones  $a_j = \text{Logistic}(\sum_i w_{i,j} a_i)$

# Visualisation de la rétropropagation

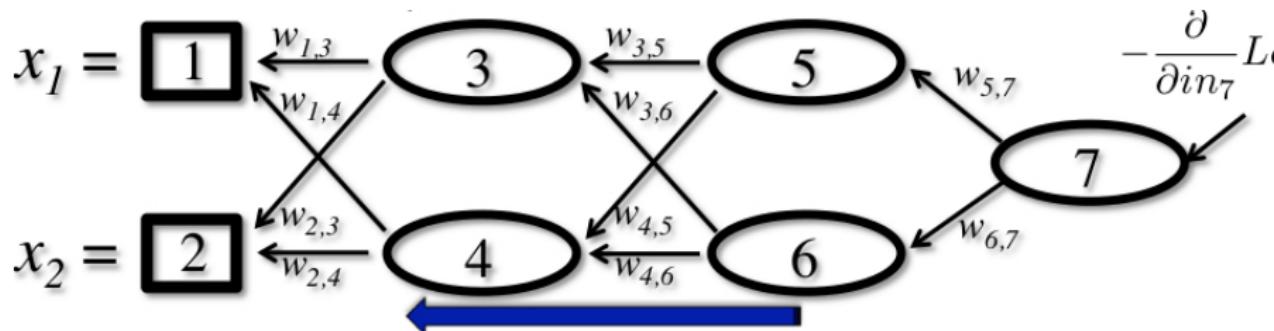
L'algorithme d'apprentissage commence par une **propagation avant**



$$a_j = \text{Logistic}\left(\sum_i w_{i,j} a_i\right)$$

# Visualisation de la rétropropagation

Ensuite, le gradient (le Delta) sur la sortie est calculé

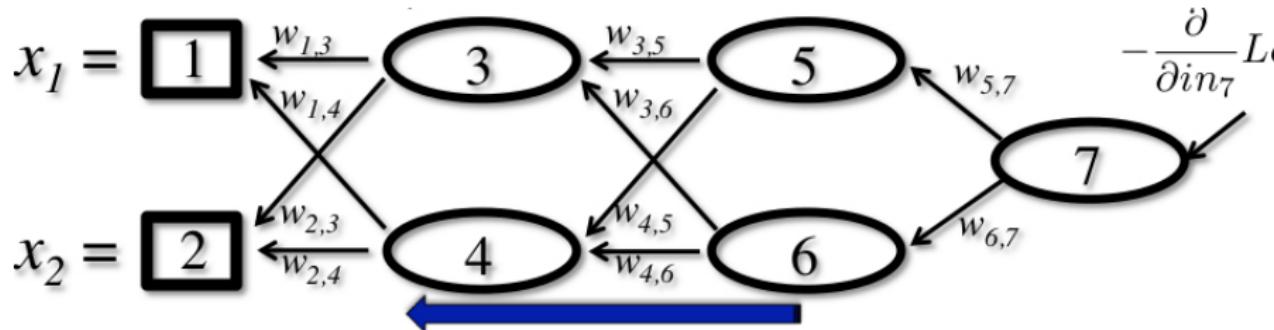


$$\Delta[i] = \text{Logistic}(in_i) * (1 - \text{Logistic}(in_i)) * (y_i - a_i)$$

$$in_i = \sum_{k \in \text{precedents}(i)} w_{k,i} a_k$$

# Visualisation de la rétropropagation

Le gradient (le Delta) est rétropropagé

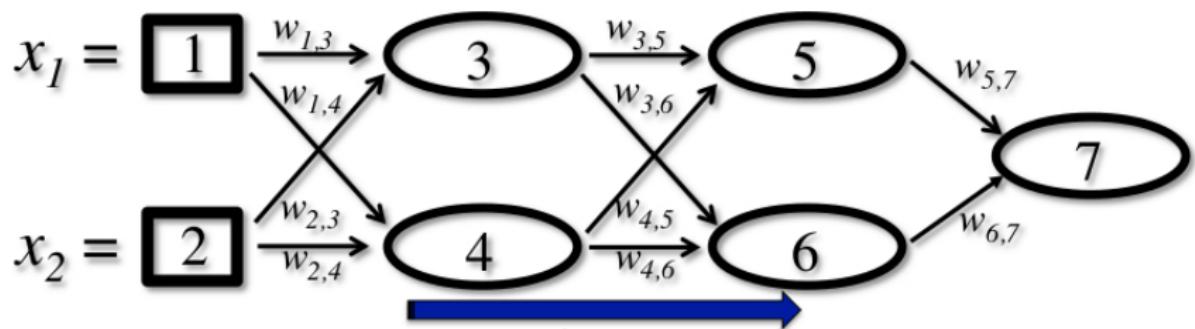


$$\Delta[i] \leftarrow Logistic(in_i) * (1 - Logistic(in_i)) * \sum_{j \in suivants(i)} w_{i,j} \Delta[j]$$

$$in_i = \sum_{k \in precedents(i)} w_{k,i} a_k$$

# Visualisation de la rétropropagation

Mettre à jour les poids  $w_{i,j}$  de tous le réseau en appliquant la règle d'apprentissage généralisée



$$w_{i,j} \leftarrow w_{i,j} + \alpha * \Delta[j] * a_i$$

# Algorithme de rétropropagation

## Algorithme de rétropropagation (exemples, réseau)

```

initialiser les poids du réseau  $w_{i,j}$  aléatoirement
while le critère d'arrêt n'est pas atteint do
    /* propagation des entrées pour le calcul des sorties */
    for chaque neurone  $j$  de la couche d'entrée do
         $a_j \leftarrow x_j$ 
    end for
    for  $l = 2$  to  $L$  do
        for chaque neurone  $j$  de la couche  $l$  do
             $a_j \leftarrow Logistic(\sum_i w_{i,j} a_i)$ 
        end for
    end for
    /* rétropropagation des  $\Delta$  de la couche de sortie jusqu'à la couche d'entrée */
    for chaque neurone  $i$  de la couche de sortie do
         $\Delta[i] \leftarrow Logistic(in_i) * (1 - Logistic(in_i)) * (y_i - a_i)$ 
    end for
    for  $l = L - 1$  to 1 do
        for chaque neurone  $i$  de la couche  $l$  do
             $in_i \leftarrow \sum_{k \in precedents(i)} w_{k,i} a_k$ 
             $\Delta[i] \leftarrow Logistic(in_i) * (1 - Logistic(in_i)) * \sum_{j \in suivants(i)} w_{i,j} \Delta[j]$ 
        end for
    end for

```

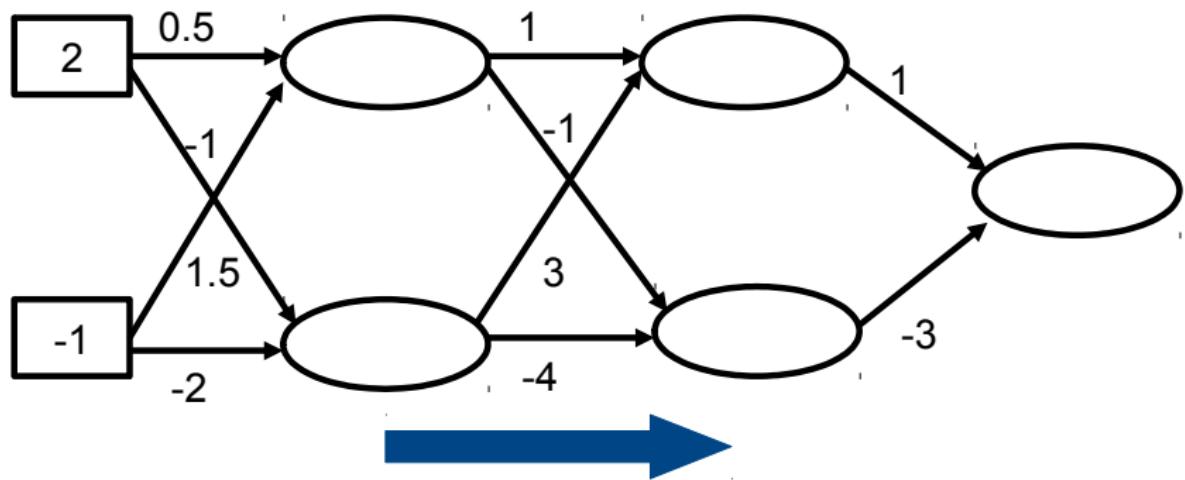
# Algorithme de rétropropagation

## Algorithme de rétropropagation (suite)

```
/* mise à jour des poids du réseau en utilisant les  $\Delta$  */  
for chaque poids  $w_{i,j}$  do  
     $w_{i,j} \leftarrow w_{i,j} + \alpha * \Delta[j] * a_i$   
endfor  
end while
```

## Exemple de déroulement

Exemple :  $x=[2, -1]$ ,  $y=0$

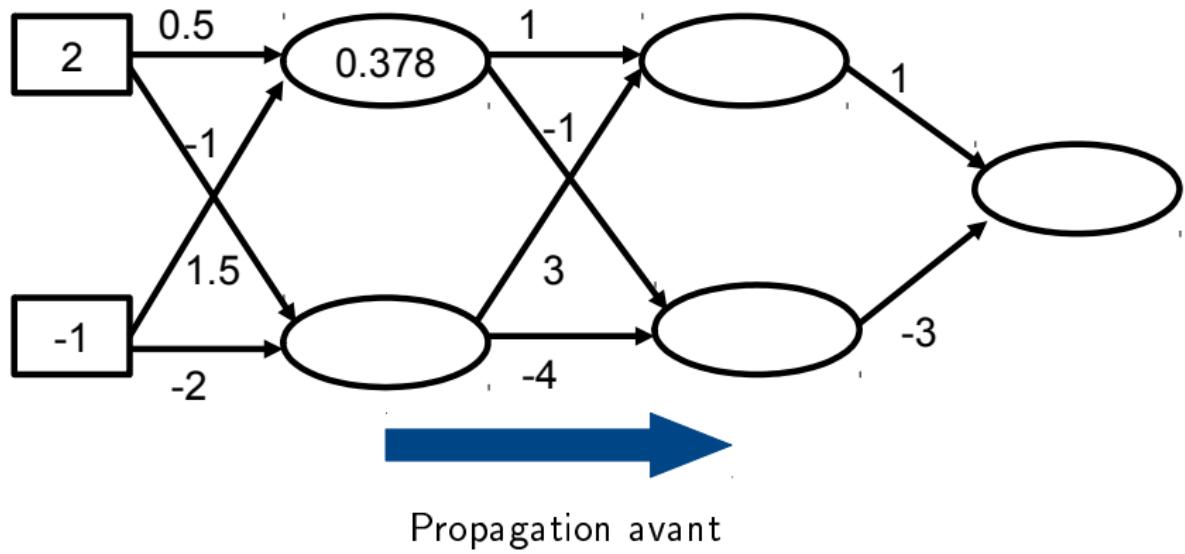


Propagation avant

$$a_j = \text{Logistic}\left(\sum_i w_{i,j} a_i\right)$$

## Exemple de déroulement

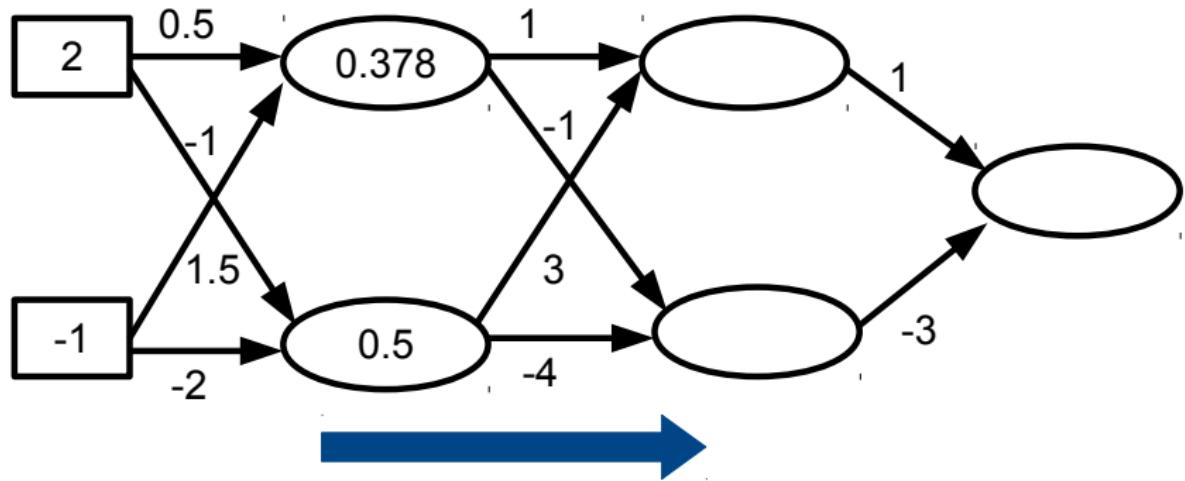
Exemple :  $x=[2, -1]$ ,  $y=0$



$$\text{Logistic}(0.5 * 2 + 1.5 * -1) = \text{Logistic}(-0.5) = 0.378$$

# Exemple de déroulement

Exemple :  $x=[2, -1]$ ,  $y=0$

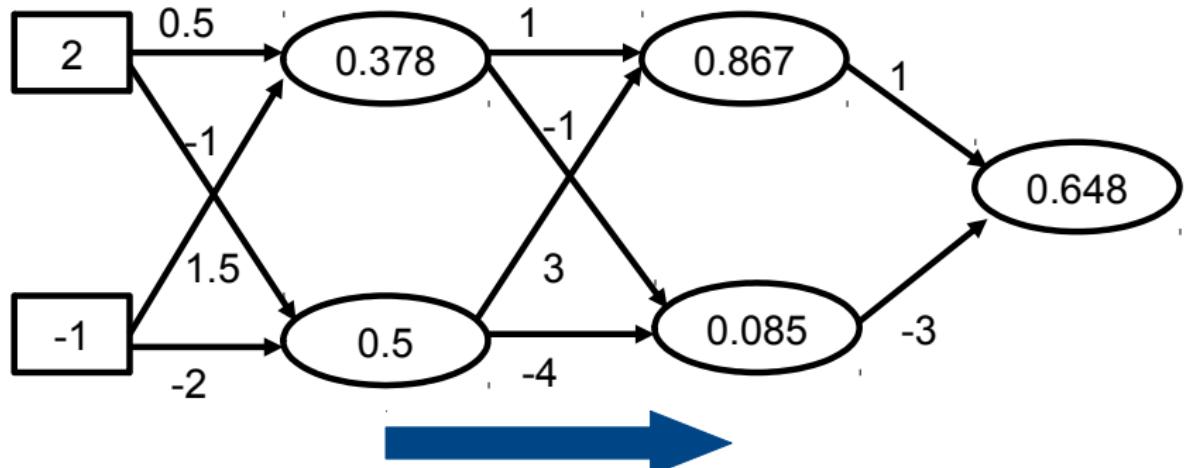


Propagation avant

$$\text{Logistic}(-1 * 2 + -2 * -1) = \text{Logistic}(0) = 0.5$$

## Exemple de déroulement

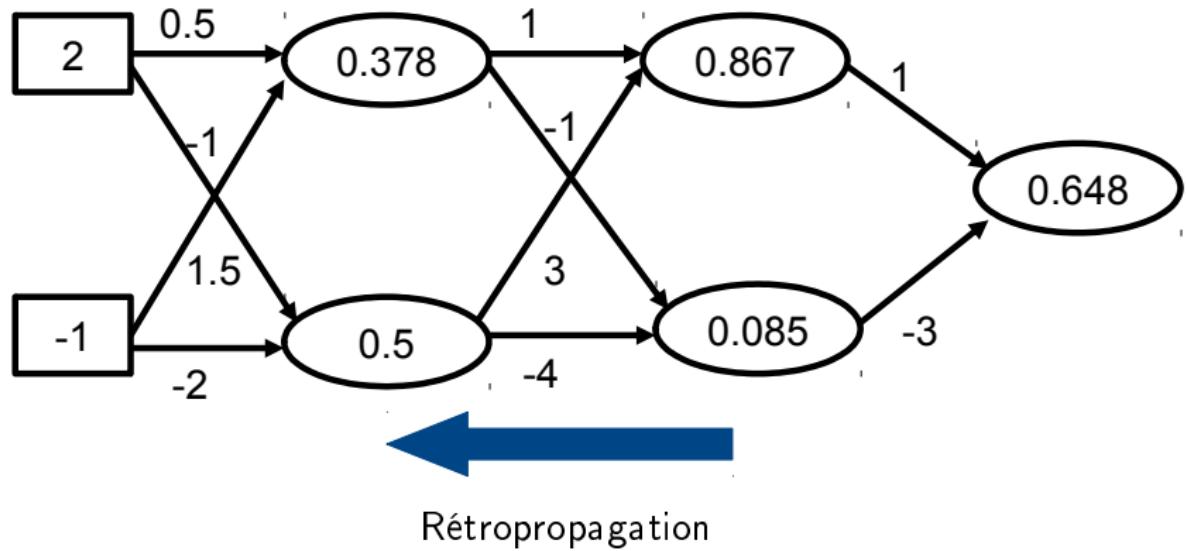
Exemple :  $x=[2, -1]$ ,  $y=0$



Propagation avant

## Exemple de déroulement

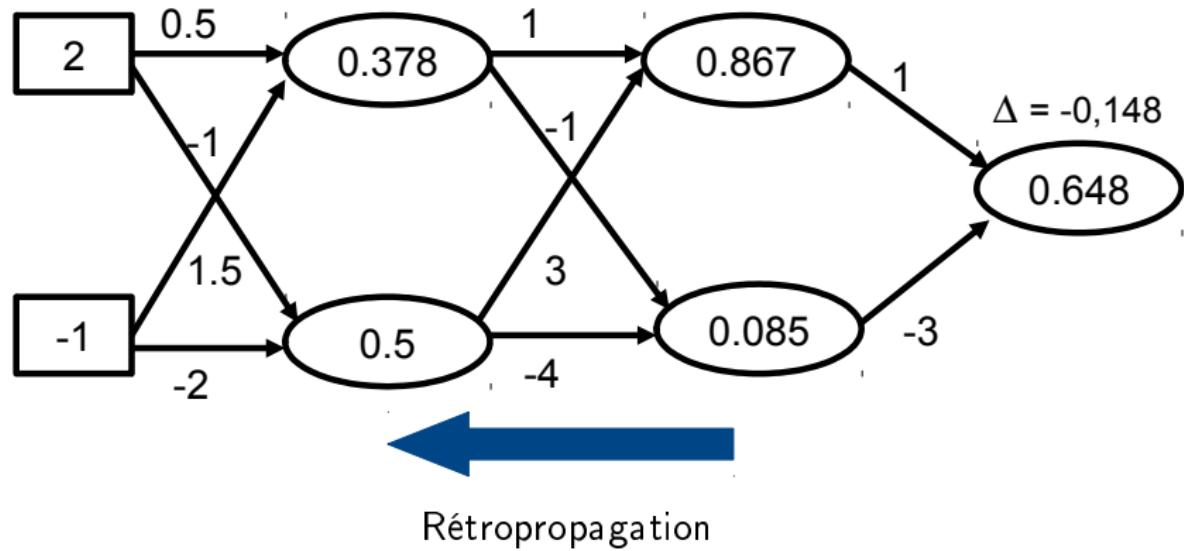
Exemple :  $x=[2, -1]$ ,  $y=0$



$$\Delta[i] = \text{Logistic}(in_i) * (1 - \text{Logistic}(in_i)) * (y_i - a_i)$$

## Exemple de déroulement

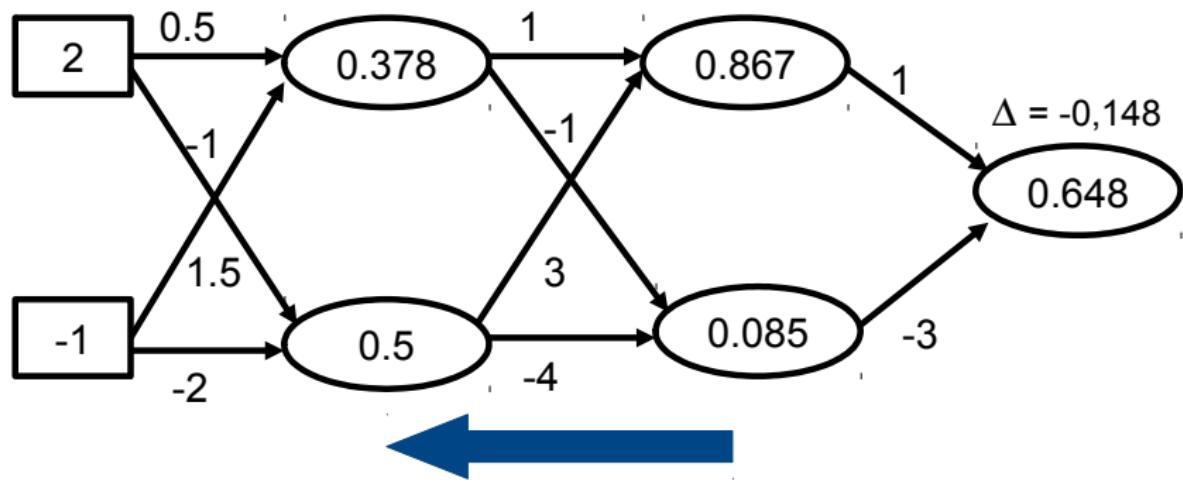
Exemple :  $x=[2, -1]$ ,  $y=0$



$$\Delta = 0.648 * (1 - 0.648) * (0 - 0.648) = -0.148$$

# Exemple de déroulement

Exemple :  $x=[2, -1]$ ,  $y=0$

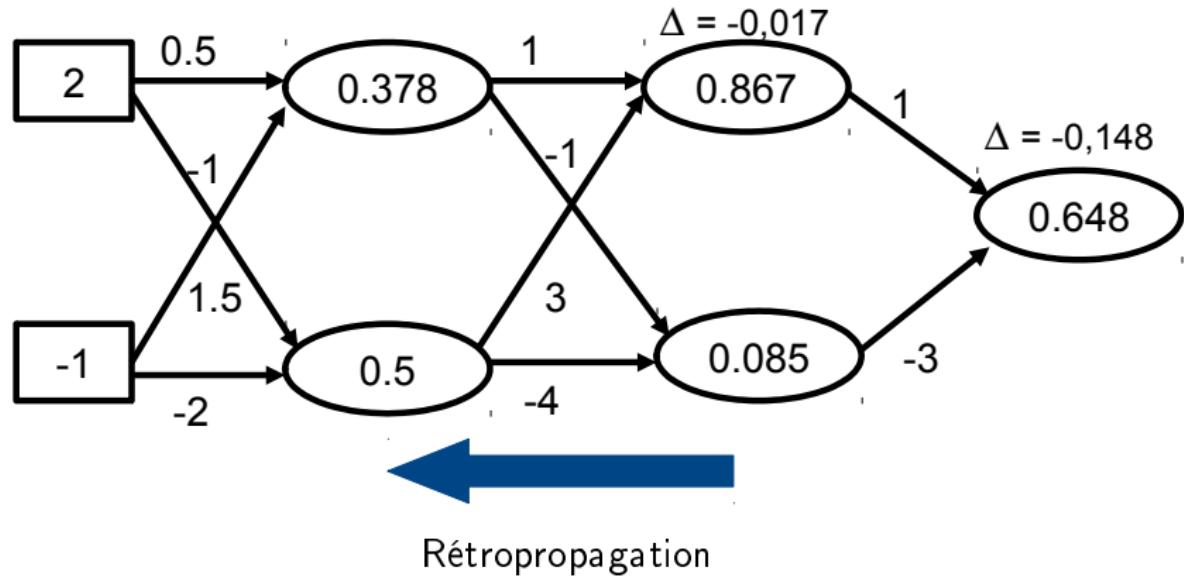


Rétropropagation

$$\Delta[i] = \text{Logistic}(in_i) * (1 - \text{Logistic}(in_i)) * \sum_{j \in \text{suivants}(i)} w_{i,j} * \Delta[j]$$

# Exemple de déroulement

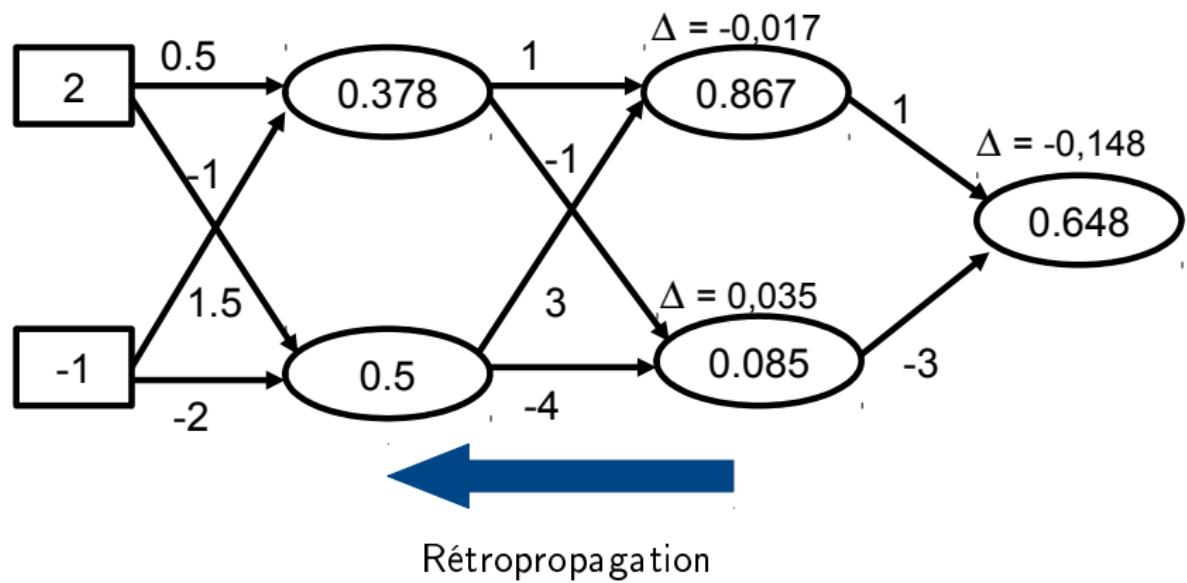
Exemple :  $x=[2, -1]$ ,  $y=0$



$$\Delta = 0.867 * (1 - 0.867) * (1 * -0.148) = -0.017$$

# Exemple de déroulement

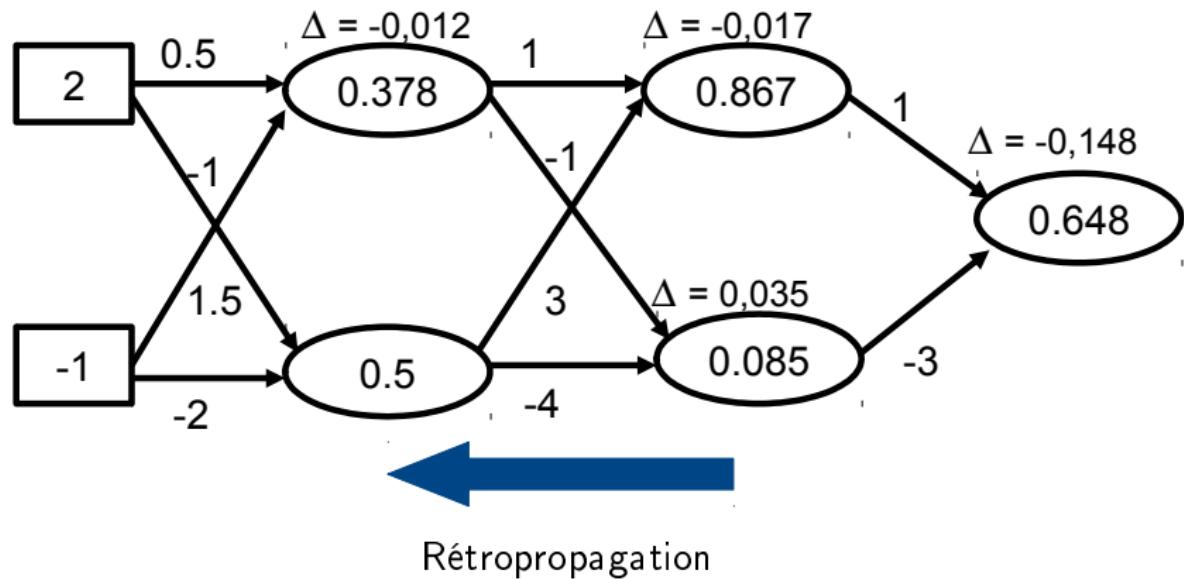
Exemple :  $x=[2, -1]$ ,  $y=0$



$$\Delta = 0.085 * (1 - 0.085) * (-3 * -0.148) = 0.035$$

## Exemple de déroulement

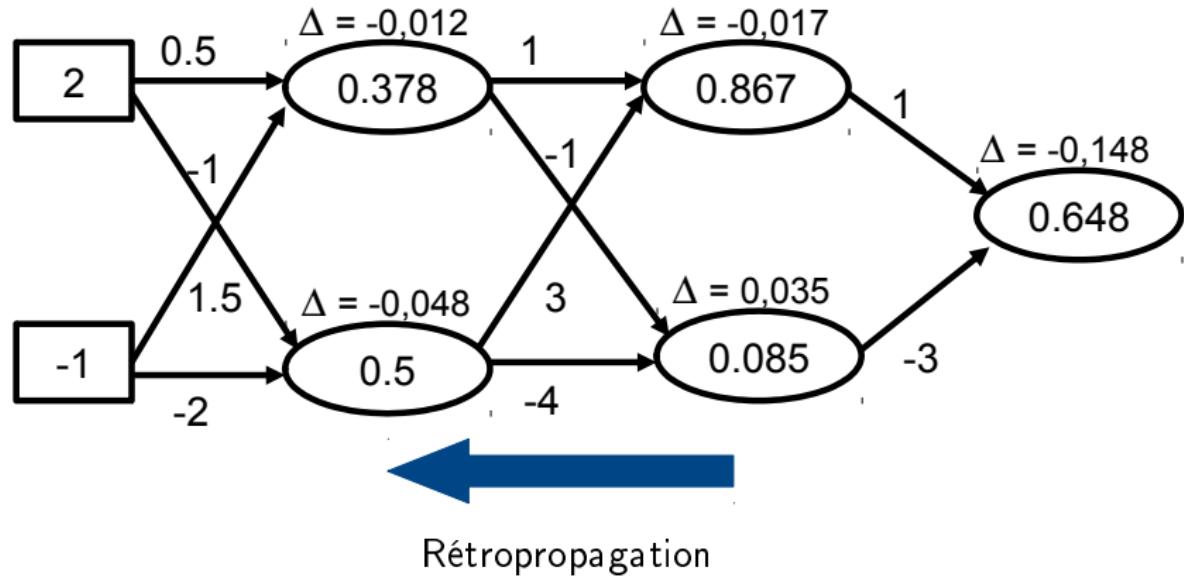
Exemple :  $x=[2, -1]$ ,  $y=0$



$$\Delta = 0.378 * (1 - 0.378) * (1 * -0.017 + -1 * 0.035) = -0.012$$

# Exemple de déroulement

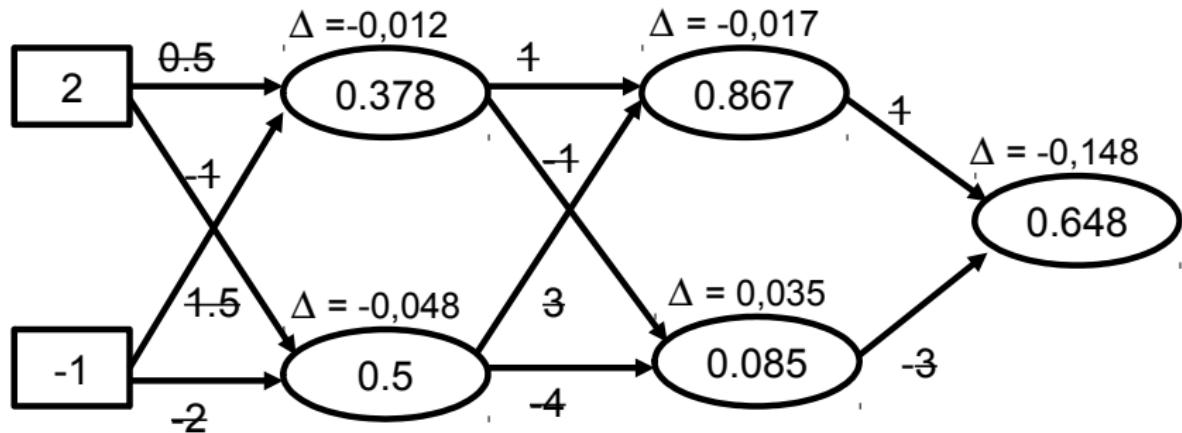
Exemple :  $x=[2, -1]$ ,  $y=0$



$$\Delta = 0.5 * (1 - 0.5) * (3 * -0.017 + -4 * 0.035) = -0.048$$

# Exemple de déroulement

Exemple :  $x=[2, -1]$ ,  $y=0$

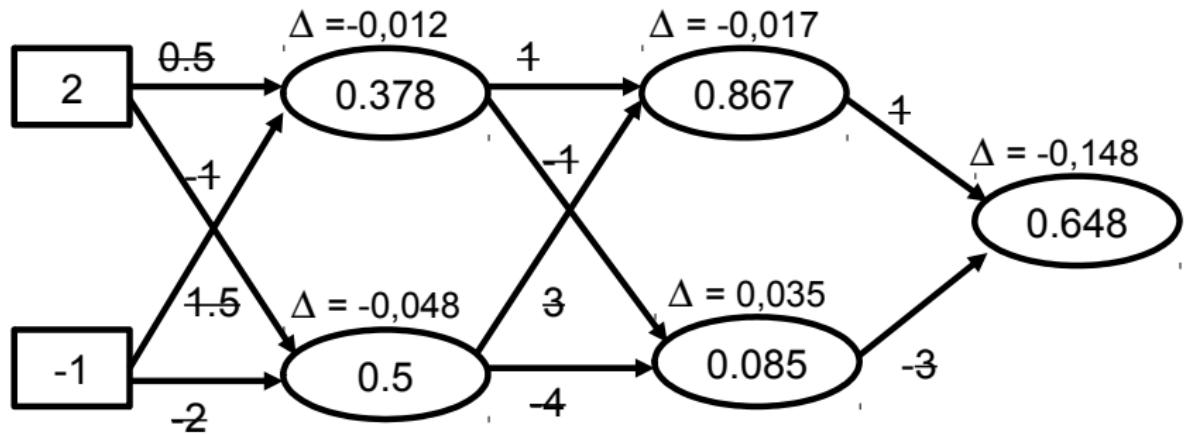


Mise à jours des poids avec  $\alpha = 1$

$$w_{i,j} \leftarrow w_{i,j} + \alpha * \Delta[j] * a_i$$

# Exemple de déroulement

Exemple :  $x=[2, -1]$ ,  $y=0$



$$w_{1,3} \leftarrow 0.5 + 1 * -0.012 * 2 = 0.48 \quad w_{3,5} \leftarrow 1 + 1 * -0.017 * 0.378 = 0.99$$

$$w_{1,4} \leftarrow -1 + 1 * -0.048 * 2 = -1.10 \quad w_{3,6} \leftarrow -1 + 1 * 0.035 * 0.378 = -0.99 \quad w_{5,7} \leftarrow 1 + 1 * -0.148 * 0.867 = 0.87$$

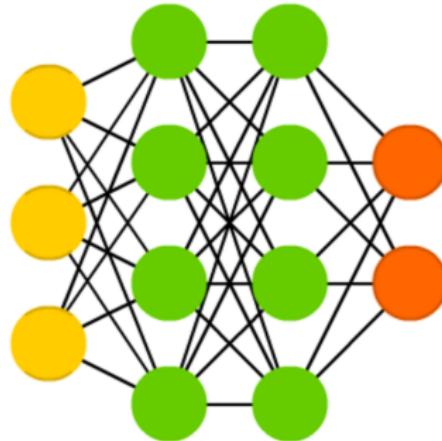
$$w_{2,3} \leftarrow 1.5 + 1 * -0.012 * -1 = 1.51 \quad w_{4,5} \leftarrow 3 + 1 * -0.017 * 0.5 = 3$$

$$w_{6,7} \leftarrow -3 + 1 * -0.148 * 0.085 = -3.0$$

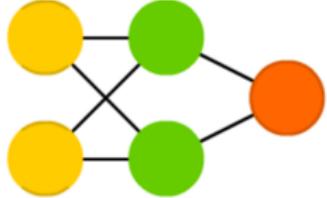
$$w_{2,4} \leftarrow -2 + 1 * -0.048 * -1 = -1.95 \quad w_{4,6} \leftarrow -4 + 1 * 0.035 * 0.5 = -3.98$$

# Types de réseaux de neurones

## Deep Feed Forward (DFF)

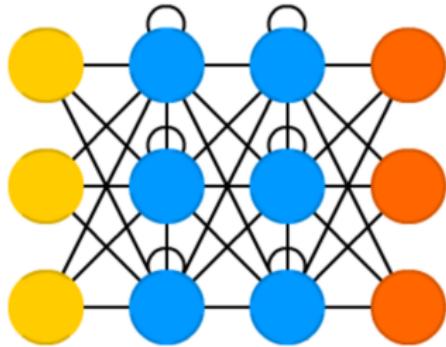


## Feed Forward (FF)

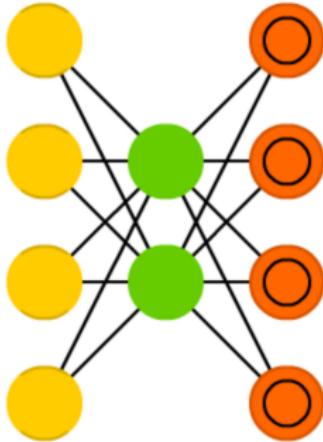


# Types de réseaux de neurones

Recurrent Neural Network (RNN)



Auto Encoder (AE)



# Reconnaissance d'images

## Classification



CAT

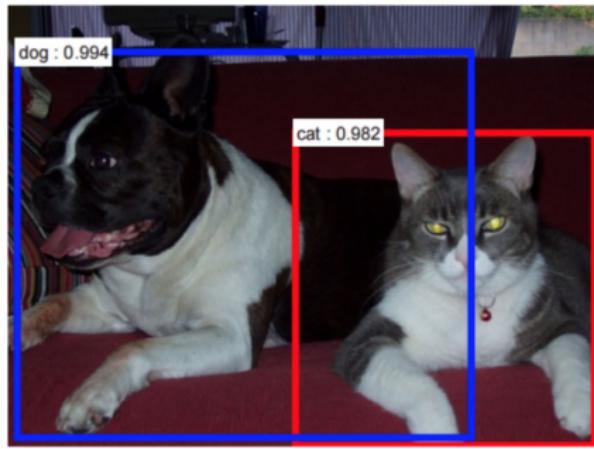
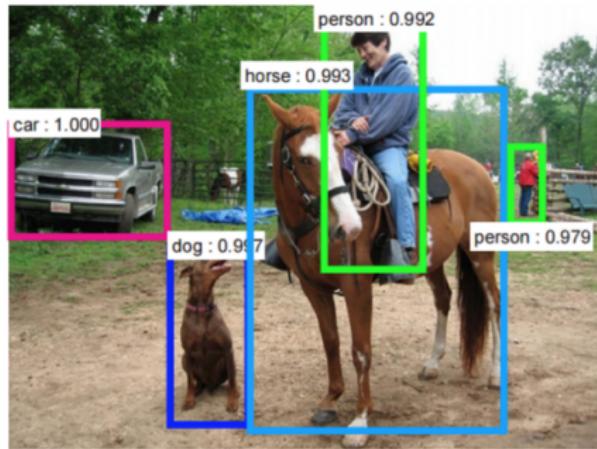


CAT



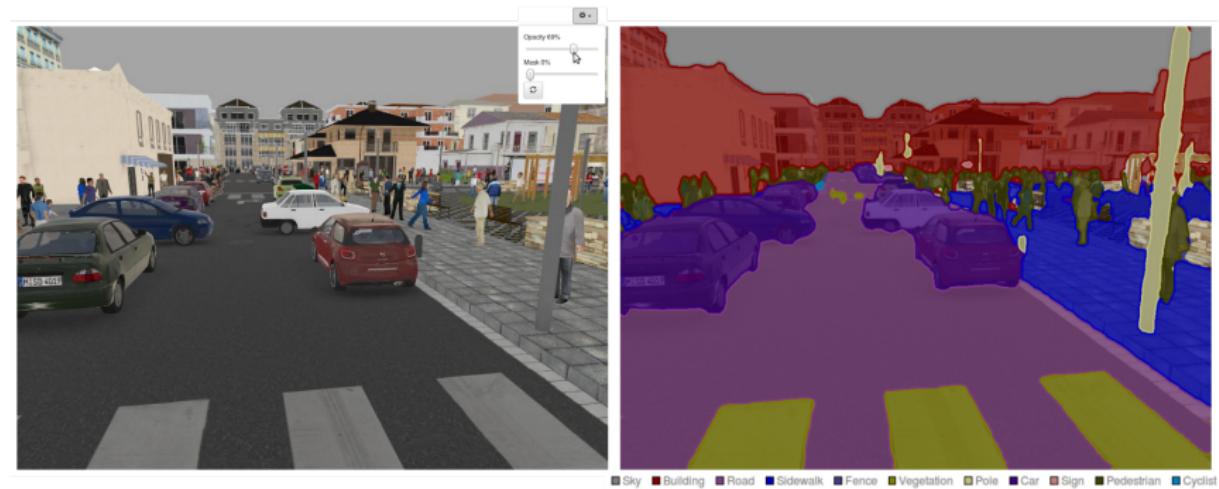
# Reconnaissance d'images

## Détection d'objets

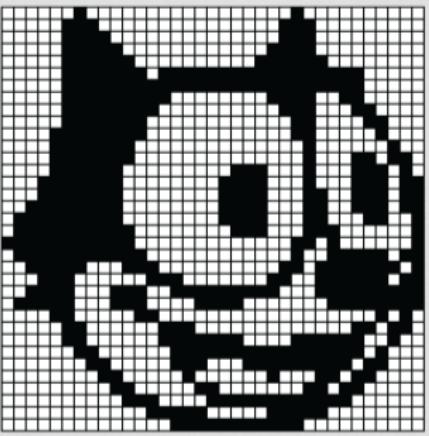


# Reconnaissance d'images

## Segmentation



## Reconnaissance d'images : L'image = matrice



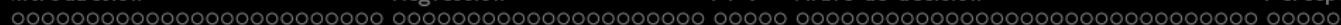
35x35

# Reconnaissance d'images : L'image = matrice

	165	187	209	58	7	
	14	125	233	201	98	159
253	144	120	251	41	147	204
67	100	32	241	23	165	30
209	118	124	27	59	201	79
210	236	105	169	19	218	156
35	178	199	197	4	14	218
115	104	34	111	19	196	
32	69	231	203	74		

# Reconnaissance d'images : convolution

<b>Identité</b>	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
<b>La détection de contours</b>	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
<b>Amélioration de la netteté</b>	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
<b>Box blur</b> (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
<b>Filtre de Gauss 3 × 3</b> (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

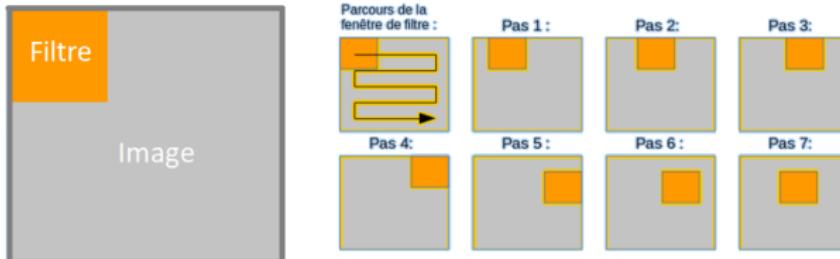


# Reconnaissance d'images : convolution

- Le filtre est une matrice de valeurs généralement de petites dimensions et bien souvent carrées. La taille de filtre la plus utilisée est de 3 par 3.

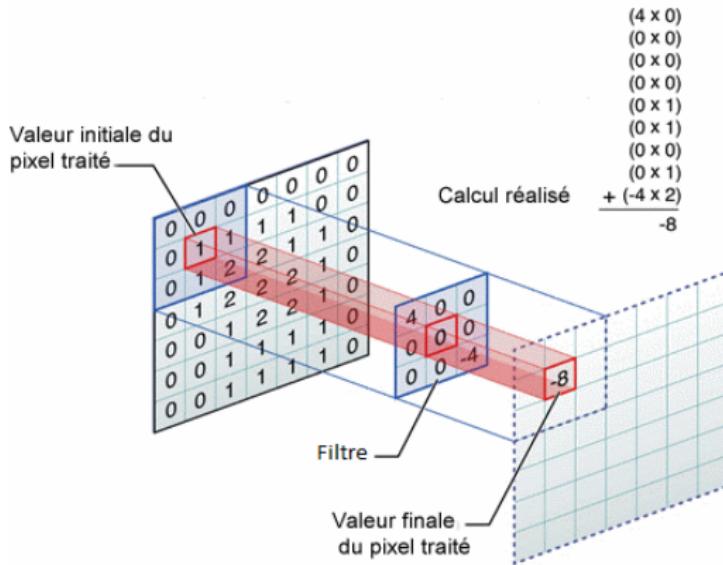
1	0	1
0	1	0
1	0	1

- Un filtre sert à faire ressortir certaines caractéristiques d'une image donnée (couleur, contour, luminosité, netteté, etc...).



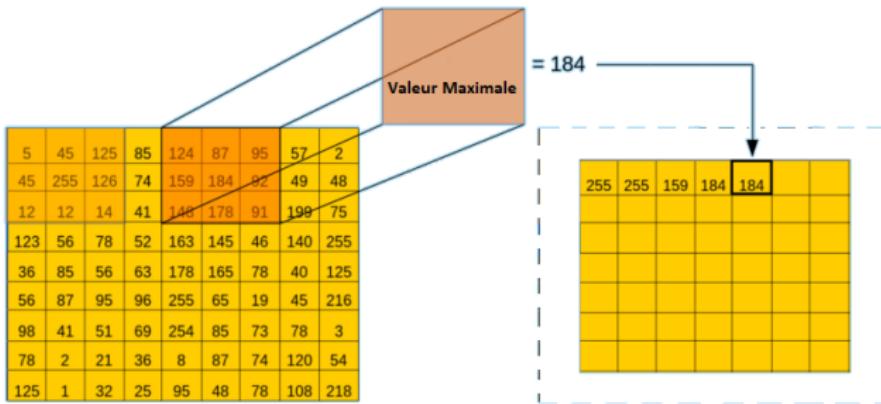
# Reconnaissance d'images : convolution

- Il est préférable de ne pas utiliser de filtres trop petits ou un nombre de filtres trop important car cela peut entraîner un sur-apprentissage.



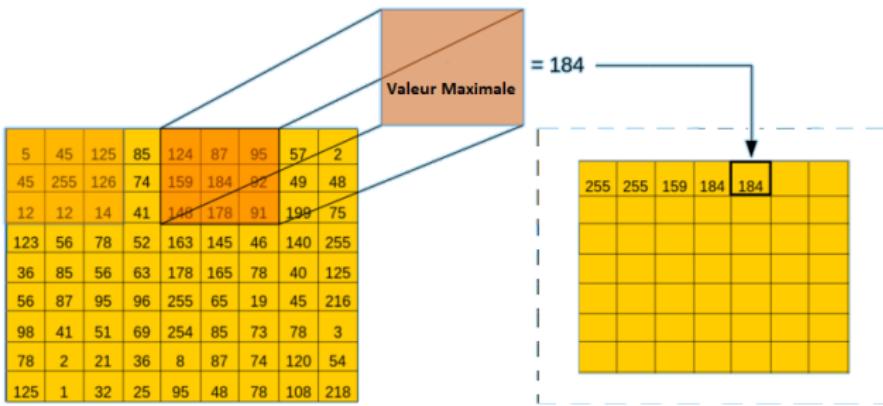
# Reconnaissance d'images : le pooling

- Le Pooling permet de réduire une image tout en conservant les caractéristiques pertinentes.
- L'image de 9 par 9 pixels de départ est réduite en une image de 7 par 7 pixels.



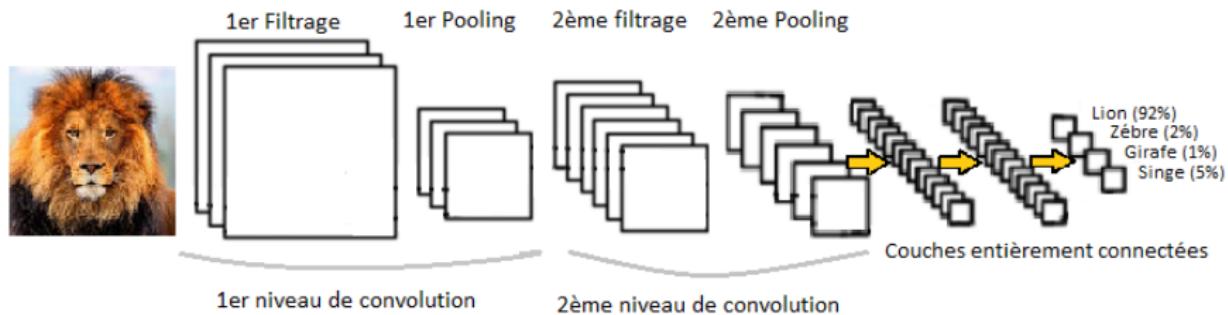
# Reconnaissance d'images : le pooling

- Il existe d'autres méthodes que le "Max Pooling" :
  - Average Pooling** : moyenne de toutes les valeurs recouvertes par la tuile.
  - Stochastic Pooling** : ne retient qu'une seule valeur en se basant sur une méthode probabiliste.



# Reconnaissance d'images : la classification

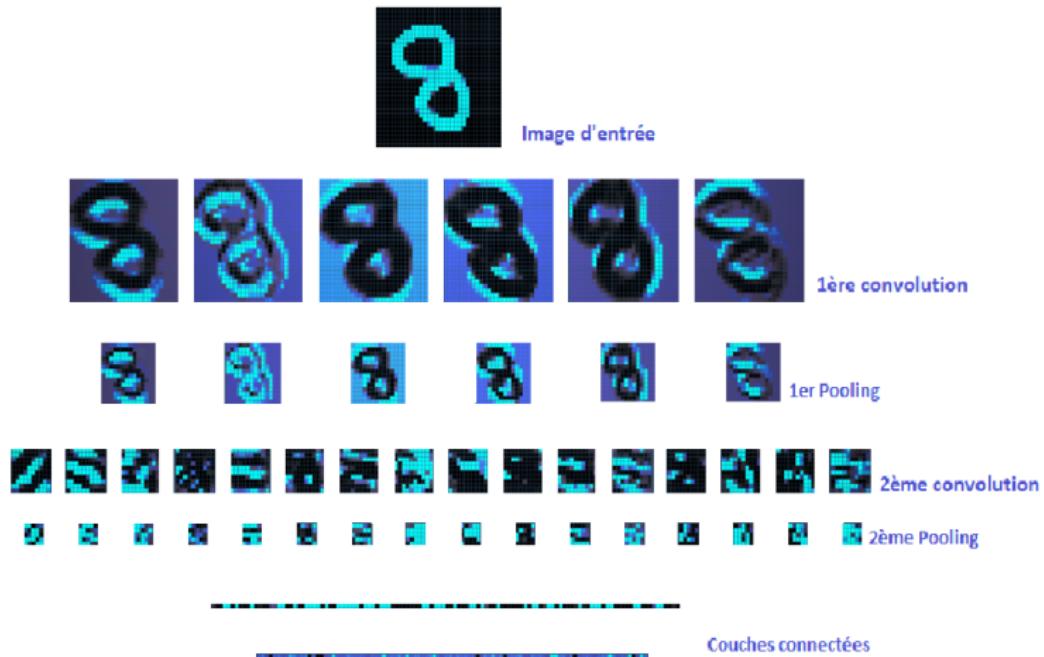
Les patterns obtenues en sortie de la convolution sont injectés comme données d'entrée dans un réseau neuronal classique.





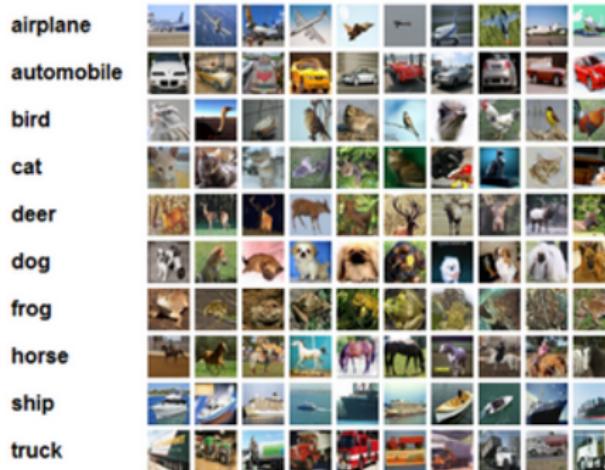
# Reconnaissance d'images : la classification

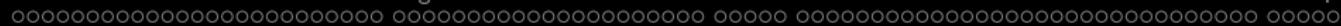
Les patterns obtenues en sortie de la convolution sont injectés comme données d'entrée dans un réseau neuronal classique.



# Reconnaissance d'images

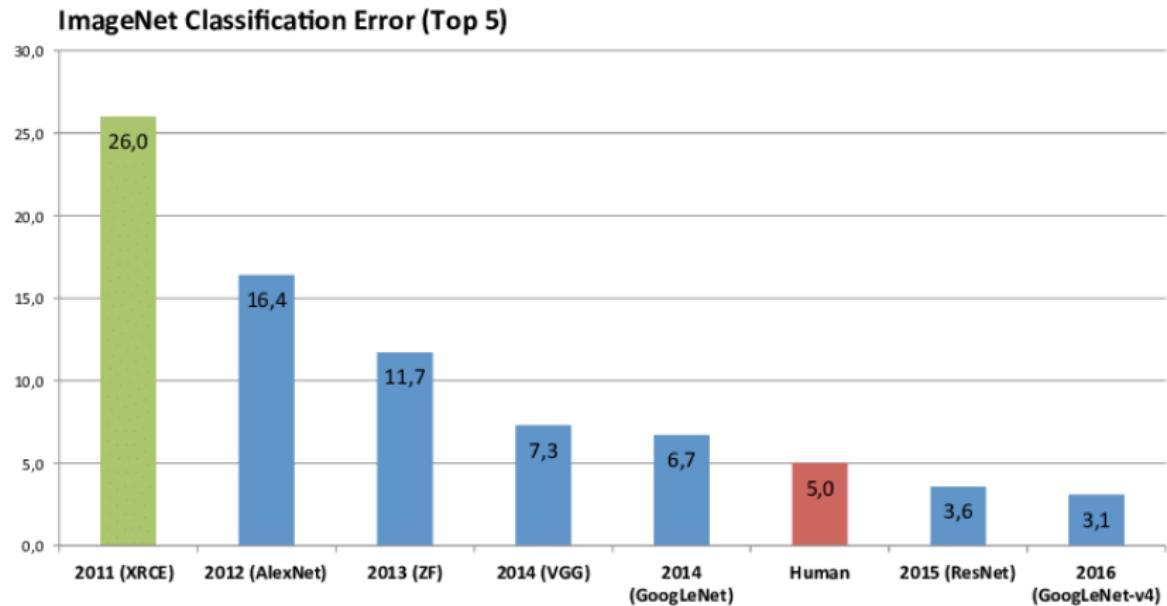
ImageNet est une base de données d'images annotées à destination des travaux de recherche en vision par ordinateur.





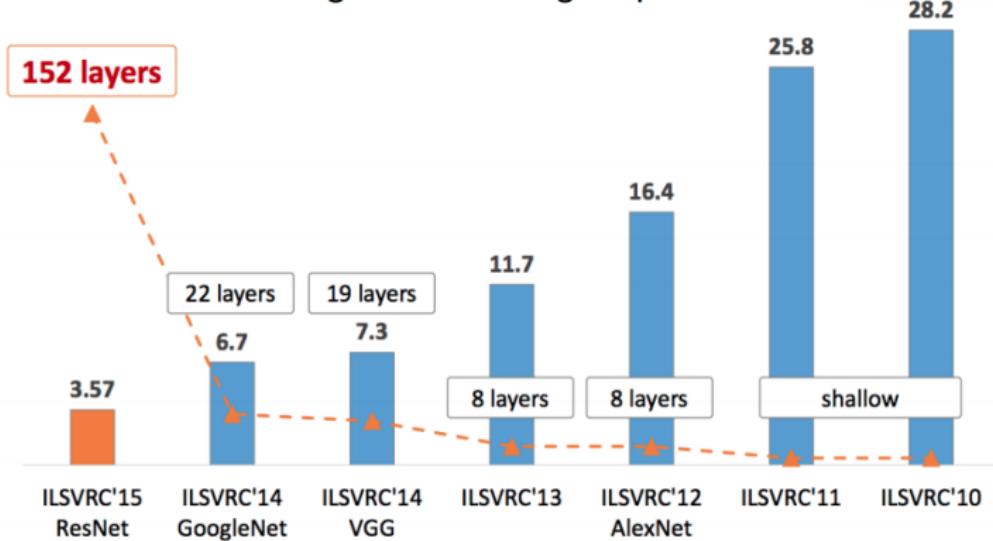
# Reconnaissance d'images

Le challenge de reconnaissance visuelle à grande échelle d'ImageNet (ILSVCR)

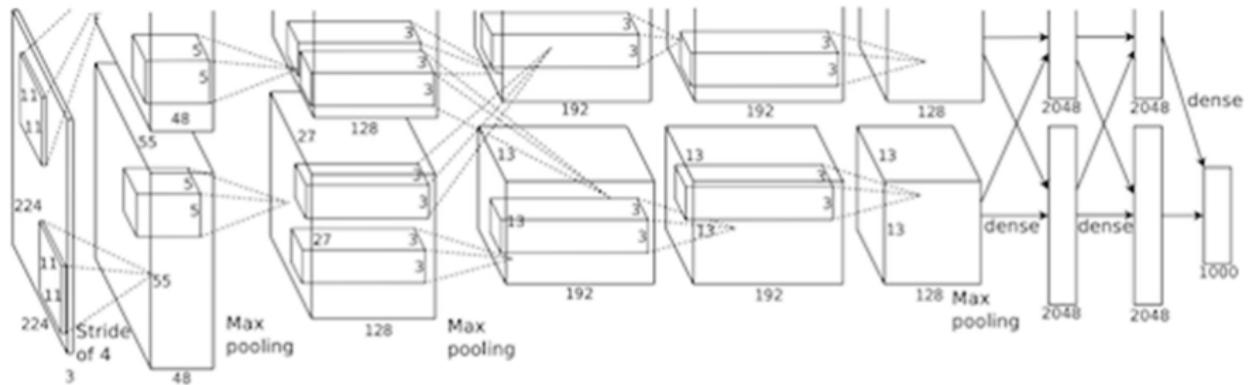


# Reconnaissance d'images

## Classification: ImageNet Challenge top-5 error

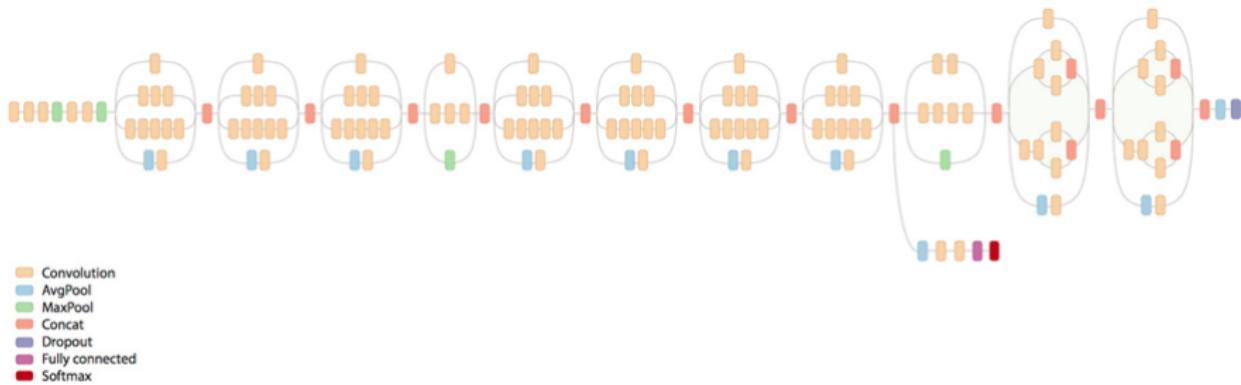


# Reconnaissance d'images : AlexNet 2012



AlexNet architecture (May look weird because there are two different “streams”. This is because the training process was so computationally expensive that they had to split the training onto 2 GPUs)

# Reconnaissance d'images : GoogLeNet 2014



Another view of GoogleNet's architecture.