Apprentissage automatique et IA générative

Master DIMI · Design d’interface multimédia et internet

Contenus

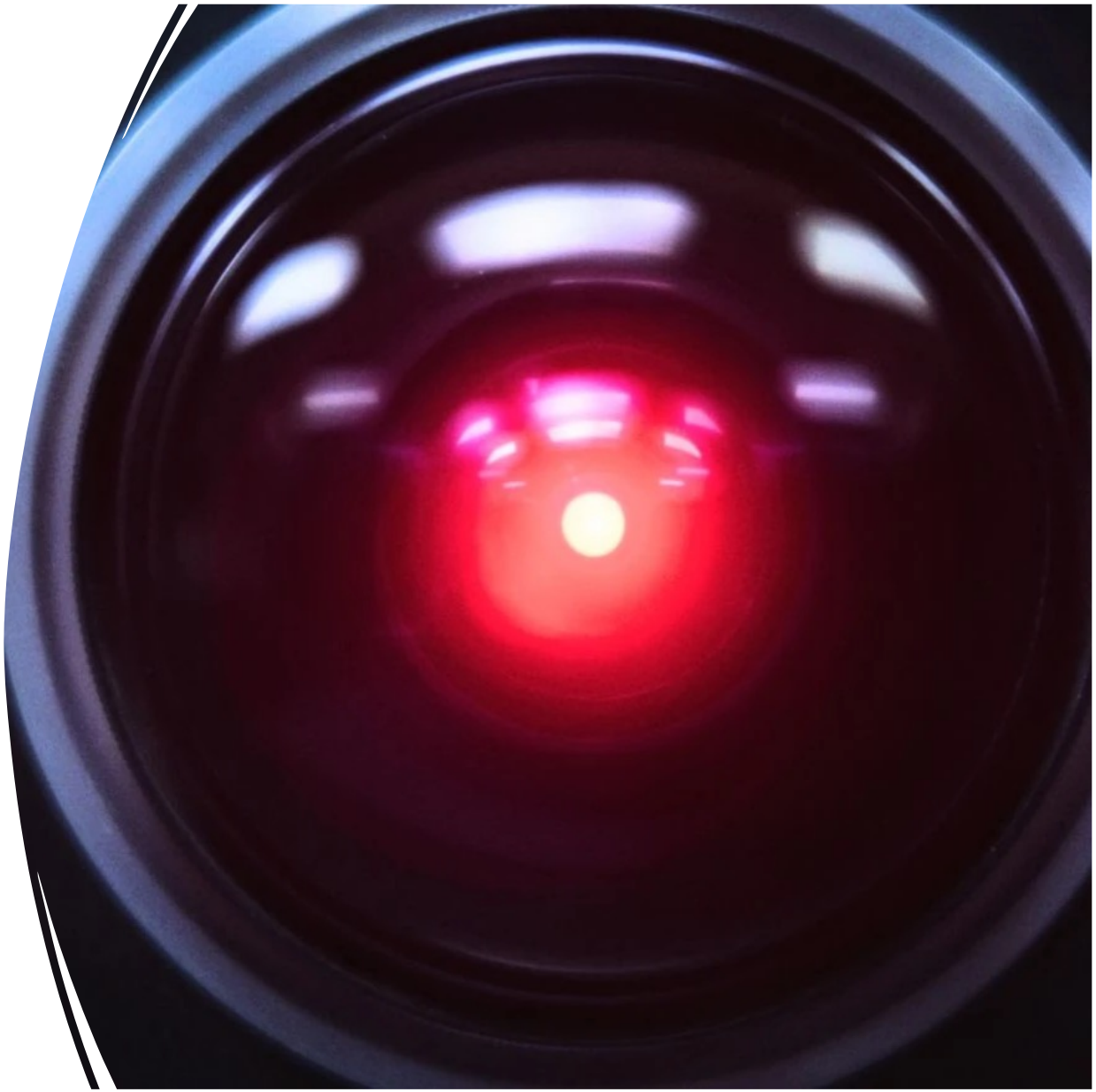
1. Introduction : IA et apprentissage automatique
2. Apprendre à partir d'exemples : régression, classification, k-NN
3. Combiner des modèles : arbres de décision, forêts aléatoires
4. Séparer les données : SVM (l'idée)
5. Réseaux de neurones
6. Réseaux de convolution : comment une machine « voit » une image
7. Bien entraîner un modèle : surapprentissage et régularisation
8. Séquences : réseaux récurrents
9. Représentations : embeddings, auto-encodeurs, GANs
10. IA générative : modèles de langage et génération de code
11. Génération d'images
12. IA et design d'interface : enjeux et bonnes pratiques

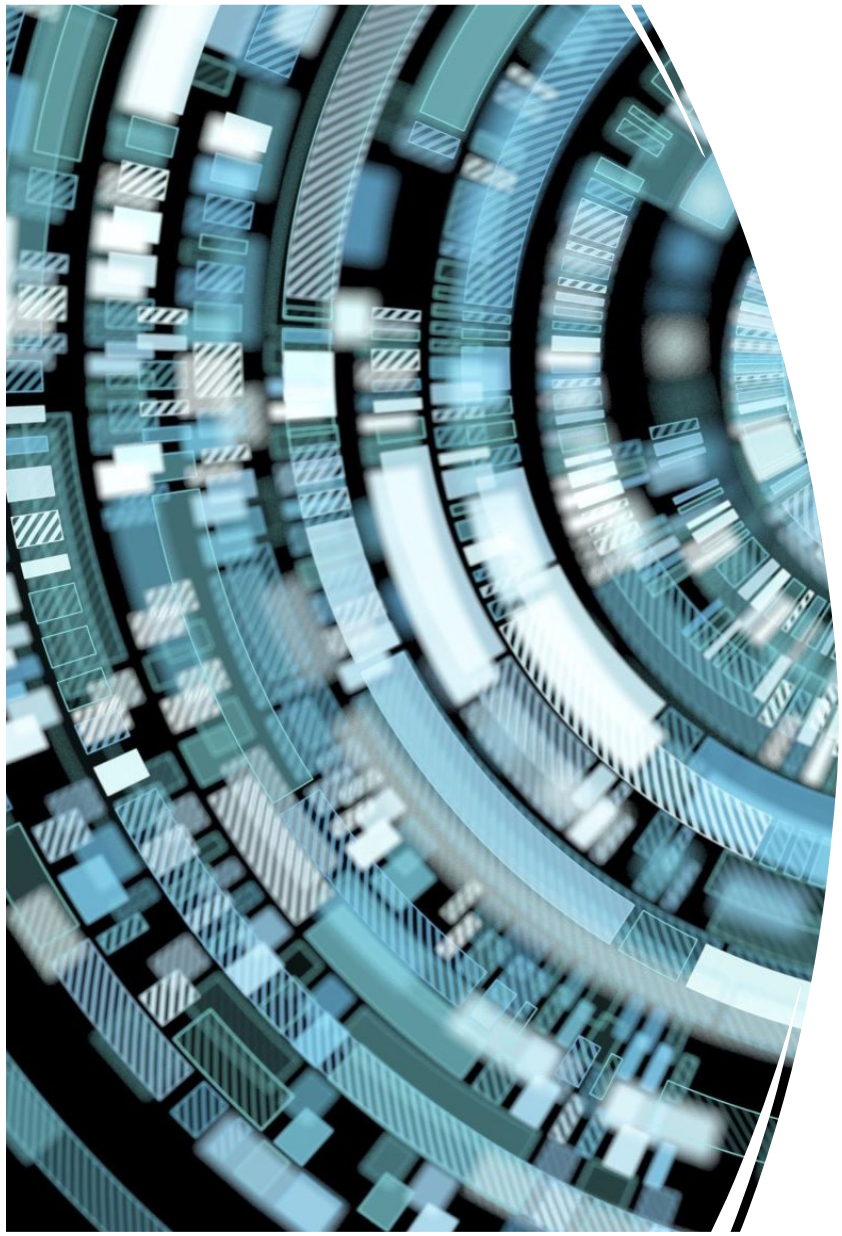
1. Introduction

Intelligence Artificielle et Apprentissage Automatique

C’est quoi l’Intelligence Artificielle?

- – Modèles et méthodes capables de simuler l’intelligence humaine
  - A.Turing “Computing Machinery and Intelligence” (1950)
    - “Can machines *think*”?
    - Test de Turing
  - Lien très fort (perçu) entre capacité linguistique et intelligence





IA, Apprentissage Automatique, Deep Learning…?

- IA n’est pas apprentissage
- IA n’est pas “Deep Learning”

- Apprentissage Automatique: (un) moyen pour atteindre l’IA
- Deep Learning: un type de modèle d'Apprentissage Automatique basé sur les réseaux neuronales

Les étapes clé

- 1950 : Alan Turing propose le « jeu de l'imitation » (test de Turing)
- 1957 : Frank Rosenblatt invente le **perceptron**, premier neurone artificiel
- 1998 : LeNet (Yann Le Cun) lit les chiffres manuscrits des chèques
- 2012 : AlexNet, un réseau de neurones gagne le concours de reconnaissance d'images ImageNet
- 2013 : Word2Vec, des mots transformés en vecteurs « de sens »
- 2014 : GANs, premiers réseaux capables de générer des images réalistes
- 2017 : le **Transformer**, base de tous les modèles de langage actuels
- 2021 : DALL·E et CLIP, du texte vers l'image
- 2022 : Stable Diffusion, Midjourney, puis **ChatGPT**
- 2023 à aujourd'hui : modèles multimodaux (texte, image, son, vidéo) et génération d'interfaces et de code

Apprentissage Automatique

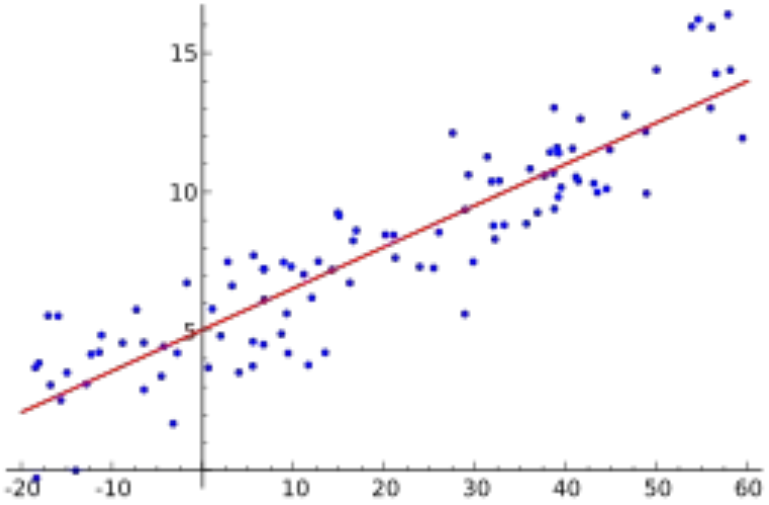
- Tom Mitchell (1998): Un programme informatique “apprend” d’une expérience E par rapport à une tâche T et une mesure de performance P si sa performance sur T mesurée par P s’améliore avec l’expérience E
- Exemple: prédiction de spam
  - T: classifier les e-mails en spam/pas spam
  - E: un ensemble d’e-mails annotés spam/pas spam par un utilisateur
  - P: pourcentage d’e-mails correctement identifiées en tant que spam

Apprentissage Supervisé

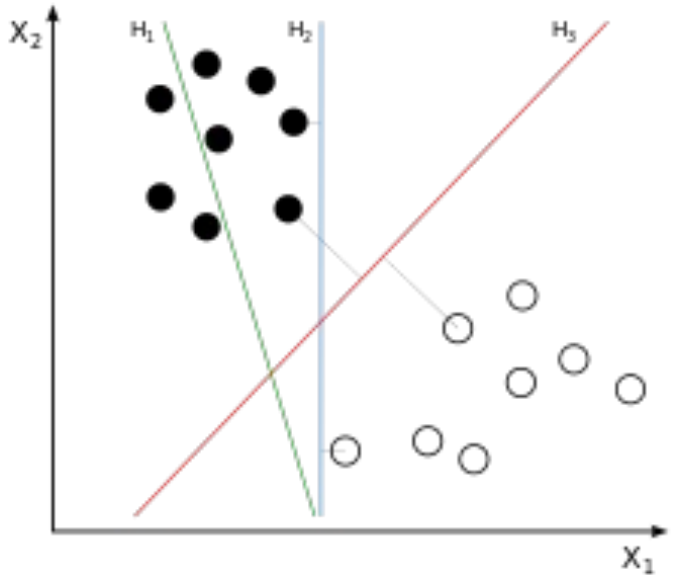
- On apprend à partir d'exemples **étiquetés** : chaque exemple vient avec la « bonne réponse »
- Le modèle cherche une règle qui relie les données (x) à la réponse (y)
- Ensuite, il applique cette règle à des données nouvelles
- **Exemple :** 10 000 photos étiquetées « chat » ou « chien » ; le modèle reconnaît ensuite une photo qu'il n'a jamais vue
- **Exemple design :** des milliers de maquettes notées « lisible / pas lisible » ; prédire la lisibilité d'une nouvelle maquette

Régression ou Classification?

- étiquette réelle: **régression**
  - exemple: mm de pluie

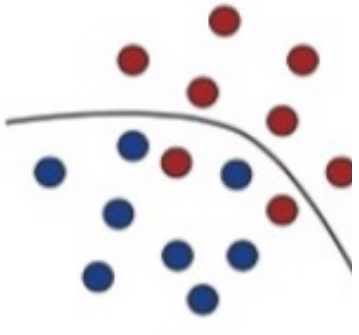


- étiquette discrète: **classification**
  - exemple: pluie/pas pluie



Classification

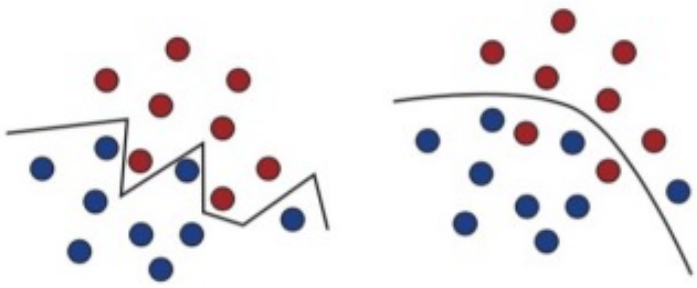
- Chaque point = un exemple (ici : un visiteur d'un site)
- Couleur = la classe (rouge : s'est abonné ; bleu : ne s'est pas abonné)
- Position = ses **caractéristiques** (features), ex. temps passé sur le site et nombre de pages vues
- Objectif : apprendre la frontière qui sépare les deux couleurs, pour prédire ce que fera un nouveau visiteur

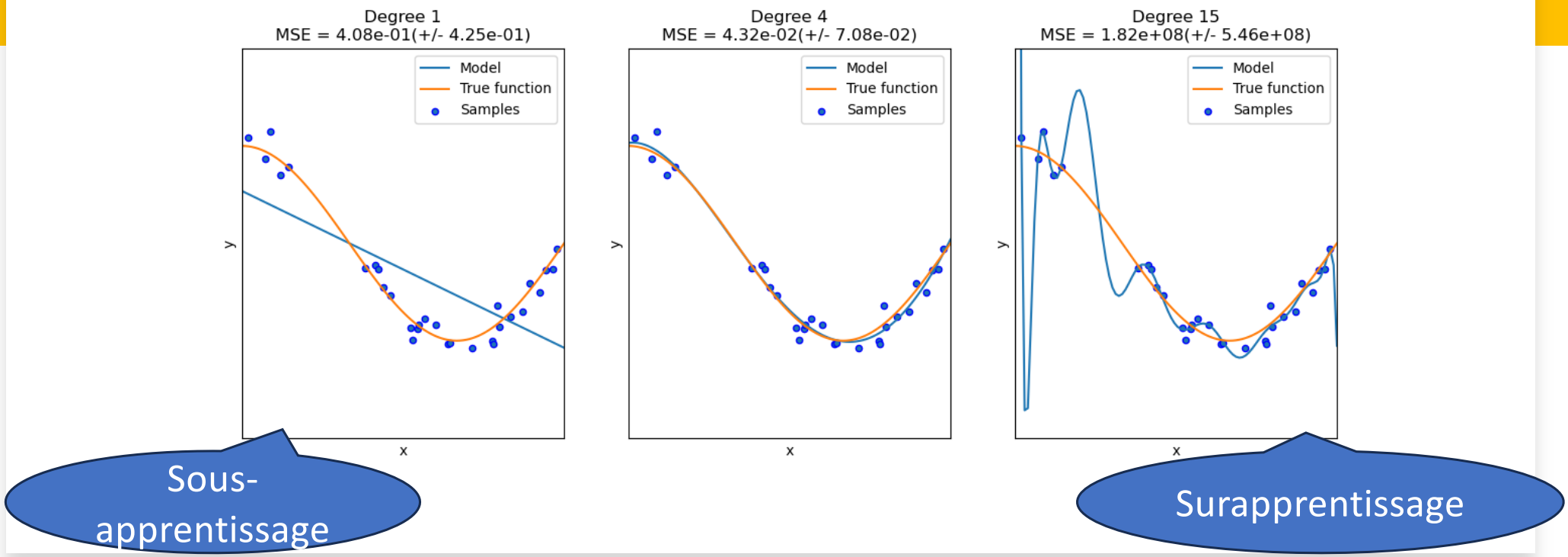


Fonction objectif (Loss)

- Mesure « à quel point le modèle se trompe »
- Apprendre = réduire cette erreur petit à petit
- **Exemple :** le modèle prédit 3 mm de pluie, il en tombe 5 : erreur de 2
- **Lecture de la formule :** pour chaque exemple i, on calcule l'écart (réel − prédit), on le met au carré, puis on additionne
- Attention : une erreur nulle sur les exemples ne garantit pas un bon modèle ; il a pu apprendre « par cœur » (à droite)

$$E = \sum_{i=1}^{k} (y(i) - y'(i))^2$$





Dilemme Biais-Variance

- Sous-apprentissage : modèle trop simple, il rate la tendance
- Surapprentissage : modèle trop complexe, il apprend aussi le bruit
- Comme un étudiant qui mémorise l'examen blanc sans comprendre le cours

Test/Validation

- On garde une partie des données **cachée** pendant l'apprentissage, pour vérifier ensuite
- **Hold-out :** par ex. 80 % pour apprendre, 20 % pour tester
- **Validation croisée :** on découpe en k blocs et on teste à tour de rôle sur chaque bloc
- Analogie : on ne note pas un étudiant sur les exercices déjà corrigés en TD

$\mathscr{S}_1$	$\mathscr{S}_2$	$\mathscr{S}_3$	$\mathscr{S}_4$	$\mathscr{S}_5$	
A	A	A	A	T	→ coût d'erreur sur T : R_1
A	A	A	T	A	→ coût d'erreur sur T : R_2
A	A	T	A	A	→ coût d'erreur sur T : R_3
A	T	A	A	A	→ coût d'erreur sur T : R_4
T	A	A	A	A	→ coût d'erreur sur T : R_5

Erreur de validation croisée : $\frac{1}{5}(R_1 + R_2 + R_3 + R_4 + R_5)$

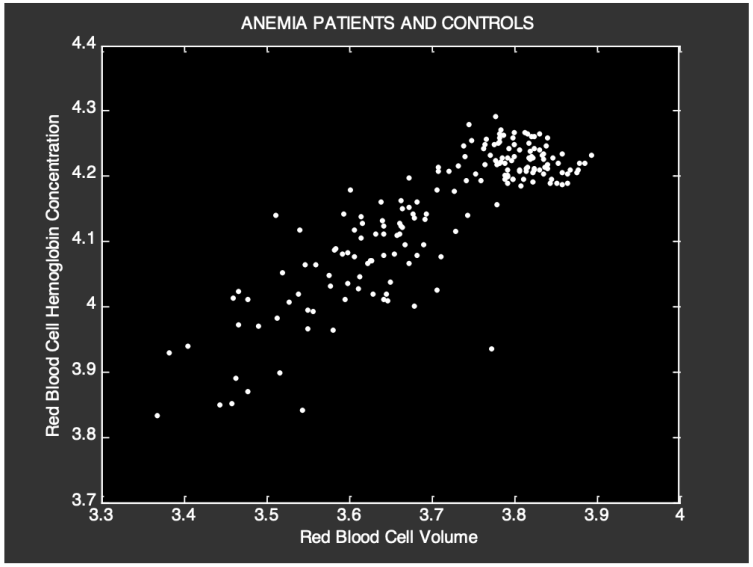
Apprentissage non-supervisé

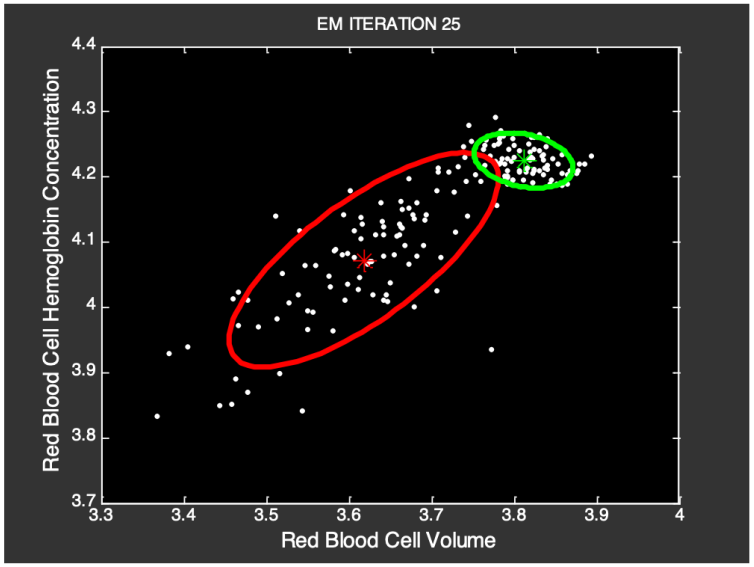
- Données **sans étiquette** : personne n'a donné la « bonne réponse »
- Le modèle trouve tout seul des groupes ou des structures
- Usages : regrouper (clustering), résumer des données très riches (réduction de dimension)

Exemple

On regroupe les visiteurs d'un site selon leur navigation. Trois profils apparaissent : « explorateurs », « acheteurs pressés », « comparateurs

Clustering : l'algorithme E-M





Images from P. Smyth, ICS278, slides

Régression : prédire un nombre

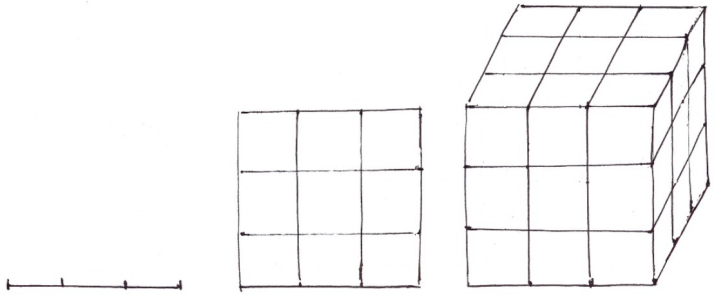
- Prédire une **valeur numérique** à partir d'autres valeurs
- On cherche la droite qui passe « au plus près » de tous les points
- Cette droite est une règle simple : prix = a × surface + b
- a et b sont appris à partir des données
- On choisit a et b qui rendent la plus petite possible l'erreur $\Sigma (y_i - (a \cdot x_i + b))^2$ (méthode des moindres carrés)

Exemple

Prix d'un appartement selon sa surface : 30 m² à 250 k€, 60 m² à 480 k€, etc. Le modèle apprend « environ 8 k€ par m² » et peut estimer le prix d'un 45 m² qu'il n'a jamais vu.

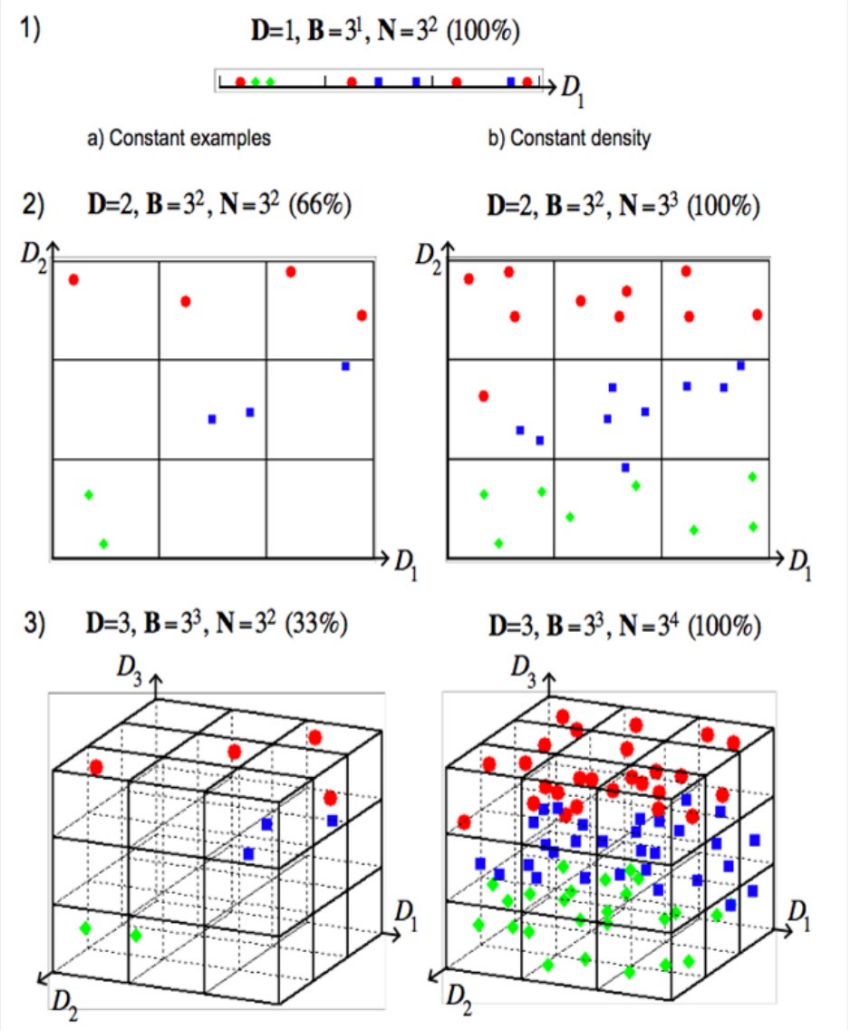
La malédiction de la dimension («curse of dimensionality»)

- La question de base: de combien de données j'ai besoin pour « apprendre » correctement?
- Problème relatif aux propriétés des espaces de grande dimension
- Relation entre nombre de dimensions et taille des échantillons d'apprentissage
- Exemple: estimation d'une distribution de probabilité à partir d'un échantillon de m points avec n^k caractéristiques: k=1, 2, 3



Curse of dimensionality (exemple)

- 3 classes
  - Même nombre d'observations plongées dans des dimensions croissantes
  - Augmentation exponentielle du nombre de points nécessaires pour garder la même densité
  - Augmentation de la probabilité de trouver des grandes portions de l'espace des variables vide



Régression avec la méthode des moindres carrées (Ordinary Least Squares)

- $f_w(\boldsymbol{x}) = b + \boldsymbol{w}\boldsymbol{x}$
  - $b \in \mathbb{R}$ « bias » ou intercept
  - $\boldsymbol{w} \in \mathbb{R}^d$ paramètres
- Les paramètres b et w sont obtenus par la méthode des moindres carrés
- En minimisant la somme des carrés des résidus du modèle sur la base d'apprentissage:$\sum_{n=1}^{N} |y_n - f_w(\boldsymbol{x})|^2$
- Si x est un scalaire et **x** = (x,x^2 ,...,x^p) est constitué de puissances de x (jusqu'au degré p > 0), on voit que la régression polynomiale apparaît comme un cas particulier de la régression linéaire multivariée

Régression avec la méthode des moindres carrées (Ordinary Least Squares)

$$W = \begin{pmatrix} b \\ \boldsymbol{w} \end{pmatrix}, \quad Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}, \quad X = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1d} \\ 1 & x_{21} & x_{22} & \dots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & x_{N2} & \dots & x_{Nd} \end{pmatrix}$$

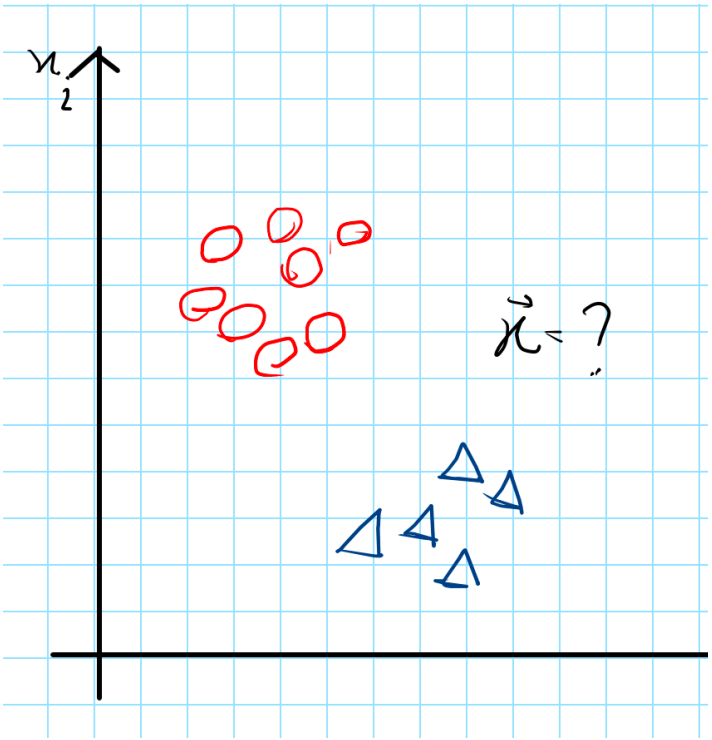
Proposition 1.1 *Avec ces notations, on obtient l'estimation de W au sens des moindres carrés par les équations dites normales :*

$$W = (X^T X)^{-1} X^T Y$$

à la condition que $X^T X$ soit une matrice inversible.

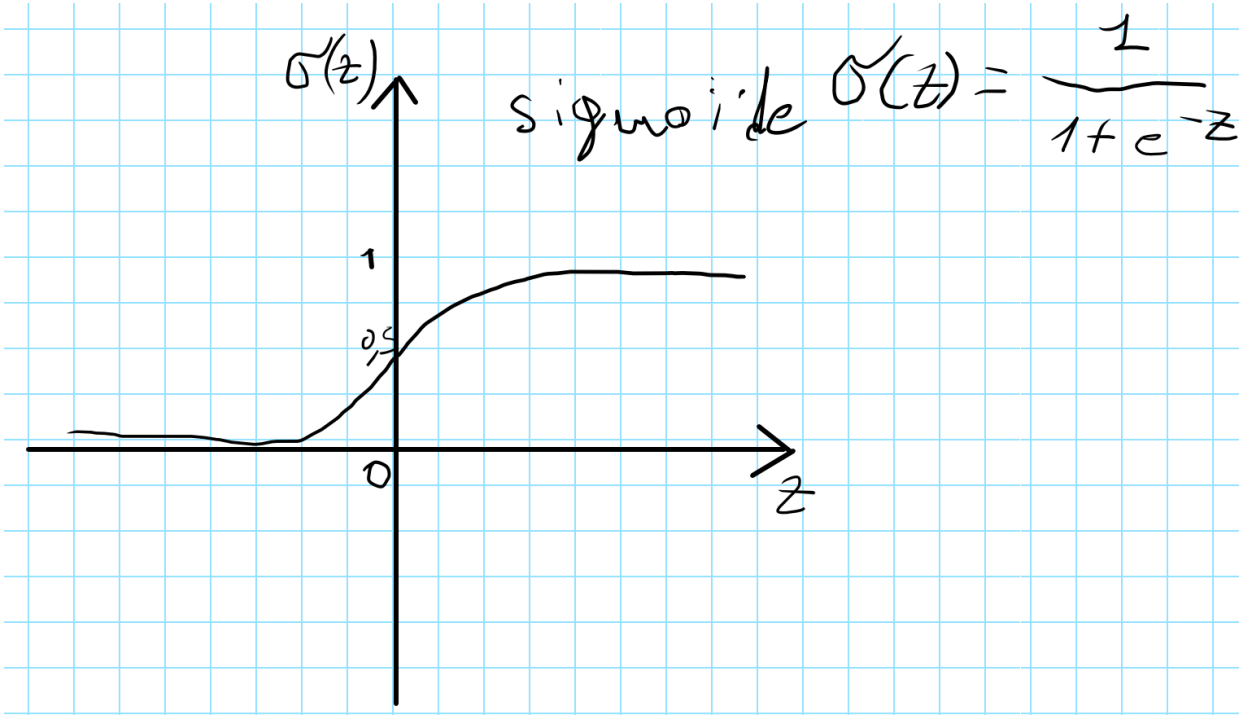
Régression Logistique

- Même idée que la régression, mais la réponse est **oui ou non**
- Le modèle calcule d'abord f(x) = b + w·x
- **Exemple :** l'utilisateur va-t-il cliquer sur ce bouton ?
- Problème : le modèle produit un nombre quelconque (-3, 12, 0,4…)
- Comment transformer ce nombre en « oui » ou « non » ?



Régression logistique : la sigmoïde

- La **sigmoïde** « écrase » n'importe quel nombre entre 0 et 1
- Formule : $\sigma(z) = 1 / (1 + e^{-z})$
- On l'interprète comme une probabilité : $P(oui) = \sigma(b + w \cdot x)$
- Plus de 0,5 : « oui » ; sinon : « non »
- **Exemple :** probabilité de clic = 0,82, donc on prédit « clic »



Régression Logistique - Likelihood

- Donc nous pouvons écrire:
  - La probabilité que X soit de classe Y=1: $\sigma(b + \boldsymbol{wx})$
  - La probabilité que X soit de classe Y=0: $1 - \sigma(b + \boldsymbol{wx})$
  - Par conséquence, la probabilité que X soit de classe Y = 0 ou 1:
    - $\sigma(b + \boldsymbol{wx})^{y} \cdot \ (1 - \sigma(b + \boldsymbol{wx}))^{1-y}$
- Avec cela, nous pouvons calculer la « likelihood » sur la totalité du dataset:
  - $\prod_i \sigma(b + \boldsymbol{wx_i})^{\boldsymbol{y_i}} \cdot \ (1 - \sigma(b + \boldsymbol{wx_i}))^{1-y_i}$
- Mais les produit de plusieurs probabilités n'est pas numériquement stable, donc on préfère calculer la « **log-likelihood** »:
  - $LL = \ \sum_i y_i \sigma(b + \boldsymbol{wx_i}) \cdot (1 - y_i)(1 - \sigma(b + \boldsymbol{wx_i}))$

Régression Logistique - optimisation

- Le coût devient E(w) = -LL(w)
- Mais nous ne pouvons pas trouver les paramètres **w** de façon analytique
- Donc nous utilisons la **descente de gradient**

Descente de gradient

- Comment le modèle trouve-t-il ses bons paramètres ?
- On part au hasard, puis on ajuste petit à petit pour réduire l'erreur
- À chaque pas, on va dans la direction où l'erreur baisse le plus
- Le **learning rate** = la taille du pas : trop grand, on saute par-dessus la vallée ; trop petit, on avance très lentement
- **Lecture de la formule** (à droite) : nouveaux paramètres θ = anciens θ − η × pente de l'erreur ∇J(θ)

Analogie

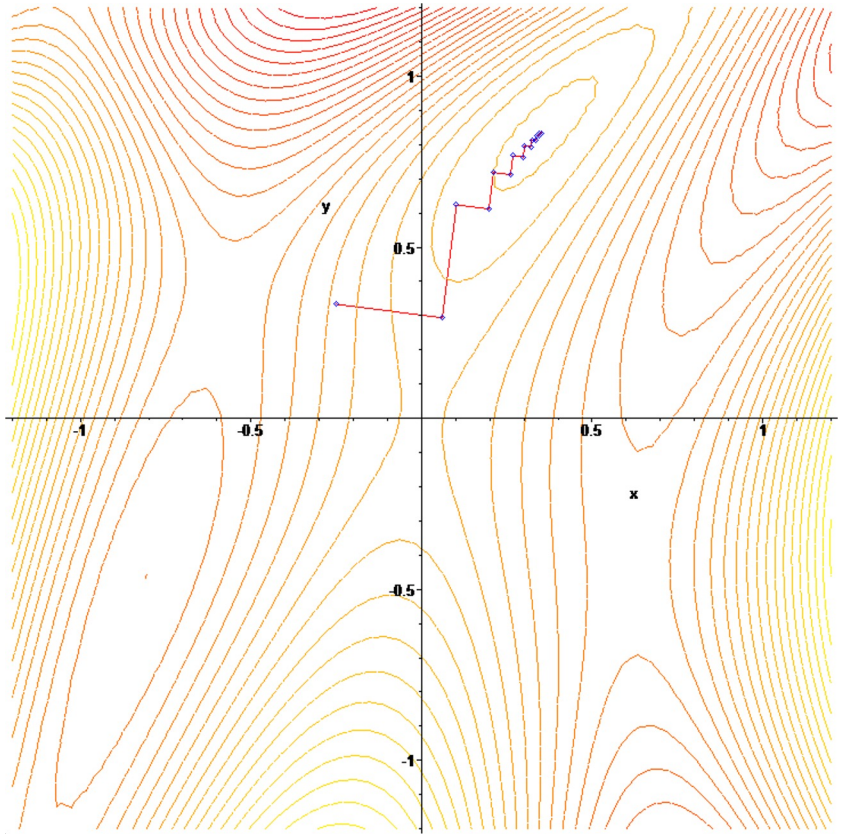
Descendre une montagne dans le brouillard : on ne voit pas la vallée, mais on sent la pente sous ses pieds. On fait un pas vers le bas, puis un autre, puis un autre…

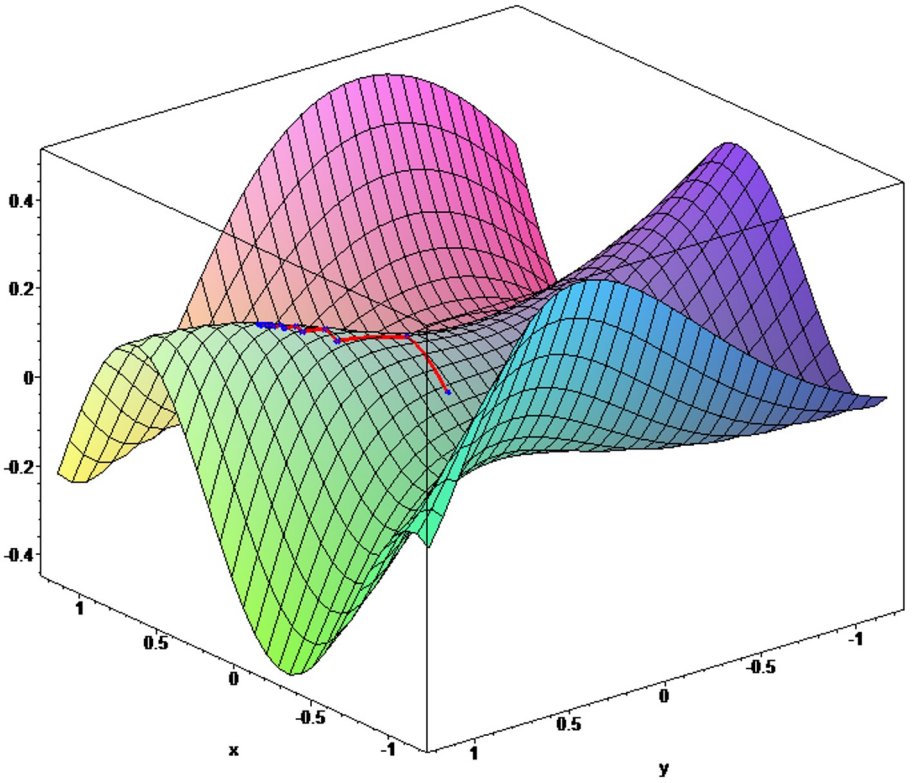
$$\theta = \theta - \eta \nabla_\theta J(\theta)$$

Descente de gradient

- Minimiser une fonction réelle différentiable définie sur un espace euclidien
- Supposons ***J(ϑ)*** l'erreur de la fonction objective, paramètres **θ**.
- Trouver θ qui minimise J(θ) :
  - mettre à jour les paramètres dans le sens inverse du gradient de la fonction objectif : $\nabla_\vartheta J(\vartheta)$
- **η** *learning rate* (pas)

$$\theta = \theta - \eta \bigtriangledown_\theta J(\theta)$$





Descente de gradient

k-NN

- Idée très simple : « dis-moi qui sont tes voisins, je te dirai qui tu es »
- Pour un nouvel exemple, on regarde les k exemples les plus proches
- Classification : vote de la majorité des voisins
- Régression : moyenne des valeurs des voisins
- Pas de vraie phase d'apprentissage : on garde tous les exemples en mémoire

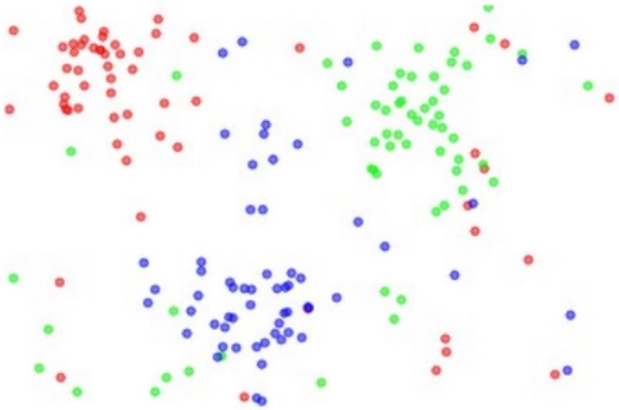
k-NN : un exemple de recommandation

- Un nouvel utilisateur a aimé 3 films
- On cherche les 5 utilisateurs aux goûts les plus proches (k = 5)
- On recommande ce que la majorité d'entre eux a aimé
- Choix de k : petit, le modèle est sensible au bruit ; grand, ses décisions deviennent trop « moyennes »

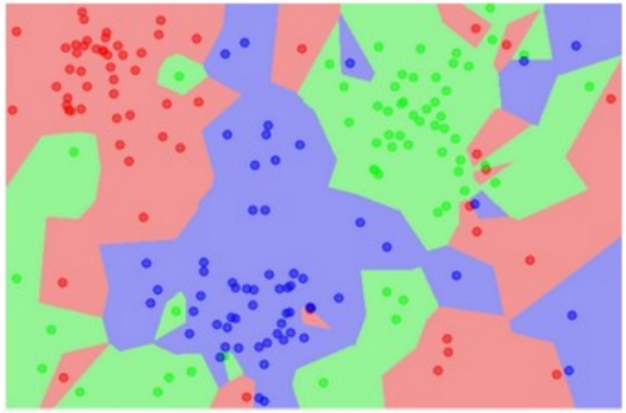
Dans les systèmes de reccomandation

« Les clients qui ont acheté cet article ont aussi acheté… » : c'est le principe des plus proches voisins.

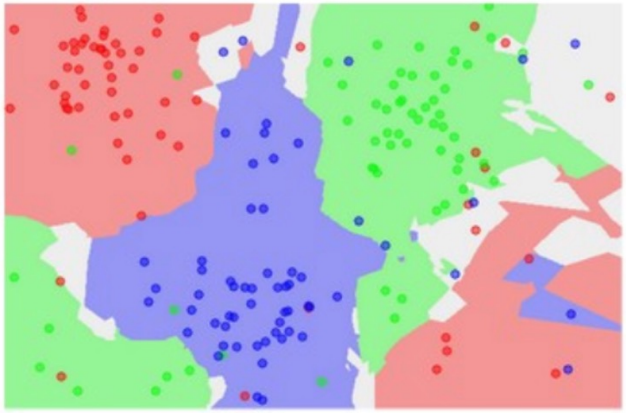
the data



NN classifier



5-NN classifier



2. Combiner des modèles

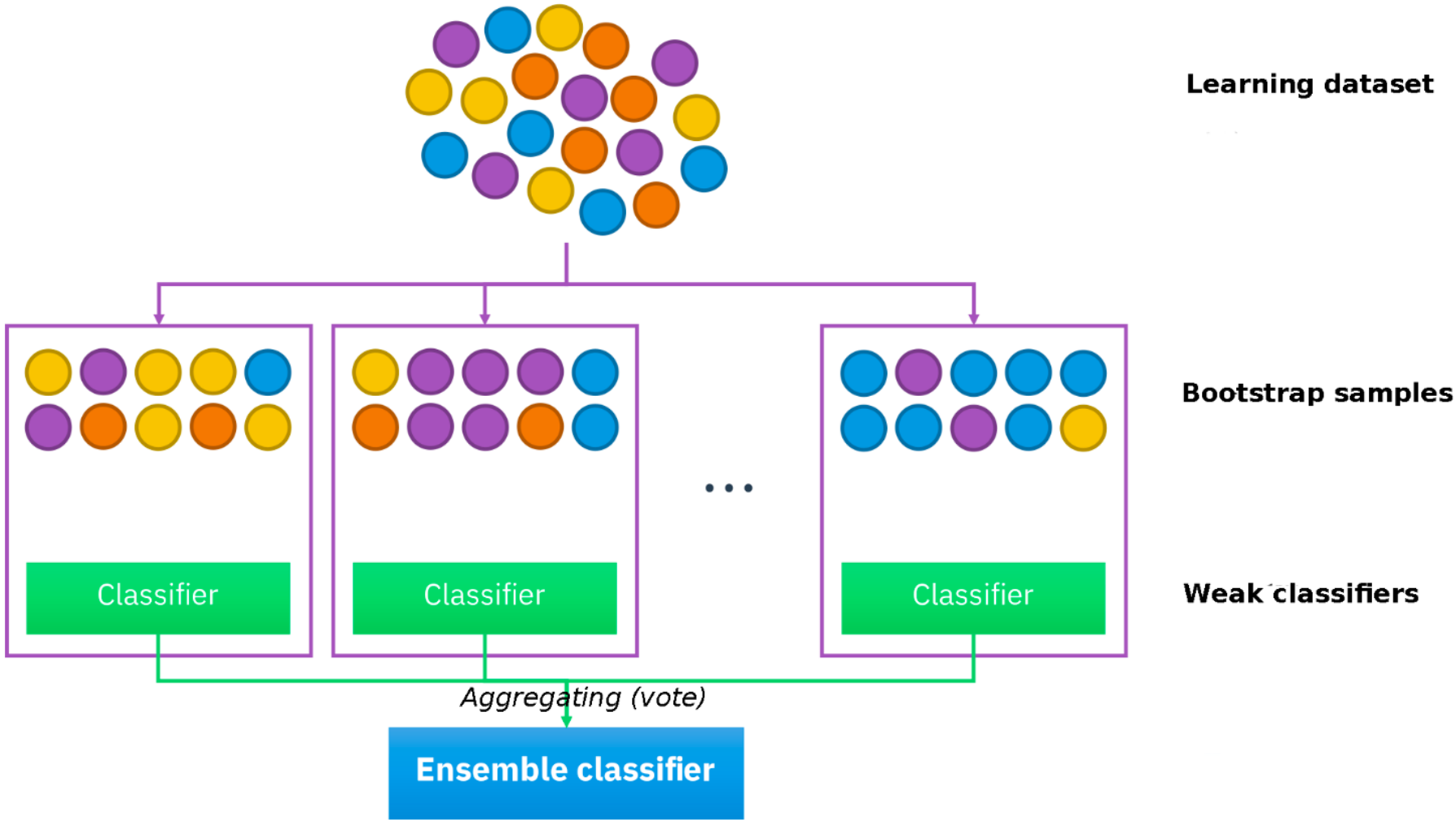
Arbres de décision, bagging, forêts aléatoires, boosting

La sagesse des foules

- Un **classifieur faible** se trompe souvent, mais un peu moins que le hasard
- Idée : en faire voter beaucoup et garder la réponse majoritaire
- Si leurs erreurs sont indépendantes, la majorité se trompe beaucoup plus rarement
- **Exemple :** 20 classifieurs qui se trompent chacun 30 % du temps ; la majorité se trompe moins de 5 % du temps

Bagging

- On entraîne plusieurs modèles, chacun sur un **tirage au hasard** des données
- Tirage « avec remise » : un même exemple peut être tiré plusieurs fois
- Puis on fait voter tous les modèles
- Nom : **B**ootstrap **AGG**regat**ING**


Learning dataset
Bootstrap samples
Classifier
Classifier
Classifier
Weak classifiers
Aggregating (vote)
Ensemble classifier

Classifieurs faibles

- Considérons un problème de classification biclasse, pour lequel on dispose de *N* classifieurs donnant des prédictions indépendantes.
- On suppose que chaque classifieur a une probabilité d'erreur de $p < 1/2$
- Si *p* est juste inférieure à 1/2, le classifieur est à peine meilleur qu'une réponse aléatoire
  - *Classifieur faible (weak classifier)*
- Si on choisit la réponse majoritaire entre les N classifieurs, est-ce qu'elle sera meilleure que la réponse individuelle des classifieurs?

Classifieurs faibles

- La probabilité qu'exactement *k* classifieurs parmi les *N* prédisent la mauvaise classe est (selon la loi binomiale) :

$$\binom{N}{k} p^k (1-p)^{N-k}$$

- La probabilité que la majorité des classifieurs prédise la mauvaise classe est donc:

$$\sum_{k=\lceil N/2 \rceil}^{N} \binom{N}{k} p^k (1-p)^{N-k} \leqslant e^{-2N(p-1/2)^2}$$

- Il se trouve que cette valeur décroît assez vite avec *N* .
  - Par exemple, si on considère *N* = 20 classifieurs de probabilité d'erreur individuelle *p* = 0.3, alors la probabilité qu'au moins 10 classifieurs parmi les 20 fournissent une mauvaise réponse vaut 0, 048.

On a donc tout intérêt à associer des classifieurs faibles pour en faire un classifieur fort!

Forêts aléatoires

- Forêt aléatoire = bagging appliqué à des **arbres de décision**
- Un arbre de décision est une suite de questions simples, comme au jeu « Qui est-ce ? »
- Avantage : on peut lire et expliquer sa décision

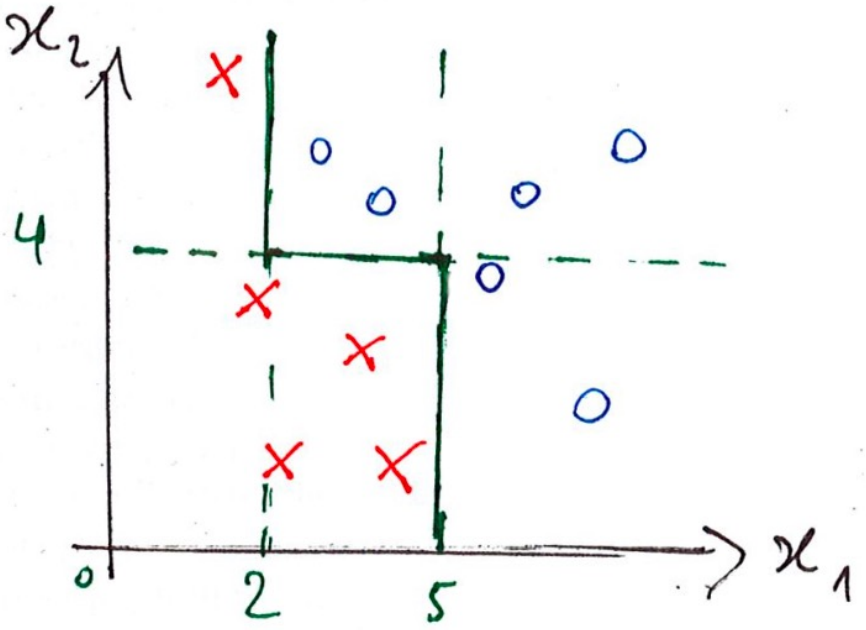
Exemple

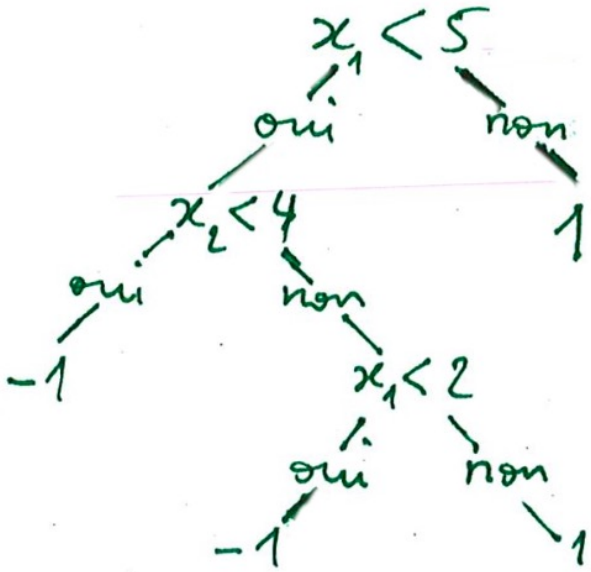
Faut-il afficher la pop-up d'inscription ?

- Visiteur nouveau ? Oui
- Temps sur la page > 30 s ? Oui

→ Afficher la pop-up

Exemple d'arbre de décision





Comment l'arbre choisit ses questions

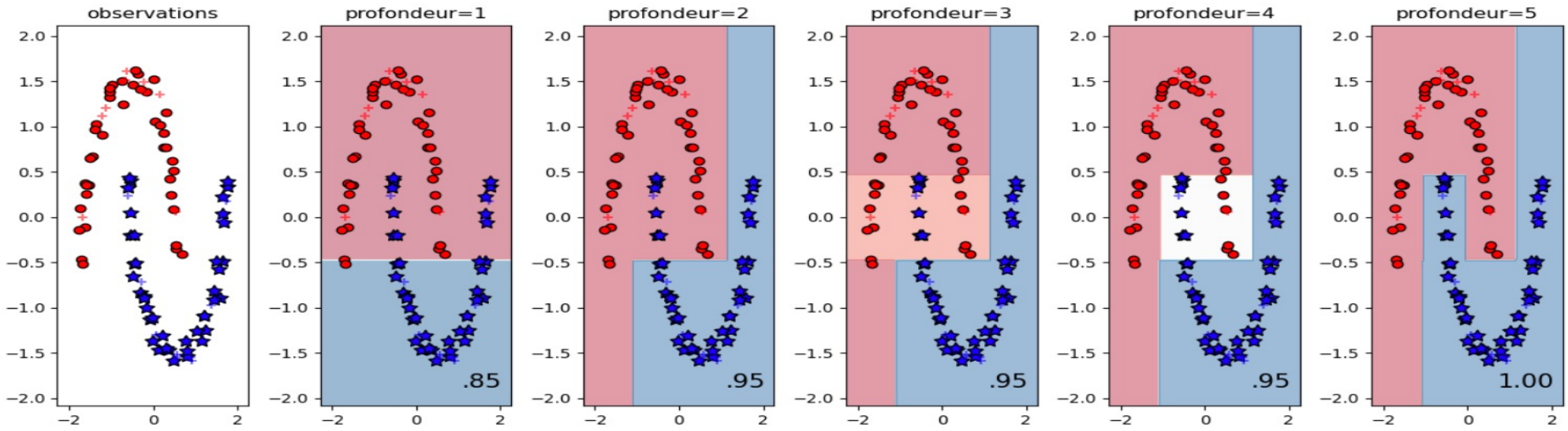
- Notons p_k la proportion d'observations dans le sous-ensemble courant appartenant à la classe k (k ∈ {1,...,K} dans le cas de K classes)
- Plusieurs critères de segmentation C sont possibles, les plus utiles sont :
  - L'entropie croisée

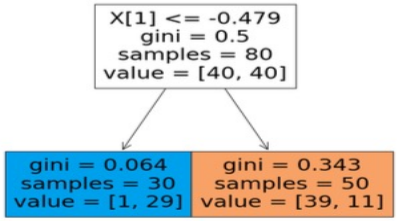
$$C = -\sum_{k=1}^{K} p_k \log(p_k)$$

  - Le critère de Gini

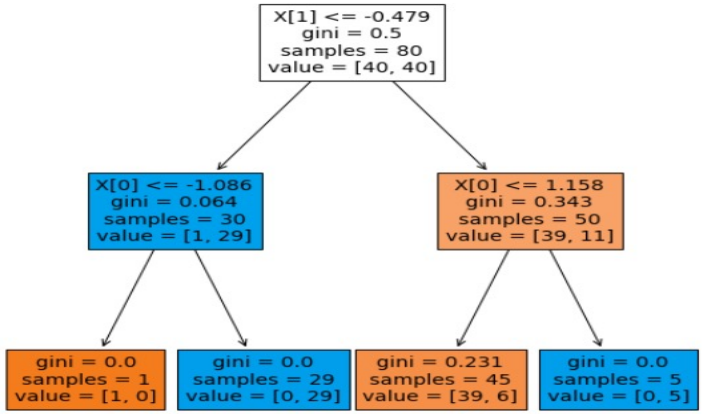
$$C = \sum_{k=1}^{K} p_k (1 - p_k)$$

- Dans le cas biclasse C = 0 si $p_k = 0$ ou $p_k = 1$, et C est maximal pour $p_k = 1/2$

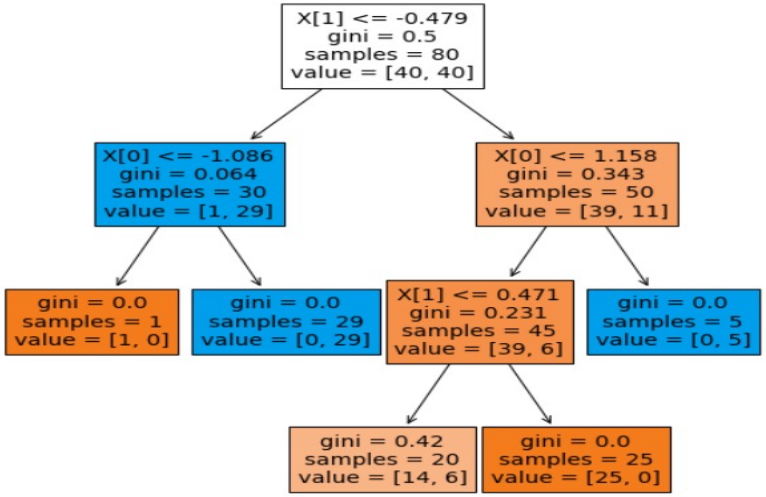




profondeur=1



profondeur=2

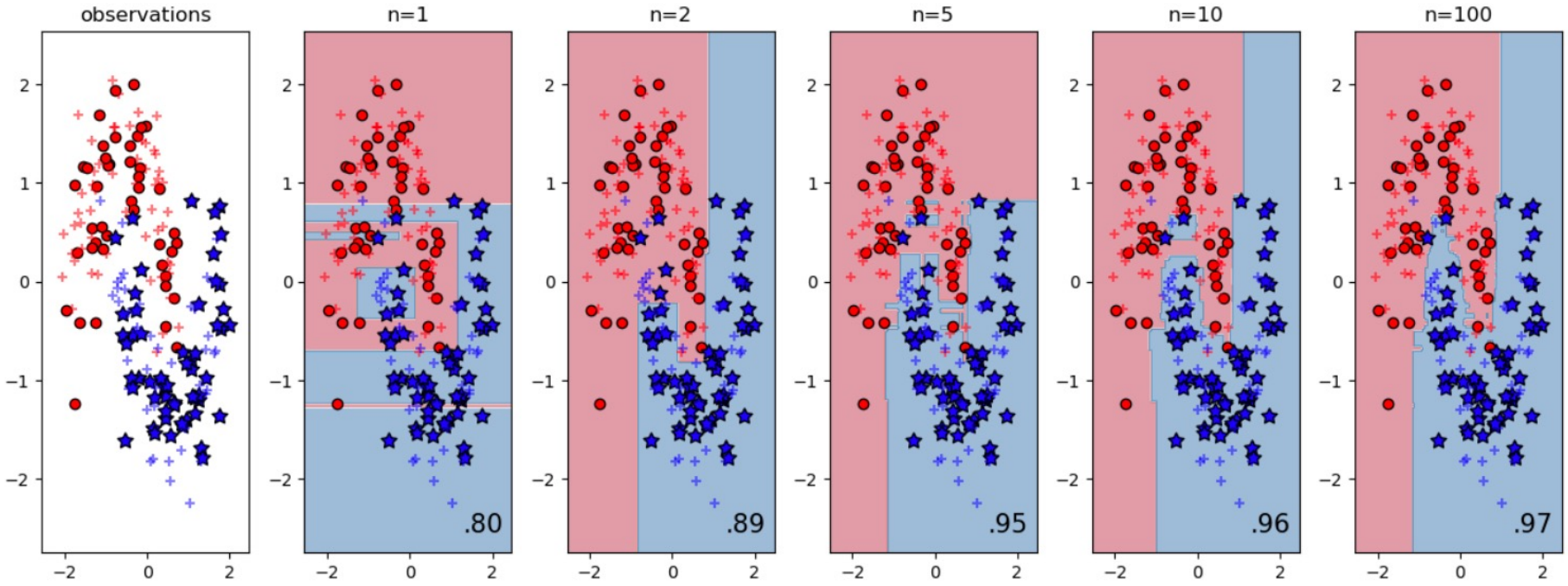


profondeur=3

Forêts aléatoires

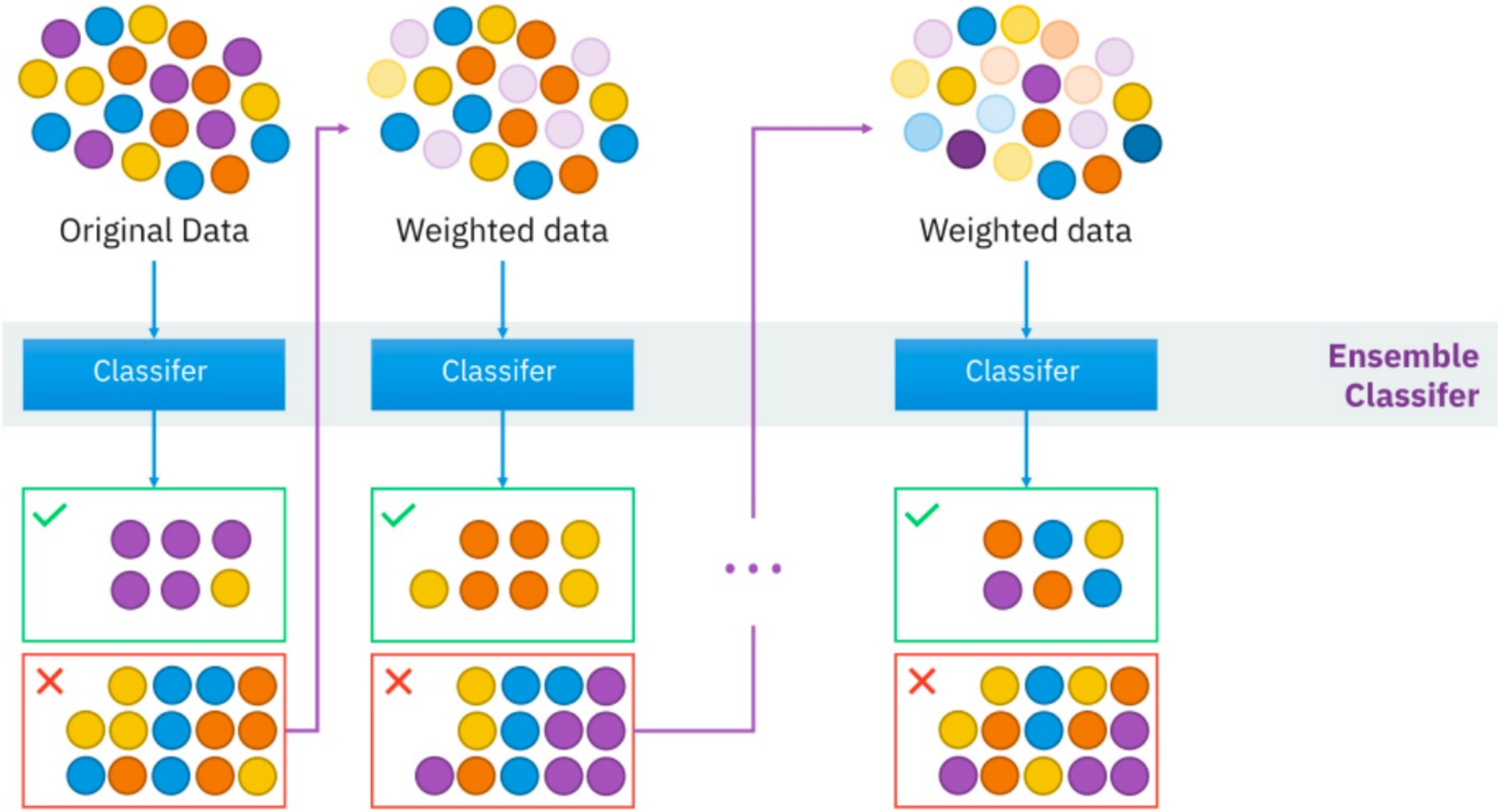
- Un seul arbre dépend beaucoup des exemples vus : il est instable
- Une forêt de centaines d'arbres qui votent est beaucoup plus robuste
- Prix à payer : on ne peut plus lire facilement la décision (« boîte noire »)
- Réglages principaux : nombre d'arbres, nombre de caractéristiques tirées au hasard

Exemple de RF avec un nombre d’arbres de décision croissant



Boosting

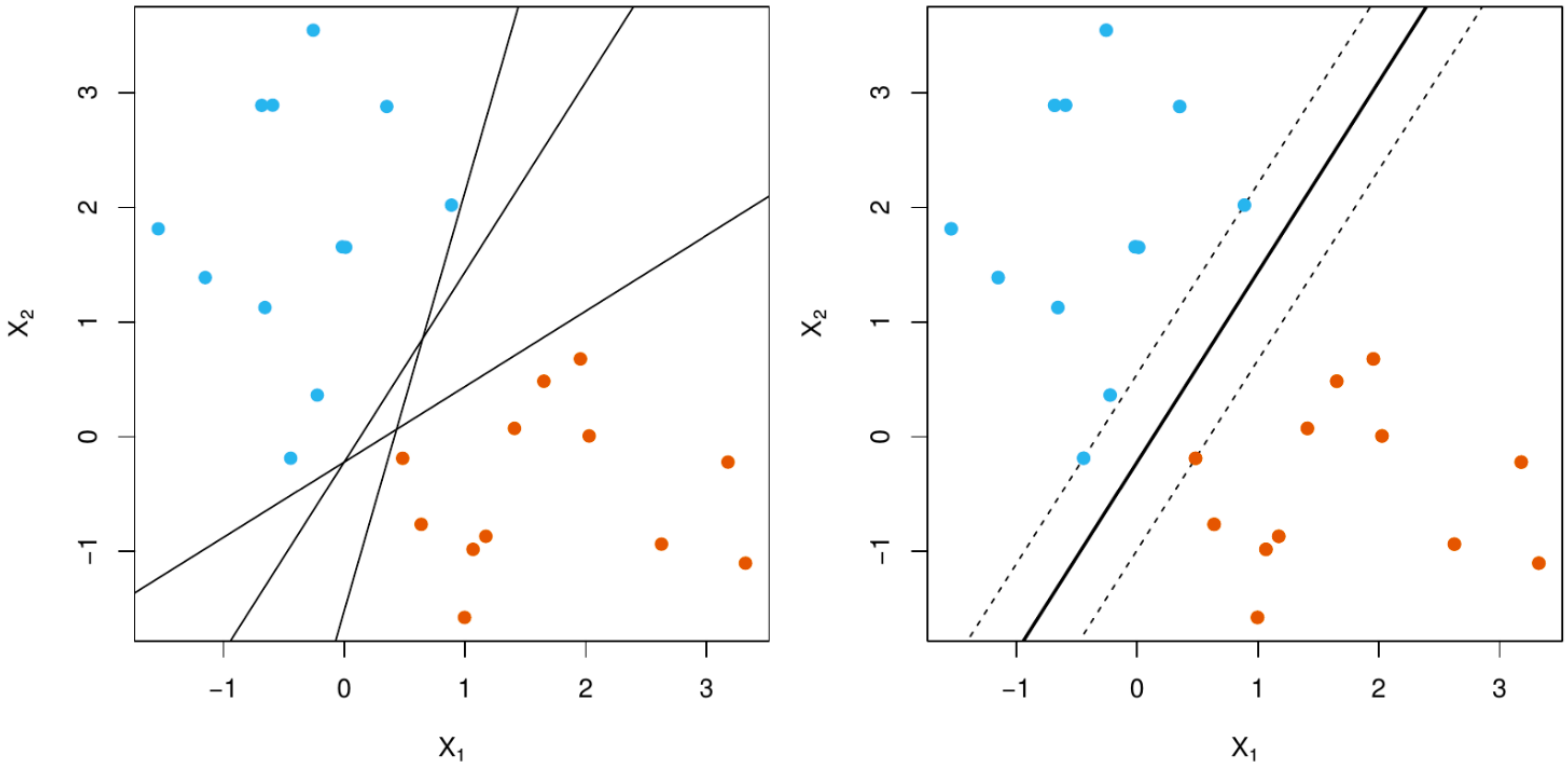
- Les modèles sont entraînés **l'un après l'autre**
- Chaque nouveau modèle se concentre sur les exemples mal classés par les précédents
- Réponse finale : vote pondéré, les meilleurs modèles comptent plus
- Exemples : AdaBoost, XGBoost (très utilisé sur les données en tableau)
- Analogie : réviser en priorité les exercices ratés


Original Data
Weighted data
Weighted data
Classifer
Classifer
Classifer
Ensemble
Classifer

3. Machines à vecteurs de support

SVM : l’idée, sans les équations

Notion de marge maximale



SVM : l’idée

- Trouver la frontière qui sépare deux classes avec la **plus grande marge** possible
- Marge = la « zone de sécurité » entre la frontière et les points les plus proches
- Ces points les plus proches sont les **vecteurs de support** : ce sont eux qui décident
- Frontière : $w \cdot x + b = 0$; un point est classé selon le signe de $w \cdot x + b$
- Marge souple : on accepte quelques erreurs quand les données se chevauchent

Notations

- N observations étiquetées de la base d'apprentissage seront notées $(\mathbf{x_n}, y_n)$, avec $y_n \in \{-1,1\}$
- $\mathbf{x}_i \in R^d$
- Equation hyperplan : $\mathbf{w} \cdot \mathbf{x} + b = 0$
- On cherche:
- $f(x)=w \cdot x+b$ ($b \in R, w \in R^d$) tel qu'une nouvelle observation x est classée selon : $f(x)>0 \Rightarrow x \in C_1$, et $f(x)<0 \Rightarrow x \in C_{-1}$

Rappel : distance de $\boldsymbol{x}'$ au plan $[\boldsymbol{w} \cdot \boldsymbol{x} + b = 0]$: $|\boldsymbol{w} \cdot \boldsymbol{x}' + b| / ||\boldsymbol{w}||_2$

Marge maximale (formalisation)

- Les hyperplans séparateurs de paramètres (**w**,b) vérifient, pour tout n entre 1 et N :

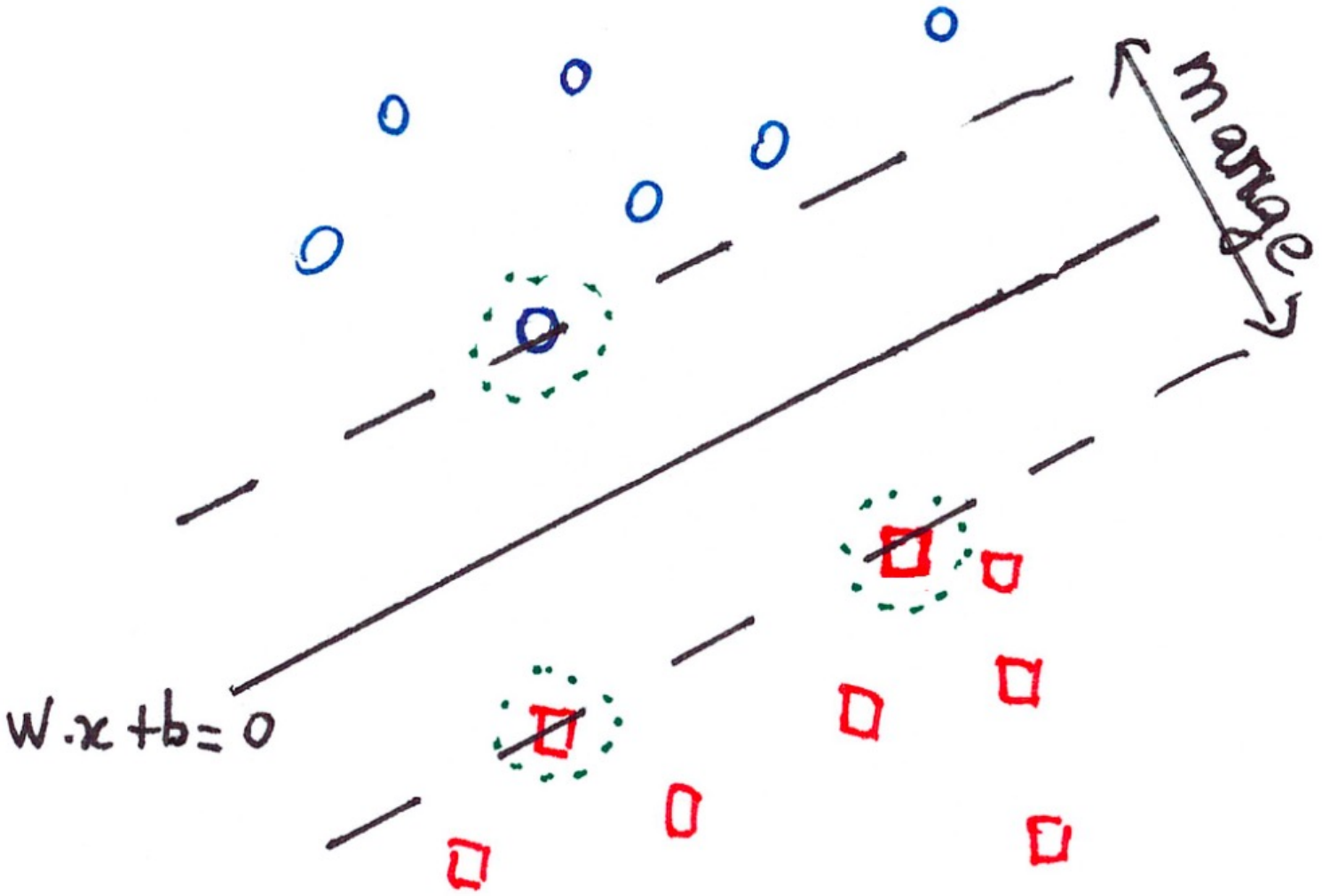
$$\begin{cases} \boldsymbol{w}\cdot\boldsymbol{x}_n + b > 0 & \text{si } y_n = 1 \\ \boldsymbol{w}\cdot\boldsymbol{x}_n + b < 0 & \text{si } y_n = -1 \end{cases}$$

- Ainsi, pour tout n : $y_n(\boldsymbol{w}\cdot\boldsymbol{x}_n + b) = |\boldsymbol{w}\cdot\boldsymbol{x}_n + b|$
- La plus petite distance entre les observations et un hyperplan séparateur de paramètres (**w**,b) est donc :

$$\frac{1}{\|\boldsymbol{w}\|_2} \min_{1\leqslant n\leqslant N} y_n(\boldsymbol{w}\cdot\boldsymbol{x}_n + b)$$

- Les paramètres du séparateur de plus grande marge sont donc:

$$\arg\max_{\boldsymbol{w},b} \left\{ \frac{1}{\|\boldsymbol{w}\|_2} \min_{1\leqslant n\leqslant N} y_n(\boldsymbol{w}\cdot\boldsymbol{x}_n + b) \right\}$$


marge
W.x+b=0

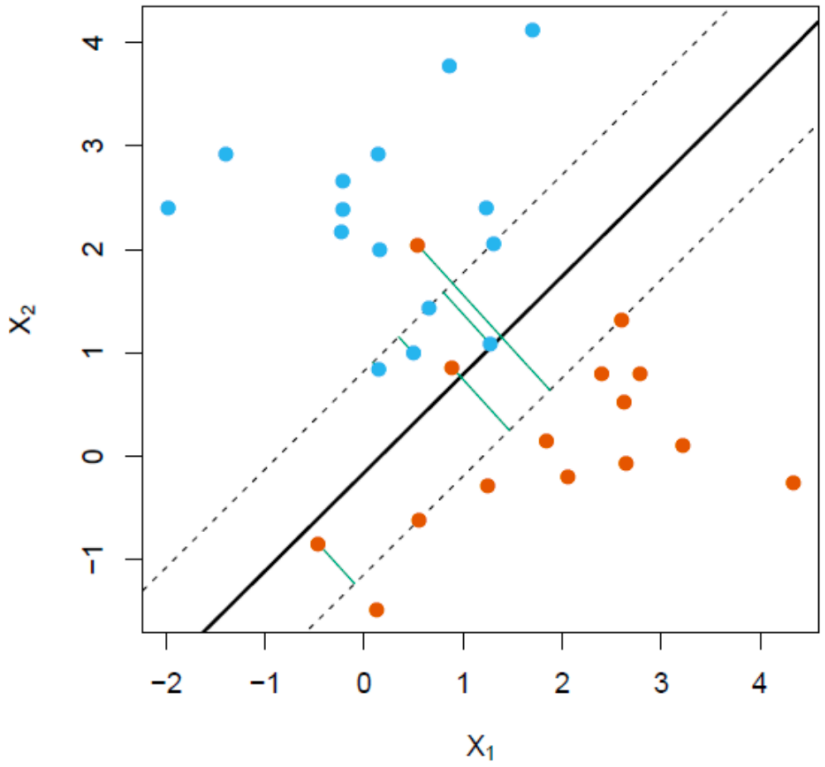
Marge souple

- Si le problème n'est pas linéairement séparable, c'est impossible de trouver une marge maximale
  - Solution: **marge souple**, qui permet d'accepter un certain nombre d'erreurs de classification
- Une variable d'écart à la marge (**slack variable**) pour l'observation $\mathbf{x}_n$ est un réel $\xi_n > 0$ tel que :

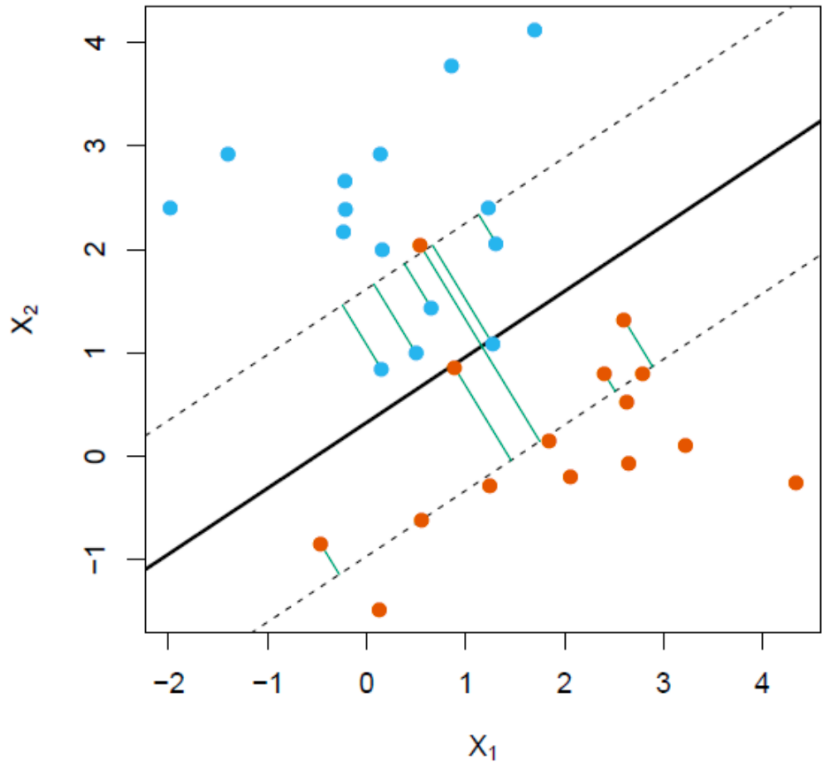
$$\xi_n = \begin{cases} 0 & \text{si l'observation } \boldsymbol{x}_n \text{ satisfait la contrainte sur la marge, } i.e.\ y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) \geqslant 1 \\ |y_n - (\boldsymbol{w} \cdot \boldsymbol{x}_n + b)| & \text{sinon} \end{cases}$$

Dans tous les cas, comme $y_n \in \{-1, 1\}$, la variable d'écart de la n-ème observation s'exprime ainsi :

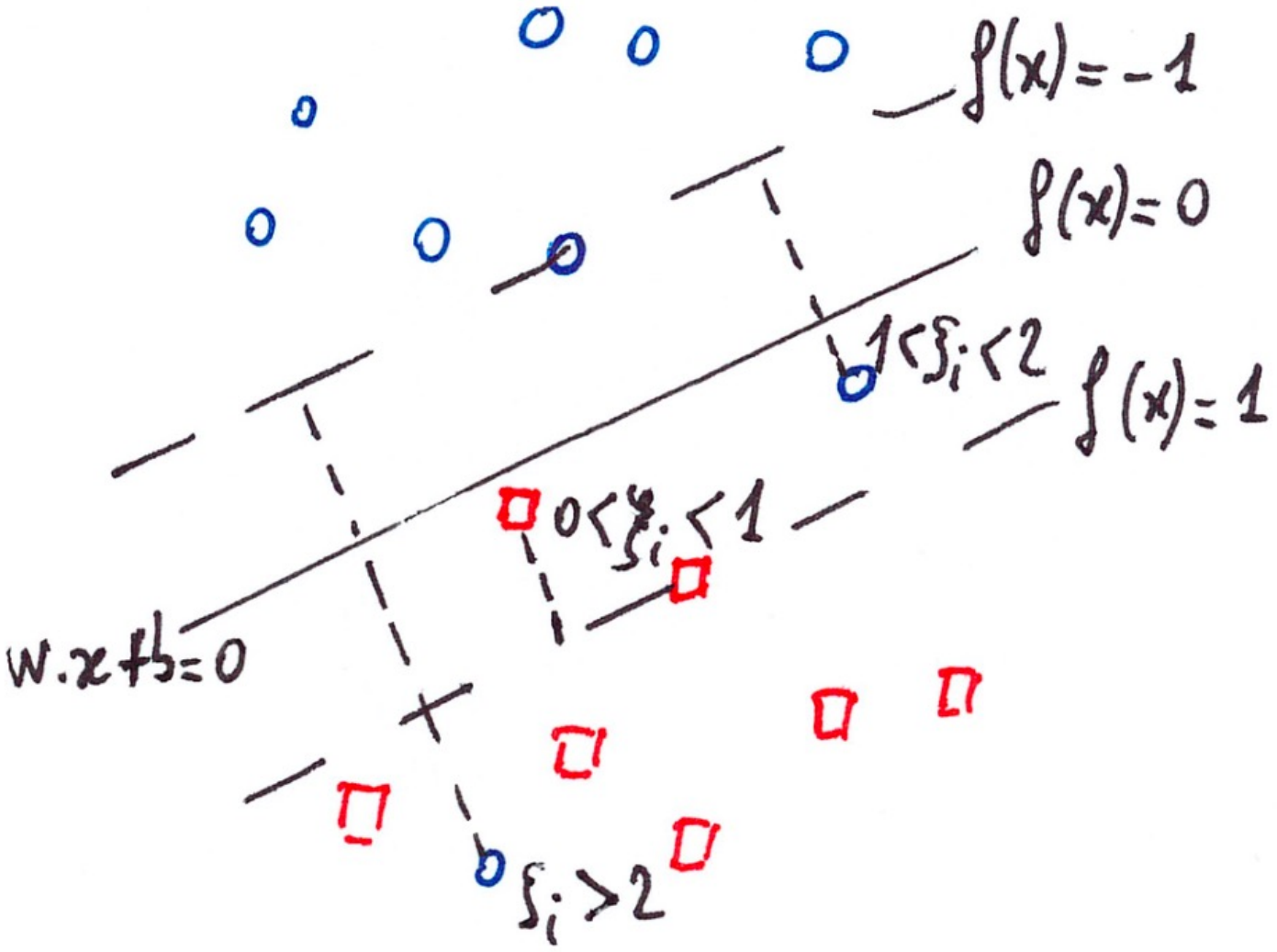
$$\xi_n = \max\{0, 1 - y_n(\boldsymbol{w} \cdot x_n + b)\}$$



C grand
peu d'observations ≪ dans ≫ la marge
marge petite

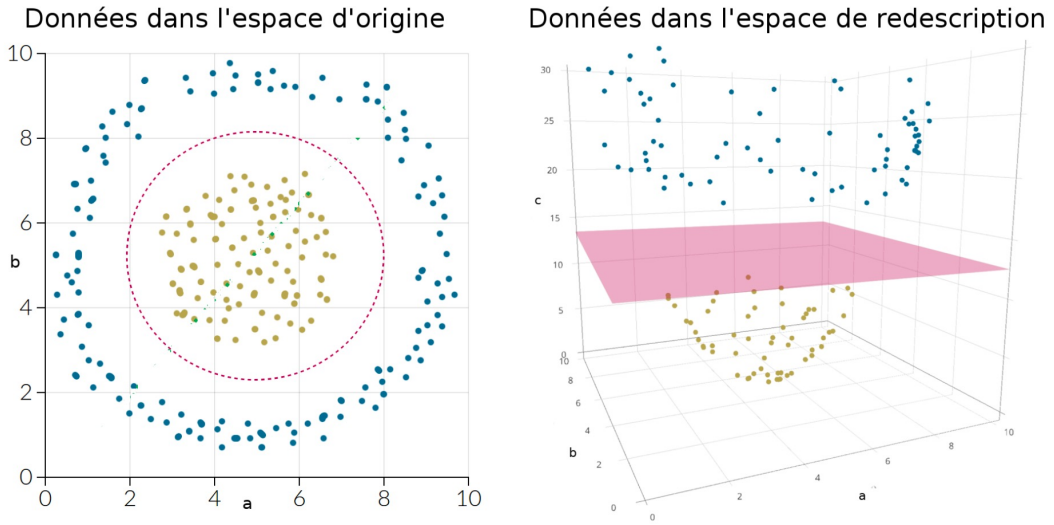


C faible
plus d'observations ≪ dans ≫ la marge
marge grande


f(x) = -1
f(x) = 0
1 < ξi < 2
f(x) = 1
0 < ξi < 1
w.x + b = 0
ξi > 2

L’astuce du noyau

- Problème : la frontière d’un SVM est une ligne droite
- Astuce : on « déplie » les données dans un espace avec une dimension de plus, où une séparation droite devient possible
- **Exemple :** points rangés en cercle ; on ajoute la hauteur « distance au centre » et un plan les sépare

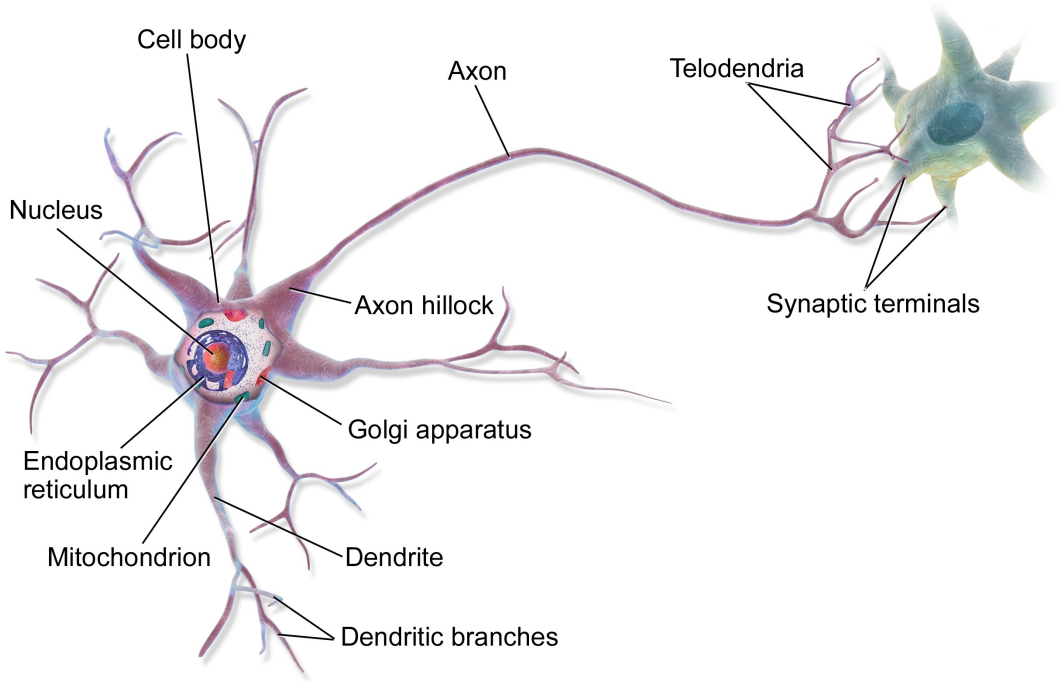


4. Réseaux neuronaux artificiels

Perceptron, réseaux multicouches, apprentissage

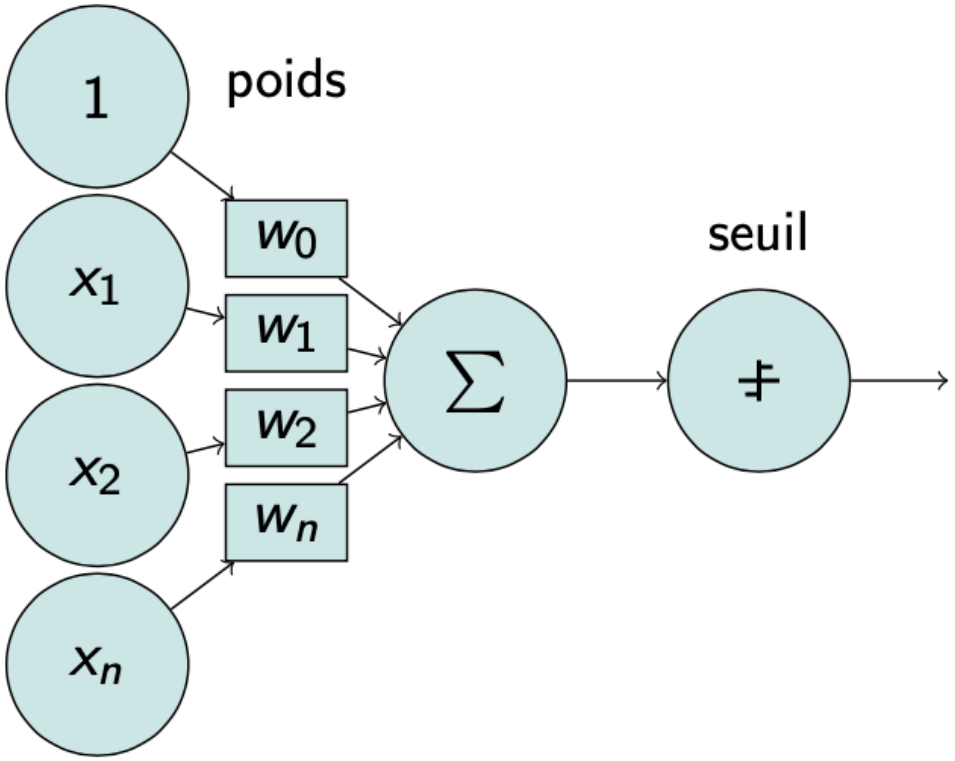
Le perceptron de Rosenblatt

- Modèle inspiré par le neurone réel
- C'est le type de *réseau de neurones* le plus simple
- Fonctionnement unidirectionnel (pas de « retropropagation »)



Le perceptron de Rosenblatt

- Entrées x : les informations (ex. météo, distance, prix du billet)
- Poids w : l'importance de chaque entrée, apprise à partir d'exemples
- Le neurone fait la somme pondérée : $w_1x_1 + w_2x_2 + \ldots + w_nx_n$
- Si elle dépasse un **seuil** : réponse « oui » (1), sinon « non »
- **Exemple :** « Est-ce que je vais au concert ce soir ? »

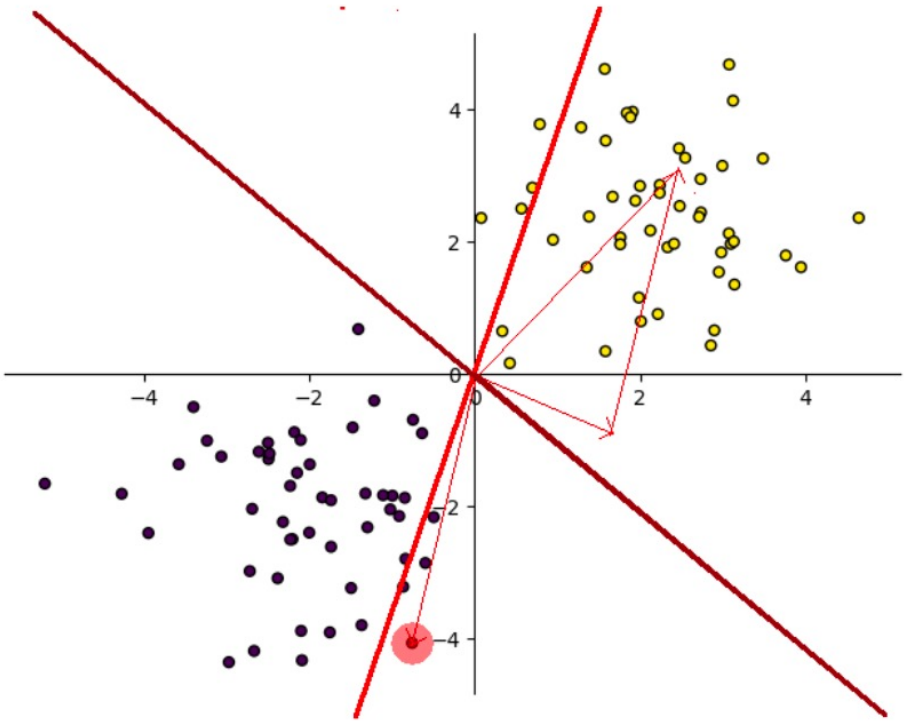


Comment apprend le perceptron ?

- On commence avec des poids à zéro
- Prédiction : $\hat{y} = \text{signe}(w \cdot x)$, soit +1 ou −1
- Pour chaque exemple mal classé ($\hat{y} \neq y$), on corrige : $\mathbf{w \leftarrow w + y \cdot x}$
- Lecture : on pousse les poids dans la direction de la bonne réponse
- On recommence jusqu'à ce qu'il n'y ait plus d'erreur
- Limite : un perceptron ne sait tracer qu'une ligne droite

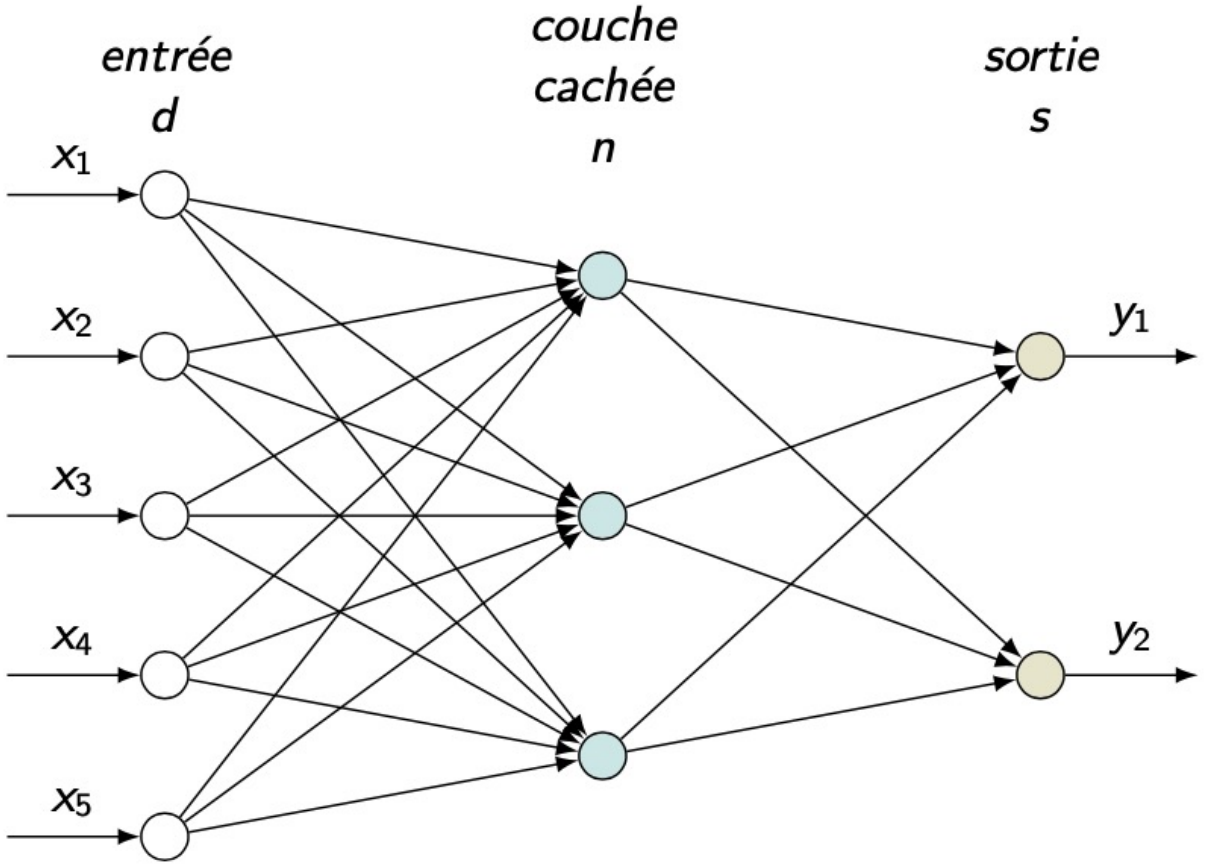
Visualisation

jaune : 1 mauve : -1



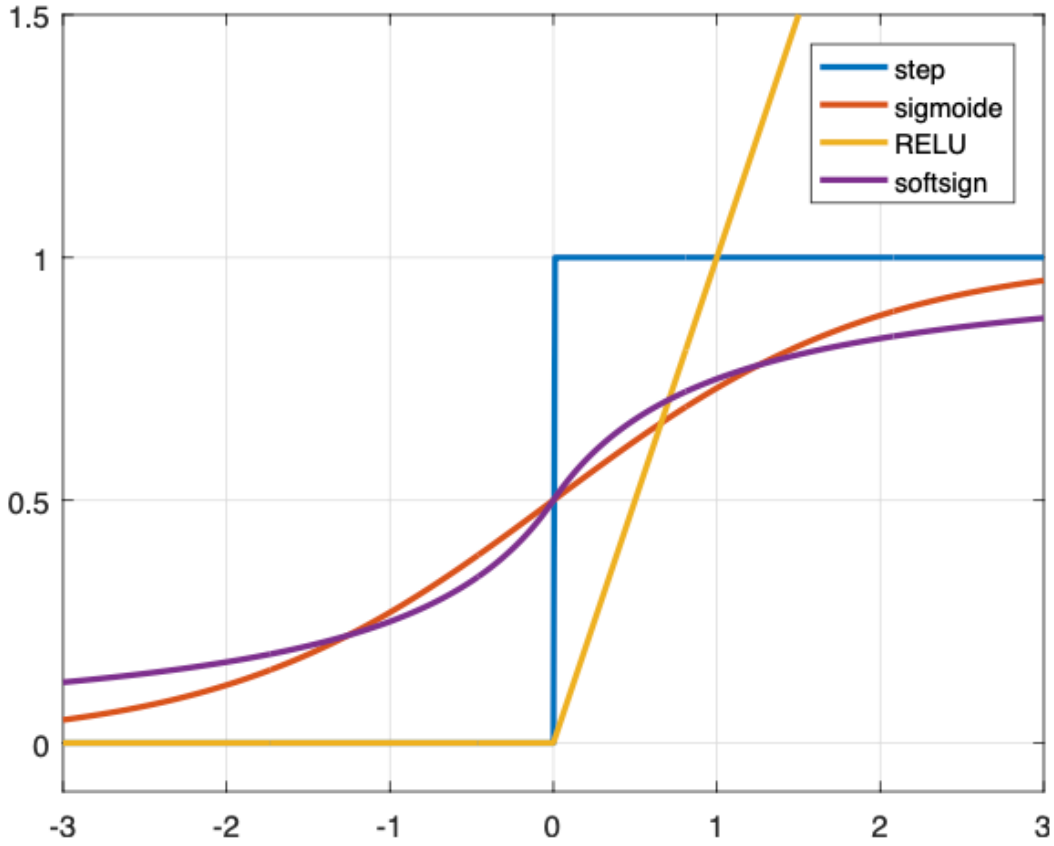
À l'étape $n+1$

Perceptron multicouche



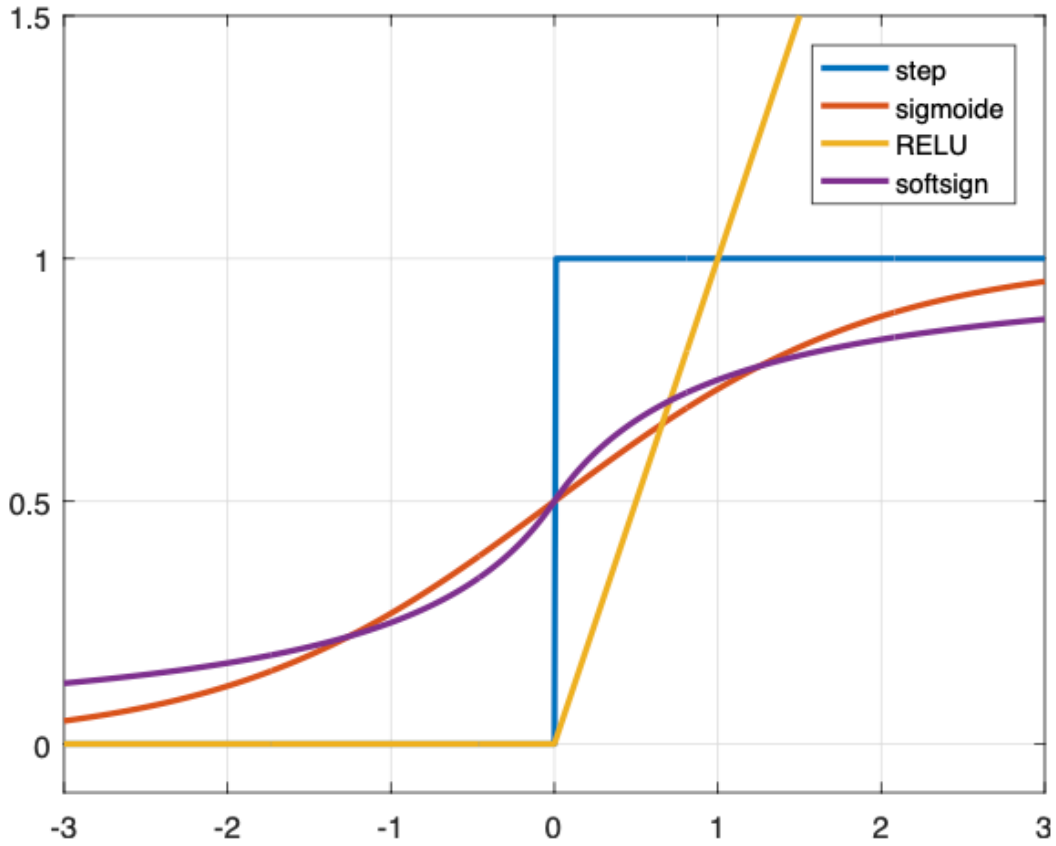
Fonctions d'activation

- Chaque neurone calcule $z = \sigma(b + \Sigma\, w_i\, x_i)$: somme pondérée, puis **fonction d'activation** σ
- Elle introduit des « courbes » : sans elle, le réseau ne ferait que des lignes droites
- Les plus courantes : sigmoïde, tanh et ReLU (la plus utilisée aujourd'hui) : ReLU(a) = max(0, a)



Fonctions d'activation

step (voir perceptron)	$s(a) = \begin{cases} 0 \text{ si } a < 0 \\ 1 \text{ si } a \geqslant 0 \end{cases}$
sigmoide	$\sigma(a) = \frac{1}{1+e^{-a}}$
tangente hyperbolique	$\tanh(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}} = 2\sigma(2a) - 1$
Rectified Linear Unit	$\mathrm{ReLU}(a) = \max\{0, a\}$
softsign	$\mathrm{softsign}(a) = \frac{a}{1+\lvert a\rvert}$



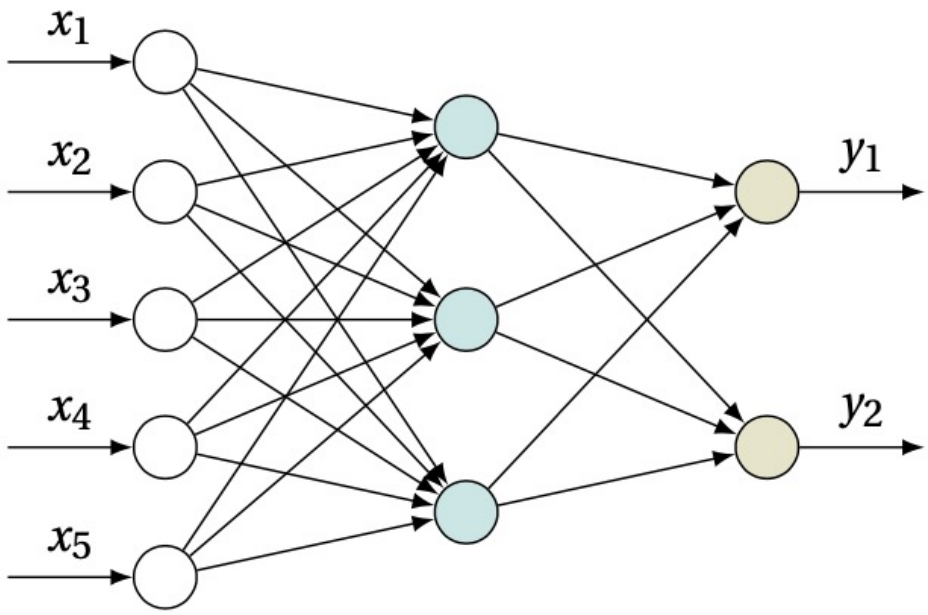
Notations

- B = (X_n,Y_n) n=1...N : base d'apprentissage
- Y_n = (y_{n1}, . . . y_{ns}) ∈ R^s si régression (s neurones de sortie)
- Y_n = 0 ou 1 si classification biclasse
- Y_n = (0,0,1,0,...,0) si classification s > 2 classes
- Sortie du perceptron sur X_n = (x_{n1}, . . . , x_{nd}) (entrée : *d* neurones): $\hat{Y}_n = (\hat{y}_{n1}, \dots \hat{y}_{ns})$

Sortie d'un MLP de régression

- Autant d'unités de sortie que la dimension souhaitée
  - Ex: ici, nous pouvons prédire $\boldsymbol{y} \in R^2$ à partir de $\boldsymbol{x} \in R^5$
- Pas d'activation sur la dernière couche
- J-eme composante de la sortie est calculée:

$$y_j = \sum_i w_{ji} z_i + b_j$$

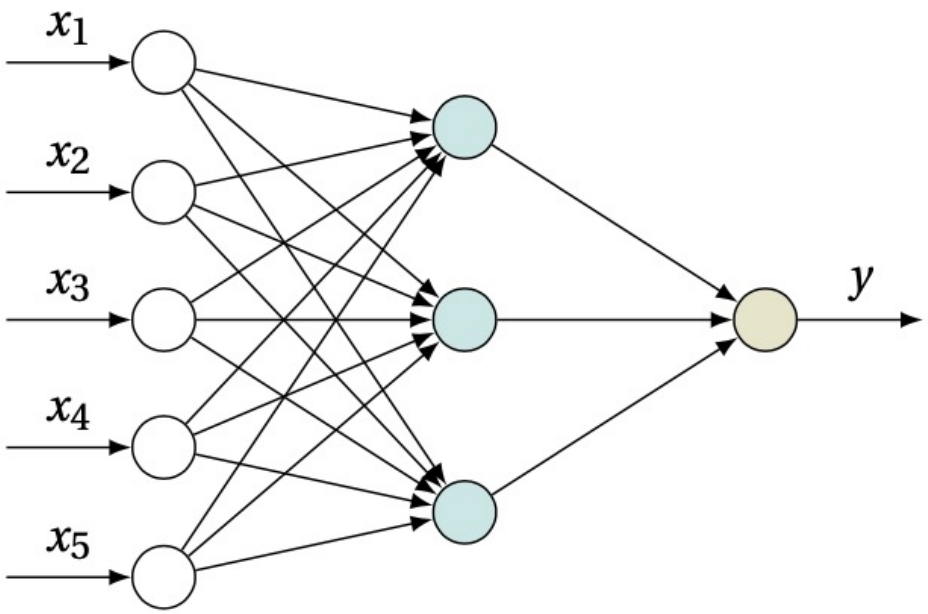


z_i est le signal émis par le neurone *i* de la couche cachée

Sortie pour classification binaire

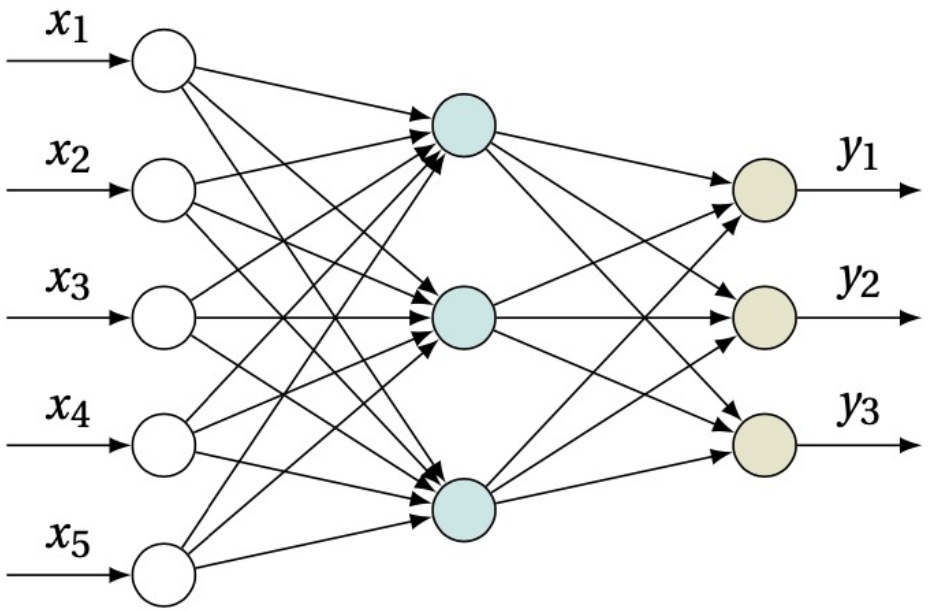
- Un seul neurone de sortie
- Sa valeur y = σ(somme pondérée des neurones cachés z_i), entre 0 et 1, est une probabilité (formule ci-dessous)
- Plus de 0,5 : classe 1 ; sinon : classe 2
- **Exemple :** « cet e-mail est un spam » avec une probabilité de 0,93

$$y = \sigma\left(\sum_i w_i z_i + b\right)$$



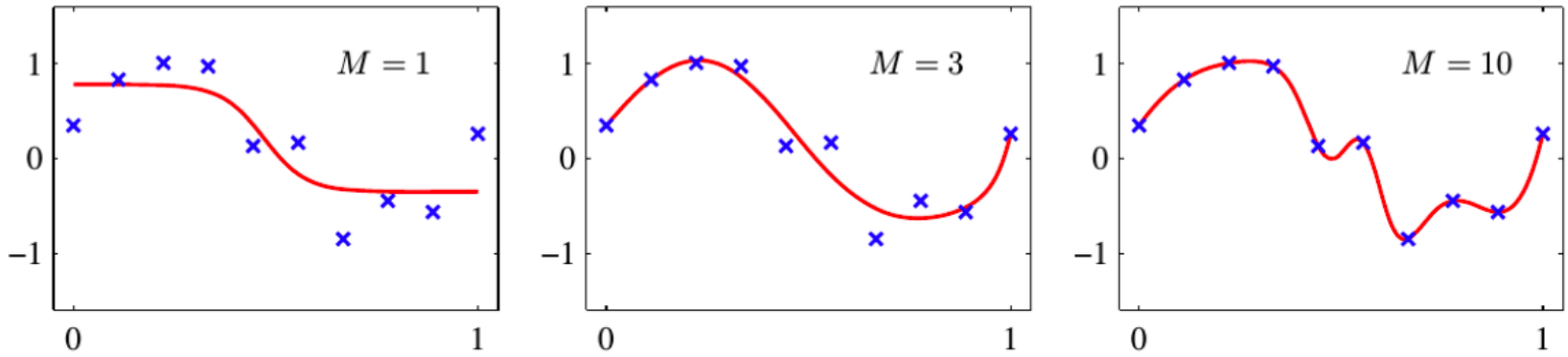
Sortie pour classification multiclasse

- Un neurone de sortie par classe
- Encodage **one-hot** : chat = (1, 0, 0), chien = (0, 1, 0), oiseau = (0, 0, 1)
- La fonction **softmax** (ci-dessous) transforme les sorties en probabilités dont la somme fait 1 : on prend l'exponentielle de chaque sortie et on divise par le total
- **Exemple :** chat 0,80 · chien 0,15 · oiseau 0,05 ; le réseau répond « chat »



$$y_i = \frac{1}{\sum_{k=1}^{K} \exp(a_k)} \exp(a_i)$$

Que peut apprendre un réseau ?



M : nombre de neurones dans la couche cachée
Illustration : C. Bishop, *Pattern Recognition and Machine Learning*, Springer 2006

$\rightarrow$ possible grâce à la non-linéarité σ

Mais : c'est un résultat d'*existence* uniquement
$\rightarrow$ ne dit pas comment choisir le nombre de neurones cachés, ni comment fixer les poids

Coût d'erreur

- Le risque empirique (coût d'erreur moyen sur la base d'apprentissage) est la moyenne des coûts des erreurs E_n sur chaque observation étiquetée (x_n , y_n) :

$$E(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N} E_n(\boldsymbol{w})$$

- Il faut choisir la fonction En en fonction du problème traité. Les choix les plus courants sont :

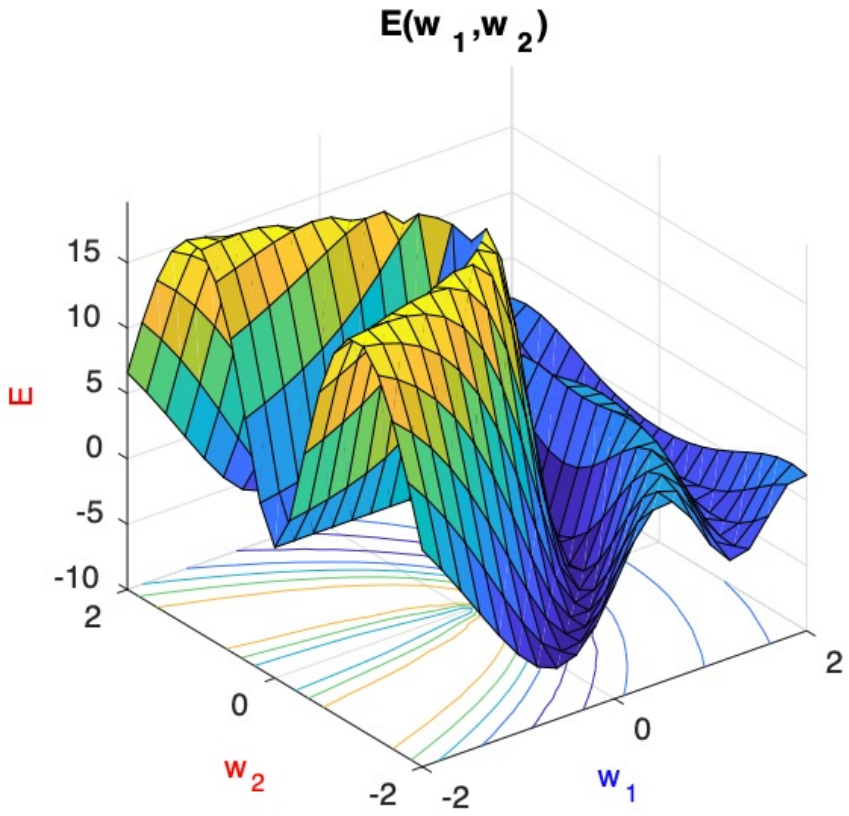
$$E_n(\boldsymbol{w}) = \tfrac{1}{2}||Y_n - \hat{Y}_n||_2^2 \text{ pour régression}$$

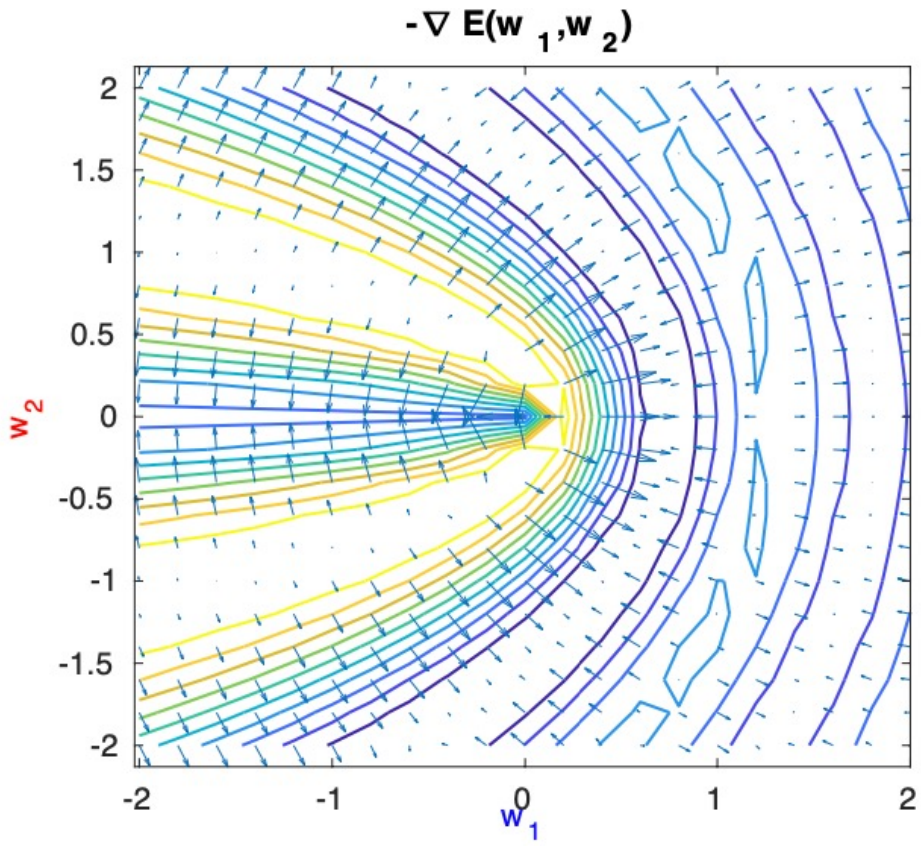
$$E_n(\boldsymbol{w}) = -\sum_{j=1}^{s} Y_{nj}\log(\hat{Y}_{nj}) \text{ pour classif. } s > 2 \text{ (entropie croisée)}$$

$$E_n(\boldsymbol{w}) = -Y_n\log(\widehat{Y}_n) - (1 - Y_n)\log(1 - \widehat{Y}_n) \text{ pour classif. biclasse}$$

- **Objectif** : choisir des poids w qui minimisent E(w)

Minimisation de l’erreur par descente de gradient





Minimisation par descente de gradient

Rappel : $\nabla E(w_1, w_2) = \begin{pmatrix} \frac{\partial E}{\partial w_1}(w_1, w_2) \\ \frac{\partial E}{\partial w_2}(w_1, w_2) \end{pmatrix}$

Algorithme de descente de gradient à pas fixe $\eta > 0$:
si $\mathbf{w} = (w_1, w_2)$, on itère : $\mathbf{w}_{n+1} = \mathbf{w}_n - \eta \nabla E(\mathbf{w}_n)$

soit, composante par composante : $\forall\, i \in \{1, 2\},\ w_i \leftarrow w_i - \eta \frac{\partial E}{\partial w_i}(w_1, w_2)$

Ici : η est le taux d'apprentissage

Minimisation par descente de gradient

On minimise $E(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} E_n(\boldsymbol{w})$

Par linéarité : $\nabla E(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} \nabla E_n(\boldsymbol{w})$

$\rightarrow$ algorithme de descente standard :

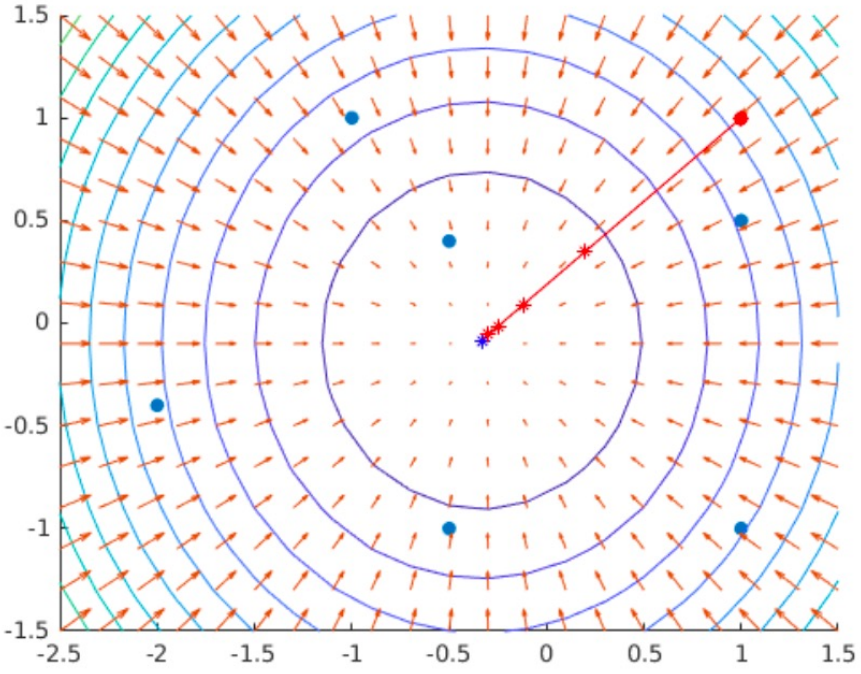
$$w_{ji} \leftarrow w_{ji} - \frac{\eta}{N} \sum_{n=1}^{N} \frac{\partial E_n}{\partial w_{ji}}(\boldsymbol{w})$$

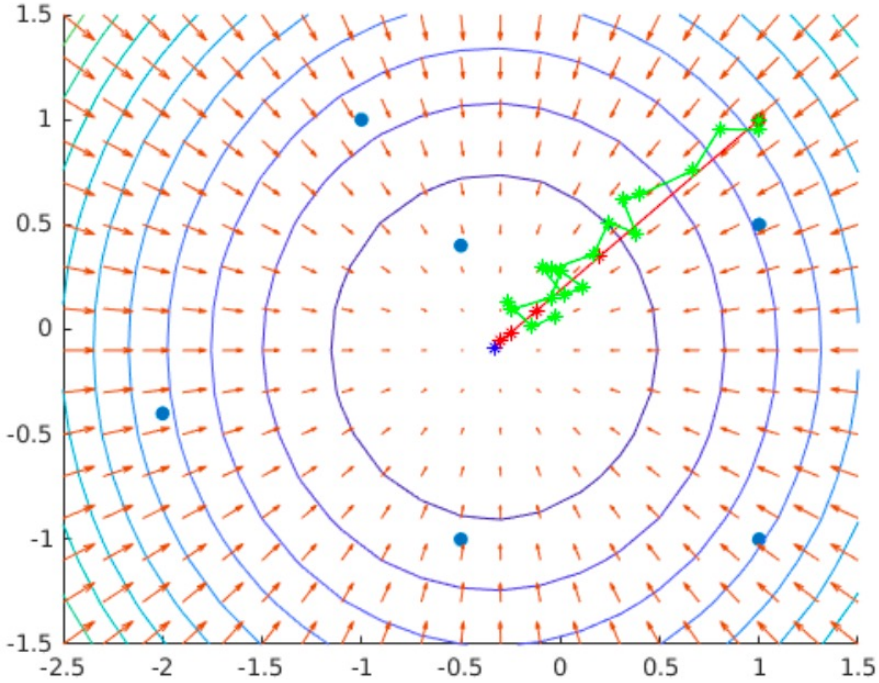
Problème : si N est très grand ? (plusieurs millions ?)
Vitesse de convergence ? Somme possible ?

SGD (Descente de gradient stochastique)

- Estimation du gradient sur un sous-ensemble (***Batch***) des données d'entrainement
  - M = $|B_{SGD}|$
  - M << N
- $w_{ji} \leftarrow w_{ji} - \eta \frac{\eta}{|B_{sgd}|} \sum_{n \in Bsgd} \frac{\partial E_n}{\partial w_{ji}}(\mathbf{w})$

Comparaison GD vs. SGD

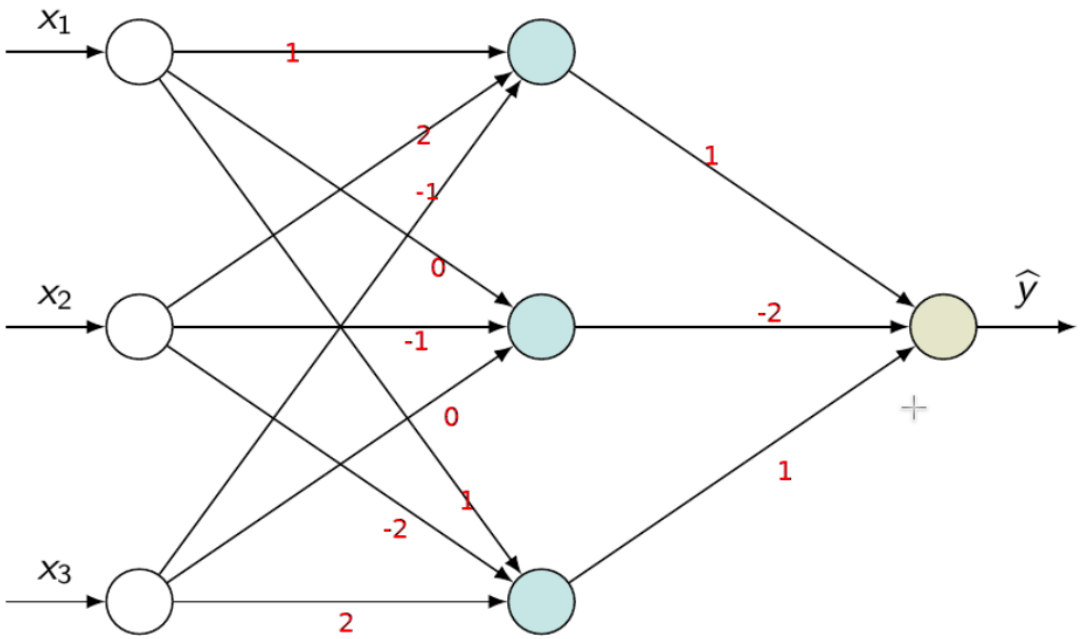




Calcul des poids

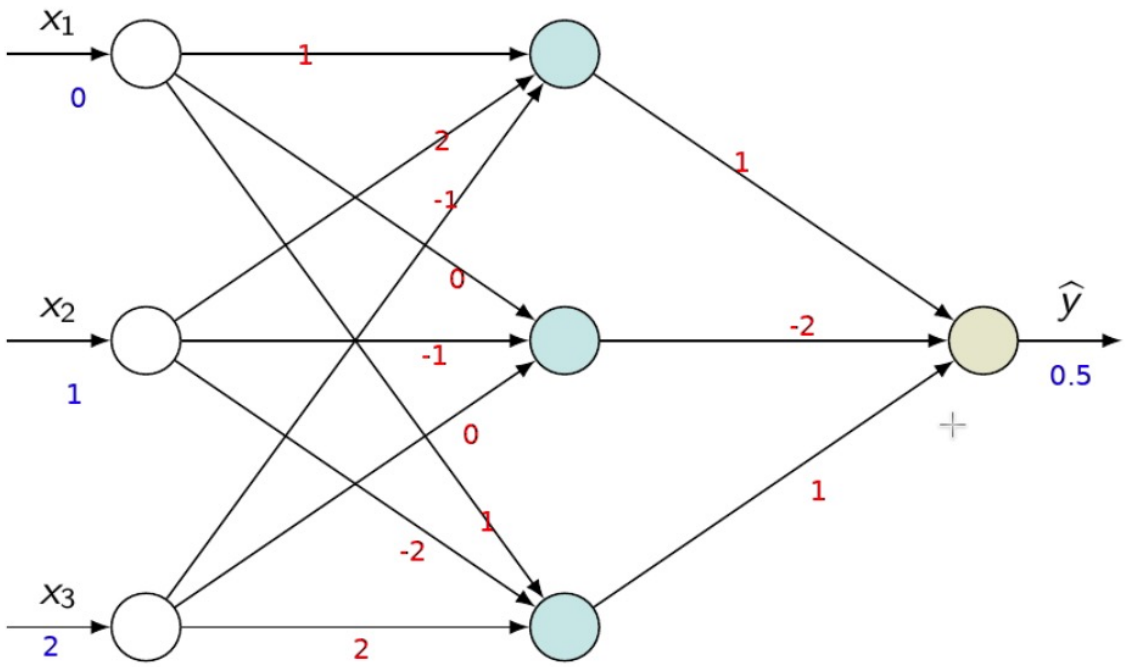
- Un réseau peut avoir des millions de poids : comment savoir lesquels corriger ?
- **Rétropropagation** (backpropagation) : on part de l'erreur en sortie et on remonte couche par couche
- Chaque poids est corrigé selon sa « part de responsabilité » dans l'erreur
- Exemple pas à pas sur les diapositives suivantes

Rétropropagation: exemple



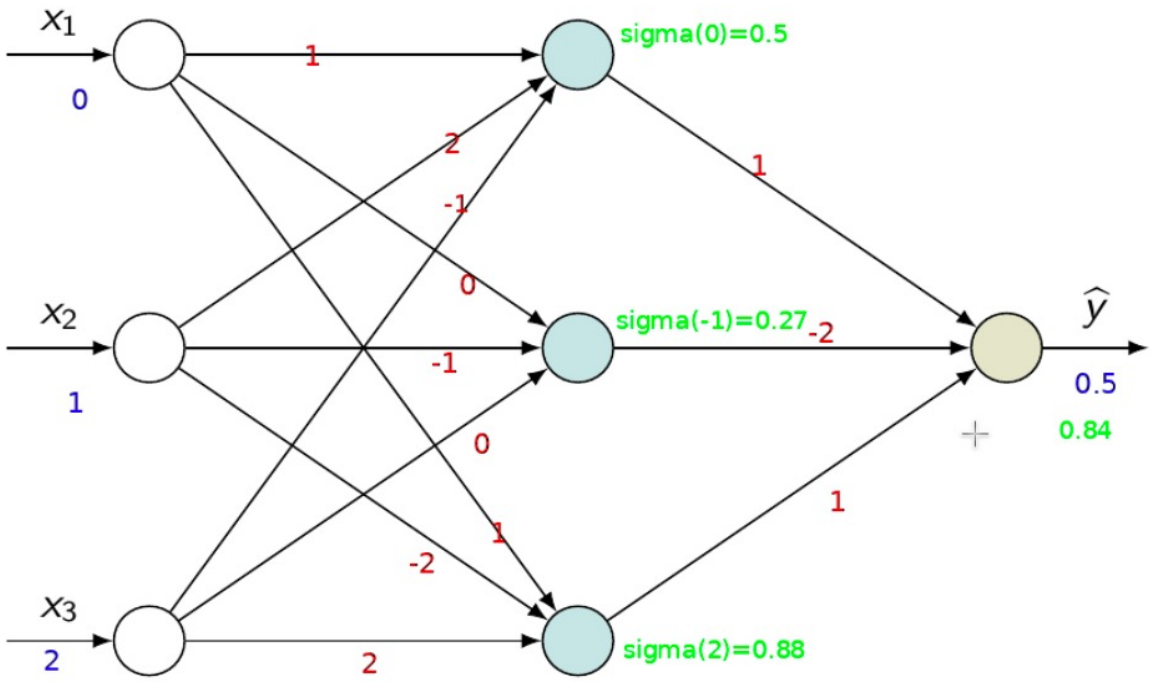
Exemple de MLP (régression)
activation aux neurones cachés : sigmoïde

Rétropropagation: exemple



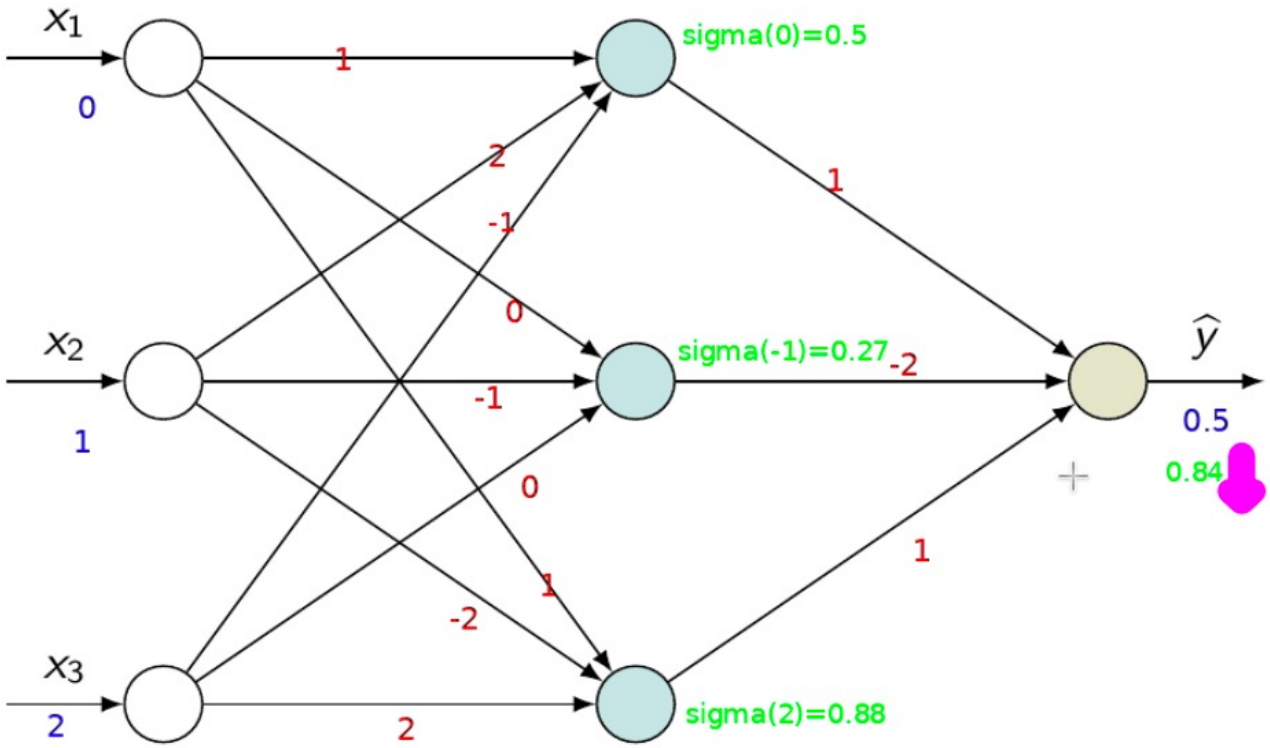
Observation étiquetée de la base d'apprentissage :
$\boldsymbol{x} = (0, 1, 2)$, $y = 0.5$

Rétropropagation: exemple



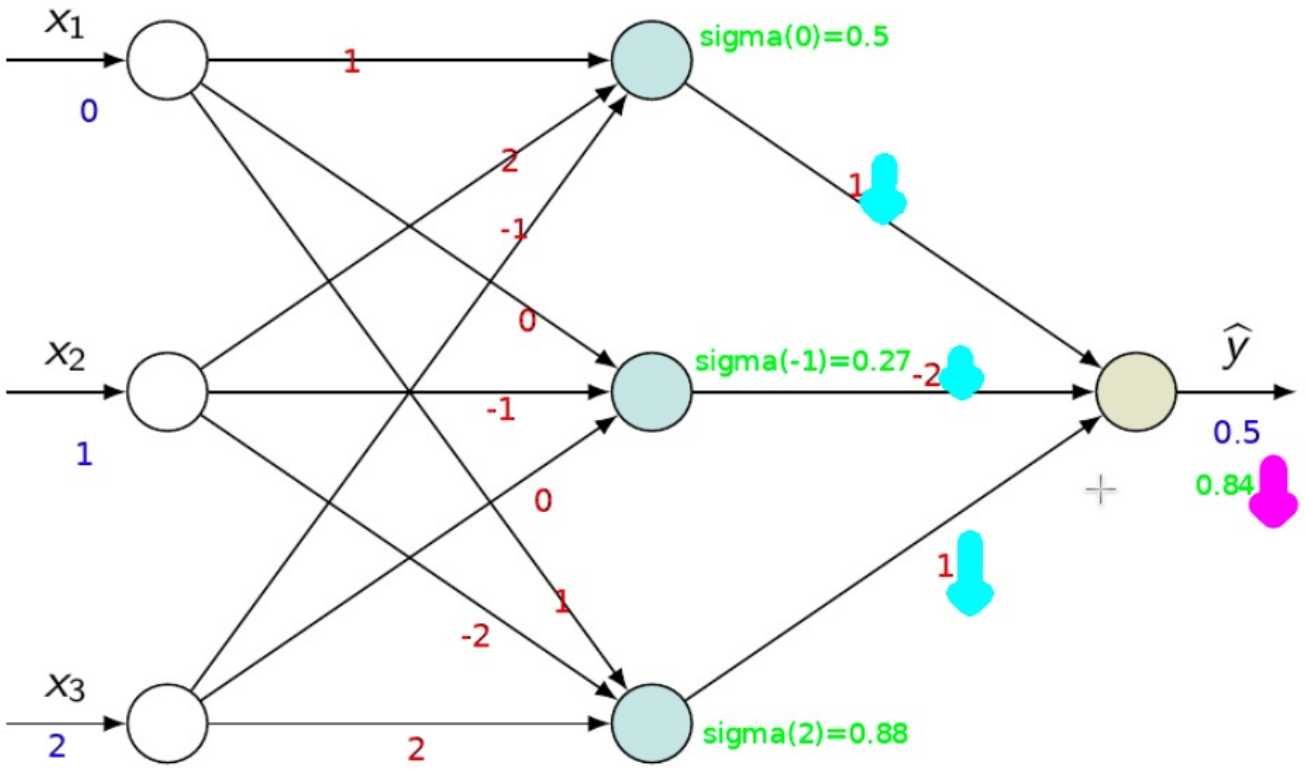
Étape « avant » (forward): calcul du résultat selon les poids et les entrées courantes

Rétropropagation: exemple



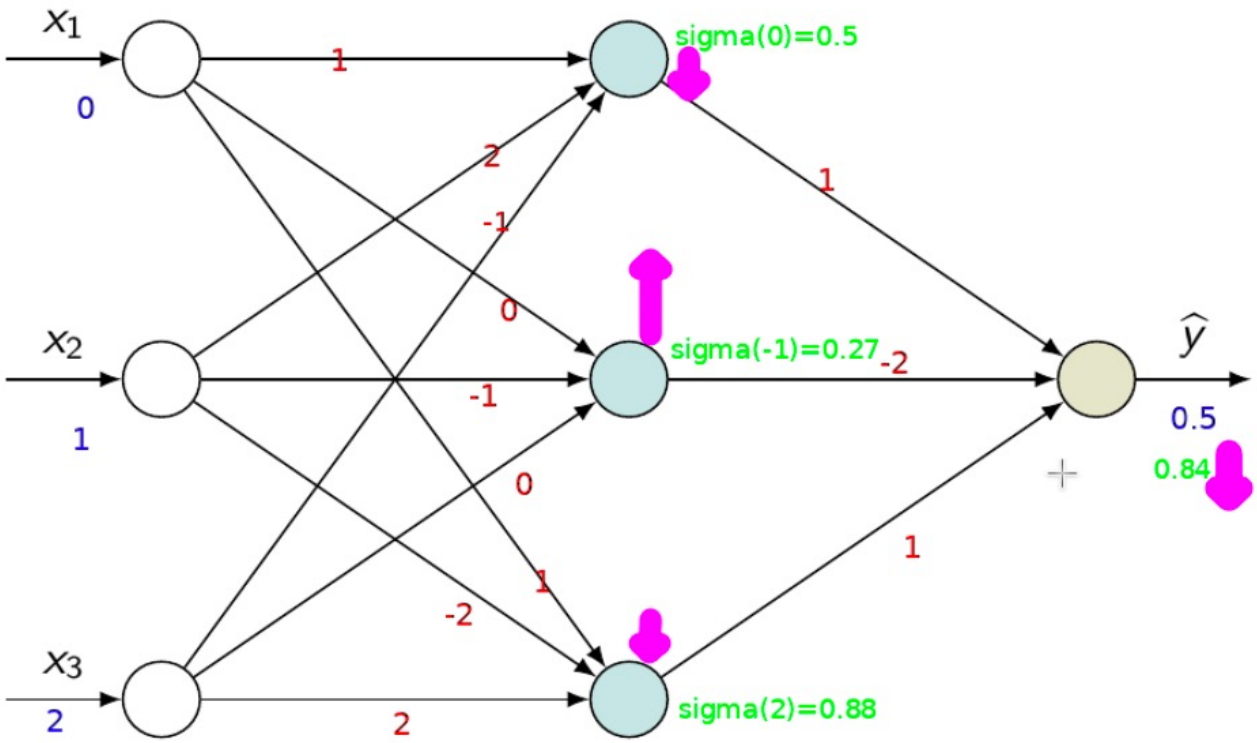
Estimation de l'erreur par rapport à la cible

Rétropropagation: exemple



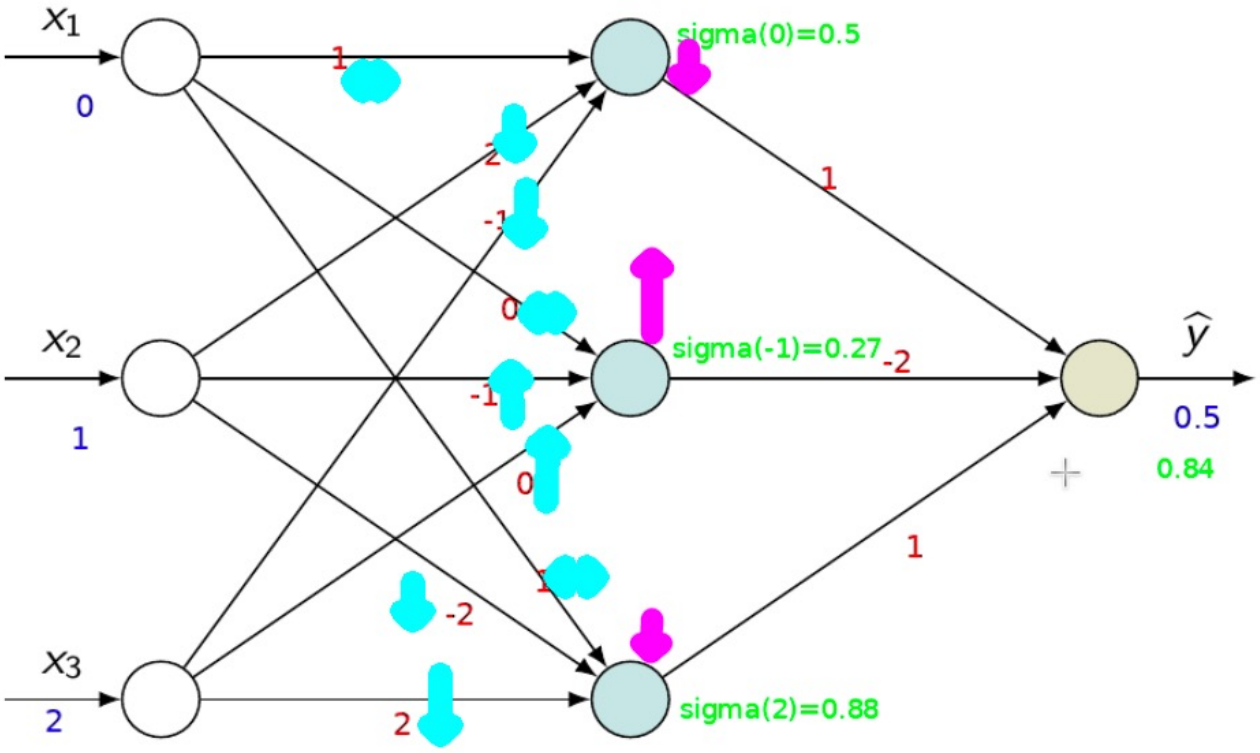
Pour baisser la valeur de sortie finale, il faut baisser les valeurs de ce qui rentre dans le dernier neurone ($w_{ij}z_{ij}$): donc il faut modifier les w_{ij}

Rétropropagation: exemple



Et par conséquent, il faut modifier aussi les z_{ij} -

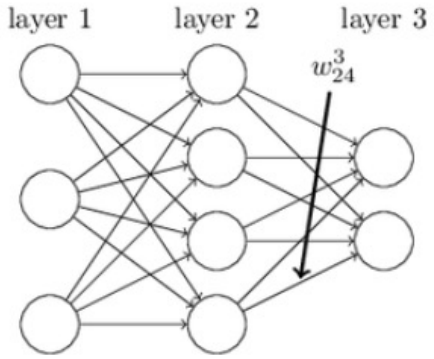
Rétropropagation: exemple



Mais donc il faut aussi modifier les poids entre l'entrée et la couche cachée

Notation

w^l_{jk} : weight among the k^{th} neuron of the $l-1$ layer to the j^{th} neuron of the l layer.

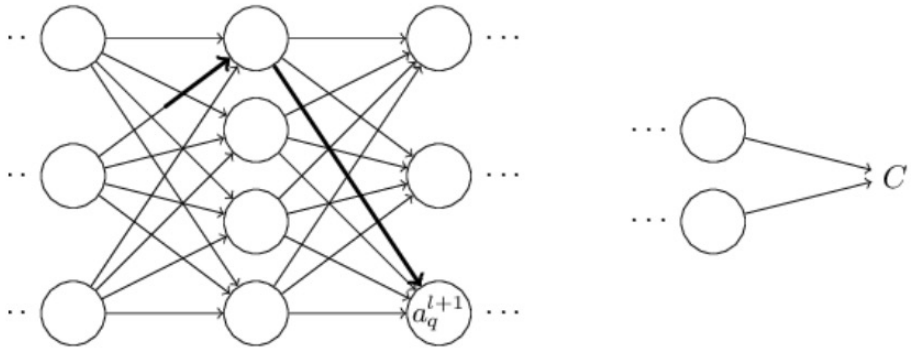


Algorithme de rétropropagation

- **Entrée**: les activations a^l de la couche l
- **Étape avant (forward)**: pour chaque couche l = 2, 3, … , L calculer $z^l = w^l \alpha^{(l-1)} + b^l$ et $a^l = \sigma(z^l)$
- **Calculer l'erreur**: $\delta^L = \nabla E \odot \sigma'(z^L)$
- **Rétropropagation de l'erreur**: pour chaque l = L-1, L-2, … calculer $\delta^l = ((w^{l+1})\sigma^{l+1}) \odot \sigma'(z^L)$
- **Sortie**: gradient de la fonction coût, $\frac{\partial E}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ et $\frac{\partial E}{\partial b^l_j} = \delta^l_j$ pour le biais

The big picture

- Δw_{lj}^{k} : small change in weight w_{lj}^{k}
- causes change of the output activation from the corresponding neuron:
  $\Delta \alpha_j^l \approx \frac{\partial \alpha_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$
- causing changes in all activations in the next layer
- Thus: $\Delta \alpha_q^{l+1} \approx \frac{\partial \alpha_q^{l+1}}{\partial \alpha_j^l} \Delta \alpha_j^l$



The big picture

Cascading changes to next layers cause change in the output:

- Single path:

$$\Delta C \approx \frac{\partial C}{\partial a_m^L}\frac{\partial a_m^L}{\partial a_n^{L-1}}\frac{\partial a_n^{L-1}}{\partial a_p^{L-2}}\cdots\frac{\partial a_q^{l+1}}{\partial a_j^l}\frac{\partial a_j^l}{\partial w_{jk}^l}\Delta w_{jk}^l$$

- Sum over all paths:

$$\Delta C \approx \Sigma_{mnp\ldots q}\frac{\partial C}{\partial a_m^L}\frac{\partial a_m^L}{\partial a_n^{L-1}}\frac{\partial a_n^{L-1}}{\partial a_p^{L-2}}\cdots\frac{\partial a_q^{l+1}}{\partial a_j^l}\frac{\partial a_j^l}{\partial w_{jk}^l}\Delta w_{jk}^l$$

- Thus effect of a change in a single weight on the total cost/error:

$$\frac{\partial C}{\partial w_{jk}^l} \approx \Sigma_{mnp\ldots q}\frac{\partial C}{\partial a_m^L}\frac{\partial a_m^L}{\partial a_n^{L-1}}\frac{\partial a_n^{L-1}}{\partial a_p^{L-2}}\cdots\frac{\partial a_q^{l+1}}{\partial a_j^l}\frac{\partial a_j^l}{\partial w_{jk}^l}$$

- Weights update:

$$w_k^{'} = w_k - \eta\frac{\partial C}{\partial w_k}$$

$$b_l^{'} = b_l - \eta\frac{\partial C}{\partial b_l}$$

En résumé : entraîner un réseau

1. **Propagation avant** : calculer la prédiction
2. **Mesurer l'erreur** (loss)
3. **Rétropropagation** : répartir l'erreur sur les poids
4. **Ajuster les poids** (descente de gradient)

On répète sur des milliers d'exemples, plusieurs fois (les « époques »)

À tester

TensorFlow Playground (playground.tensorflow.org) : on construit un petit réseau dans le navigateur et on le regarde apprendre en direct, sans code.

5. Réseaux de convolution

Comment une machine « voit » une image

Convolution

- Opération fréquente dans le monde du traitement des signaux
  - H fonction transformant un signal d'entrée *e* en signal de sortie *s = H(e)*
- Propriétés:
  - H est linéaire:si e_1 et e_2 sont des signaux et $\alpha \in R$, alors $H(e_1+\alpha e_2)=H(e_1)+\alpha H(e_2)$;
  - H est continu;
  - H est invariant par **translation**:si $\forall t$, $H(e(t))=s(t)$,alors $\forall t$, $H(e(t-\tau))=s(t-\tau)$
- On dénote donc avec

$$H(e) = h * s$$

le produit de convolution

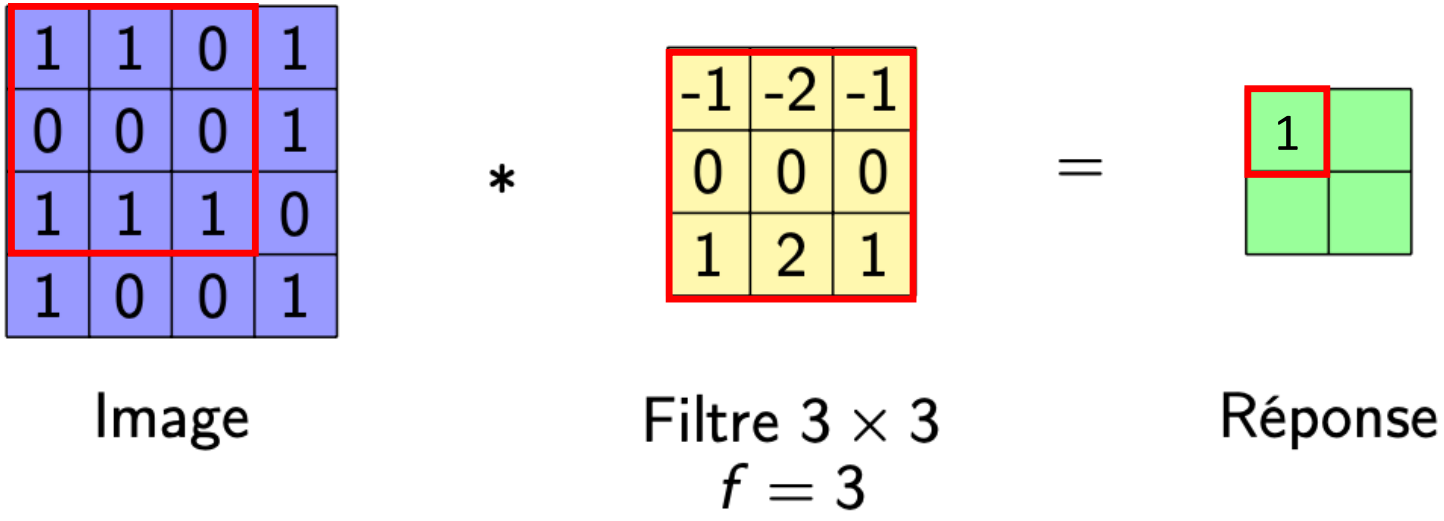
Convolution

- Une **convolution** = faire glisser un petit filtre sur toute l'image
- À chaque position : réponse = Σ (pixel × coefficient du filtre), voir le calcul détaillé sur les diapositives suivantes
- Dans un réseau de convolution (CNN), les filtres ne sont pas choisis à la main : ils sont **appris**

Vous le faites déjà

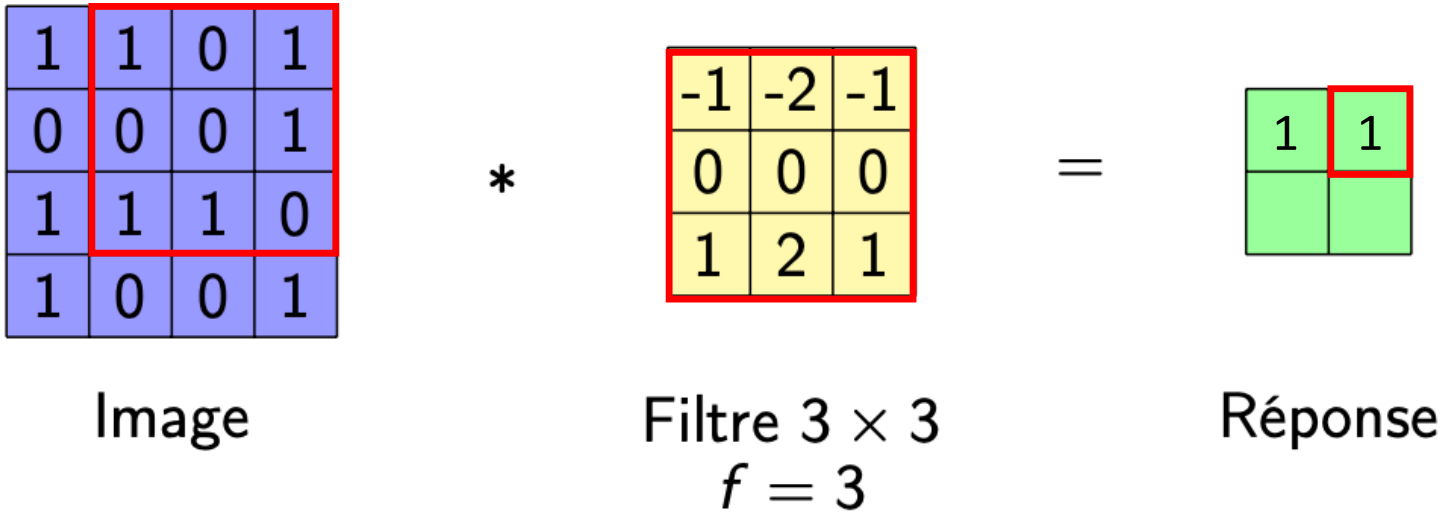
Les filtres « flou », « netteté » ou « détection de contours » de Photoshop ou GIMP sont des convolutions. Le filtre de la diapositive suivante détecte les bords horizontaux.

Convolution (images)



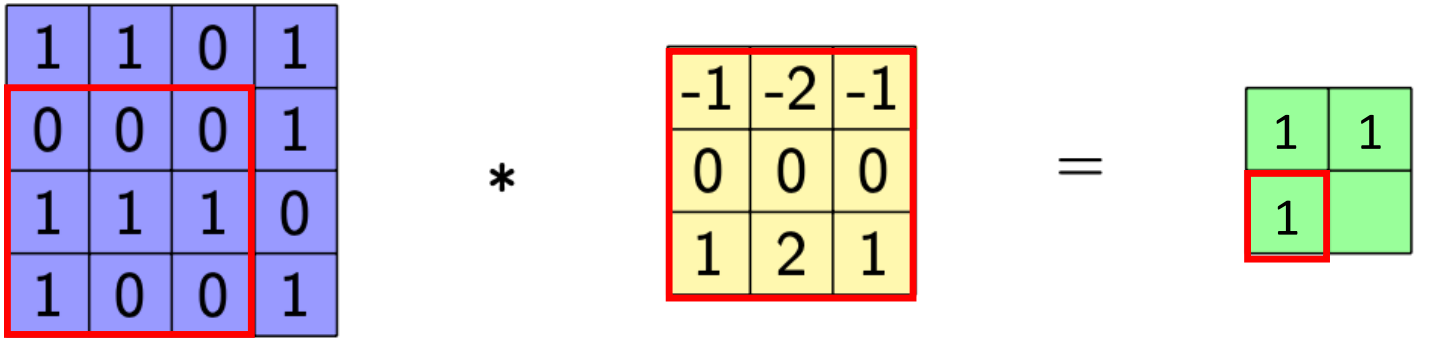
$$1 * -1 + 1 * -2 + 0 * -1 + 0 * 0 + 0 * 0 + 0 * 0 + 1 * 1 + 1 * 2 + 1 * 1 = 1$$

Convolution (images)



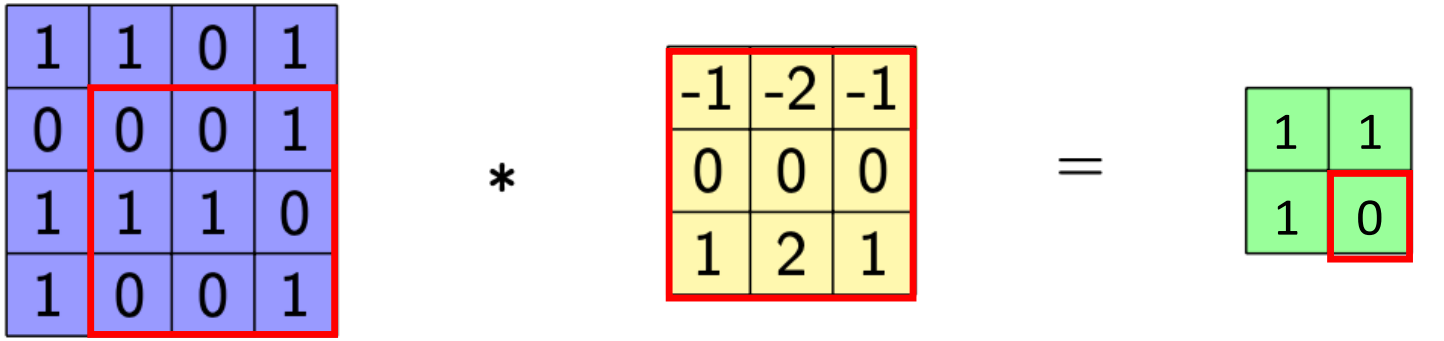
$1 * -1 + 0 * -2 + 1 * -1 + 0 * 0 + 0 * 0 + 1 * 0 + 1 * 1 + 1 * 2 + 0 * 1 = 1$

Convolution (images)



$$0 * -1 + 0 * -2 + 0 * -1 + 1 * 0 + 1 * 0 + 1 * 0 + 1 * 1 + 0 * 2 + 0 * 1 = 1$$

Convolution (images)



Image

Filtre 3×3
$f = 3$

Réponse

$$0 * -1 + 0 * -2 + 1 * -1 + 1 * 0 + 1 * 0 + 0 * 0 + 0 * 1 + 0 * 2 + 1 * 1 = 0$$

Note sur les dimensions

- Étant donnés :
  - une image I en niveaux de gris (un seul canal couleur) de dimension $w \times h$,
  - un filtre K de dimension f (le filtre est toujours un carré)
- Alors la dimension de la réponse I * K de l'image I au filtre K est

$$(w - f + 1) \times (h - f + 1)$$

Convolution de volumes

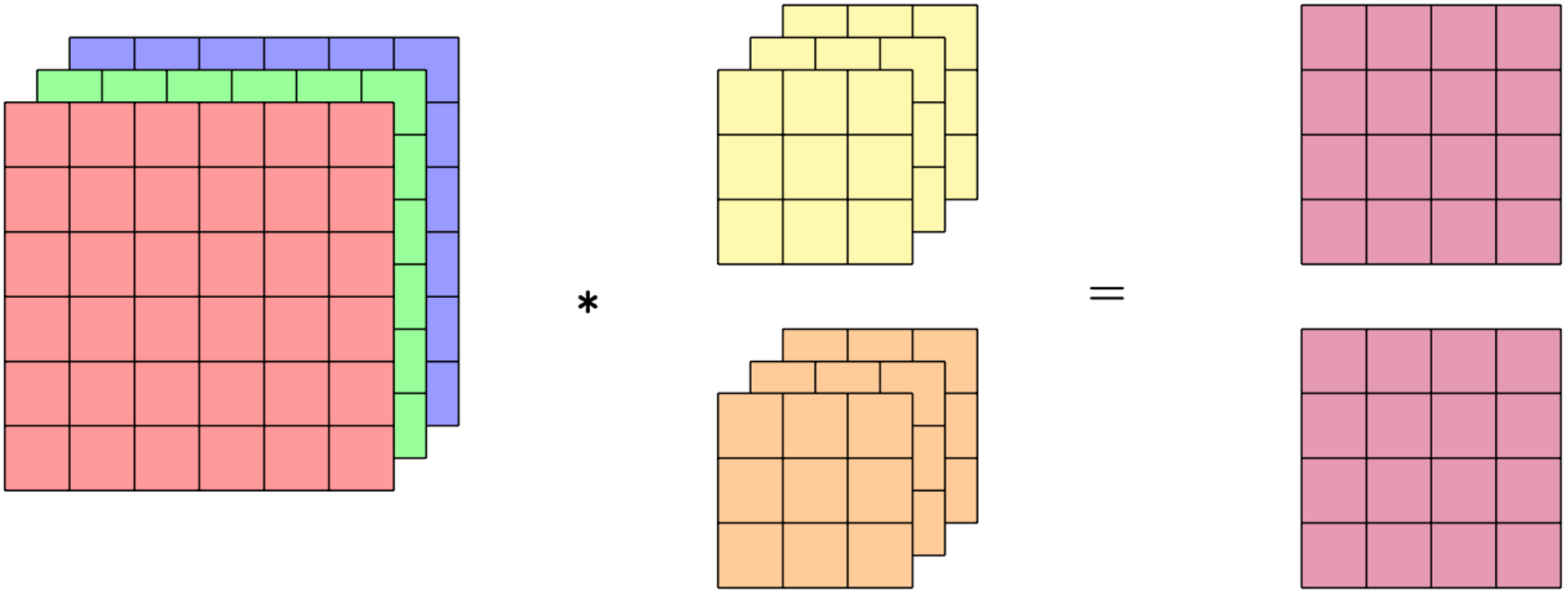


Image
$6 \times 6 \times \mathbf{3}$

Filtres
$3 \times 3 \times \mathbf{3} \times 2$

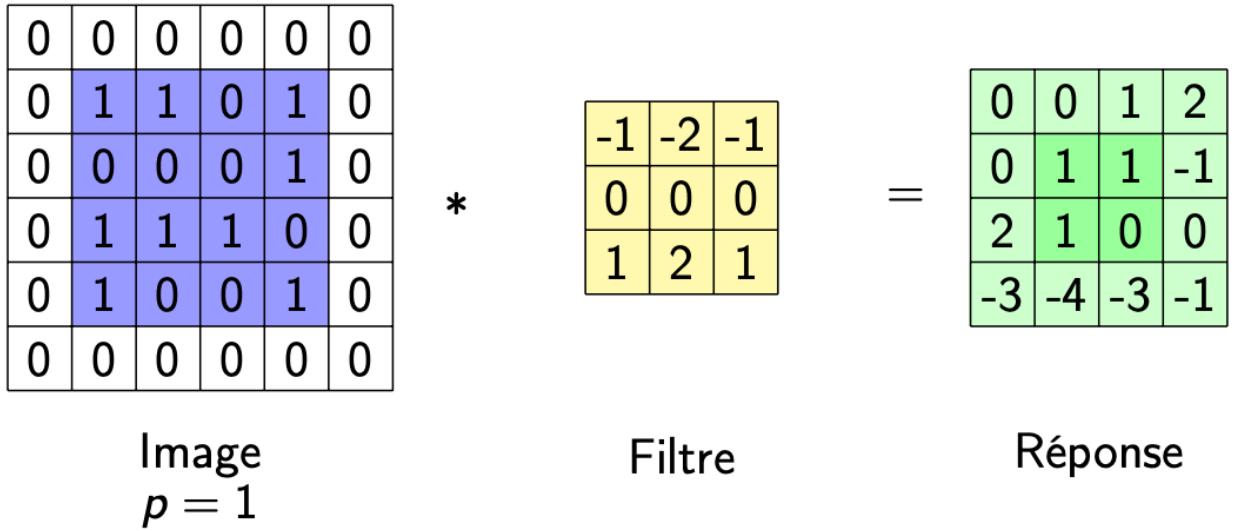
Réponse
$4 \times 4 \times 2$

Couche de Convolution

- Chaque pixel de l'image en entrée est une entrée du réseau (X)
- Les poids connectent ces entrées aux filtres
  - Donc: chaque « pixel » d'un filtre est une unité du réseau
- La taille des filtres détermine la taille de la couche de convolution
- Il y a 2 types de paramètres dans une couche de convolution :
  - Les coefficients des filtres de convolution : il y en a donc w x h x #canaux x #filtres
  - Les biais additionnés à la réponse des filtres de convolution, avant l'application de la fonction d'activation. Il y a exactement un biais par filtre de convolution.
- Ainsi dans l'exemple ci-dessus, il y a 3x3x3x2 + 2 = 56 paramètres.

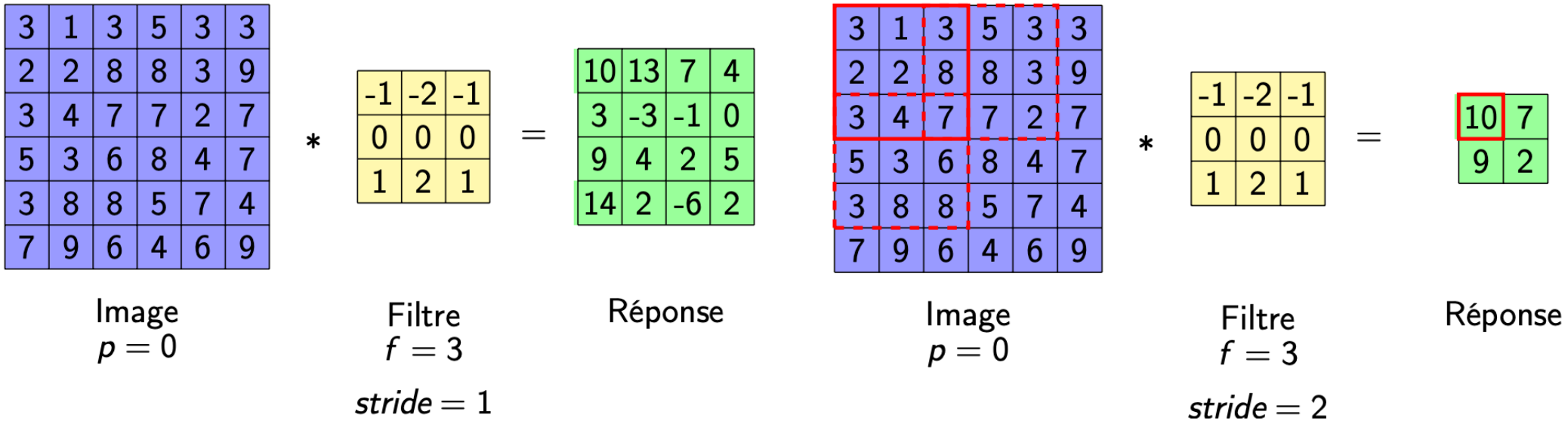
Padding

- Ajout de zéros (zero-padding) aux bords.
- Permet par exemple d'obtenir une réponse de même dimension que l'image d'entrée, ce qui facilite le chaînage des couches de convolution dans un réseau de neurones.



Stride

- Pas du filtre



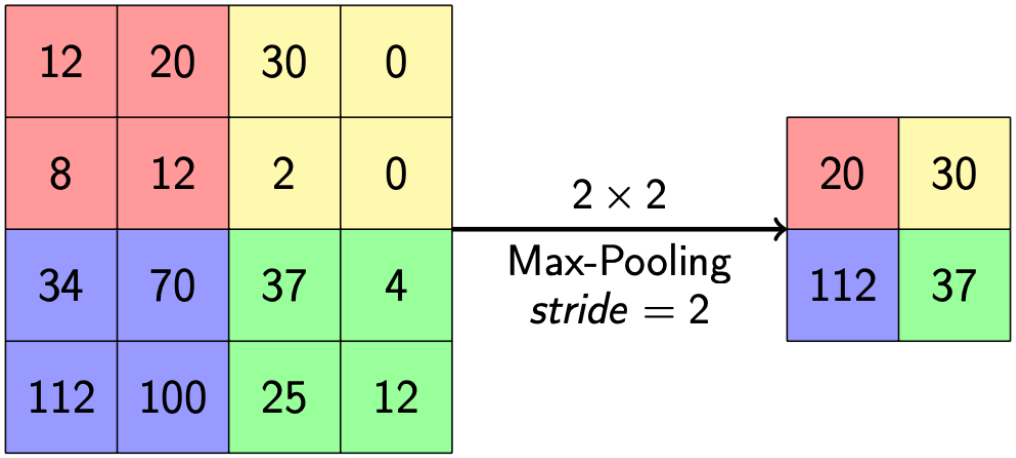
Effet du stride sur les dimensions

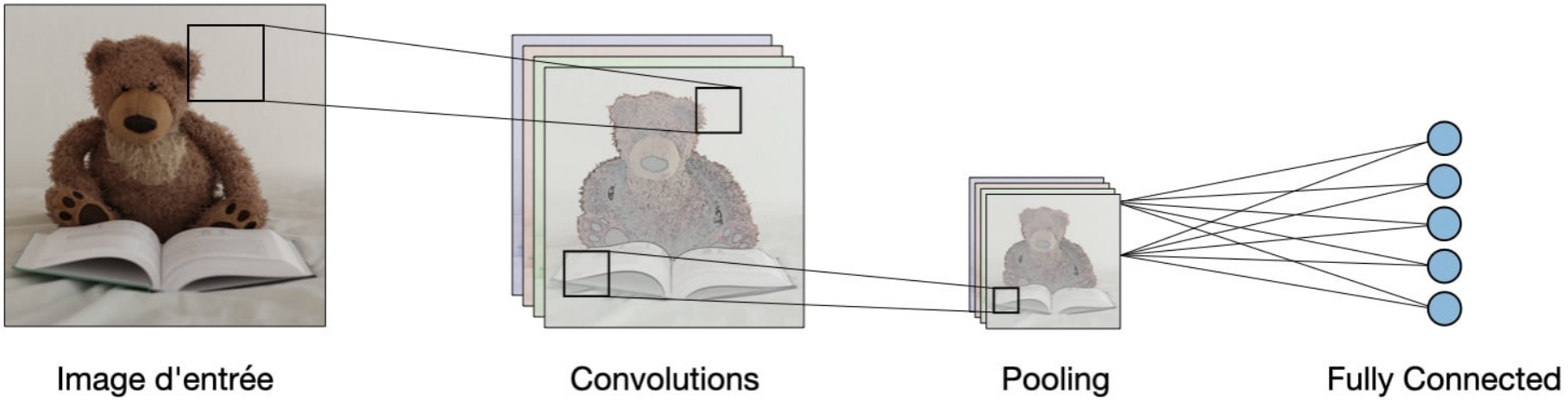
- Étant donnés :
  - une image I en niveaux de gris (un seul canal couleur) de dimension w × h,
  - un filtre K de dimension f ,
  - avec un padding p et
  - un stride s
- Alors la dimension de la réponse I * K de l'image I au filtre K est :

$$\lfloor \frac{w + 2p - f}{s} + 1 \rfloor \times \lfloor \frac{h + 2p - f}{s} + 1 \rfloor$$

Pooling

- Permet de réduire la dimension des tenseurs, pour contrebalancer la multiplication des réponses aux filtres de convolution.
- Préserve les hautes réponses des filtres de convolution.
- Introduit une invariance à la translation.
- Pas de paramètres à apprendre !
- 2 types: max et mean (moyenne)



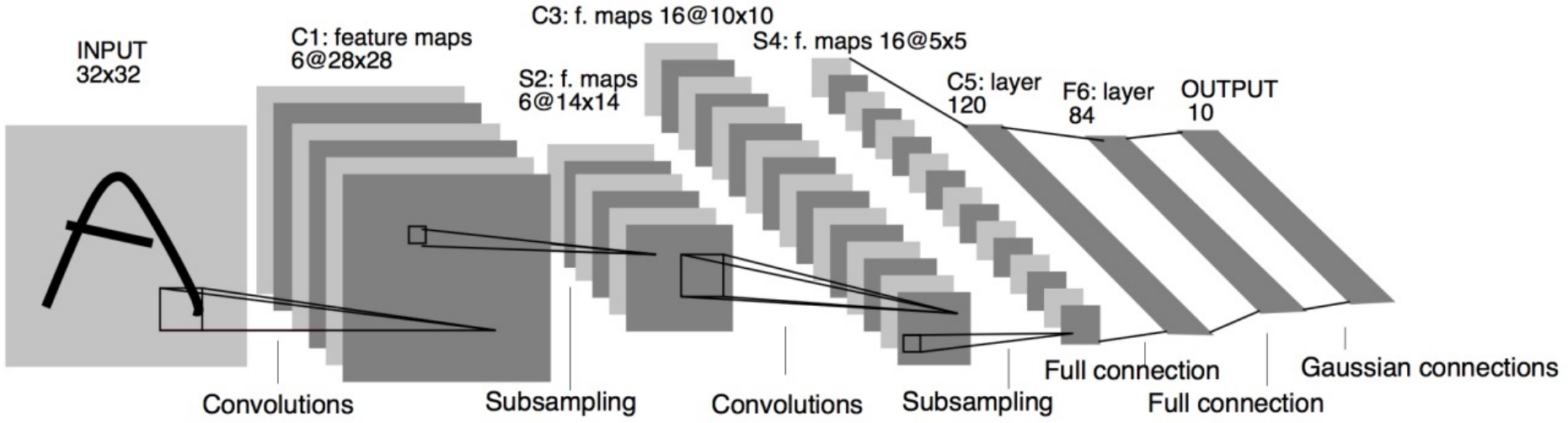


Architecture d'un réseau convolutif (CNN)

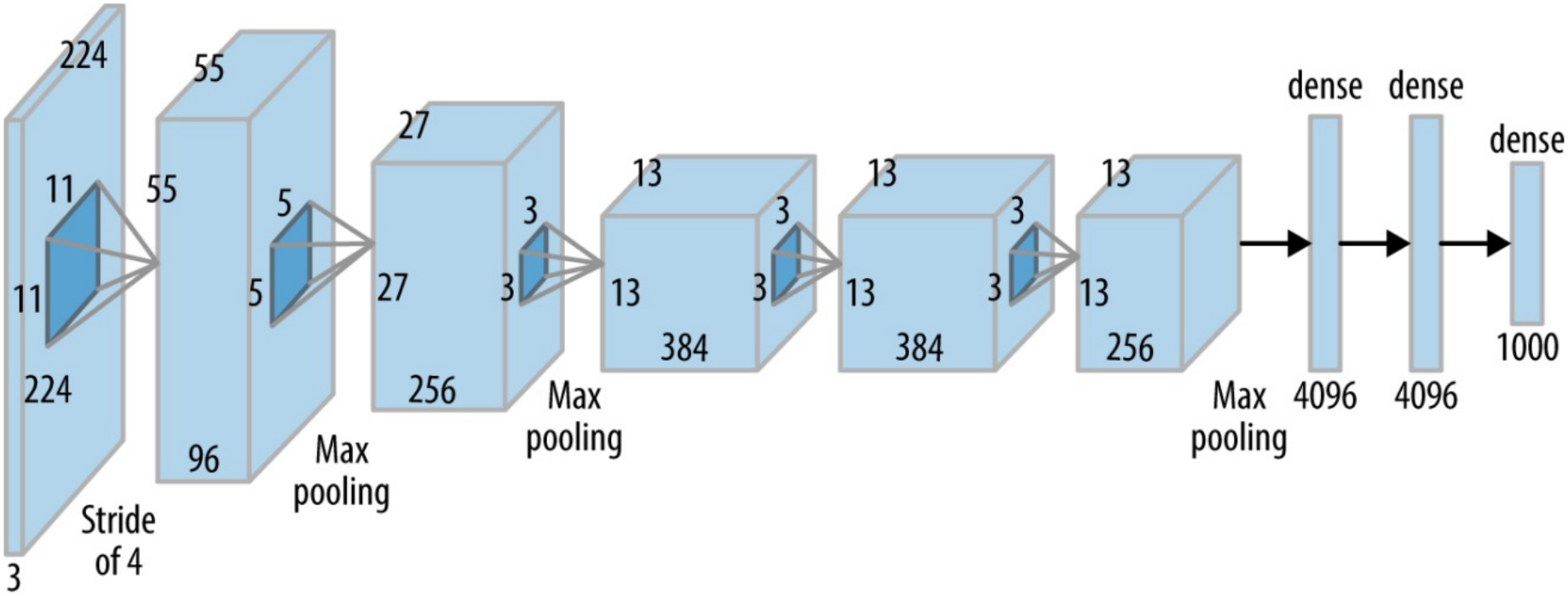
- On trouve 3 types de couches dans un réseau de neurones convolutif typique :
- Des couches de convolution, combinées à des couches de pooling dans les premières couches du réseau.
- Des couches complètement connectées dans les dernières couches du réseau.

https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels

LeNet (Yann Le Cun, 1998)



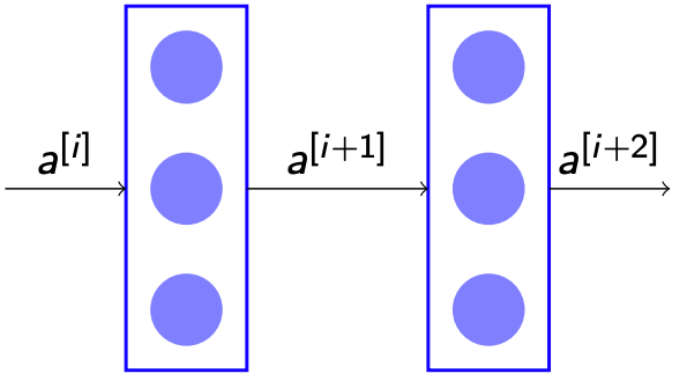
AlexNet (2012)



Toujours plus profond ?

- Plus un réseau est profond, plus il est puissant… mais plus il est difficile à entraîner
- Le signal d'erreur s'affaiblit en remontant les couches (« évanescence du gradient »)
- Solution : des **raccourcis** qui sautent des couches (blocs résiduels) : y = F(x) + x, le bloc n'apprend que la correction F(x) à apporter à x
- C'est ce qui a permis ResNet, avec 152 couches

Blocs résiduels



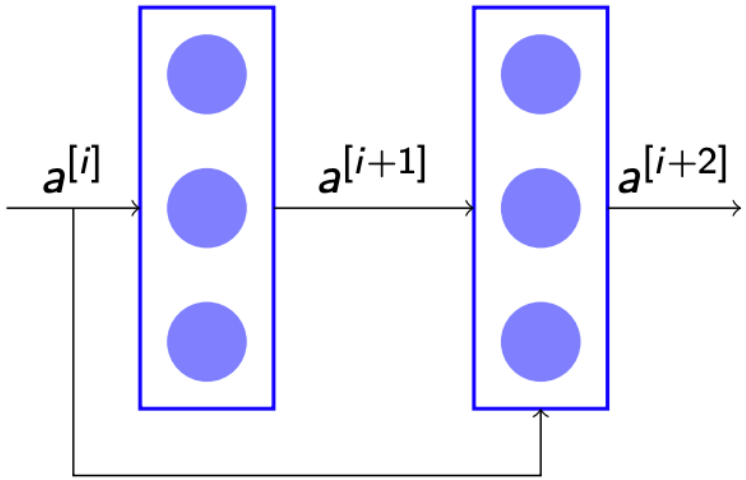
Cette vue abstrait les liaisons synaptiques entre les 2 couches $i+1$ et $i+2$.
On a ici par exemple :

$$a^{[i+1]} = \texttt{ReLU}(W^{[i+1]}a^{[i]} + b^{[i+1]})$$

et

$$a^{[i+2]} = \texttt{ReLU}(W^{[i+2]}a^{[i+1]} + b^{[i+2]})$$

Blocs résiduels



On peut ajouter une connexion comme présenté ci-dessus (*skip connection*) pour former un **bloc résiduel**, d'équation :

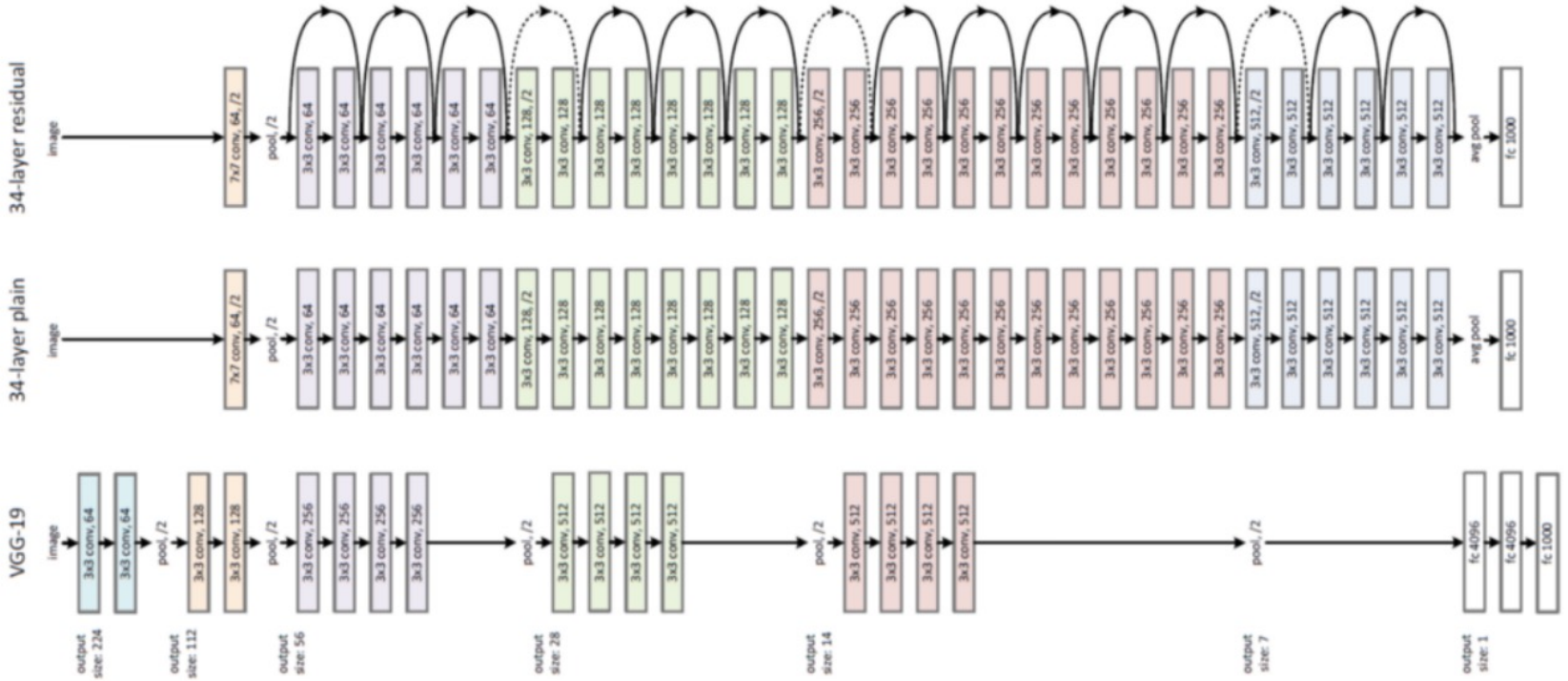
$$a^{[i+2]} = \texttt{ReLU}(W^{[i+2]}a^{[i+1]} + b^{[i+2]} + a^{[i]})$$

Blocs Résiduels

- Une condition importante est que les dimensions de $a^{[i]}$ et $a^{[i+2]}$ soient identiques !
- On utilise donc souvent des convolutions qui conservent la dimension dans les réseaux résiduels.
- Alors que l'entraînement vise d'ordinaire à estimer une fonction F telle que **y** = F(x), on cherche ici la fonction telle que : **y** = F(x) + x.
- En d'autres termes, la fonction F estime le résidu **y** − x (d'où le nom de bloc résiduel).

ResNet

- Réseau qui a remporté le challenge ImageNet en 2015 avec 152 couches



6. Bien entraîner un modèle

Optimisation, régularisation, dropout

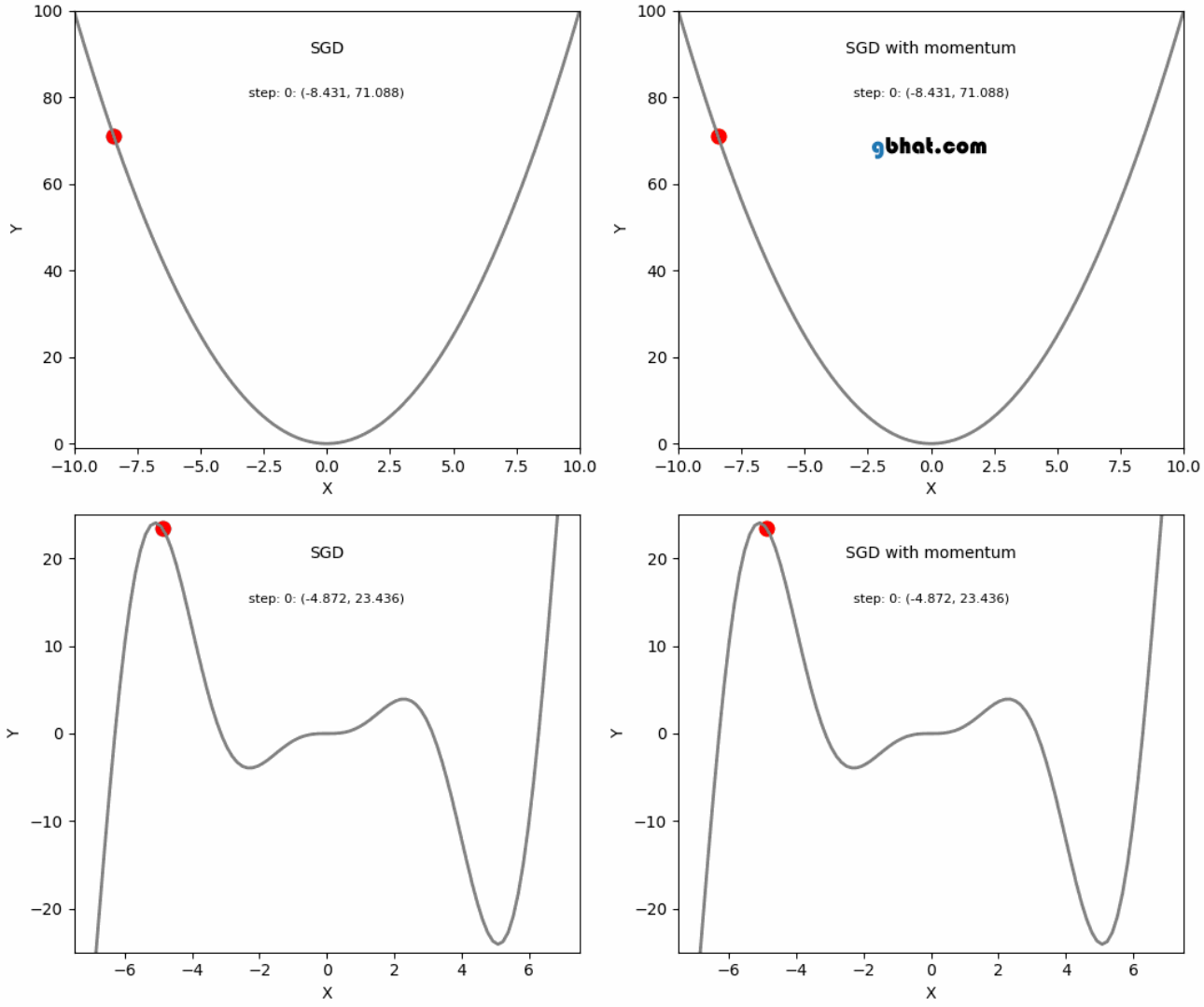
Optimisation

- En général, on parle d'optimisation quand on a les trois éléments suivants:
  - Une **fonction objectif** à minimiser ou maximiser
  - Des **variables de décision** qui déterminent le « comportement » de la fonction objectif
  - Des **contraintes:** conditions qui doivent être satisfaites par les variables de décision pour que les valeurs correspondantes soient acceptables
- Dans le contexte des réseaux de neurones, on parle souvent d'optimisation pour la descente au gradient
  - stratégies pour choisir le bon « learning rate » α

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla f(\boldsymbol{w})$$

Momentum

- Le principe général du **momentum** (ou accélération) est d'utiliser de l'information des itérations précédentes pour choisir un meilleur pas



Gradient accéléré (gradient de Nesterov)

- Le gradient n'est pas évalué en le point courant w_k, mais en une combinaison linéaire de ce point avec le pas précédent $w_k - w_{k-1}$
- Ce second terme est appelé terme de *momentum*
- Donc on réécrit la descente au gradient comme suit:

$$\begin{cases} \boldsymbol{w}_{k+1} &= \boldsymbol{z}_k - \alpha_k \nabla f(\boldsymbol{z}_k) \\ \boldsymbol{z}_{k+1} &= \boldsymbol{w}_{k+1} + \beta_{k+1}(\boldsymbol{w}_{k+1} - \boldsymbol{w}_k). \end{cases}$$

- Pour commencer on met $z_0 = w$
- A noter que maintenant on a un deuxième hyperparamètre, $\boldsymbol{\beta_k}$

AdaGrad

- Méthode qui prend en compte l'histoire du gradient
  - Le gradient est « accumulé » dans une variable v^w_t
- (notation utilisée dans la doc PyTorch avec η pour le learning rate)

$$v_t^w = v_{t-1}^w + (\nabla w_t)^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t^w + \epsilon}} * \nabla w_t$$

- Utile pour des problèmes avec gradients parcimonieux («sparse» en anglais), pour lesquels les gradients ont tendance à avoir beaucoup de composantes nulles
- Dans une telle situation, on peux ajuster le pas uniquement pour les coordonnées non nulles

RMSProp

- RMSProp (Root Mean Square Propagation) procède de manière similaire à AdaGrad en jouant sur les composantes du gradient

$$v_t^w = \beta * v_{t-1}^w + (1 - \beta)(\nabla w_t)^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t^w + \epsilon}} * \nabla w_t$$

- On voit donc que la méthode peut donner plus de poids aux gradients précédents (grâce au paramètre β) plutôt qu'au gradient de l'itération : cette idée permet aux longueurs de pas de décroître moins rapidement que celles de AdaGrad

Adam

- (ADAptive Moment estimation) peut être vu comme combinant le concept de momentum avec celui d'accumulation d'information sur les gradients précédents utilisés par les algorithmes précédents

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla w_t$$

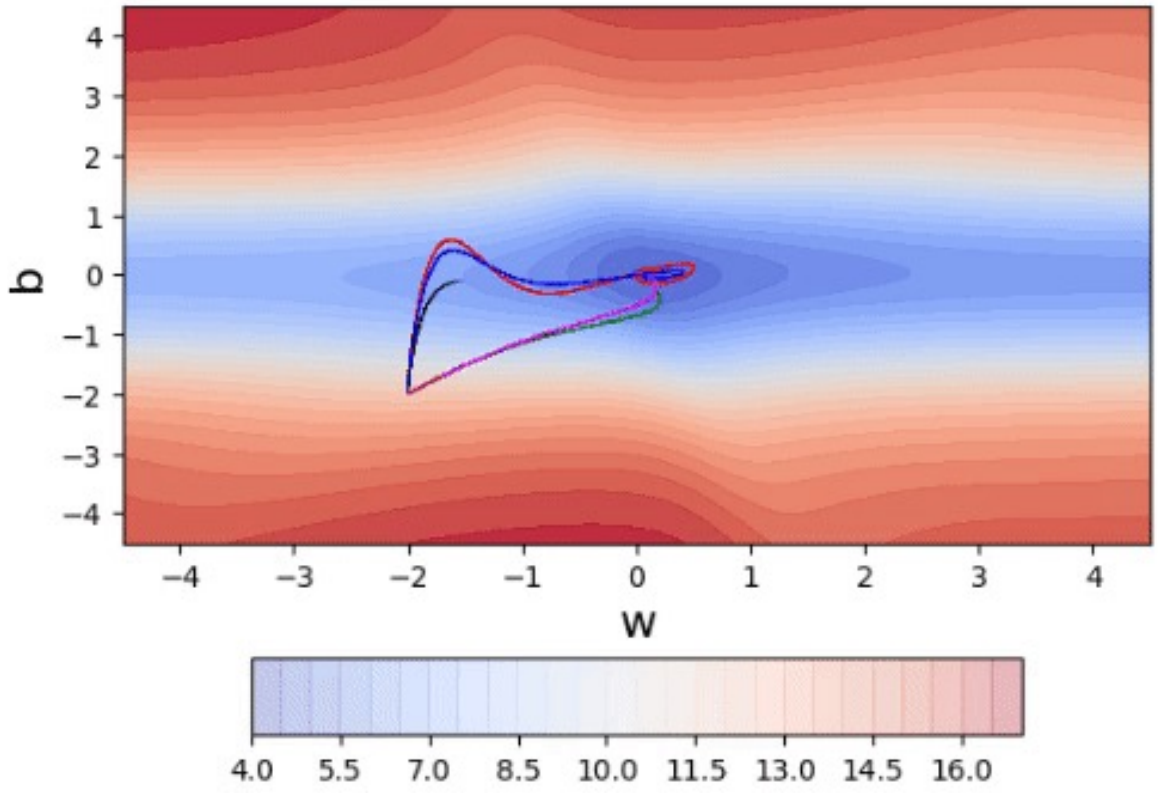
$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * (\nabla w_t)^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} * \hat{m}_t$$

Adam

- Algorithme le plus performant sur les tâches de Traitement Automatique de la Langue
- Paramètres typiques:
  - $\beta_1 = 0.9$, $\beta_2 = 0.999$
  - $\eta = 0.001$ ou 0.0001



Régularisation

- But : éviter que le modèle apprenne « par cœur » (surapprentissage)
- Principe : on pénalise les modèles trop compliqués
- **Lecture de la formule** (à droite) : on minimise erreur f(w) + λ × complexité Ω(w) ; λ règle la force de la pénalité
- **Exemple :** régularisation L2 (weight decay) ; on empêche les poids de devenir trop grands

Analogie

Mieux vaut une explication simple qui marche presque toujours qu'une explication compliquée qui ne marche que sur les exemples déjà vus.

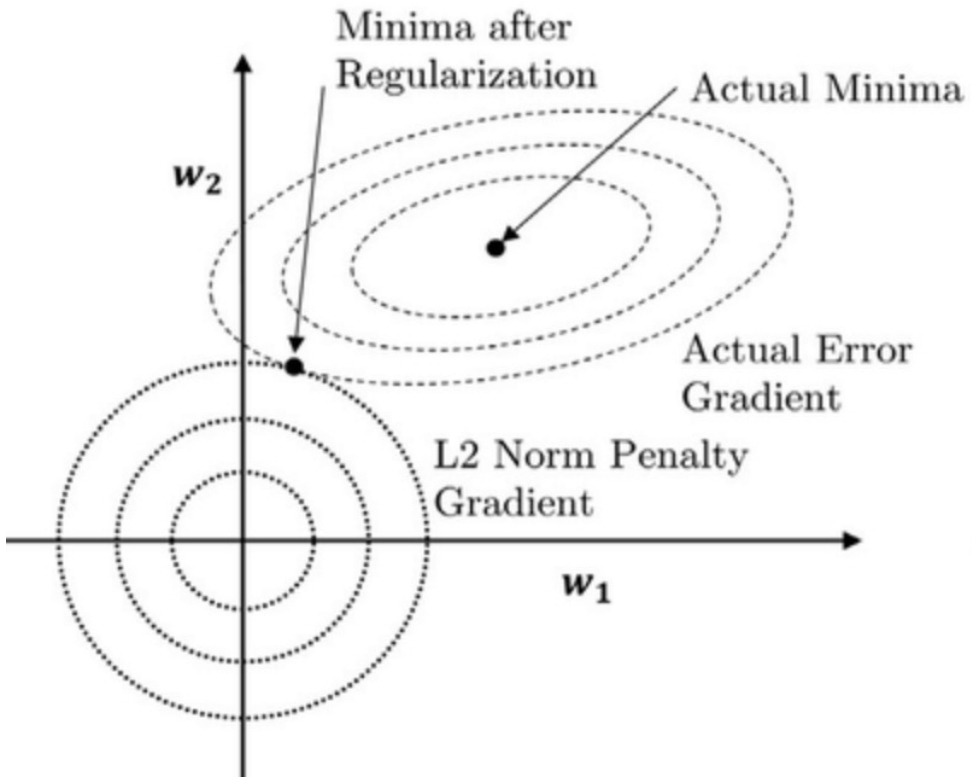
$$\underset{\boldsymbol{w}\in\mathbb{R}^d}{\text{minimiser}}\ f(\boldsymbol{w}) \quad + \lambda\Omega(\boldsymbol{w}).$$

Régularisation L2 (Ridge)

- Le terme de régularisation est la norme carrée des poids divisé par 2:

$$\underset{\boldsymbol{w}\in\mathbb{R}^d}{\text{minimiser}}\, f(\boldsymbol{w}) + \frac{\lambda}{2}\|\boldsymbol{w}\|^2.$$

- Technique aussi appelé **weight decay** dans les réseaux de neurones



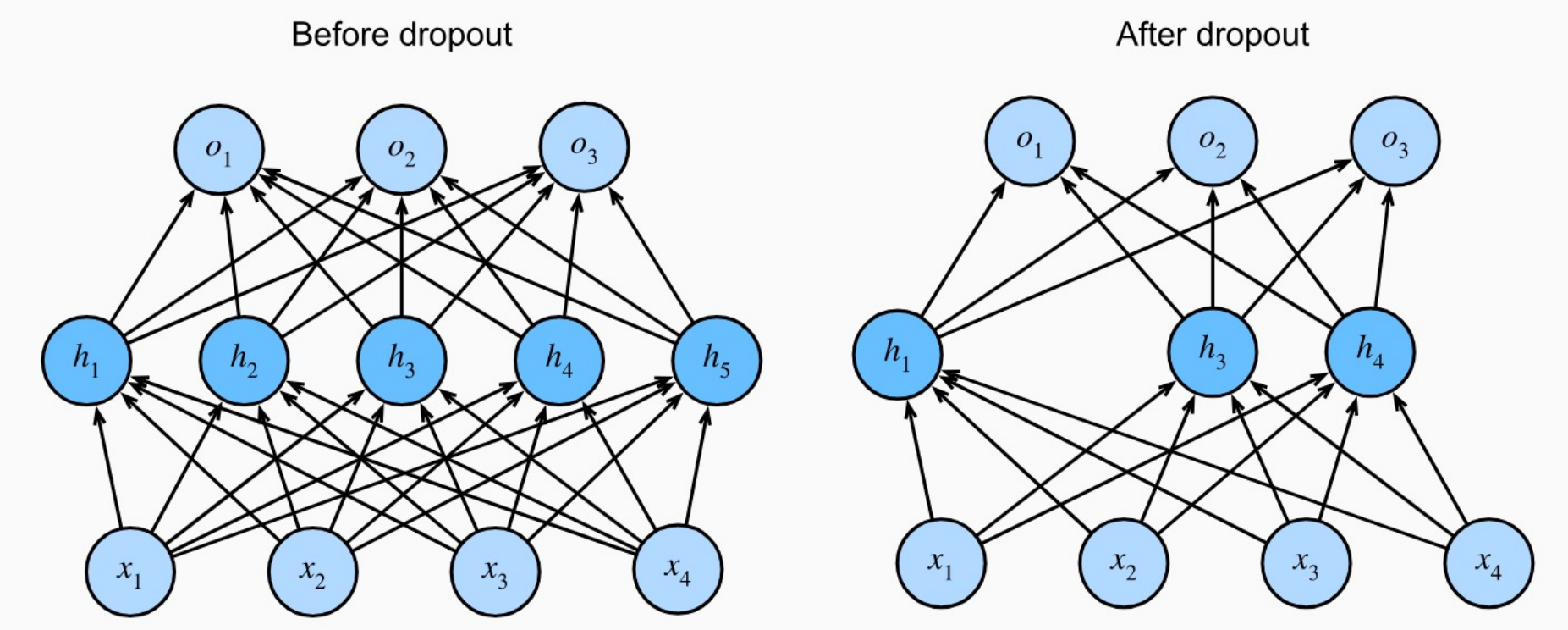
Dropout

- Technique propre aux réseaux de neurones
- Pendant l'entraînement, on « éteint » au hasard une partie des neurones
- Le réseau ne peut plus compter sur un seul neurone : il apprend des indices plus variés et plus généraux

Analogie

Une équipe où chacun doit pouvoir remplacer un collègue absent devient plus solide.

Dropout

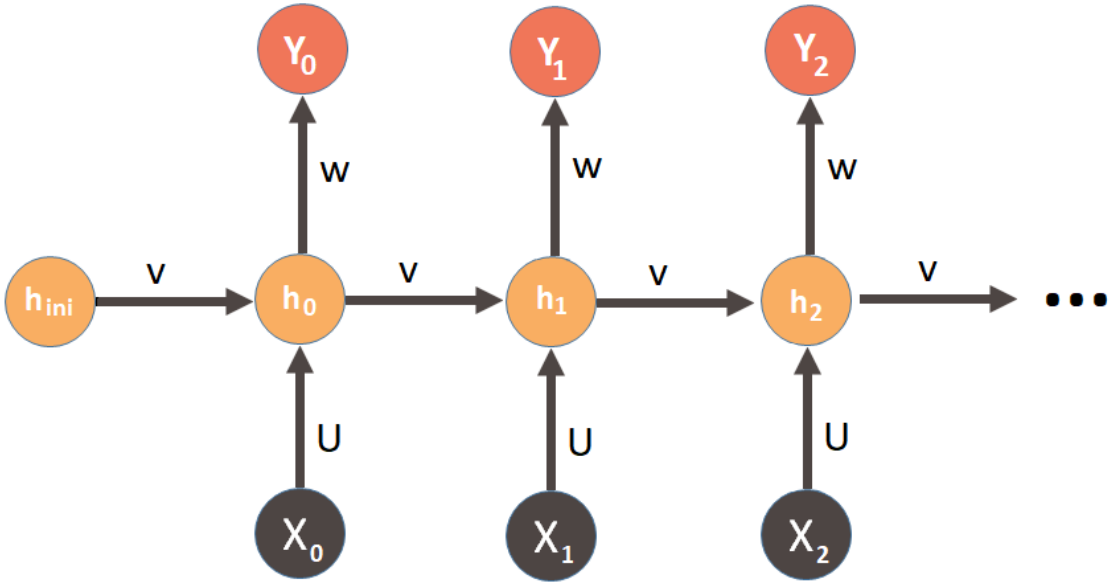

Before dropout
o_1
o_2
o_3
h_1
h_2
h_3
h_4
h_5
x_1
x_2
x_3
x_4
After dropout
o_1
o_2
o_3
h_1
h_3
h_4
x_1
x_2
x_3
x_4

7. Réseaux récurrents

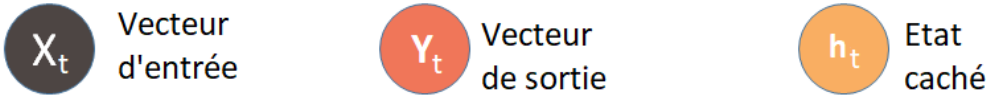
Traiter des séquences : texte, son, gestes

Réseaux récurrents (RNN)

- Pour les données **en séquence** : texte, parole, musique, trajectoire de la souris
- Le réseau garde une « mémoire » h_t : elle combine l'entrée x_t et la mémoire précédente h_{t-1} (formule ci-dessous)
- **Exemple :** le clavier du smartphone qui propose le mot suivant



$$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$

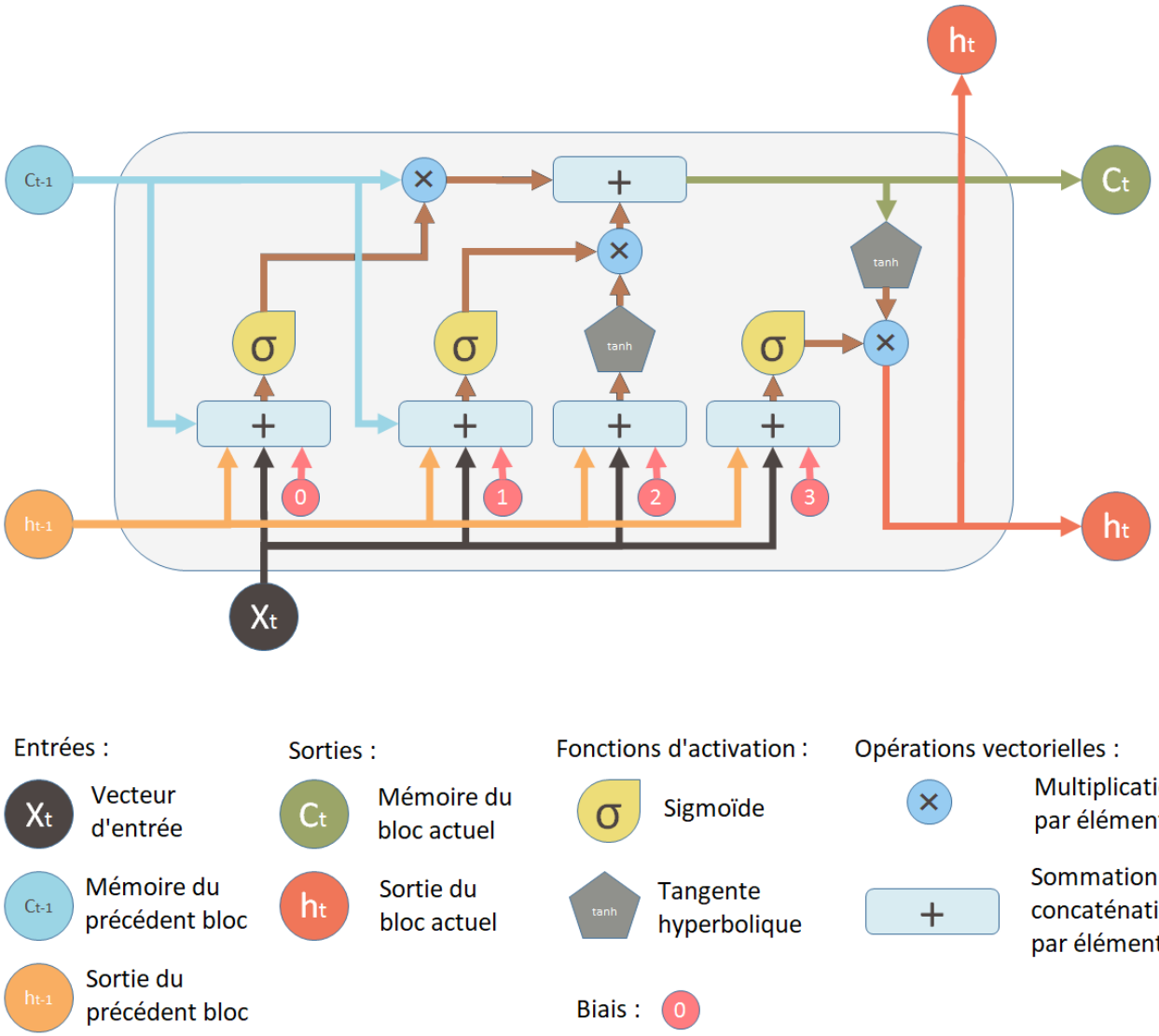


RNNs

- Problème: lorsque la séquence à traiter est trop longue, la rétropropagation du gradient de l'erreur peut soit devenir beaucoup trop grande et exploser, soit au contraire devenir beaucoup trop petite.
- Les LSTMs (Long Short-Term Memory) de Hochreiter et Schmidhuber proposent une solution à ce problème en prenant en compte un vecteur mémoire via un système de portes (gates) et d'états.

LSTMs

- Portes (gates):
  - **Forget** Gate (capacité à oublier de l'information, quand celle-ci est inutile)
  - **Input** Gate (capacité à prendre en compte de nouvelles informations utiles)
  - **Output** Gate (quel est l'état de la cellule à l'instant t sachant la forget gate et la input gate)
- Etats
  - Hidden state (état caché)
  - Cell state (état de la cellule)



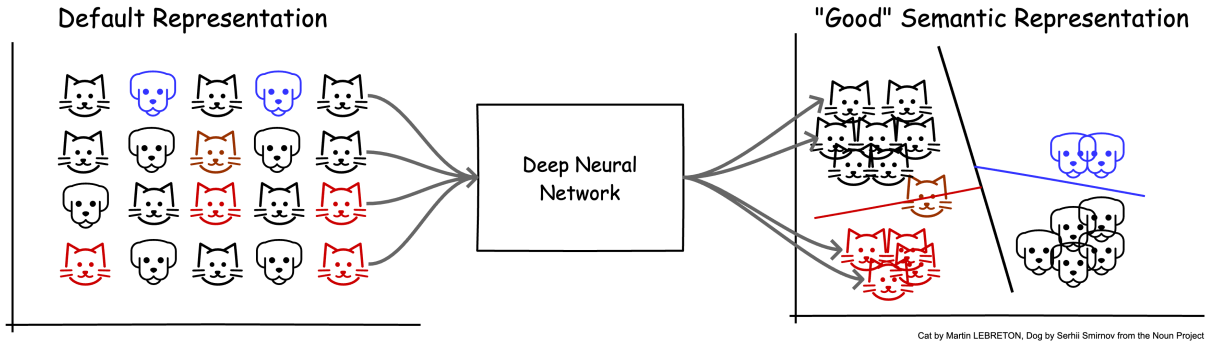
https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html#torch.nn.LSTM

8. Apprendre des représentations

Embeddings, auto-encodeurs, GANs

Apprentissage de représentations

- (En anglais: « representation learning »)
- Objectif: transformer des données (brutes – pixels, mots) en entrée dans des formes facilement « manipulables » (normalement pour les grouper)
- Aussi d'une façon qui préserve leurs caractéristiques sémantiques
  - **Exemple**: images en tant que bitmap -> vecteurs résultat d'une transformation par une CNN entraînée à classifier les images

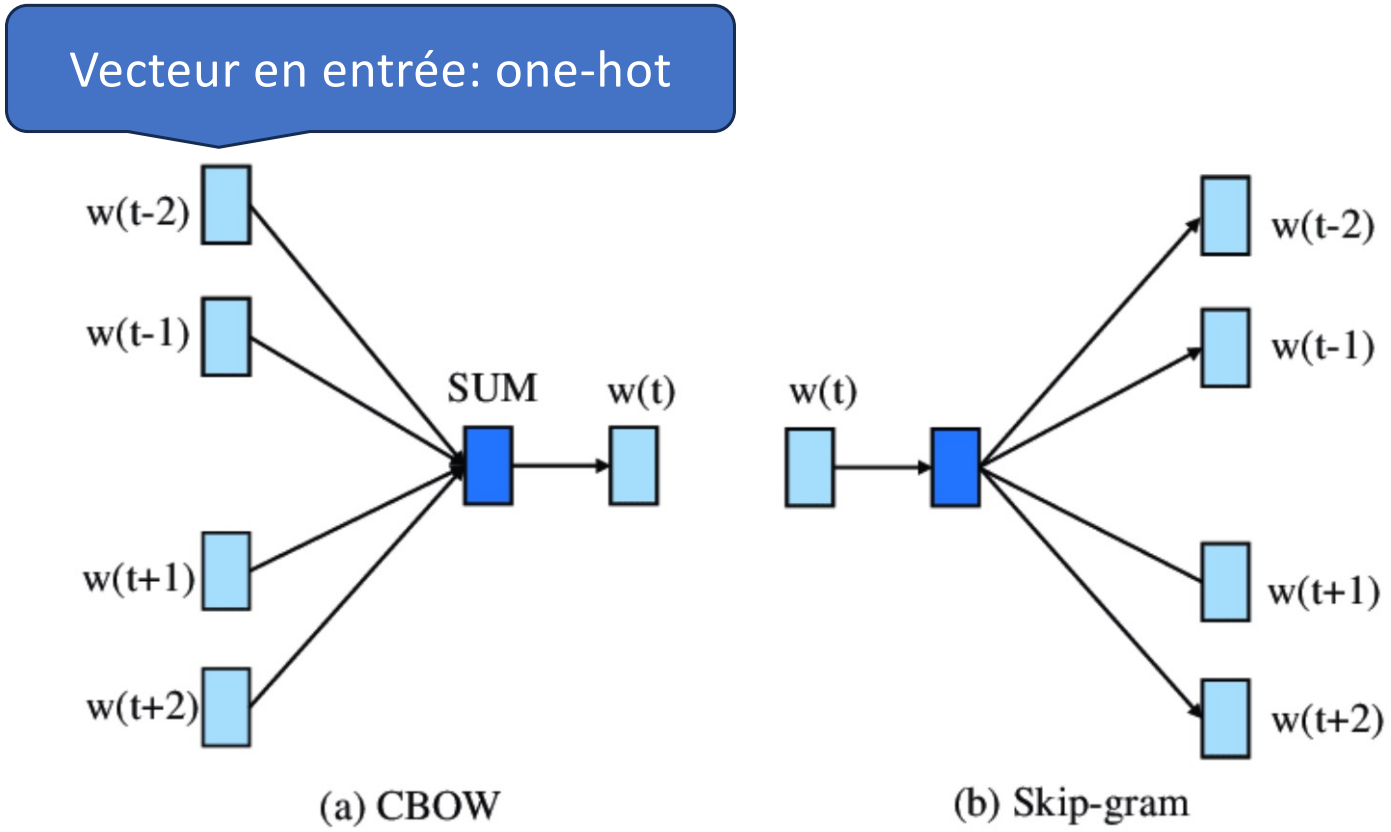


Word2vec (Mikolov, 2013)

- Algorithme pour transformer des mots dans des vecteurs « sémantiques »
  - **Word embeddings**
- Il se base sur l'idée d'une **sémantique distributionnelle**
  - C'est-à-dire, si deux mots apparaissent dans des contextes similaires, alors ils sont sémantiquement similaires
  - Exemple:
    - Phrase 1 : « J'ai acheté une **pomme** à l'épicerie et maintenant je vais la manger. »
    - Phrase 2 : « Je vais manger la banane que j'ai acheté à l'épicerie. »
  - (donc *pomme* est similaire à *banane* car on peut manger les deux et les acheter au même endroit)

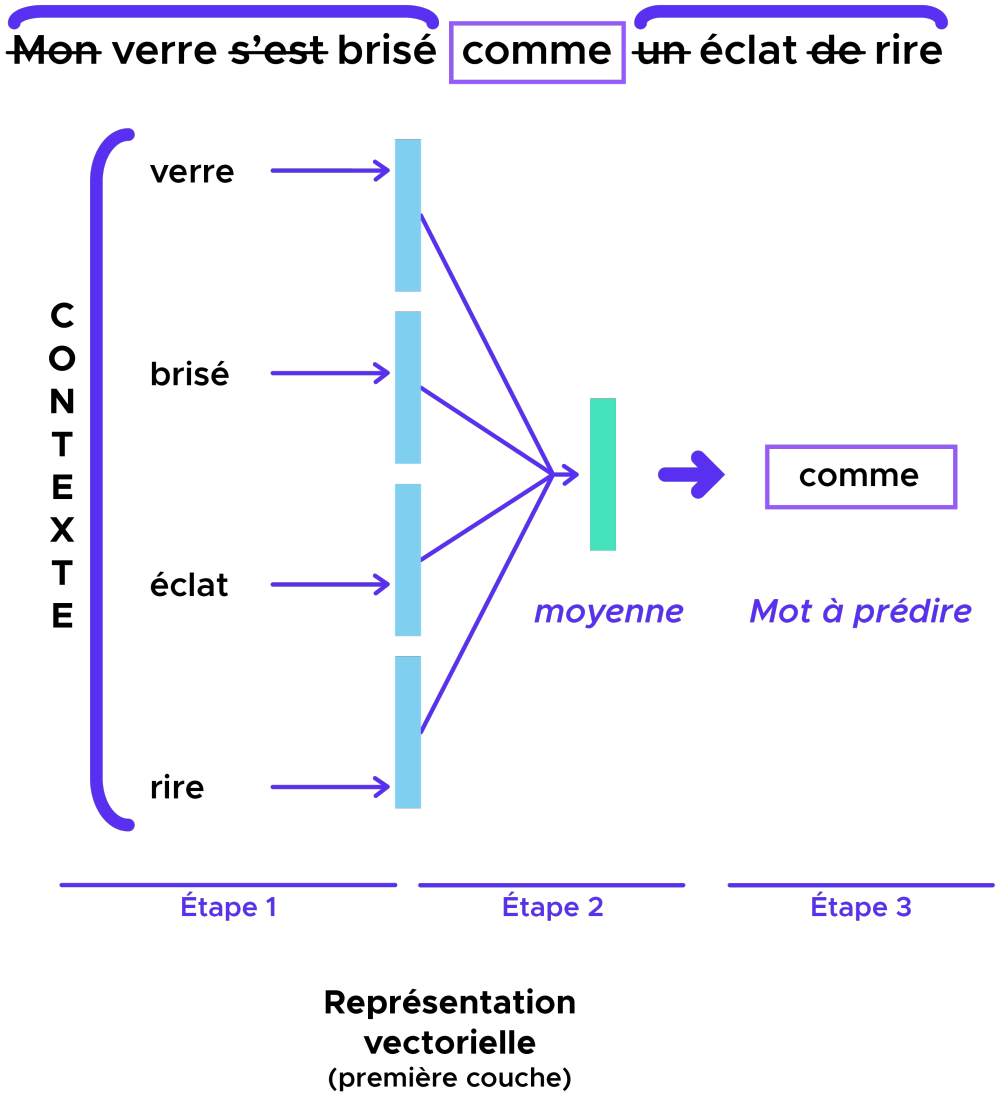
word2vec

- Il existe deux variantes du Word2vec:
  - Common Bag Of Words (CBOW) et Skip-gram
- Les deux utilisent un réseau de neurones avec une couche cachée:



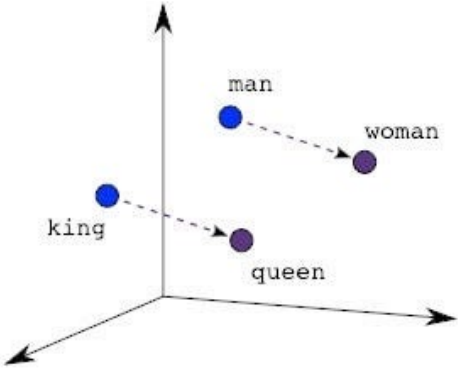
CBOW : Le modèle est nourri par le contexte, et prédit le mot cible. Le résultat de la couche cachée est la nouvelle représentation du mot

Skip Gram : Le modèle est nourri par le mot cible, et prédit les mots du contexte. Le rés
de la couche cachée est la nouvelle représentation du mot

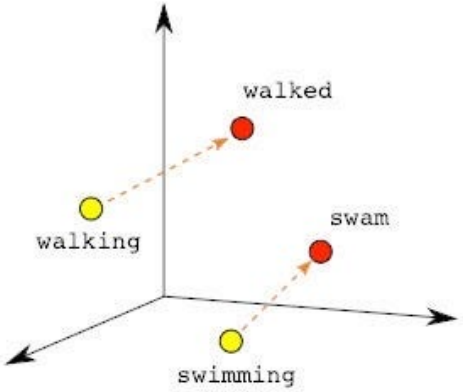


Représentation vectorielle
(première couche)

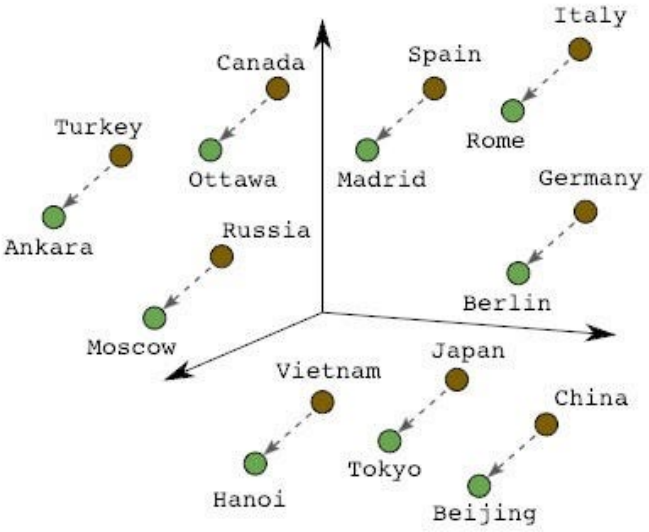
Word2vec - propriétés



Male-Female



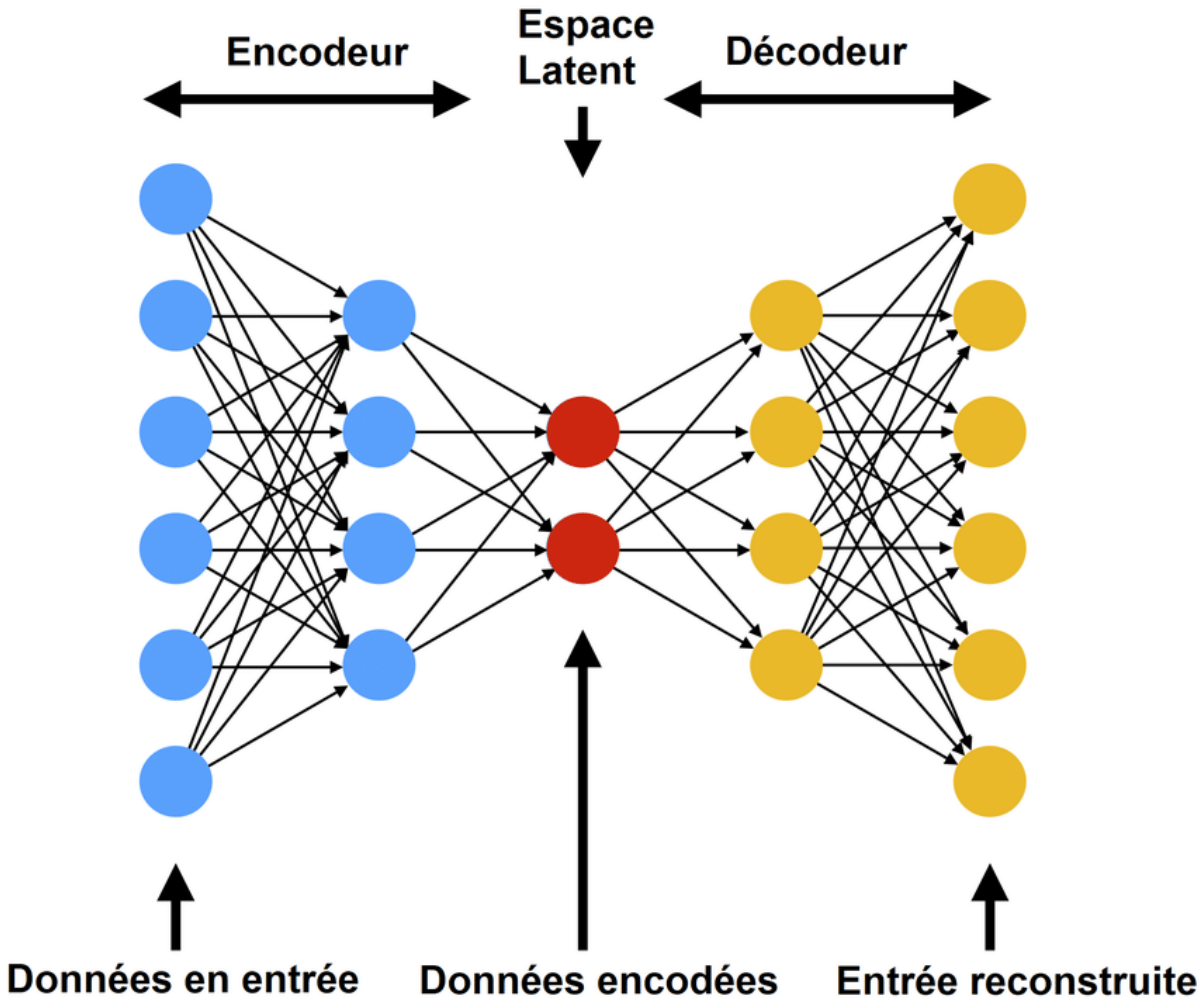
Verb Tense



Country-Capital

Auto-encodeurs

- Réseaux de neurones où la sortie est de la même taille que l'entrée
- Dont le but est d'apprendre une représentation **latente**
- Très fréquemment utilisés en génération d'images
- Architecture typique:
- Fonction à apprendre: $h(x, W, b) \approx x$



Fonction de perte (loss) d'un auto-encodeur

- Encodage:

$$\mathbf{z} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

- Reconstruction (décodage):

$$\mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{z} + \mathbf{b}')$$

- Perte: minimisation de l'erreur de reconstruction

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2$$

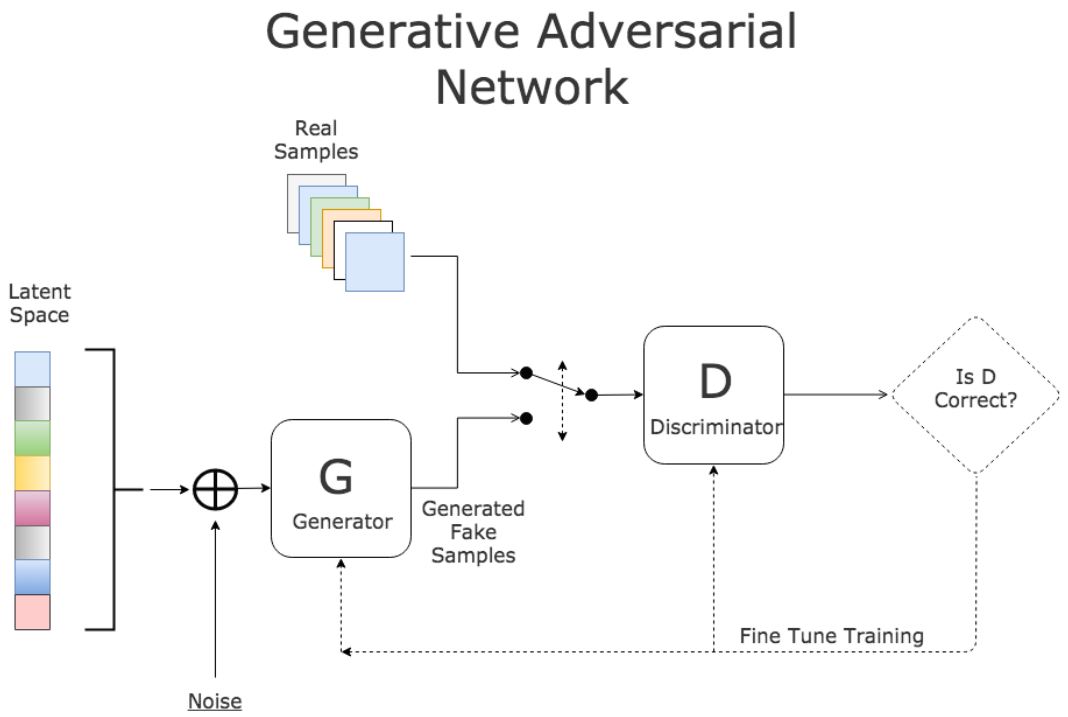
Variantes

- Auto-encodeur **débruiteur**:
  - Un auto-encodeur débruiteur prend une entrée partiellement corrompue et apprend à récupérer l'entrée originale débruitée
- Auto-encodeur **épars**:
  - En imposant la parcimonie sur les unités cachées durant l'apprentissage (tout en ayant un plus grand nombre d'unités cachées que d'entrées), un auto-encodeur peut apprendre des structures utiles dans les données d'entrées
- Auto-encodeur variationnel (VAE):
  - Il utilise l'approche variationnelle pour l'apprentissage de la représentation latente, ce qui se traduit par une composante de perte additionnelle et un algorithme d'apprentissage spécifique fondé sur un estimateur bayésien variationnel du gradient stochastique

$$\mathcal{L}(\phi, \theta, \mathbf{x}) = D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big(\log p_\theta(\mathbf{x}|\mathbf{z})\big)$$

GAN (Generative Adversarial Network)

- (Réseaux antagoniste génératif)
- Un GAN est un modèle génératif où deux réseaux de neurones sont placés en compétition dans un scénario de théorie des jeux.
  - Le premier réseau est le **générateur**, il génère un échantillon (ex. une image)
  - Son adversaire, le **discriminateur** essaie de détecter si un échantillon est réel ou bien s'il est le résultat du générateur.
- Ainsi, le générateur est entraîné avec comme but de tromper le discriminateur



9. IA générative : le langage

Des n-grammes aux LLM, génération de code

Les modèles de langage

- Nous permettent de savoir si une chaine de caractères appartient ou pas à un langage
- Anciennement (1960 - …): modèles basés sur des **grammaires**
  - Ex: Context-Free Grammars (N.Chomsky)
- Mais les grammaires ne sont pas suffisantes:
- Frases grammaticales mais pas logiques:
  - Le verre boit de l'eau
- Frases non grammaticales mais quand même compréhensibles:
  - Aujourd'hui moi manger pasta

Les modèles de langage probabilistes

- Idée: la probabilité d'observer des phrases correctes (dans un grand corpus de textes) est **supérieure** à la probabilité d'observer des phrases incorrectes

- Traduction:
  - La lumière est très forte -> … en anglais ?
  - p(« light is very bright ») > p(« light is very loud »)
- Orthographie:
  - p(« j'arrive dans 5 minutes ») > p(« j'arrive dans 5 menuets »)

Les modèles de langage probabilistes

- Pour calculer les probabilités il faut **compter** combien de fois on observe une phrase dans un corpus
- On obtient la probabilité en divisant le nombre d'occurrences de la phrase par le nombre de documents dans le corpus
- Plus le corpus est grand, meilleure sera l'estimation de la probabilité
- Problème: la fréquence d'une phrase diminue sensiblement si la phrase est longue
  - -> Modèles à n-grammes

Modèles à n-grammes: un exemple

- Nous allons **générer du texte** avec un modèle a **2-grammes**
- D'abord il nous faut un corpus où calculer les probabilités

- La petite ville de Verrières peut passer pour l'une des plus jolies de la Franche-Comté. Ses maisons blanches avec leurs toits pointus de tuiles rouges, s'étendent sur la pente d' une colline, dont des touffes de vigoureux châtaigniers marquent les moindres sinuosités. Le Doubs coule à quelques centaines de pieds au-dessous de ses fortifications, bâties jadis par les Espagnols, et maintenant ruinées.
- Verrières est abritée du côté du nord par une haute montagne, c'est une des branches du Jura. Les cimes brisées du Verra se couvrent de neige dès les premiers froids d'octobre. Un torrent, qui se précipite de la montagne, traverse Verrières avant de se jeter dans le Doubs, et donne le mouvement à un grand nombre de scies à bois ; c'est une industrie fort simple et qui procure un certain bien-être à la majeure partie des habitants plus paysans que bourgeois.

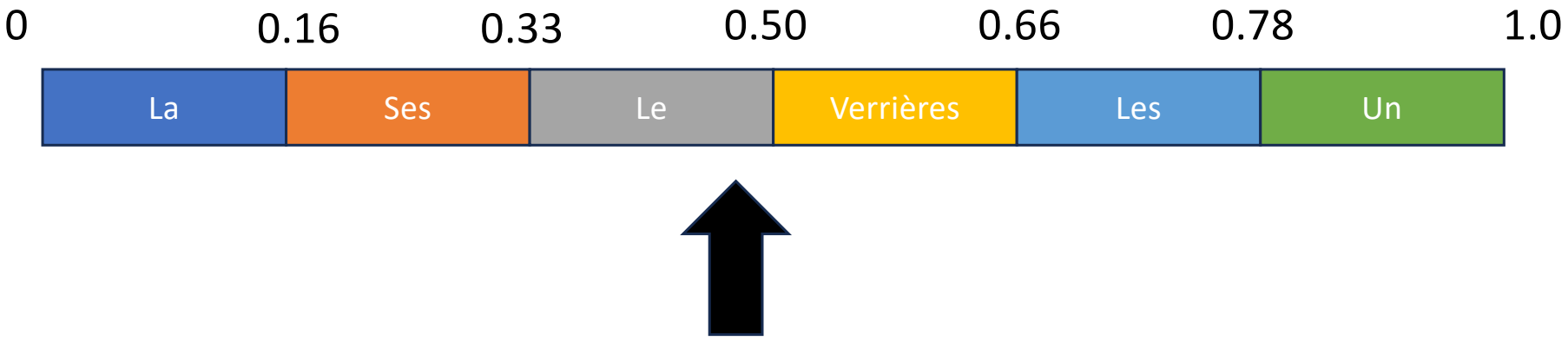
Générer du texte

- 6 possibilités différentes pour commencer une phrase:

- **La** petite ville de Verrières peut passer pour l'une des plus jolies de la Franche-Comté. **Ses** maisons blanches avec leurs toits pointus de tuiles rouges, s'étendent sur la pente d' une colline, dont des touffes de vigoureux châtaigniers marquent les moindres sinuosités. **Le** Doubs coule à quelques centaines de pieds au-dessous de ses fortifications, bâties jadis par les Espagnols, et maintenant ruinées.
- **Verrières** est abritée du côté du nord par une haute montagne, c'est une des branches du Jura. **Les** cimes brisées du Verra se couvrent de neige dès les premiers froids d'octobre. **Un** torrent, qui se précipite de la montagne, traverse Verrières avant de se jeter dans le Doubs, et donne le mouvement à un grand nombre de scies à bois ; c'est une industrie fort simple et qui procure un certain bien-être à la majeure partie des habitants plus paysans que bourgeois.

Générer du texte

Mécanisme d'échantillonage aléatoire simple sur les tokens possibles:



0.09

La

Générer du texte

- Les mots possibles sachant que le dernier mot est « la »:

> **La** **petite** ville de Verrières peut passer pour l'une des plus jolies de **la** **Franche-Comté**. Ses maisons blanches avec leurs toits pointus de tuiles rouges, s'étendent sur la pente d' une colline, dont des touffes de vigoureux châtaigniers marquent les moindres sinuosités. Le Doubs coule à quelques centaines de pieds au-dessous de ses fortifications, bâties jadis par les Espagnols, et maintenant ruinées.
> Verrières est abritée du côté du nord par une haute montagne, c'est une des branches du Jura. Les cimes brisées du Verra se couvrent de neige dès les premiers froids d'octobre. Un torrent, qui se précipite de **la** **montagne**, traverse Verrières avant de se jeter dans le Doubs, et donne le mouvement à un grand nombre de scies à bois ; c'est une industrie fort simple et qui procure un certain bien-être à la majeure partie des habitants plus paysans que bourgeois.

La **montagne** …

Modèles à n-grammes

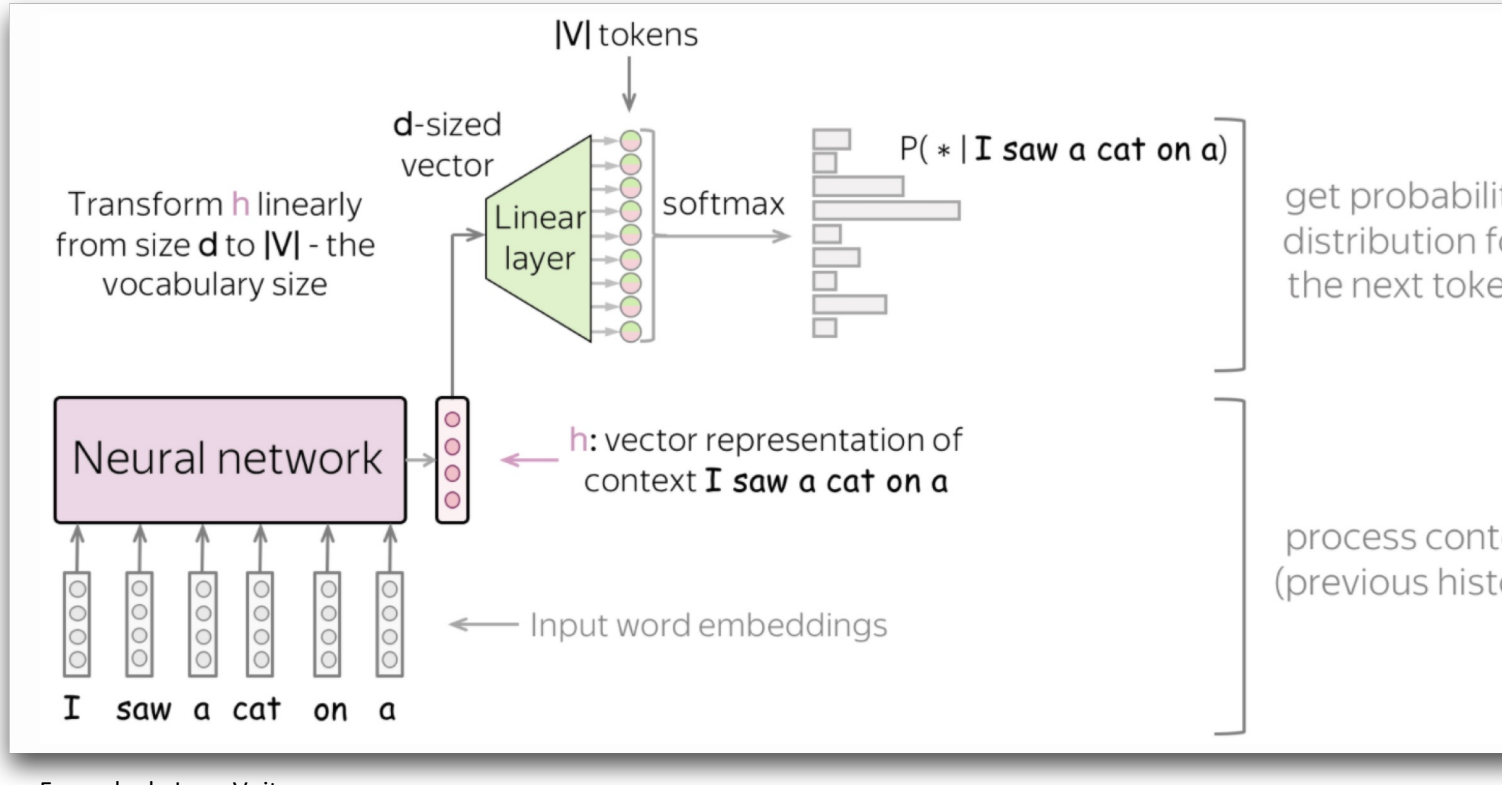
- Mais le résultat de génération successives n'a pas forcément du sens:

> La petite ville de Verrières peut passer pour l'une des plus jolies de la Franche-Comté. Ses maisons blanches avec leurs toits pointus de tuiles rouges, s'étendent sur la pente d' une colline, dont des touffes de vigoureux châtaigniers marquent les moindres sinuosités. Le Doubs coule à quelques centaines de pieds au-dessous de ses fortifications, bâties jadis par les Espagnols, et maintenant ruinées.
> Verrières est abritée du côté du nord par une haute montagne, c'est une des branches du Jura. Les cimes brisées du Verra se couvrent de neige dès les premiers froids d'octobre. Un torrent, **qui se** précipite de **la montagne**, traverse Verrières avant de **se jeter dans** le Doubs**, et** donne **le mouvement** à un grand nombre de scies à bois ; c'est une industrie fort simple **et qui** procure un certain bien-être à la majeure partie des habitants plus paysans que bourgeois.

La montagne , et qui se jeter dans le mouvement …

Modèles de langage neuronaux

- Calculer la probabilité des mots inconnus
  - C'est résolu en utilisant les **embeddings** et une **tokenisation** à niveau de **sous-mots**
  - E.g. Util-ise, -iser, -e …



Exemple de Lena Voita

Playground

Load a preset... | Save | View code | Share | ...

Write a tagline for an ice cream shop.

Looking for ChatGPT? Try it now

Submit

0

text-davinci-003

Temperature 0.9

Maximum length 20

Stop sequences
Enter sequence and press Tab

Top P 1

Frequency penalty 0

Presence penalty 0

Best of 1

Inject start text

Inject restart text

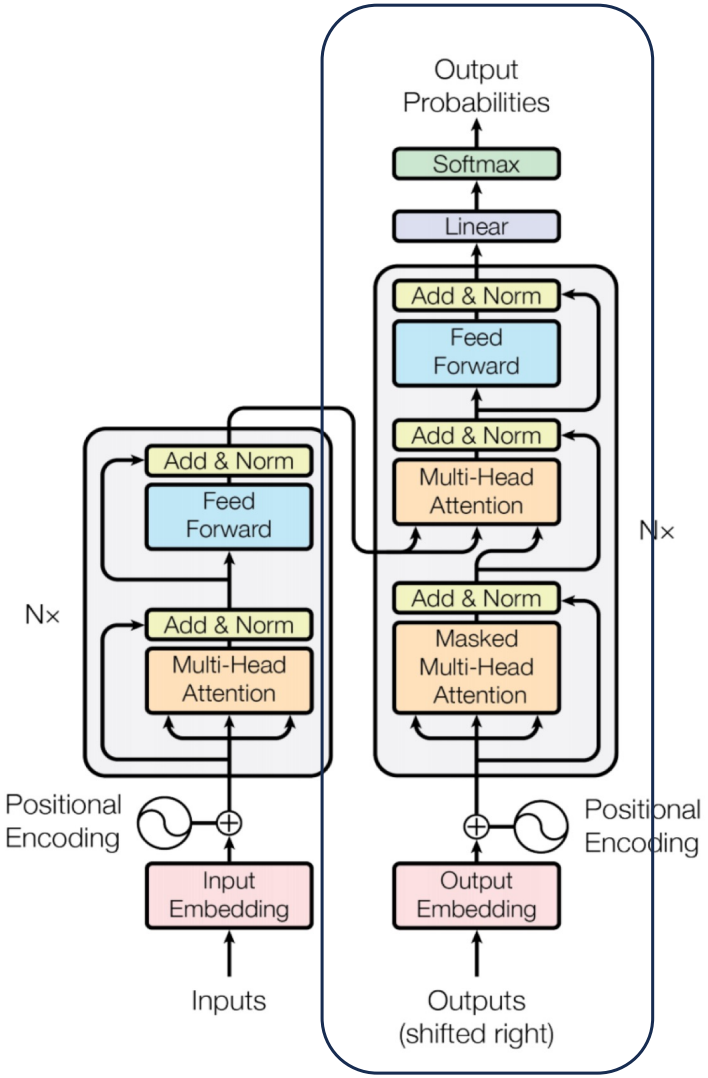
Show probabilities

Full spectrum

Generative Pre-trained Transformer (GPT)

- Le modèle neuronal utilisé par chatGPT
- Basé sur l'architecture du **transformer**
  - comme beaucoup d'autres : BERT, XLM, LLama, ...
- Les transformer sont des réseaux neuronaux complexes basés sur le principe d'(auto)-**attention**.

(Vaswani et al; 2017 "Attention is all you need")
(Brown et al; 2020 "Language models are few-shot learners")



Le mécanisme d'auto-attention

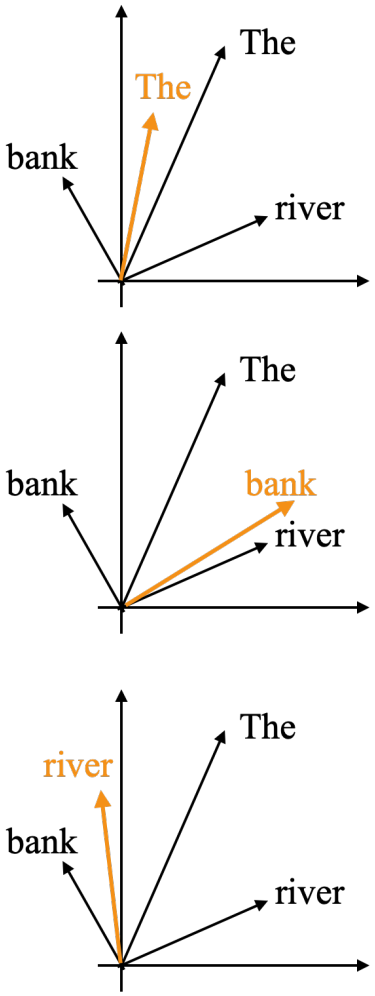
- « Bedeutung ist der Gebrauch » (L.Wittgenstein, *Investigations Philosophiques)*
  - **La signification est dans l'usage**: on comprend un mot quand on sait l'utiliser correctement

- L'auto-attention s'inspire de ce principe et au même temps reprend des concepts normalement utilisés en Recherche d'Information:
- Intuitivement:
  - "quel serait dans une phrase donnée, le mot **k** le plus utile à comprendre le sens du mot **q**"?
- La réponse est donnée par **V** (le poids de l'appariement entre k et q)
- Query, Key, Value (Q,K,V)

The

river

bank



Q

K

	The	bank	river
The	**0,84**	0,07	0,28
bank	0,14	0,01	**0,78**
river	0,02	**0,92**	0,04

softmax

self-attention matrix

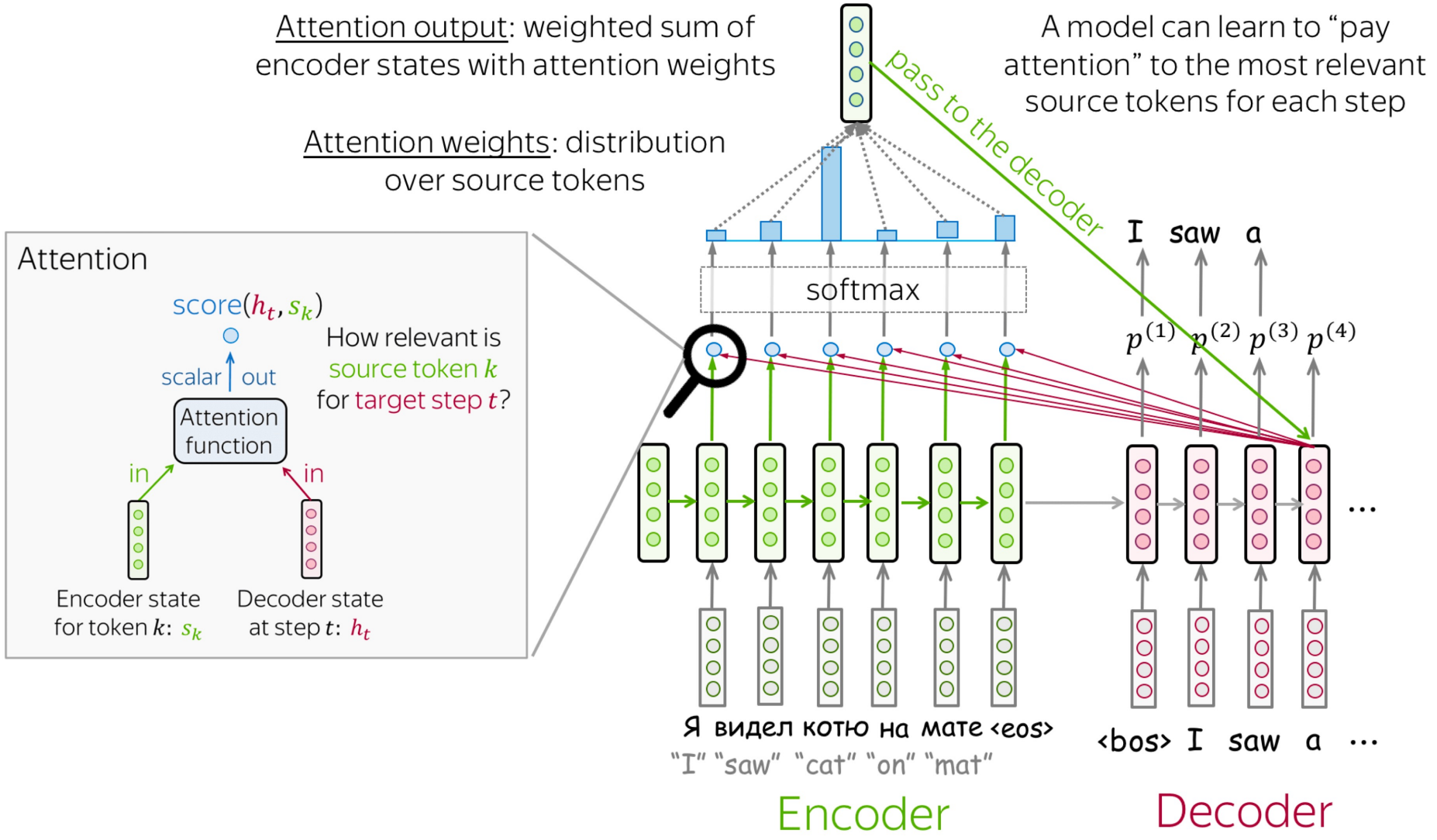
L’entrainement des modèles de langage

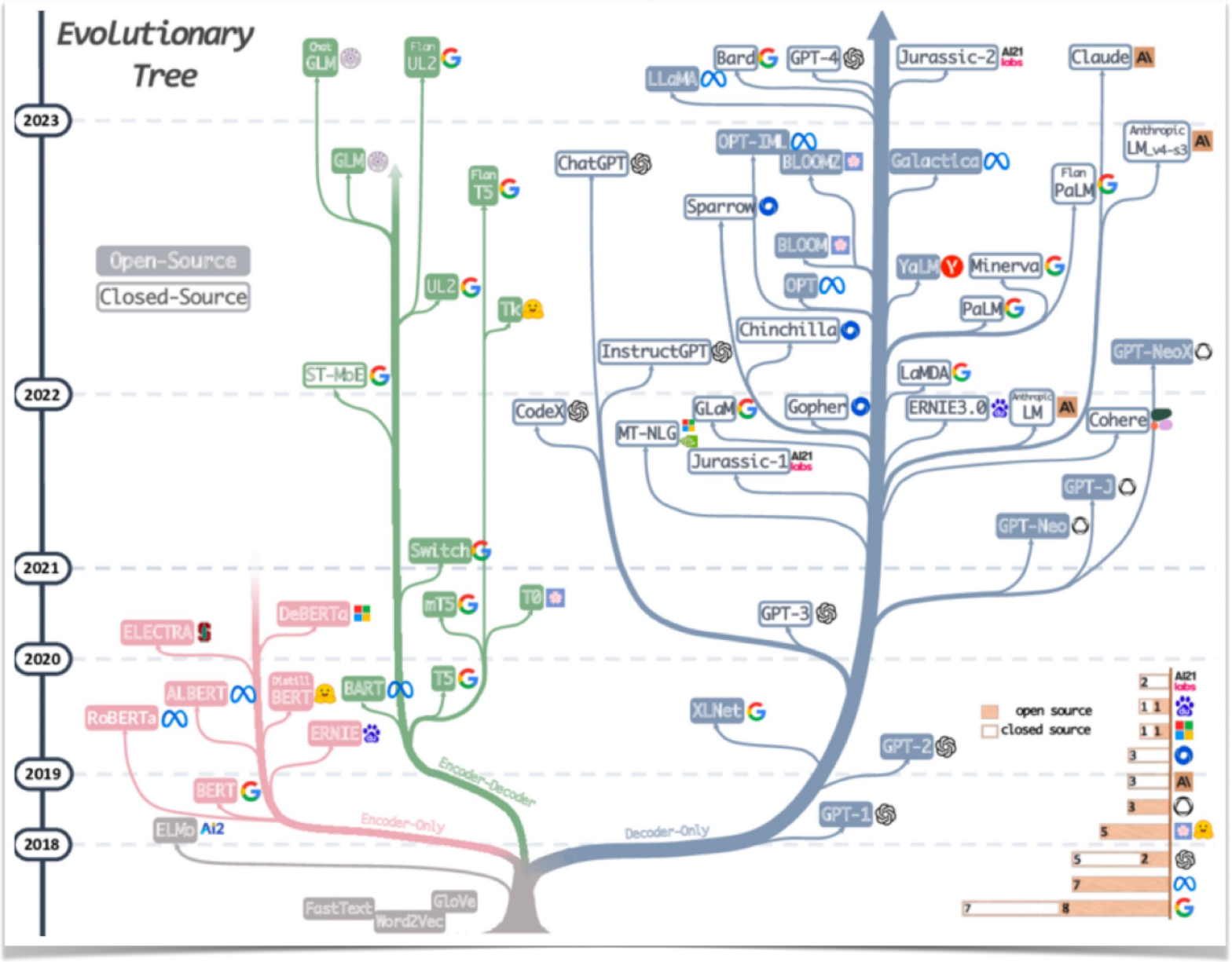
- Il faut des **très grandes collections** de textes pour estimer assez précisément deux matrices de poids:
  - $\mathbf{W_k}$ la matrice des poids des mots considérés en tant que clés
  - $\mathbf{W_q}$ la matrice des poids des mots considérés en tant que « requête »
- Tout ça est fait avec des techniques standard de réseaux de neurones (*backpropragation*)

- La **taille** du modèle est aussi un facteur pour améliorer l’estimation de ces matrices

Types de LLMs

- **Encoder-only**
  - Très utiles pour la classification
  - Exemples: BERT et variantes (RoBERTa, DistilBERT), XLNet, Electra
- **Decoder-only**
  - Spécialisés dans la prédiction du mot suivant
  - Exemples: GPT et variantes, LaMDA, PaLM
- **Encoder-decoder** (seq2seq)
  - Très utiles pour traduction automatique et résumé automatique
  - Exemples: T5, BART


Attention output: weighted sum of encoder states with attention weights
Attention weights: distribution over source tokens
A model can learn to "pay attention" to the most relevant source tokens for each step
pass to the decoder
Attention
score(h_t, s_k)
scalar
out
Attention function
in
in
How relevant is source token k for target step t?
Encoder state for token k: s_k
Decoder state at step t: h_t
softmax
$p^{(1)}$ $p^{(2)}$ $p^{(3)}$ $p^{(4)}$
I saw a
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
<bos> I saw a ...
Encoder
Decoder

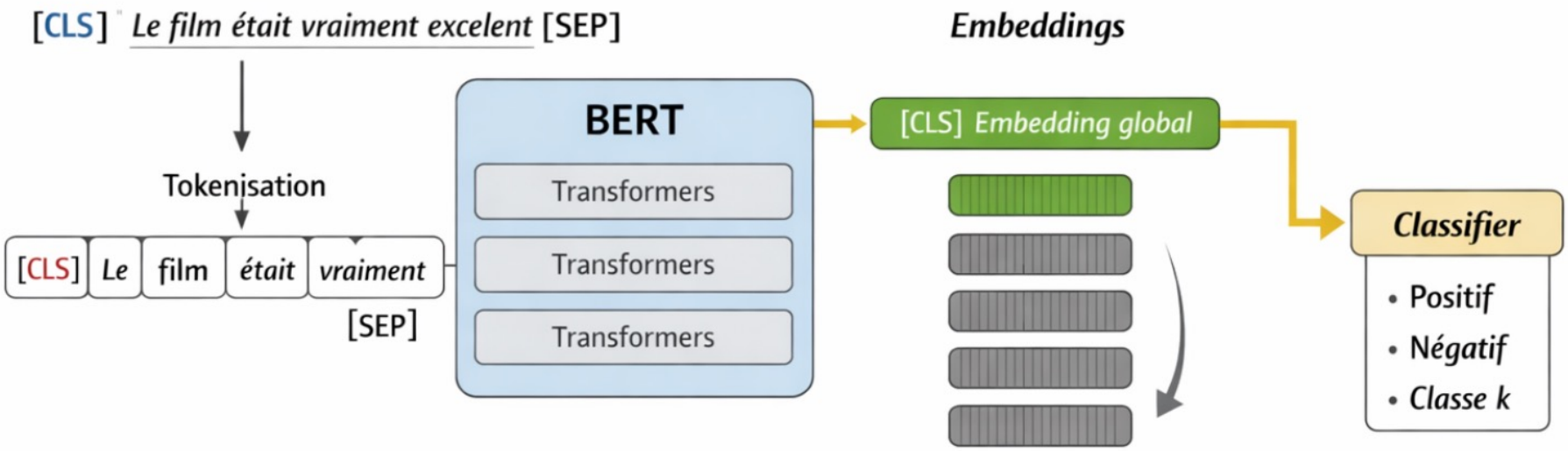

Evolutionary Tree
2023
2022
2021
2020
2019
2018
Open-Source
Closed-Source
Encoder-Only
Encoder-Decoder
Decoder-Only
FastText
GloVe
WordZVec
ELMo
BERT
RoBERTa
ALBERT
DistilBERT
ELECTRA
DeBERTa
ERNIE
BART
T5
mT5
T0
Switch
ST-MoE
GLM
ChatGLM
Flan T5
Flan UL2
UL2
Tk
GPT-3
InstructGPT
ChatGPT
CodeX
GLaM
MT-NLG
Gopher
Chinchilla
Sparrow
OPT-IML
BLOOMZ
BLOOM
OPT
Jurassic-1
LLaMA
Bard
GPT-4
Jurassic-2
Claude
Anthropic LM_v4-s3
Galactica
Flan PaLM
PaLM
YaLM
Minerva
LaMDA
ERNIE3.0
Anthropic LM
Cohere
GPT-NeoX
GPT-J
GPT-Neo
XLNet
GPT-2
GPT-1
open source
closed source

BERT (Bidirectional Encoder Representations from Transformers)

- Modèle de langage **pré-entraîné** basé sur les Transformers
- Bidirectionnel : chaque mot est contextualisé par tout le contexte gauche et droit
- Pré-entraînement sur deux tâches :
  - Masked Language Modeling (MLM)
  - Next Sentence Prediction (NSP)

- Produit des représentations vectorielles contextualisées pour chaque token

Classification avec BERT - exemple



La taille des LLMs

- Quand on parle de « taille » on fait référence au nombre de paramètres:
  - Couches, têtes d'attention, taille des embeddings
- Exemple: famille GPT-3

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Quantisation

- Exemple: modèle avec 7 milliards de paramètres
- Chaque paramètre est codé en FP16 (2 octets) ou FP32 (4 octets)
- Per l'inférence nous pouvons «caster» les paramètres à INT8 (**Quantisation**) -> réduction de la taille du modèle

Precision	RAM per parameter	Total RAM needed
FP32	4 bytes	28GB
FP16	2 bytes	14GB
INT8	1 byte	7GB

Taille des LLMs

- Tableau des besoins en RAM pour l'inférence et l'entraînement d'un modèle 7B :

- Pour info:
  - NVidia V100 32GB (8-10k €)
  - NVidia A100 40-80 GB (10-15k€)
  - NVidia H200 141GB (30-40k€)

Mode	Precision	Total RAM needed
Inference	FP16	~20-24 GB
Inference	INT8	~10-16 GB
Training	FP16	~64-80 GB
Training	FP32	~112+ GB

Bien écrire un prompt

- Donner un **rôle** et un **contexte** : pour qui, pour quoi
- Décrire précisément le **résultat attendu** : format, longueur, ton
- Fournir des **exemples** de ce qu'on veut
- **Itérer** : le premier résultat est un brouillon
- **Vérifier** : le modèle peut inventer des faits avec assurance (« hallucinations »)

Vague → précis

« Fais-moi une page d'accueil. »

→ « Page d'accueil d'une librairie de quartier, public 25-45 ans, ton chaleureux : logo, barre de recherche, 3 nouveautés en cartes, bouton « Réserver » bien contrasté, conforme WCAG AA. »

Générer du code d'interface

- Les LLM ont appris sur énormément de code : HTML, CSS, JavaScript, React…
- **Texte → interface :** on décrit l'écran, l'outil produit un prototype qui fonctionne
- **Image → code :** capture d'écran ou croquis transformé en HTML/CSS
- **Maquette → code :** design Figma converti en composants

Outils (fin 2026)

Figma Make, v0 (Vercel), Lovable, Bolt, Claude et ChatGPT, Cursor, GitHub Copilot.

Ces outils changent vite : comprendre le principe compte plus que maîtriser un outil.

Exemple : du prompt au prototype

1. **Prompt :** « Formulaire d'inscription à un festival, mobile d'abord, 3 étapes, barre de progression »
2. L'outil génère HTML, CSS et JavaScript, avec un aperçu
3. On **itère** en langage naturel : « bouton plus visible », « ajoute un mode sombre »
4. Le designer **vérifie** : cohérence avec le design system, accessibilité, messages d'erreur

En TP

Générer la même interface avec deux outils différents, puis comparer : qualité du code, accessibilité (contraste, navigation au clavier), fidélité à la maquette.

« Vibe coding » : opportunités et limites

- Prototyper en quelques minutes
- Tester des idées avec de vrais utilisateurs plus tôt
- Le designer peut créer des interactions sans attendre un développeur
- Explorer plusieurs directions en parallèle

Limites

- Code souvent difficile à maintenir
- Accessibilité fréquemment oubliée
- Interfaces « génériques » qui se ressemblent
- Failles de sécurité possibles
- Le designer reste responsable du résultat

10. Génération d’images

CLIP, diffusion, outils pour le design

CLIP (Contrastive Language–Image Pretraining)

- Objectif:
  - Apprendre une représentation commune pour les images et les textes, afin de pouvoir les comparer directement.
- Principe général:
  - Deux encodeurs séparés :
    - Encodeur image (CNN ou Vision Transformer)
    - Encodeur texte (Transformer)
  - Les deux produisent des vecteurs dans le même espace latent

CLIP (Contrastive Language–Image Pretraining)

- Entraîné sur des paires image–texte
- Objectif de l'entraînement:
  - Rapprocher les vecteurs image–texte correspondants
  - Éloigner les paires non correspondantes
  - Utilise une perte contrastive (similarité cosinus + softmax)
- Utilisation (inférence):
  - Encoder une image et plusieurs descriptions textuelles
  - Choisir le texte le plus proche de l'image dans l'espace latent
- Permet :
  - Classification zero-shot
  - Recherche image ↔ texte

Génération d'images

- Très grandes bases de données d'images annotées
  - Facilite l'apprentissage
  - Génération à partir d'un prompt
- Exemples: COCO, Laion https://laion.ai/
- Modèles principaux:
  - Dall-E (Open AI) https://labs.openai.com/
  - Midjourney https://www.imagine.art/
  - Stable Diffusion
    - https://github.com/CompVis/stable-diffusion
    - https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0

Dans cette image il y a quelque chose… vous pouvez le voir?



Stable Diffusion - entraînement

- On ajoute du bruit (gaussien) à l'image d'origine
- En même temps on entraîne un réseau de neurones à prédire la quantité de bruit présent

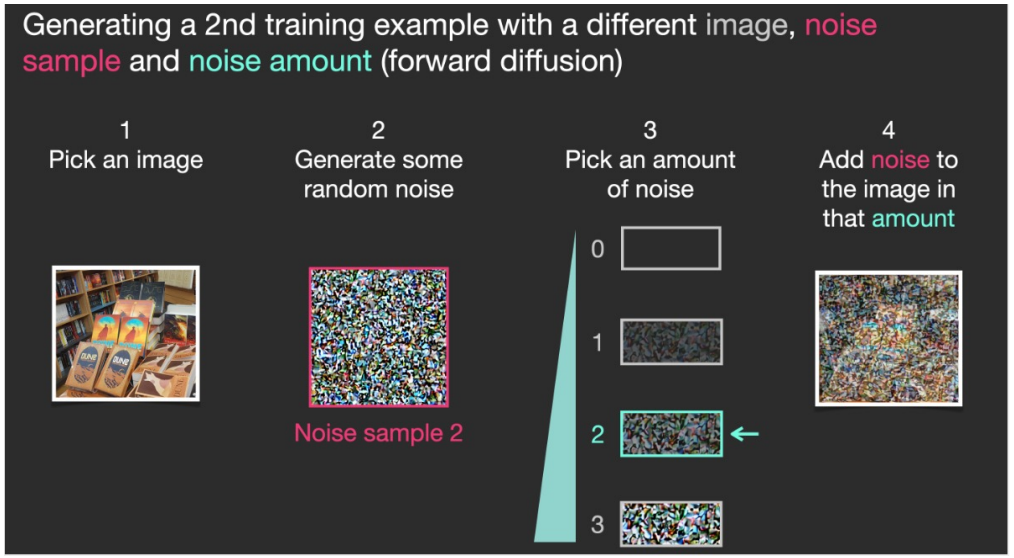
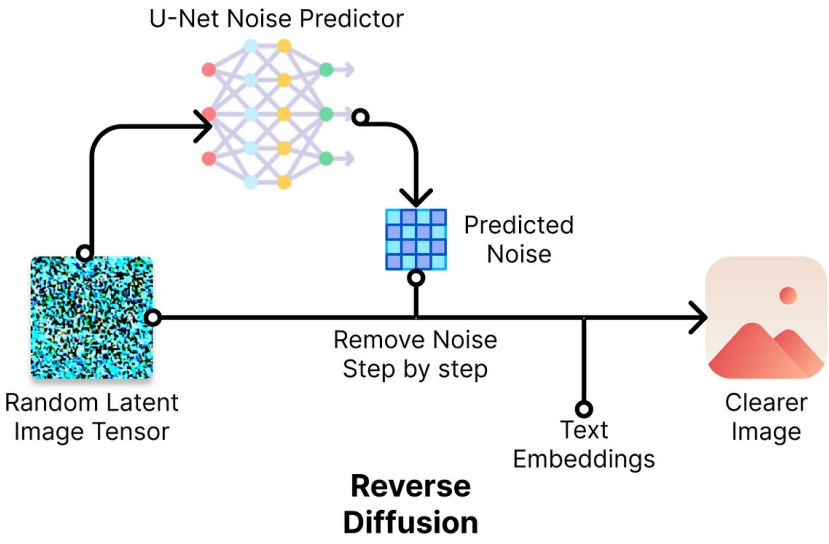


Image: Jalammar.github.io

Stable Diffusion - génération

- On détecte le niveau de bruit avec le RN appris
- On enlève la quantité de bruit détectée (opération mathématique sur vecteurs)
- On peut conditionner le résultat avec du texte, une image exemple,…



Exemple de « diffusion »

- Input: « paradise cosmic beach »



Contrôler la génération

- **Image → image :** partir d'un croquis ou d'une photo
- **Inpainting / outpainting :** modifier une zone, ou étendre l'image au-delà du cadre
- **ControlNet :** imposer une pose, des contours, une profondeur
- **LoRA :** petit ajustement du modèle pour apprendre un style ou un personnage, par ex. l'univers graphique d'une marque

Exemple

Un wireframe dessiné à la main + « style flat, palette pastel, application de méditation » → une maquette visuelle, qui respecte la composition du croquis.

L’IA générative dans le travail du designer

- **Recherche :** moodboards, variations rapides d’une idée
- **Production :** illustrations, icônes, fonds, mockups produit
- **Déclinaison :** adapter un visuel à plusieurs formats, retoucher
- **Prototypage :** interfaces générées, contenus de test réalistes

Bon réflexe

Utiliser l’IA pour explorer et accélérer, pas pour décider : le choix, la cohérence et le sens restent le travail du designer.

11. IA et design d’interface

Concevoir avec l’IA, concevoir pour l’IA

Concevoir des interfaces pour l'IA

- **Montrer l'incertitude :** le modèle peut se tromper, l'interface doit le laisser voir
- **Expliquer :** pourquoi cette recommandation ?
- **Laisser le contrôle :** modifier, annuler, refuser une suggestion
- **Recueillir du feedback :** pouce levé / baissé, corrections
- **Gérer l'attente :** affichage progressif du résultat, indicateurs de chargement

Ressources

- Google PAIR, « People + AI Guidebook »
- Microsoft, « Guidelines for Human-AI Interaction » (18 principes)

Enjeux éthiques et juridiques

- **Droits d'auteur :** données d'entraînement, propriété des images générées
- **Biais :** les modèles reproduisent les stéréotypes de leurs données (« un médecin » → surtout des hommes)
- **Transparence :** l'AI Act européen impose de signaler les contenus générés par IA (deepfakes, chatbots)
- **Environnement :** entraîner et utiliser ces modèles consomme beaucoup d'énergie
- **Métier :** quelles compétences du designer deviennent plus importantes ?

À retenir

- Apprentissage automatique = apprendre une règle à partir d'exemples
- Quatre ingrédients : des données, un modèle, une mesure d'erreur, une méthode pour la réduire
- Les modèles génératifs (texte, code, images) sont des réseaux de neurones entraînés sur d'énormes corpus
- Puissants mais faillibles : le regard critique du designer reste indispensable

Pour expérimenter sans coder

- Teachable Machine
- TensorFlow Playground
- Hugging Face Spaces (des milliers de démos en ligne)