APPRENTISSAGE SUPERVISÉ
TROISIÈME PARTIE

Les boîtes noires

Nous parlerons dans ce cours de deux types de méthodes : les **SVM** et les **réseaux de neurones**. Elles ont ceci en commun qu'on parle de ces méthodes comme des **boîtes noires** :

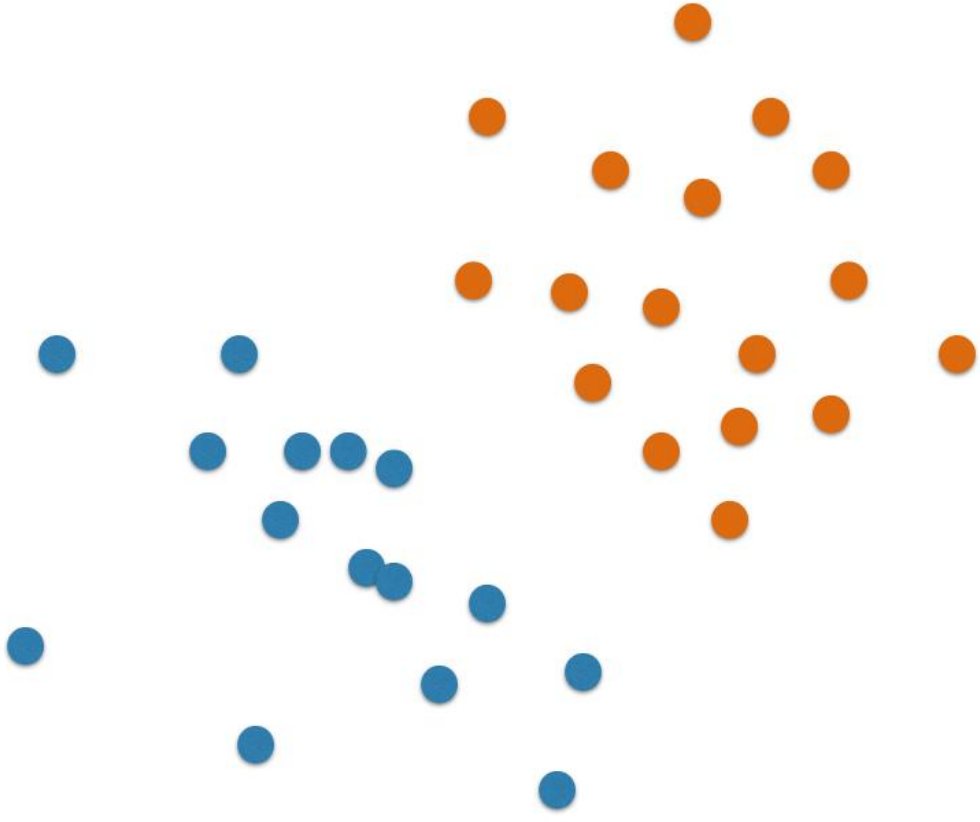
- elles reposent sur des calculs complexes;
- les résultats sont difficilement interprétables;
- de nombreux paramètres doivent être optimisés;
- elles impliquent beaucoup de "tuning" manuel;
- elles sont populaires car performantes, en particulier sur les données à structure complexe.

Les Support Vector Machines

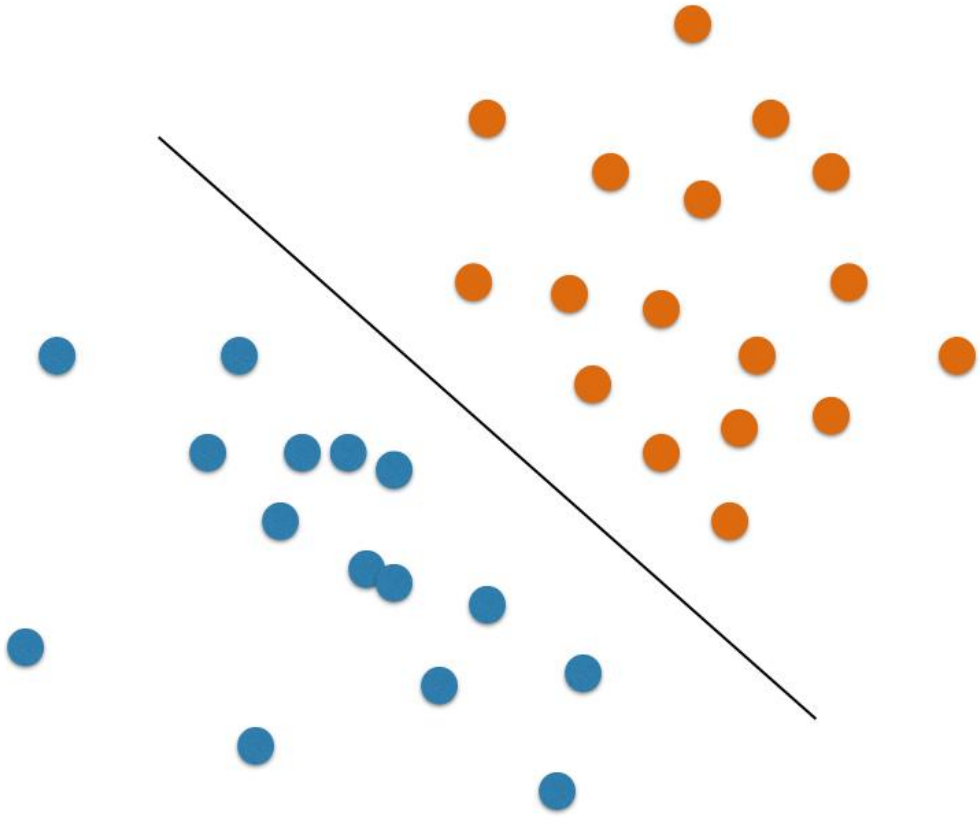
INTRODUCTION

L’arrivée des **SVMs** en **1992** marque un tournant dans l’histoire du **machine learning**.
La théorie de laquelle ces méthodes sont issues est alors nouvelle, intuitive et permet de résoudre des problèmes complexes dans un **nouveau paradigme**.
Les notions mathématiques à la base des SVMs sont difficiles. Nous n’aborderons dans ce cours que les concepts.

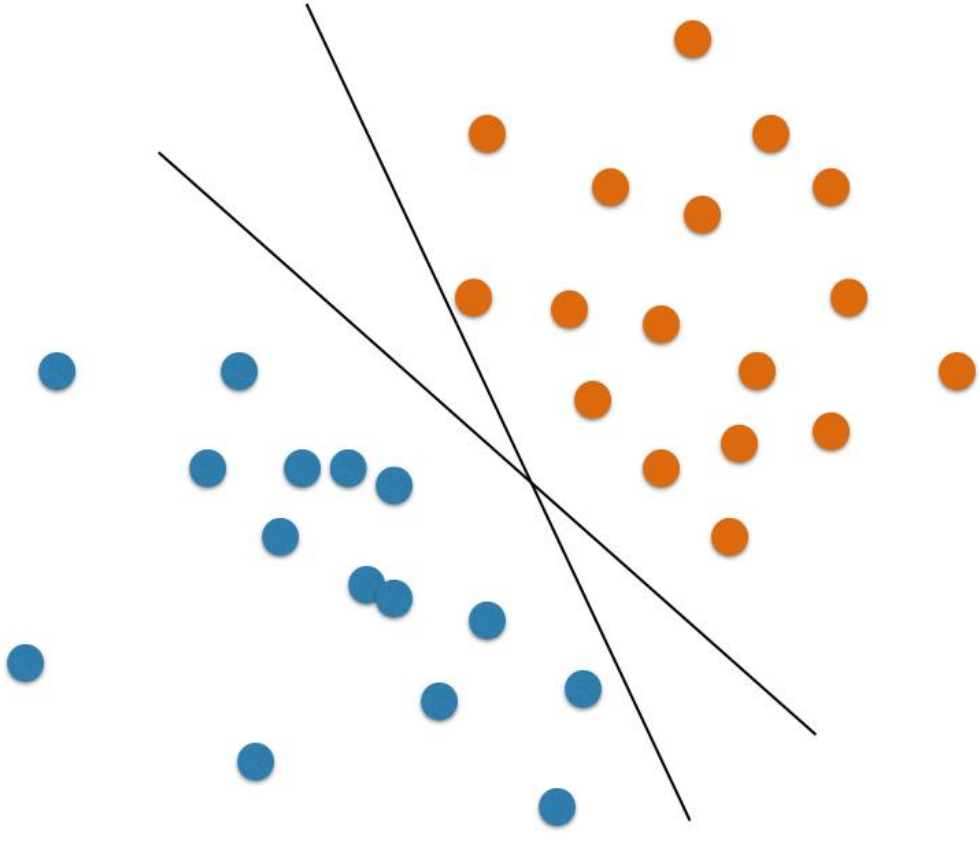
CLASSIFIEURS À VASTE MARGE: DONNÉES LINÉAIREMENT SÉPARABLES



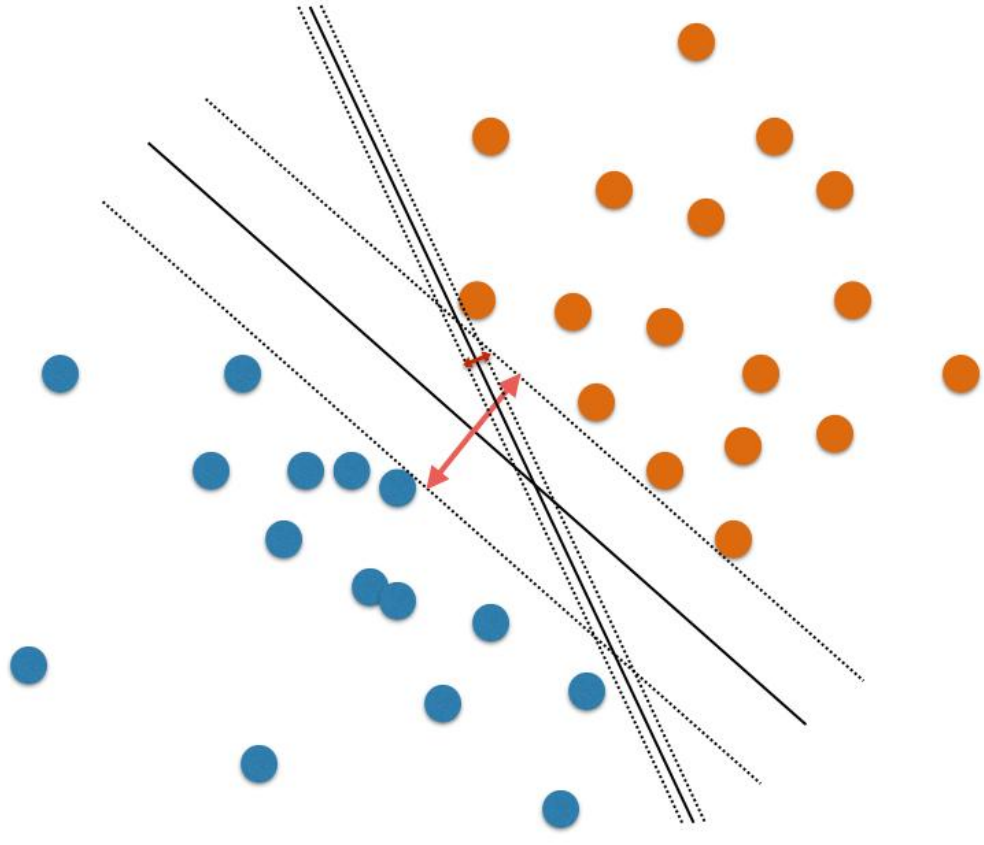
CLASSIFIEURS À VASTE MARGE: DONNÉES LINÉAIREMENT SÉPARABLES



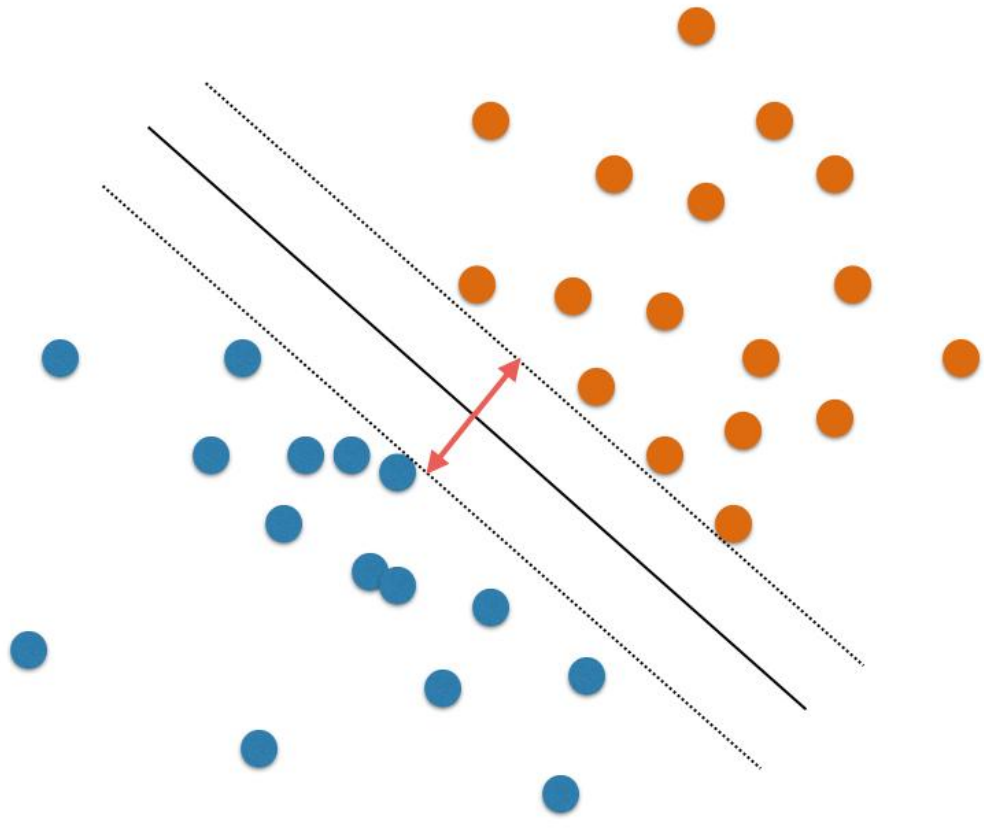
CLASSIFIEURS À VASTE MARGE: DONNÉES LINÉAIREMENT SÉPARABLES



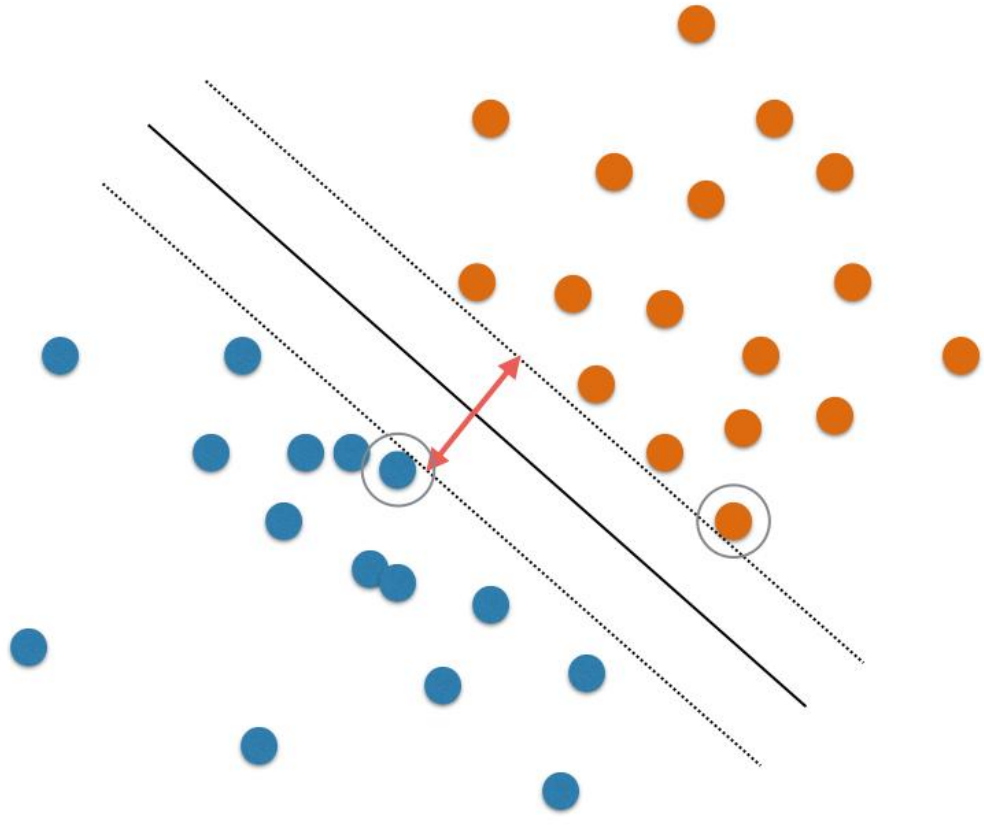
CLASSIFIEURS À VASTE MARGE: DONNÉES LINÉAIREMENT SÉPARABLES



CLASSIFIEURS À VASTE MARGE: DONNÉES LINÉAIREMENT SÉPARABLES

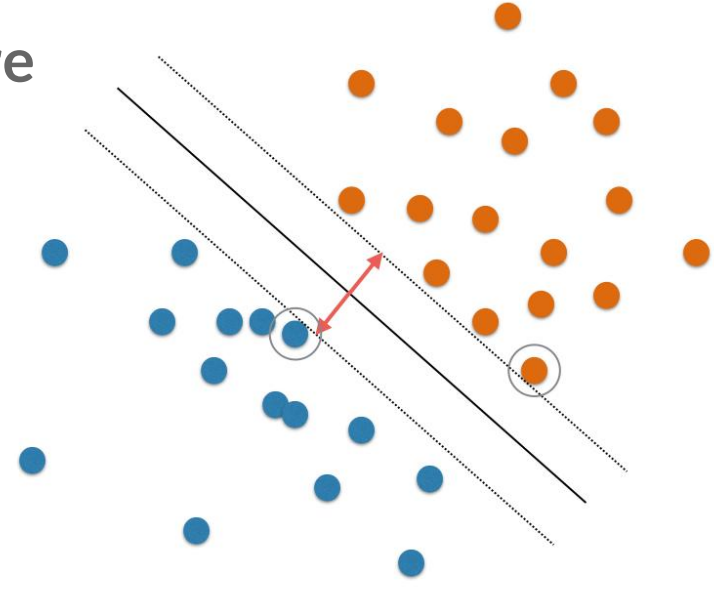


CLASSIFIEURS À VASTE MARGE: DONNÉES LINÉAIREMENT SÉPARABLES



CLASSIFIEURS À VASTE MARGE: DONNÉES LINÉAIREMENT SÉPARABLES

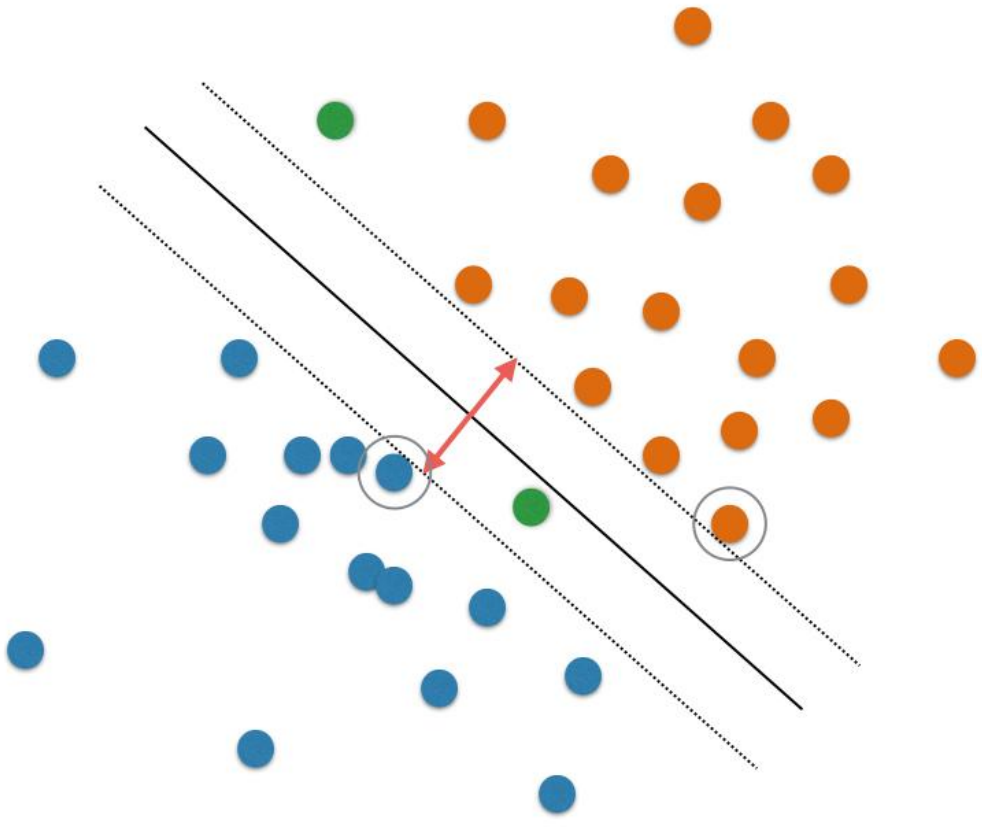
- Il existe une **infinité** de droites séparant les points.
- On cherche la **séparation linéaire** qui donne la plus grande **marge**.
- Les "bords" de cette marge s'appuient sur des **vecteurs supports**.
- On trouve cette séparation en résolvant un problème d'**optimisation convexe**.



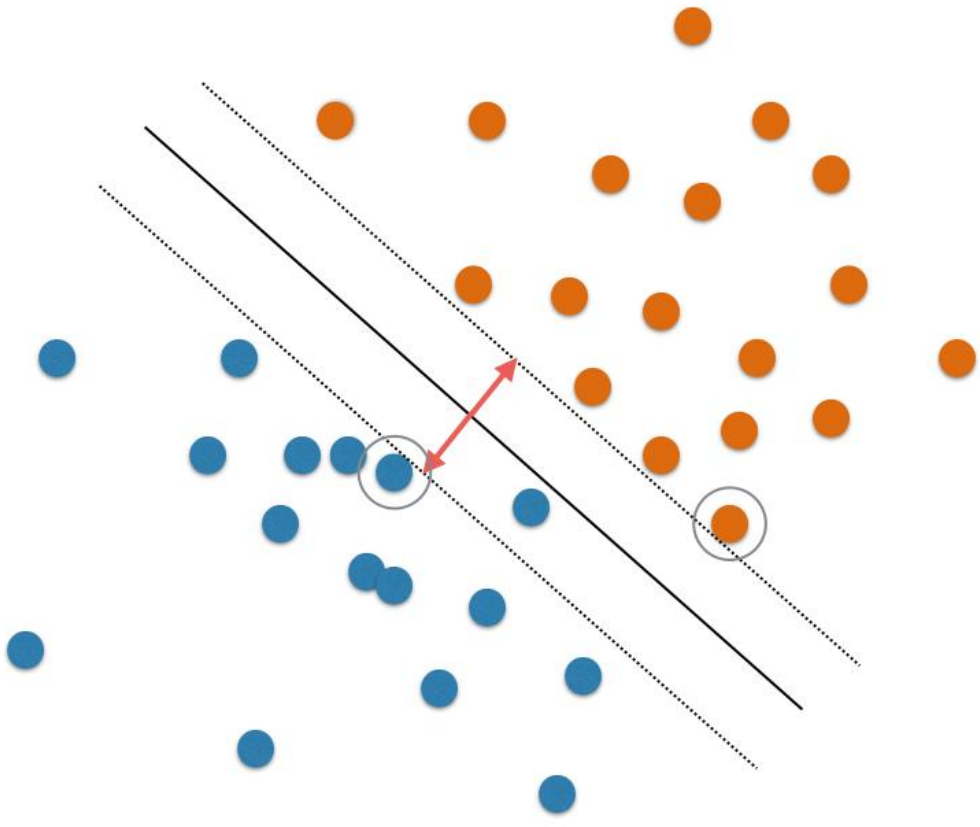
INTUITION

- Les **vecteurs supports** sont les points les plus **ambigus** et donc les plus **difficiles à classifier**.
- Ce sont ces points qui influencent le choix de la **meilleure droite** : si ces points changent, la droite change.
- En quelque sorte, on se met dans un "worst-case" scenario : puisque l'on sait classifier les points les plus ambigus, le système devrait être **robuste**.
- **Plus la marge est grande, plus l'on est sûrs de nous.**

PHASE DE TEST



PHASE DE TEST



PROBLÈME

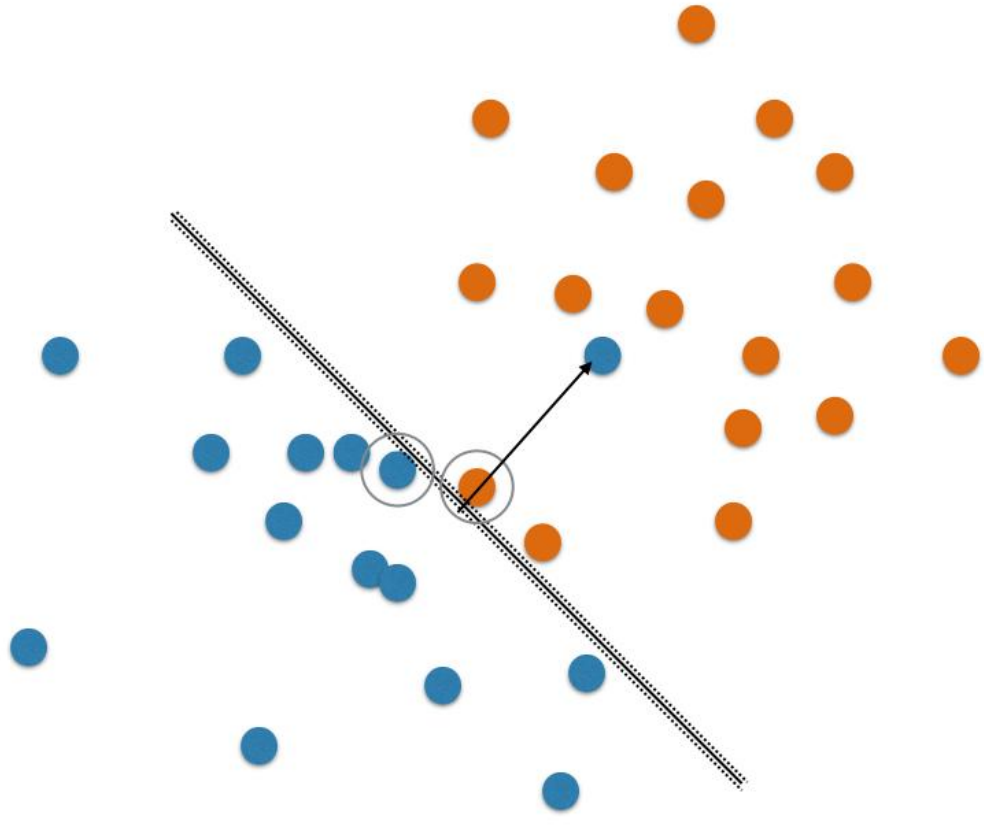
Les données sont rarement aussi simples !

CLASSIFIEURS À VASTE MARGE: CAS NON LINÉAIREMENT SÉPARABLE

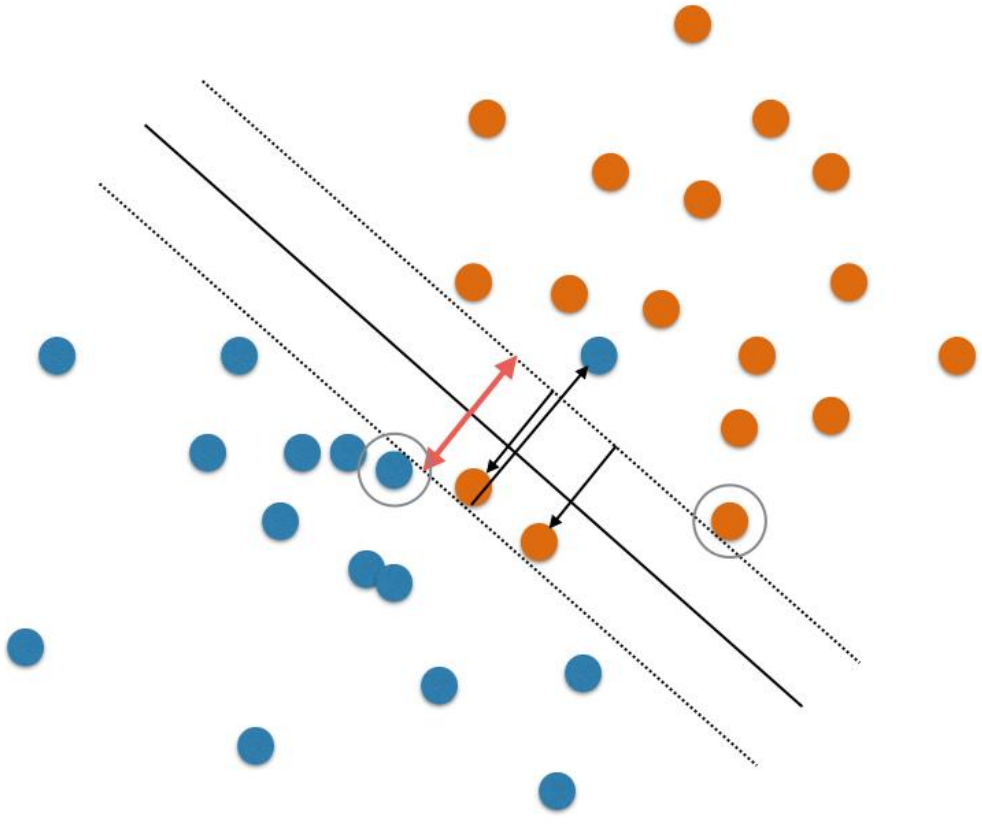
CLASSIFIEURS À VASTE MARGE: CAS NON LINÉAIREMENT SÉPARABLE

- On **autorise** des erreurs de classification sur l'**ensemble d'entraînement**. Il sera moins sensible aux **outliers** et donc plus robuste.
- Mais jusqu'à quel point autoriser ces erreurs ?

CLASSIFIEURS À VASTE MARGE: CAS NON LINÉAIREMENT SÉPARABLE

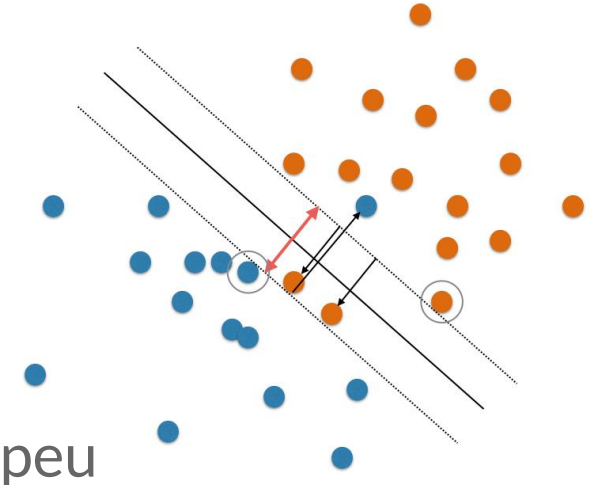


CLASSIFIEURS À VASTE MARGE: CAS NON LINÉAIREMENT SÉPARABLE



CLASSIFIEURS À VASTE MARGE: CAS NON LINÉAIREMENT SÉPARABLE

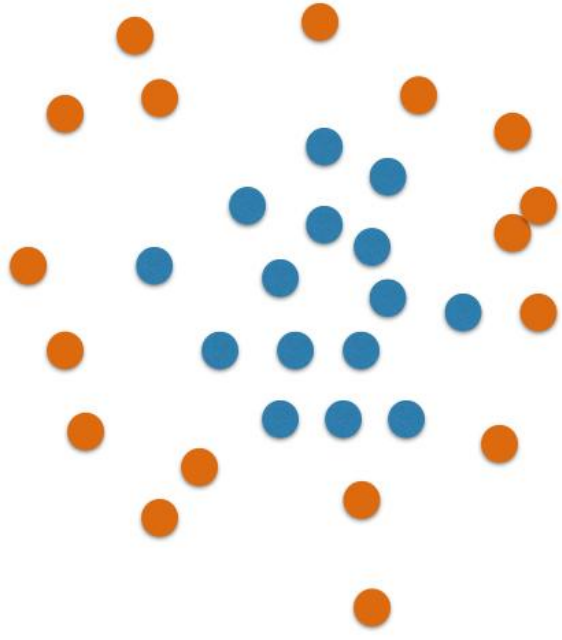
- C'est un **paramètre de régularisation** qui décide à quel point on peut faire des erreurs. Plus il est petit, plus les erreurs sont autorisées et plus la marge est large.
- On préfère des erreurs sur l'ensemble d'entraînement plutôt que du **sur-apprentissage**.
- Il faut trouver le bon **compromis** entre peu d'erreurs et une marge importante.
- Le paramètre de régularisation est optimisé en **validation croisée**.



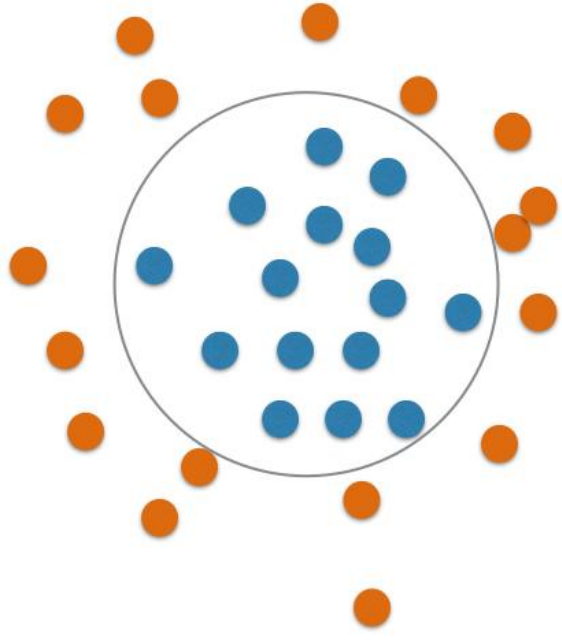
Démo

http://cs.stanford.edu/people/karpathy/svmjs/demo/

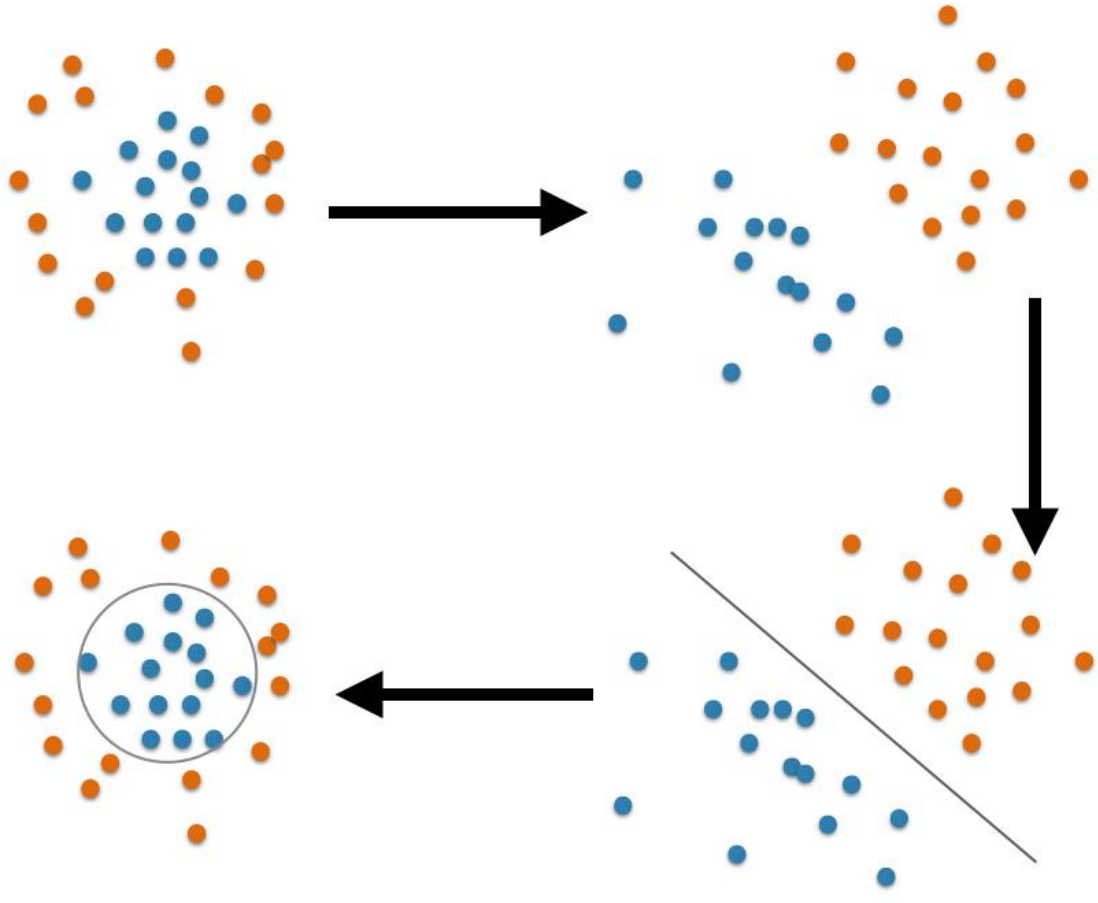
ET SI LA SÉPARATION LINÉAIRE N'EST PAS DU TOUT ADAPTÉE ?



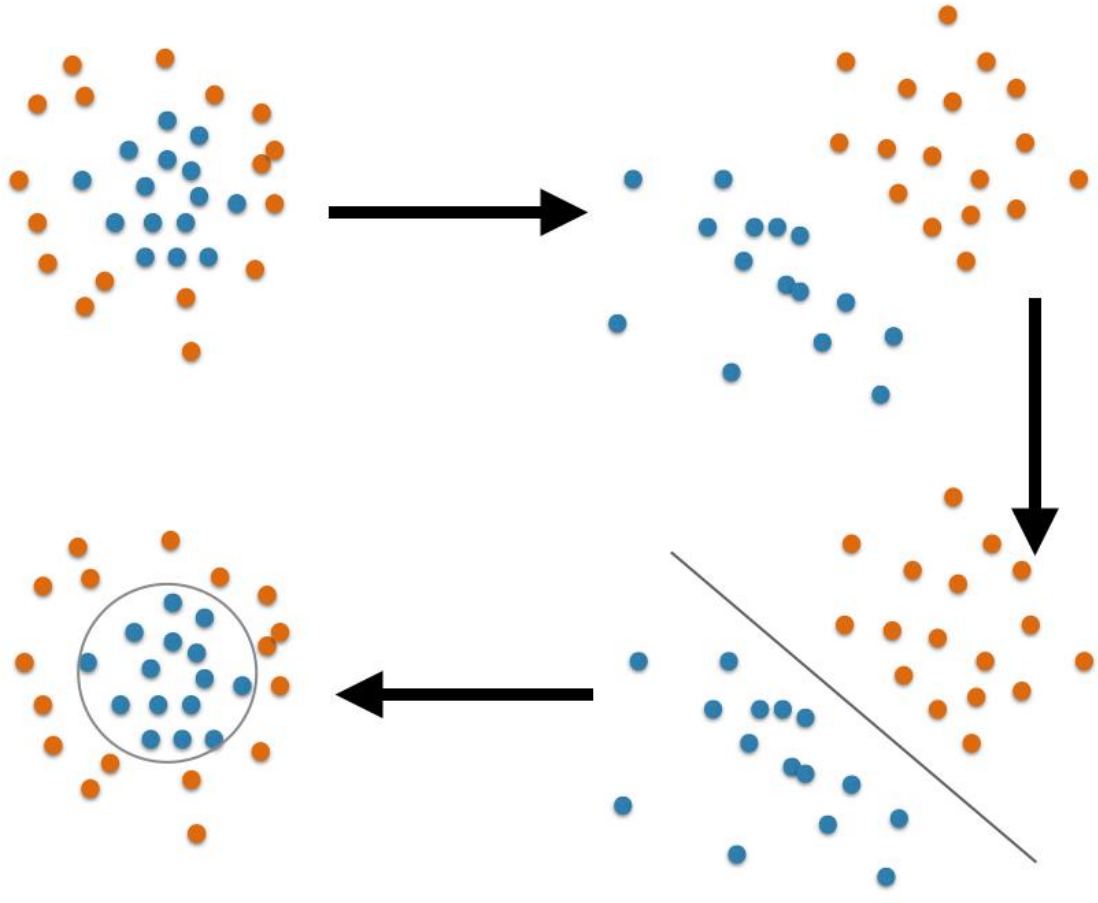
ET SI LA SÉPARATION LINÉAIRE N'EST PAS DU TOUT ADAPTÉE ?



ET SI LA SÉPARATION LINÉAIRE N'EST PAS DU TOUT ADAPTÉE ?



ET SI LA SÉPARATION LINÉAIRE N'EST PAS DU TOUT ADAPTÉE ?



ET SI LA SÉPARATION LINÉAIRE N'EST PAS DU TOUT ADAPTÉE ?

- L'idée est donc d'appliquer une **transformation** aux données pour se trouver dans un cas où le problème redevient linéaire.
- Cette transformation passe par ce qu'on appelle un **noyau**.
- Un noyau est une mesure de **similarité** souvent notée K (kernel) qui mesure à quel point deux vecteurs sont proches.
- En réalité, on n'applique pas la transformation à chaque vecteur directement mais à leur similarité deux à deux.
- Il existe différents types de noyaux. Il faut choisir celui qui est le plus adapté aux données.

Démo

http://cs.stanford.edu/people/karpathy/svmjs/demo/

COMMENT DÉFINIR LE BON NOYAU ?

Il existe des noyaux “off-the shelf”, tous prêts à être utilisés. On peut en essayer différents pour voir ce qui fonctionne le mieux. Mais ce n’est pas l’utilisation optimale de ce modèle.

Pour entraîner un SVM, on a “juste” besoin du noyau. On n’a pas besoin de savoir représenter les objets eux-mêmes.

Donc on utilise un SVM en particulier quand on sait définir la similarité entre deux objets (similarité exprimée dans le noyau), mais qu’on n’a pas de “features” pour les objets.

QUELQUES NOYAUX

- Le noyau **linéaire**

$$K(x_i, x_j) = x_{i,1}x_{j,1} + x_{i,2}x_{j,2} + \ldots + x_{i,p}x_{j,p}$$

- Le noyau **RBF**

$$K(x_i, x_j) = e^{-\gamma(x_{i,1}-x_{j,1})^2+(x_{i,2}-x_{j,2})^2+\ldots+(x_{i,p}-x_{j,p})^2}$$

- Le noyau **polynomial**

$$K(x_i, x_j) = (x_{i,1}x_{j,1} + x_{i,2}x_{j,2} + \ldots + x_{i,p}x_{j,p} + c)^p$$

- Et surtout : le noyau customisé ! Sans même savoir décrire les objets, il suffit de savoir décrire leur similarité par un noyau pour faire fonctionner un SVM.

Les noyaux **déforment** donc les distances entre les points et changent par conséquent la forme du problème.

DÉMO

SVMs non linéaires

EXEMPLE CUSTOMISÉ

SVM pour classifier des protéines. On peut comparer deux séquences comme on comparerait 2 chaînes de caractères:

Protéine 1: AGGGCTTACTA

Protéine 2: GGGCTACTAGGGGCC

EXEMPLE CUSTOMISÉ

SVM pour classifier des protéines. On peut comparer deux séquences comme on comparerait 2 chaînes de caractères:

Protéine 1: AGGGCTTACTA

Protéine 2: GGGCT ACTAGGGGCC

Par exemple une distance de Levenshtein, avec trois opérations possibles: insertion, deletion, replacement.

EXEMPLE CUSTOMISÉ

SVM pour classifier des protéines. On peut comparer deux séquences comme on comparerait 2 chaînes de caractères:

Protéine 1: AGGGCTTACTA

Protéine 2: GGGCT ACTAGGGGCC

Puisque vous êtes capables de calculer cette distance pour toutes les paires de protéines, vous avez tout ce qu'il vous faut pour entraîner un SVM avec ce noyau. A aucun moment vous n'avez eu besoin de trouver des features pour représenter chaque protéine.

EN PYTHON

```
from sklearn.svm import SVC
model = SVC(C=1, kernel = 'linear')
model.fit(Xtrain,Ytrain)
predictions = model.predict(Xtest)
```

CONCLUSION SUR LES SVMS

Avantages :

- Très performants.
- Ne nécéssitent que la similarité des objets en entrée et non les objets en eux-mêmes. On peut donc traiter images, séquences biologiques, vidéos... Il n'y a pas de limite.
- Très efficaces en grande dimension.

Inconvénients :

- Il faut pouvoir fournir un noyau adapté, sans quoi le modèle échoue.
- Difficiles à interpréter.
- Dépendent souvent de paramètres à optimiser.
- Parfois long !

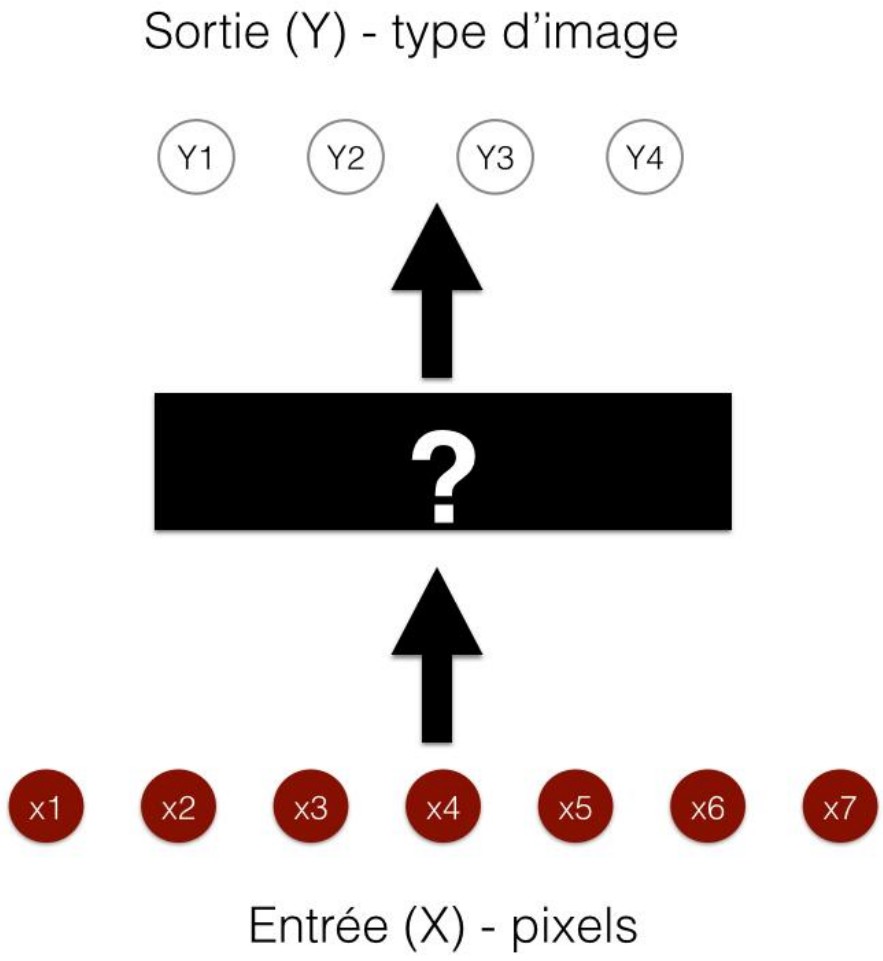
LES RÉSEAUX DE NEURONES

- Le perceptron multi-couches
- Le réseau convolutif

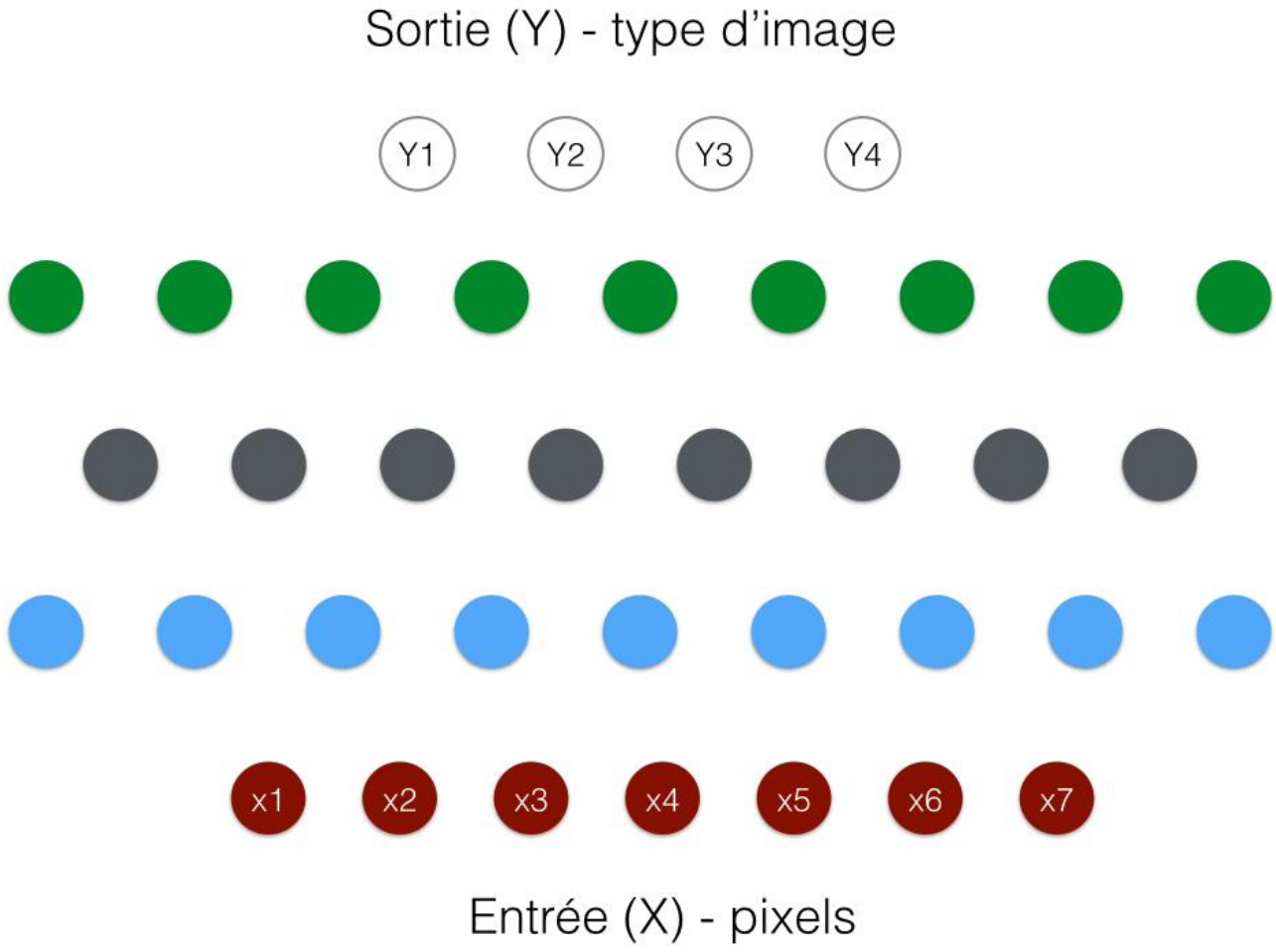
HISTOIRE

- Premières idées dans les années 50 : basé sur le fait de "copier" le fonctionnement des neurones du cerveau.
- Modèle classique : **perceptron multi-couches** introduit en 1986.
- **Théorème (Cybenko, 1989)** : Toute fonction continue bornée est estimable, avec une précision arbitraire, par un réseau à deux couches.
- Aujourd'hui : extension au **deep learning** avec en particulier les réseaux convolutifs pour les images.
- **Black box** par excellence : il est très compliqué de les interpréter / de les "tuner".

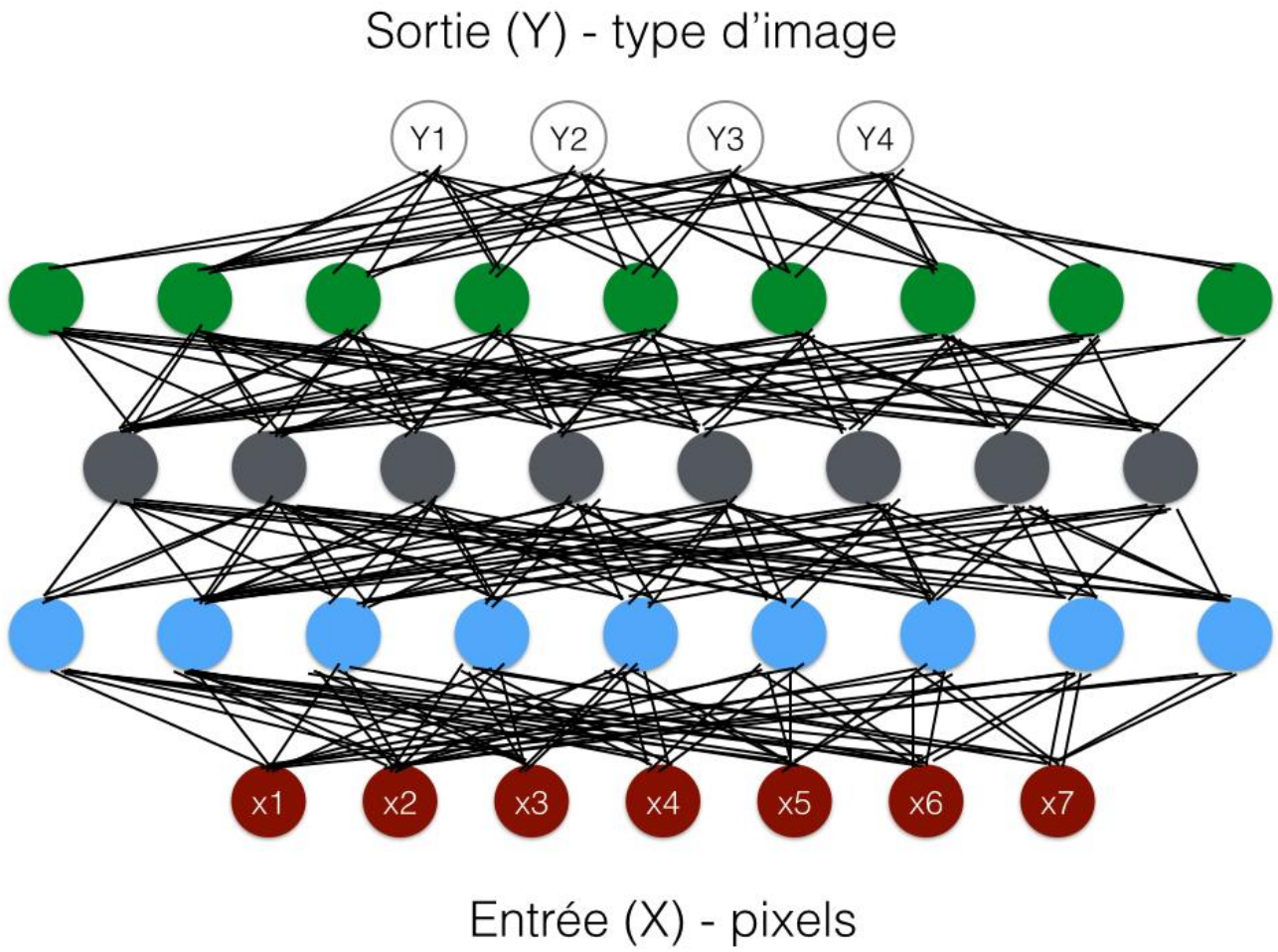
PRINCIPE


Sortie (Y) - type d'image
Y1
Y2
Y3
Y4
?
x1
x2
x3
x4
x5
x6
x7
Entrée (X) - pixels

PRINCIPE


Sortie (Y) - type d'image
Y1
Y2
Y3
Y4
x1
x2
x3
x4
x5
x6
x7
Entrée (X) - pixels

PRINCIPE


Sortie (Y) - type d'image
Y1
Y2
Y3
Y4
x1
x2
x3
x4
x5
x6
x7
Entrée (X) - pixels

PRINCIPE

Le réseau est composé:

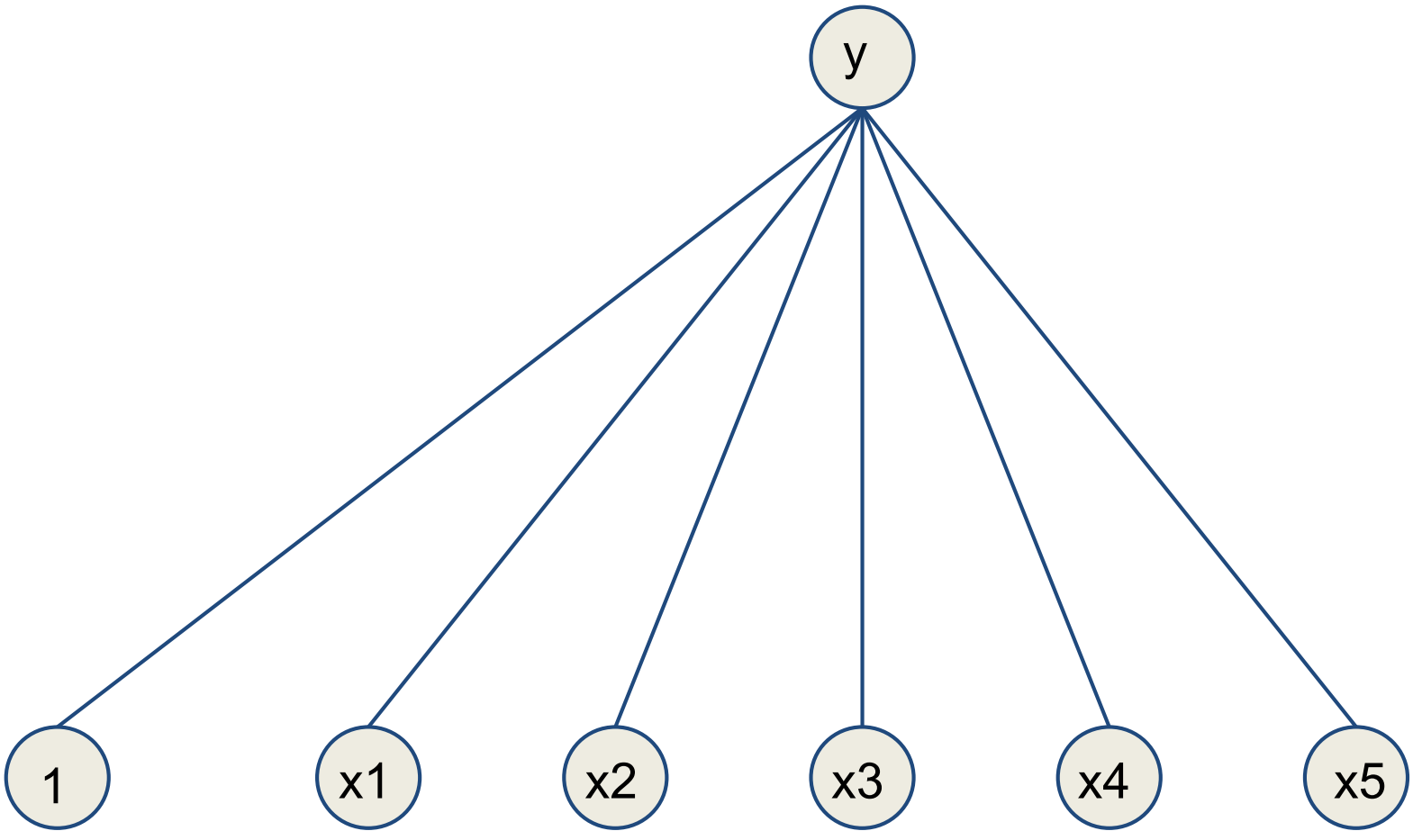
- d'une couche d'entrée (par exemple, des pixels)
- d'une couche de sortie (ce qu'on veut prédire)
- de couches intermédiaires appelées couches cachées

A partir de la première couche cachée, chaque "neurone" est une fonction de tous les neurones de la couche précédente.

Les neurones de la couche de sortie qui donnent les résultats sont donc une fonction (de fonctions de fonctions de fonctions…) des neurones de la couche d'entrée.

EXEMPLE

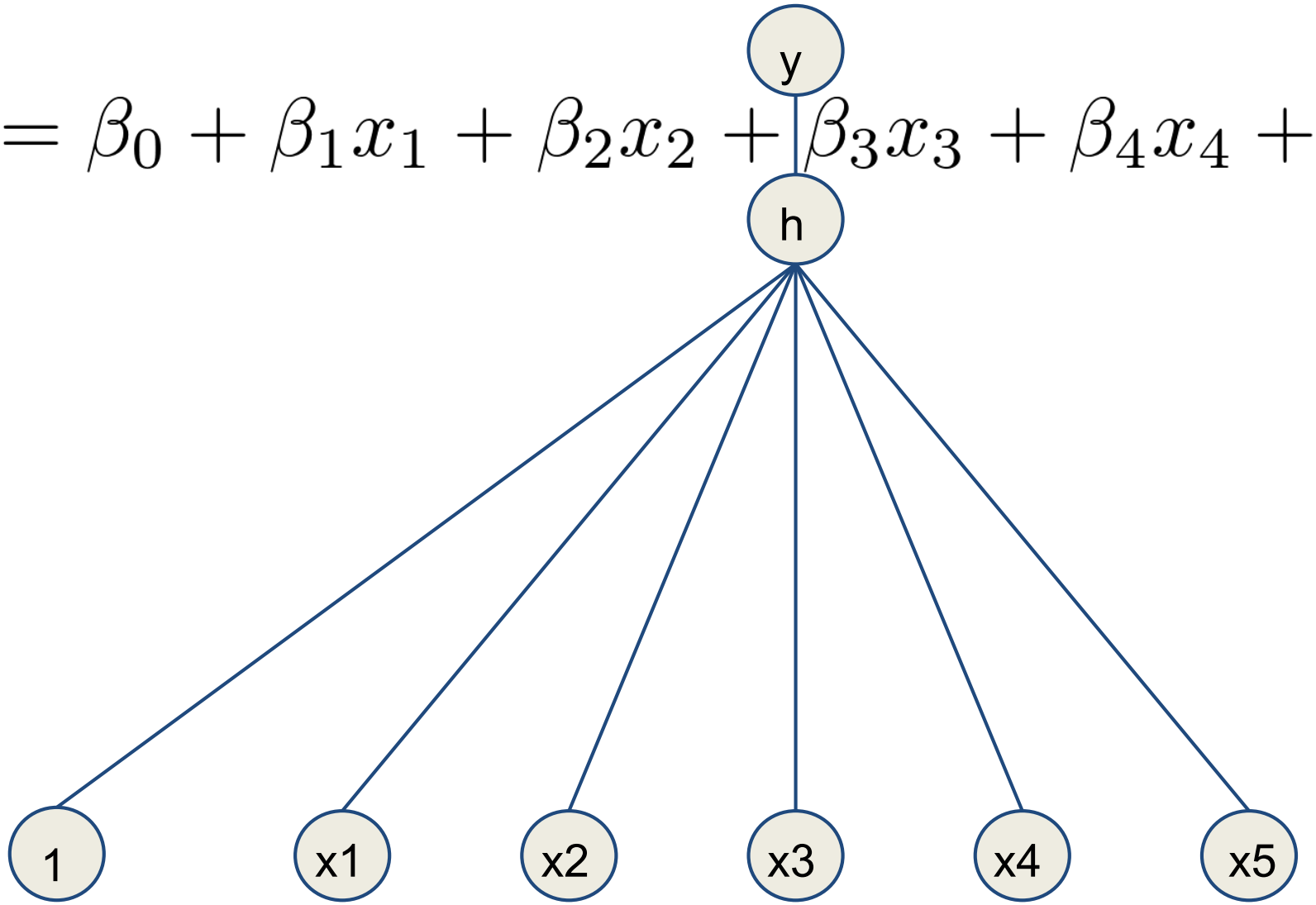
$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5$$



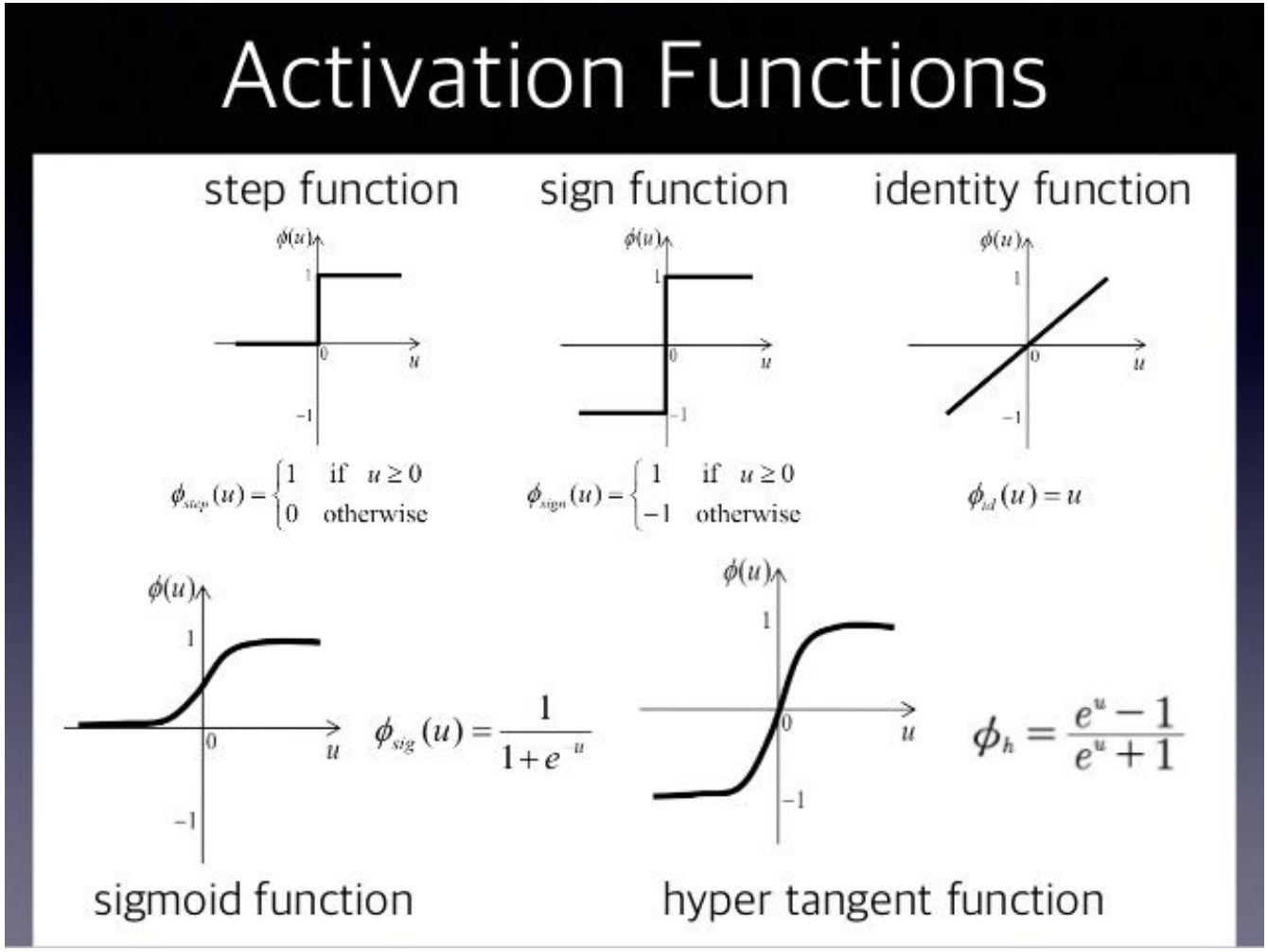
EXEMPLE

$$y = \phi(h)$$

$$h = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5$$



FONCTIONS D'ACTIVATION

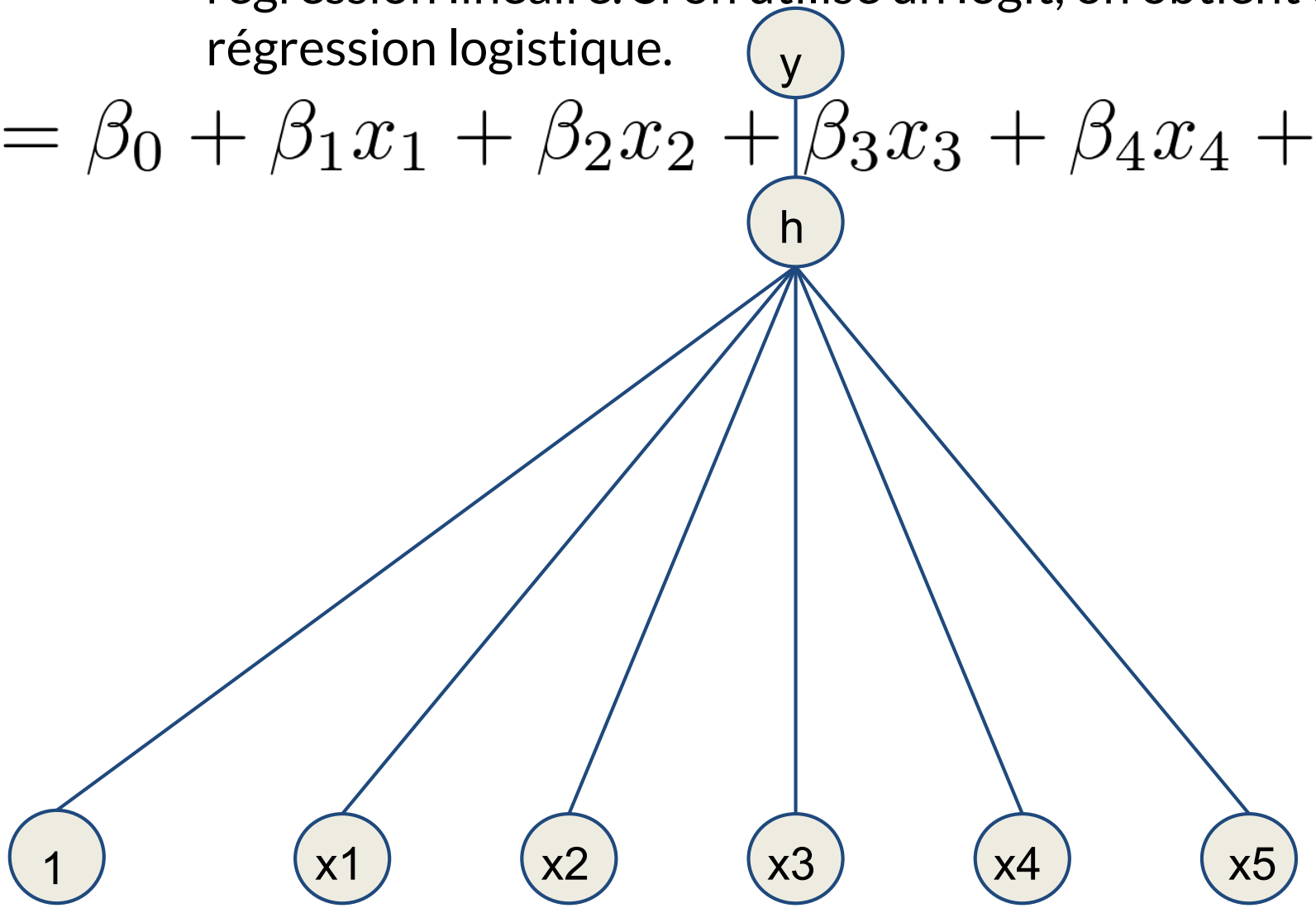


Source: [slideshare, présentation de Sung-ju Kim]

EXEMPLE

$y = \phi(h)$ Si on utilise l'activation linéaire, on obtient une régression linéaire. Si on utilise un logit, on obtient une régression logistique.

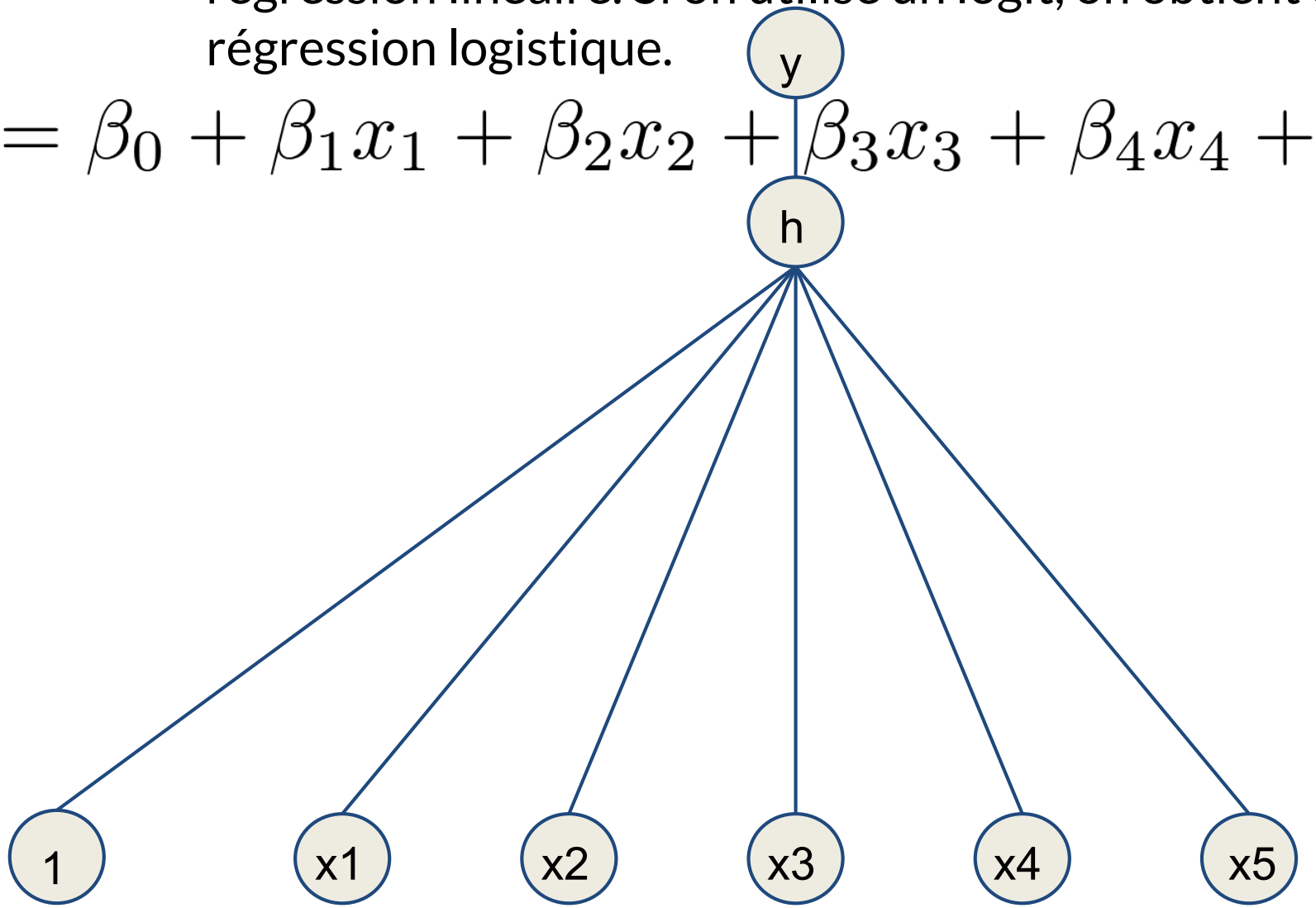
$$h = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5$$



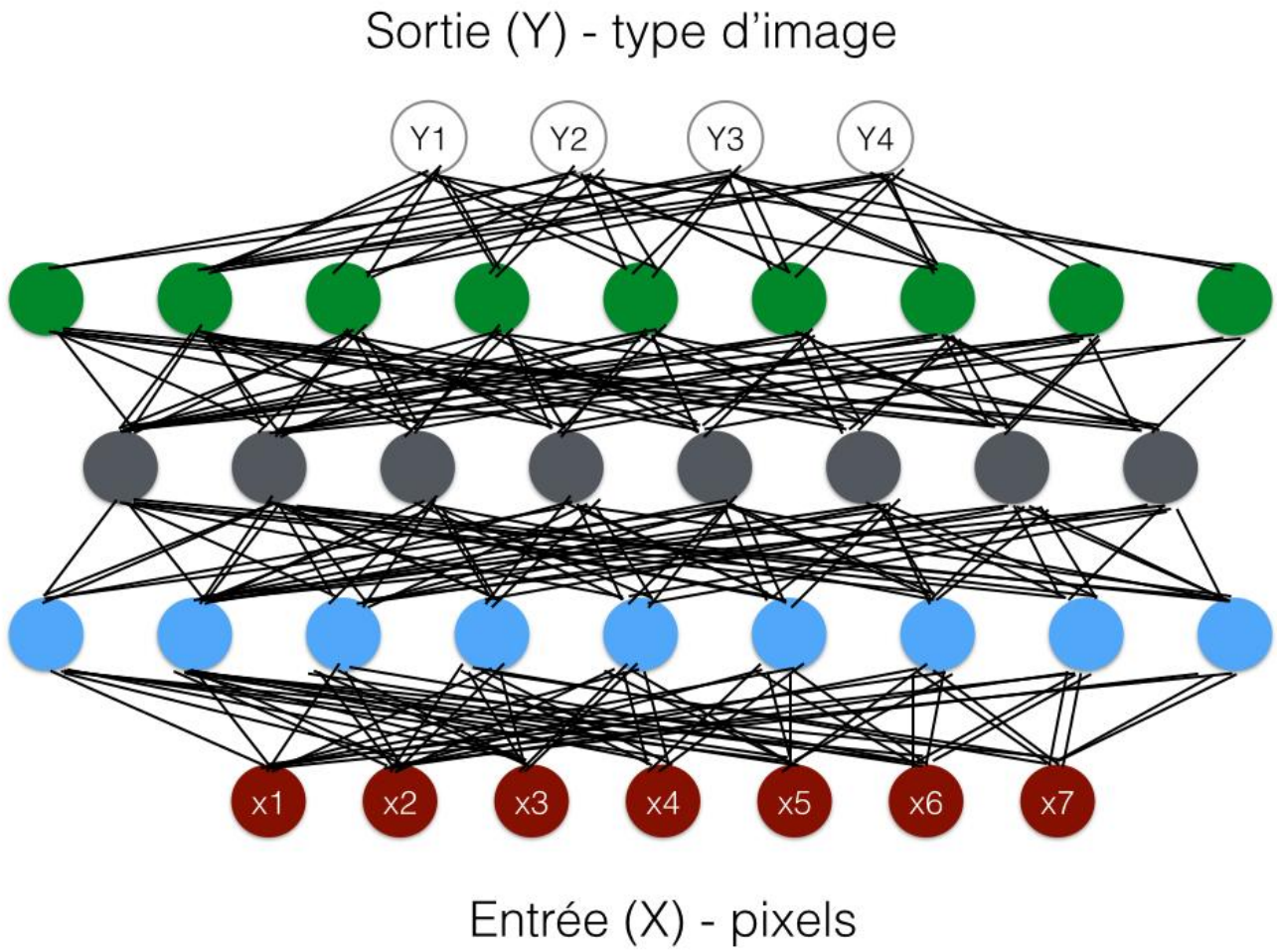
EXEMPLE

$y = \phi(h)$ Si on utilise l'activation linéaire, on obtient une régression linéaire. Si on utilise un logit, on obtient une régression logistique.

$$h = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5$$



PLUS COMPLIQUÉ



PLUS COMPLIQUÉ

Le réseau est totalement connecté: chaque neurone est une fonction de tous les neurones de la couche précédente.
Dans cette fonction, à chaque neurone est associé un paramètre.
La fonction en question est simple: c'est une fonction linéaire des neurones précédents + une fonction d'activation.

$$h_i^{(c)} = \phi(\beta_0 + \beta_1 h_1^{(c-1)} + \beta_2 h_2^{(c-1)} + \beta_3 h_3^{(c-1)} + ...)$$

COMBIEN DE PARAMÈTRES?

Vous avez un réseau qui contient:

- 1000 neurones en couche d'entrée (les poids TF-IDF par exemple, ou des pixels)
- 100 neurones en couche cachée
- 4 neurones en couche de sortie (pour un problème de classification à 4 classes possibles).

Combien de poids avez-vous en tout ?

COMBIEN DE PARAMÈTRES?

Vous avez un réseau qui contient:

- 1000 neurones en couche d'entrée (les poids TF-IDF par exemple, ou des pixels)
- 100 neurones en couche cachée
- 4 neurones en couche de sortie (pour un problème de classification à 4 classes possibles).

Combien de poids avez-vous en tout ?

Le réseau est totalement connecté. Vous avez donc:

(1000 + 1) x 100 + (100 + 1) x 4 = 100504 poids

RÉTRO-PROPAGATION

Pour trouver la valeur de ces poids, on est à nouveau sur un problème d'optimisation.

Au tout départ, les poids sont choisis aléatoirement.

Si le réseau n'a pas de couche cachée, la fonction est convexe => facile car un seul minimum.

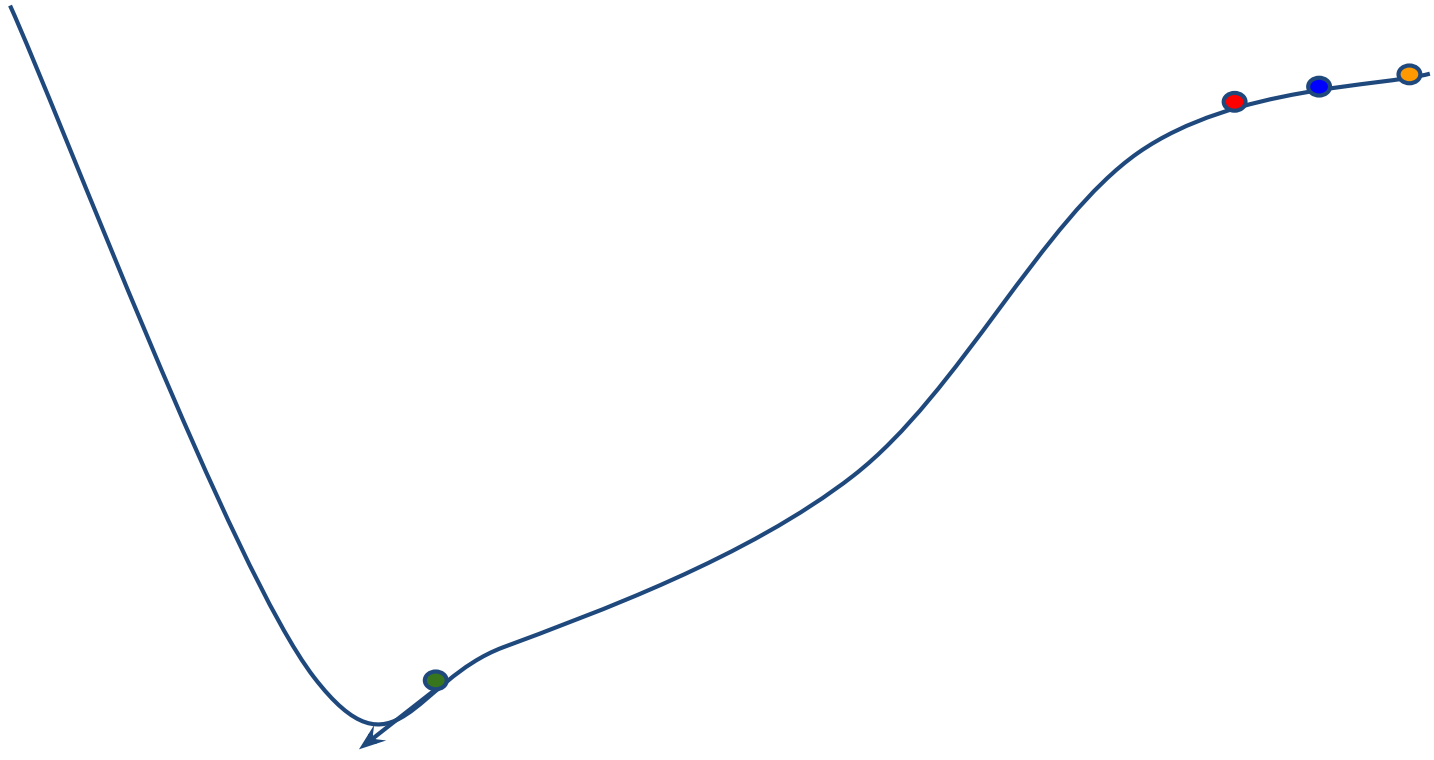
Dès qu'il y a une couche cachée, la fonction n'est plus nécessairement convexe => minima locaux!

RÉTRO-PROPAGATION

On utilise ce qu'on appelle la rétro-propagation des erreurs. C'est surtout une descente de gradient.
Idée: on a un exemple en entrée, on le fait passer dans le réseau pour obtenir une prédiction. On compare cette prédiction avec la vraie valeur de y pour notre exemple. La différence est l'erreur.
On va, avec cette erreur, redescendre dans le réseau en calculant la dérivée de l'erreur par rapport à chaque poids et en corrigeant ce poids dans la direction de son gradient.

DESCENTE DE GRADIENT

Idée de la descente de gradient: faire des petits pas dans la direction de la pente jusqu'à atteindre le minimum.



DESCENTE DE GRADIENT

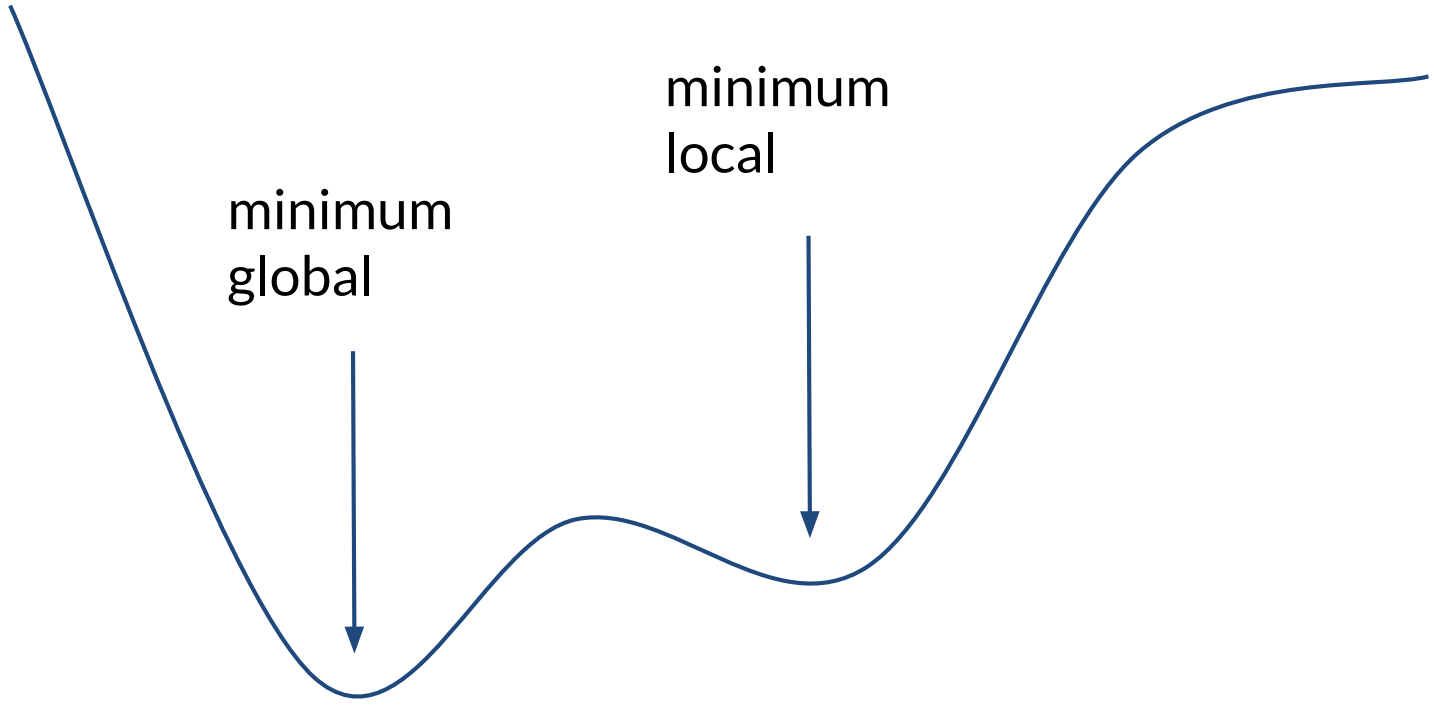
Idée de la descente de gradient: faire des petits pas dans la direction de la pente jusqu'à atteindre le minimum.

La taille des pas est ce qu'on appelle le taux d'apprentissage ou encore learning rate. Il nous dit à quel point ce que nous venons de voir doit influencer la modification des poids.

Il est généralement noté η.

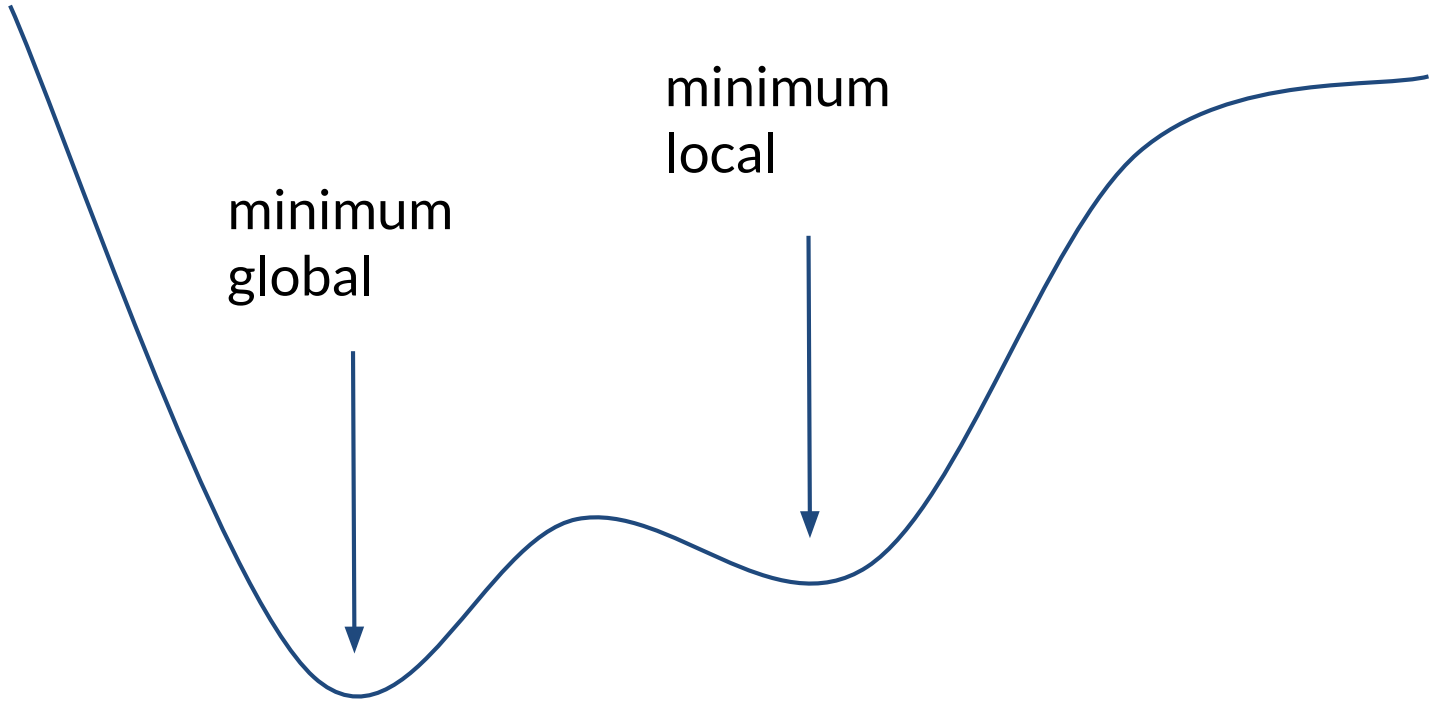
DESCENTE DE GRADIENT

Rappel: notre fonction n'est pas convexe!
Si le learning rate est trop petit, on peut rester bloqué dans un minimum local.
S'il est trop grand, on peut passer à côté du minimum global.
Aie.



DESCENTE DE GRADIENT

Solution: ne pas choisir un learning rate fixe. Par exemple, on peut choisir 1/n où n est le nombre d'observations. La première observation modifiera beaucoup le réseau, la dernière l'affinera seulement.



EXEMPLE

Je reçois une **nouvelle** image, je **prédis** grâce à mon réseau qu'il s'agit d'un chat.
Une fois au niveau de la **sortie**, je m'aperçois que c'est en fait un camion.
Je **rétro-propage** cette information pour **updater** la valeur de mes (millions de) paramètres.
Plus le réseau voit d'exemples labélisés, plus il emmagasine de connaissances et devient "bon".

PARAMÈTRES GLOBAUX

Il faut choisir un certain nombre de choses pour son réseau:

La complexité:

Le nombre de **couches** : plus elles sont nombreuses, plus on pourra prédire des choses compliquées, plus le temps de calcul est long. Si elles sont trop nombreuses, on tombe dans le **sur-apprentissage**.

Le nombre de **neurones** : idem !

Les **fonctions** d'activation et de sortie: on en propose un certain nombre. Elles dépendent en partie du type de sortie (par exemple : classification binaire vs multi-classes)

Le **learning rate** : le taux d'apprentissage. S'il est trop élevé, chaque exemple va influer beaucoup sur le réseau et on peut tomber dans un optimum **local**. S'il est faible, le réseau va être très long à optimiser. On choisit souvent un paramètre dynamique : il baisse au fur et à mesure de l'apprentissage.

BATCH ET ONLINE LEARNING

Bien que l'on ait accès à de puissants serveurs de calculs, on peut vouloir limiter l'utilisation de la mémoire pour entraîner ces algorithmes très lourds.

On peut alors envisager de ne fournir les données au réseau que **par batchs**. Ainsi, il ne traite qu'une seule partie des données à la fois, sans avoir besoin de stocker les autres. Il update les paramètres à la fin de chaque batch.

Afin que le réseau continue d'apprendre et s'améliore **au fur et à mesure**, on peut utiliser le **online learning** : le réseau reste en phase d'apprentissage "toute sa vie" et les paramètres sont updatés à chaque nouvel exemple.

RECONNAISSANCE D'IMAGES

L'avantage des réseaux de neurones est que l'on n'a pas besoin de **pré-processer** les entrées : c'est lui qui va créer les patterns/features intéressantes.

Dans le cas de la reconnaissance d'images, on peut ne lui donner en entrée que des pixels et non des résumés de l'image obtenus manuellement (handcrafted).

Problème: une image contient au moins 1M de pixels. Le problème est souvent compliqué donc il faut un réseau profond. Le nombre final de poids va vite avoisiner les 10^10. Or on a souvent bien moins d'images que ça!

RECONNAISSANCE D'IMAGES

Problème: une image contient au moins 1M de pixels. Le problème est souvent compliqué donc il faut un réseau profond. Le nombre final de poids va vite avoisiner les 10^10. Or on a souvent bien moins d'images que ça !

La solution va être de régulariser le réseau, de manière similaire à la régularisation des SVMs.

Cours prochain, avec les réseaux convolutifs.

CONCLUSION SUR LE PERCEPTRON MULTI-COUCHES

Avantages :

Potentiellement extrêmement performants, en particulier dans certains domaines comme la reconnaissance d'images ou du language.

Flexibles sur la complexité.

Inconvénients :

Peu/pas de résultats théoriques pour expliquer la performance.

Impossibles à interpréter.

Calculs extrêmement lourds.

Complexité difficile à calibrer : gros risques de **sur-apprentissage**.