Apprentissage artificiel

Jérôme Buisine

Team: IMAP (Images et Apprentissage)

5 janvier 2023

Objectifs du module

Les différents objectifs

- Comprendre la notion d'apprentissage automatique.
- Identifier la classe du problème et cibler la méthode adaptée.
- Exploiter les différentes techniques d'amélioration de performance d'un modèle.

Objectifs du module

Les différents objectifs

- Comprendre la notion d'apprentissage automatique.
- Identifier la classe du problème et cibler la méthode adaptée.
- Exploiter les différentes techniques d'amélioration de performance d'un modèle.

Remarques importantes

- Développement sous Python
- TP versionnés sous git (Gitlab)

Déroulement du module

Organisation du module

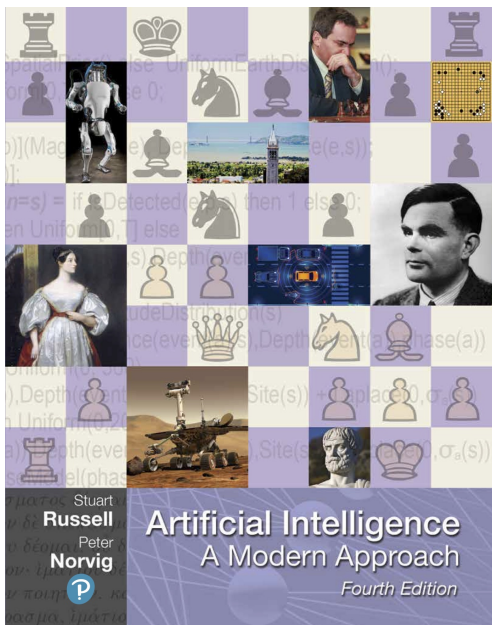
- Cours à chaque séance si nouvelle notion abordée
- **chaque TD/TP** :
  - une commit à fournir comme rendu.
  - une note.
- 2 projets de mises en situation :
  - 1 intermédiaire.
  - 1 vers la fin du module.

Note finale

note du module $= \frac{1}{2}$ examen $+ \frac{1}{2}$ (moyenne des notes TD/TP $+$ projets)

Quelques ressources

En Français



En Anglais

Plan

Plan

Plan

Définition de l'apprentissage artificiel

L'apprentissage artificiel/automatique (en anglais, *machine learning*)

Un champ d'étude de l'intelligence artificielle qui se fonde sur des approches mathématiques et statistiques pour donner aux ordinateurs la capacité d'***apprendre*** à partir de **données**, c'est-à-dire d'améliorer leurs performances à **résoudre des tâches** sans être explicitement programmés pour chacune.

Définition de l'apprentissage artificiel

L'apprentissage artificiel/automatique (en anglais, *machine learning*)

Un champ d'étude de l'intelligence artificielle qui se fonde sur des approches mathématiques et statistiques pour donner aux ordinateurs la capacité d'***apprendre*** à partir de **données**, c'est-à-dire d'améliorer leurs performances à **résoudre des tâches** sans être explicitement programmés pour chacune.

Introduction du terme

Le terme *apprentissage automatique* a été inventé en 1959 par Arthur Samuel, un Américain travaillant pour IBM et pionnier dans le domaine des jeux vidéo et de l'intelligence artificielle.

Définition plus formelle

Définition formulée par Tom M. Mitchell

« On dit d'un programme informatique qu'il apprend de l'expérience E en ce qui concerne une certaine classe de tâches T et une mesure de performance P si sa **performance** aux tâches de T, telle que mesurée par P, s'améliore avec l'expérience E. ».

Les éléments clés

- **Expérience :** capacité d'apprentissage à partir de données.
- **Tâches :** données à traiter pour exécuter la tâche (problème ciblé).
- **Performance :** évaluation des réponses aux données traitées.

Définition de l'objectif : obtention d'un modèle

Vers la notion de modèle

- construction d'un modèle de la réalité à partir de données $\rightarrow$ modèle qui apprend à reproduire ce qui a pu être observé.
- un modèle qui permet à un ordinateur de traiter des tâches (difficiles) grâce à un processus d'apprentissage ($\neq$ algorithme classique).

Domaines d'applications de l'apprentissage artificiel

- Reconnaissance de formes tels des visages (traitement de l'image).
- Traduction de texte (traitement du langage).
- Moteurs de recherche.
- Aide aux diagnostics (médical notamment).
- Au sein des jeux.
- Robotique / véhicules autonomes.
- ...

Un exemple de problème

Définition du problème

Un randonneur se balade en forêt avec un guide de haute montagne. Il remarque qu'une espèce d'arbre en particulier semble malade cette année. Le guide, connaît bien cette espèce, et pour lui, tous ne le sont pas. Pourtant à première vue, la différence visuelle est difficile.

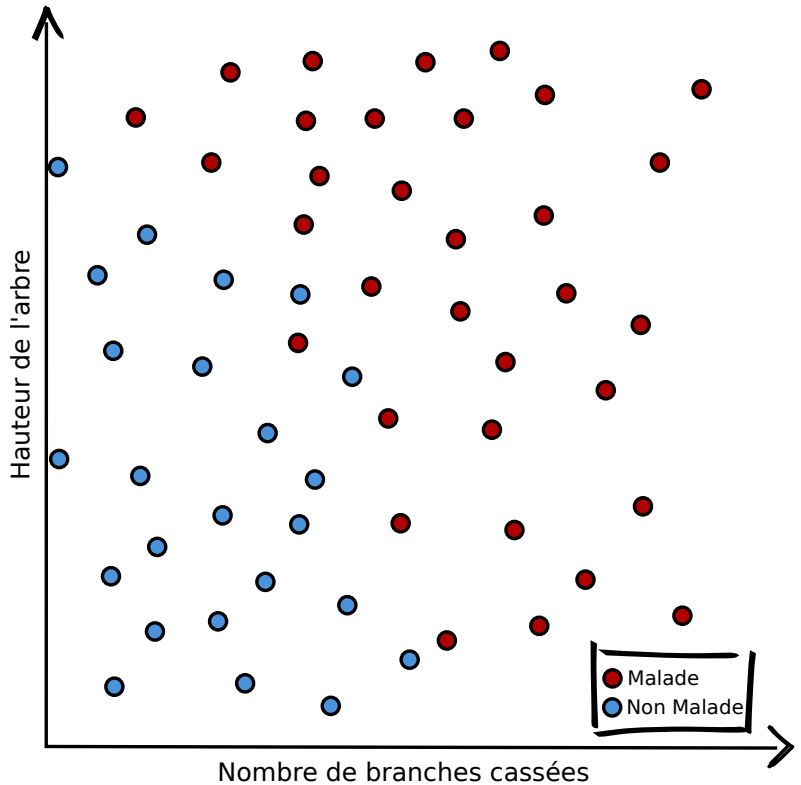
Afin de relever le pourcentage d'arbre malade pour cette année et les années suivantes, ils décident d'identifier deux critères : le nombre de branches cassées, la hauteur de l'arbre.

Le randonneur relève donc ces informations et le guide quant à lui expert dans la reconnaissance de la maladie, identifie si l'arbre est atteint de la maladie ou non.

Conception d'une base de données

À partir des données récoltées, le randonneur dresse un graphique

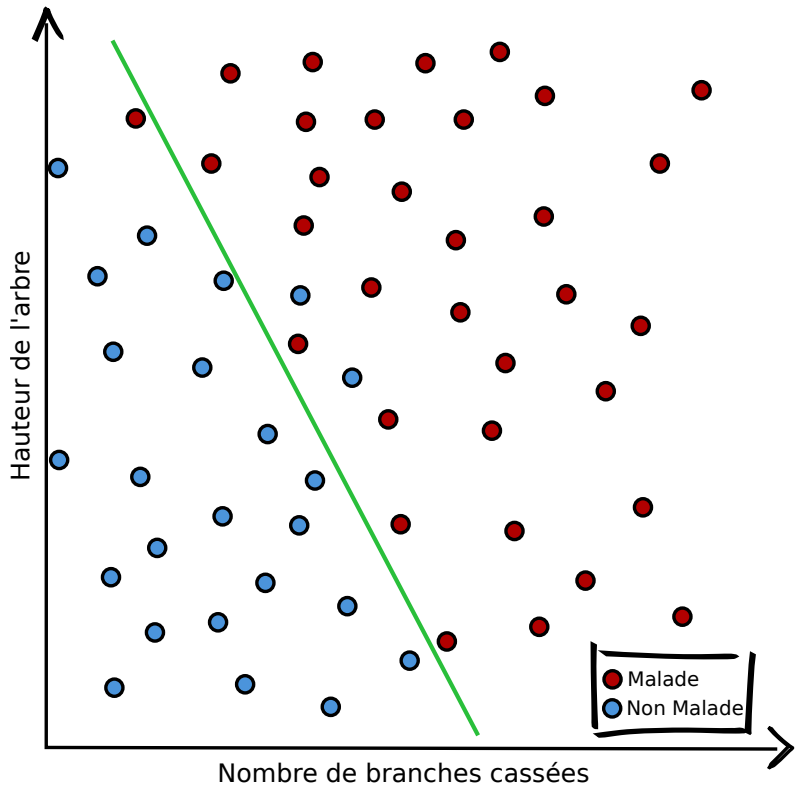
Maladie d'un arbre : représentation des données



Remarque

Les données affichées ont une certaine tendance et nous voulons trouver un modèle qui généralise la détection à partir des deux attributs

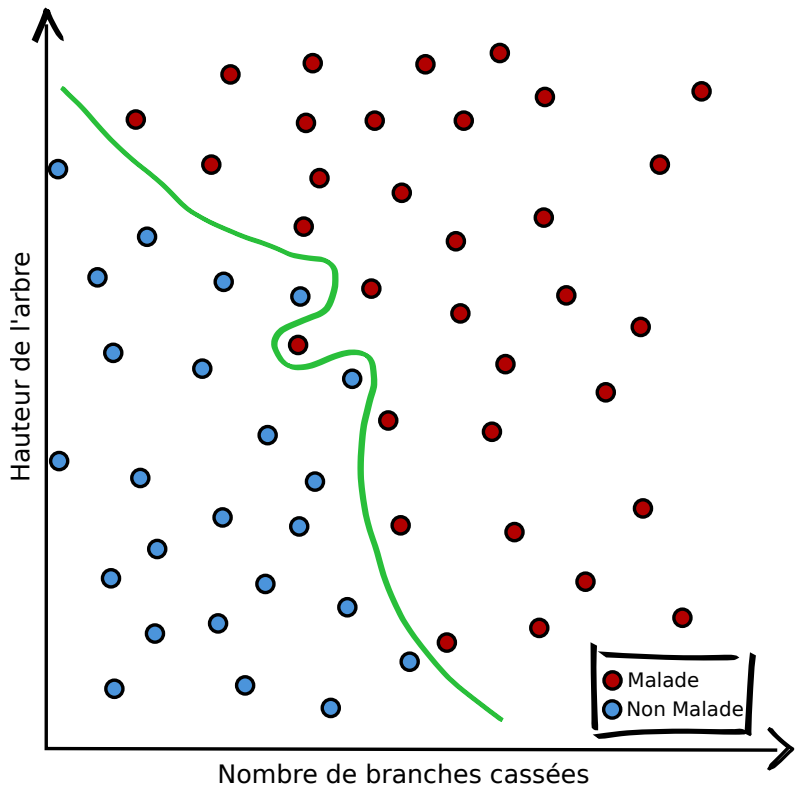
Maladie d'un arbre : modèle de séparation



Remarque

La ligne séparatrice (modèle) des données implique quelques erreurs

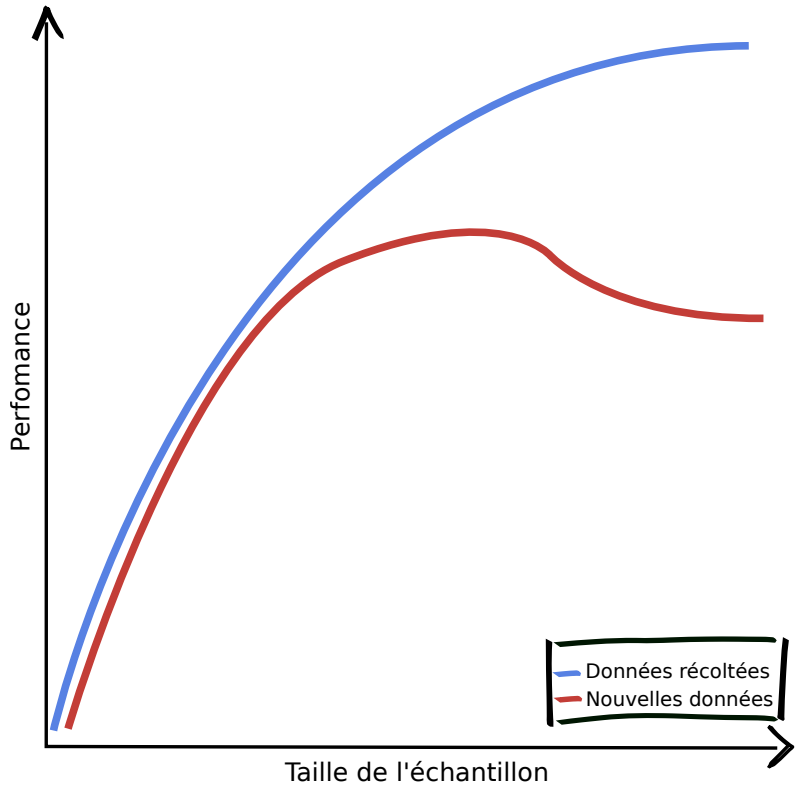
Maladie d'un arbre : modèle de séparation



Remarque

Le nouveau modèle ne commet aucune erreur ! Et pourtant...

La performance et l'importance de la généralisation



Problème du surapprentissage

Sur de nouvelles données, le modèle n'est pas performant plus le nombre d'échantillons augmentent...

Notions importantes : performance et généralisation

Mesure de performance

- Elle est un point clé d'évaluation d'un modèle (comparaisons de plusieurs modèles et choix de l'un d'entre eux).
- C'est elle qui va évaluer les erreurs, sachant que certaines erreurs peuvent être plus graves que d'autres.
- Elle doit donc être choisie judicieusement et dépend de l'application ciblée (on pourrait « préférer » 10 erreurs légères qu'une seule grave).

Notions importantes : performance et généralisation

Mesure de performance

- Elle est un point clé d'évaluation d'un modèle (comparaisons de plusieurs modèles et choix de l'un d'entre eux).
- C'est elle qui va évaluer les erreurs, sachant que certaines erreurs peuvent être plus graves que d'autres.
- Elle doit donc être choisie judicieusement et dépend de l'application ciblée (on pourrait « préférer » 10 erreurs légères qu'une seule grave).

La généralisation

- Un modèle ne doit pas être évalué uniquement sur **les données d'apprentissage**.
- On souhaite un modèle qui généralise face des données inconnues ou proches de l'inconnues.
- Classiquement, on sépare l'ensemble de données en deux sous-ensemble, un pour apprendre, le second pour vérifier les performances du modèle.

Plan

Les différents familles de modèles

Classification, régression, clustering :

- **Classification :** reconnaître des classes à partir de leurs descripteurs (caractéristiques).
- **Régression :** prédictions sur des valeurs numériques.
- **Clustering :** regrouper des ensembles d'exemples non supervisés en classes (identification de liens).

Plan

Les différents types d'apprentissage

Types d'apprentissage :

- **Apprentissage supervisé :** apprentissage sur des exemples connus (étiquetage des données par un oracle $\rightarrow$ expert).
- **Apprentissage non supervisé :** apprentissage sur des exemples non connus.
- **Apprentissage semi-supervisé :** apprentissage intermédiaire (exemples étiquetés/non étiquetés). On y trouve par exemple l'apprentissage actif.
- **Apprentissage par renforcement :** apprentissage basé sur des observations/récompenses. L'algorithme agit sur l'environnement qui en retour guide (récompense) l'algorithme.

Apprentissage supervisé

À partir de...

Un ensemble de données d'apprentissage de taille N formé de paires de données d'entrée et de sortie : $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ où y_i a été généré par une fonction inconnue $y = f(x)$.

Le modèle doit...

Découvrir une fonction h qui approche au mieux la fonction réelle f.

Les représentations de f

- Si f est une fonction continue ($y \in \mathbb{R}$) on parle alors de régression.
- Si f est une fonction discrète ($y \in \mathbb{N}$) on parle alors de classification.
- Si f est une fonction binaire ($y \in \{0, 1\}$) on parle alors d'apprentissage de concept (classification).

Apprentissage non supervisé

À partir de...

d'un ensemble de données d'apprentissage de taille N formé de données d'entrée uniquement : $(x_1), (x_2), \ldots, (x_N)$.

Le modèle doit...

Découvrir par lui-même une structure plus ou moins cachée des données (des régularités).

Apprentissage par renforcement

Au sein de...

d'un environnement où une action produit une valeur de retour, une récompense qui peut être négative ou positive.

Le modèle doit...

Apprendre un comportement étant donné une observation pour choisir la meilleure action dans le futur étant donné l'état de l'environnement.

Apprentissage par renforcement

Au sein de...

d'un environnement où une action produit une valeur de retour, une récompense qui peut être négative ou positive.

Le modèle doit...

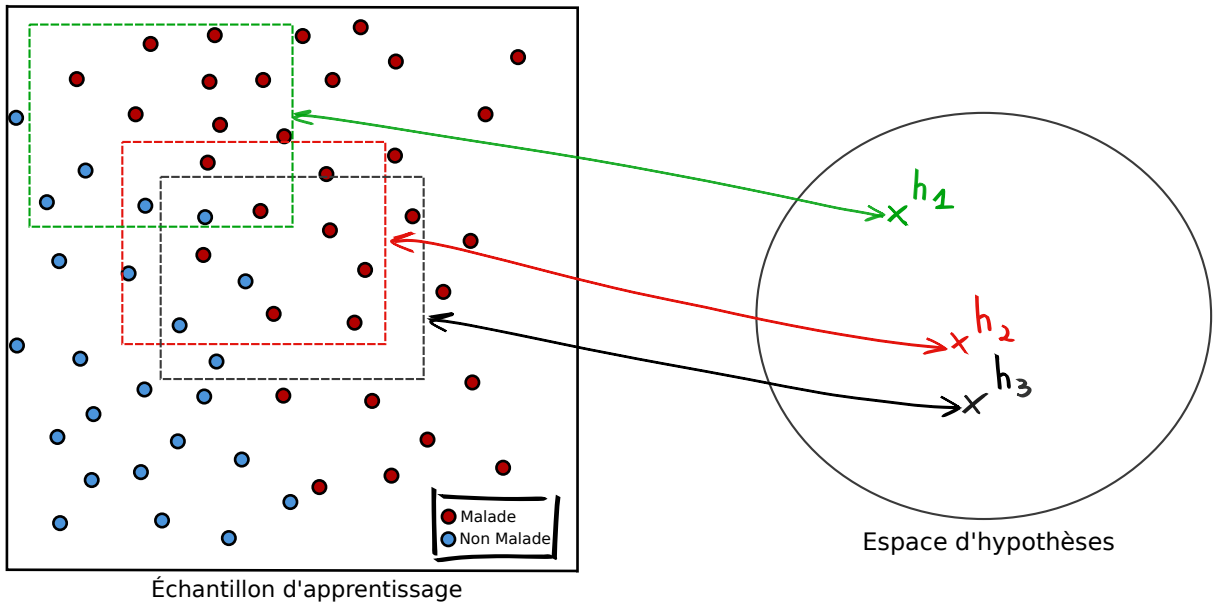
Apprendre un comportement étant donné une observation pour choisir la meilleure action dans le futur étant donné l'état de l'environnement.

Exemple : le jeu d'échec

- Environnement : état d'un plateau de jeu d'une partie.
- Actions : ensemble de coups possibles à un stade de la partie.
- Récompense : valeur apportée à une prise, bon placement, victoire, et au contraire : mauvais placement, risque de défaite.

Plan

Comparaisons de modèles : espace des hypothèses



Chaque point de $\mathcal{H}$ (une hypothèse h) correspond à un échantillon d'apprentissage

- À partir de données évaluer l'hypothèse $h \approx f$.
- Trouver un modèle offrant la meilleur performance.

Exploration de l'espace des hypothèses

- **Aucune structure :** exploration aléatoire.
- **Un voisinage existe :** optimisation de l'espace (utilisation de mesure d'évaluation).

Risque empirique vs risque réel

Risque empirique

Généralement, pour un modèle h ayant appris sur une base de données X, ses prédictions sur chaque x_i sont évaluées et comparées à $f(x_i)$ à partir d'une fonction de performance P, ou d'erreur (ici la moyenne obtenue) :

$$\text{Erreur empirique} = \sum_{i=1}^{N} P(y_i, h(x_i))$$

- avec N le nombre de données.
- x_i une donnée de la base de données X.
- $y_i = f(x_i)$

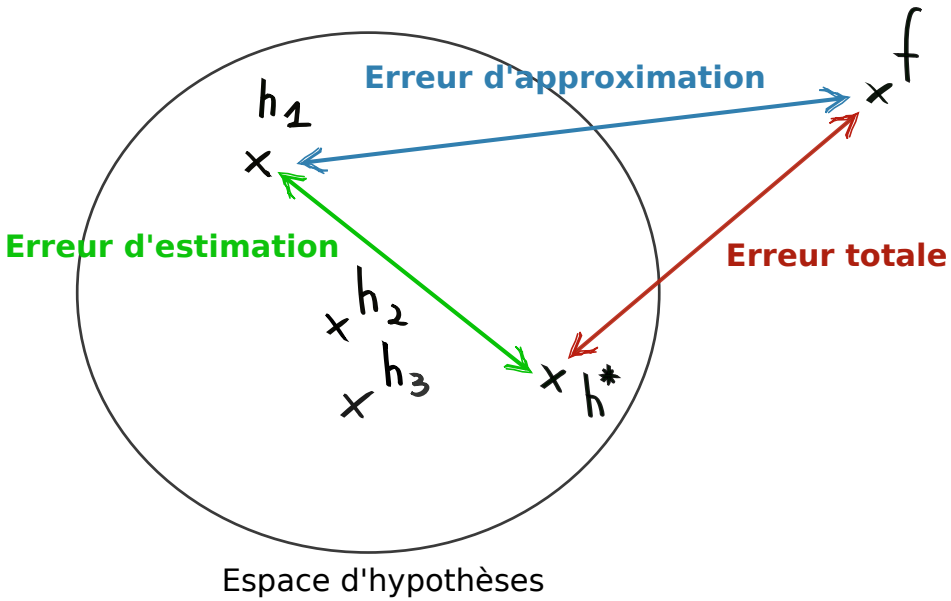
Risque réel

Le problème, c'est que l'on ne possède pas forcément toutes les données de f dans nos données, on conserve donc une :

$$\text{Erreur réelle} = \left(\sum_{i=1}^{N} P(y_i, h(x_i)) \right) + \epsilon$$

- avec ϵ, une erreur inconnue entre la fonction apprise et f.

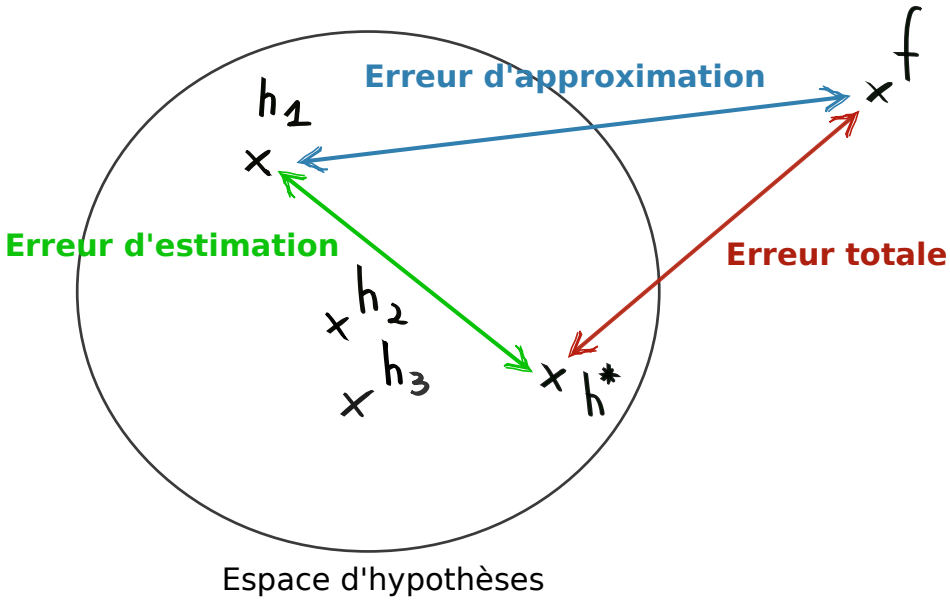
Biais et variance



Les différentes erreurs

- **Biais :** erreur d'approximation (lié au choix de l'espace des hypothèses).
- **Variance :** erreur d'estimation (différence entre la l'hypothèse obtenue à partir d'un échantillon et la meilleure hypothèse h^* de l'espace des hypothèses).
- Trouver un compromis biais/variance car :
  - agrandir l'espace $\mathcal{H} \rightarrow$ augmente la variance.
  - diminuer l'espace $\mathcal{H} \rightarrow$ augmente le biais.

Biais et variance



Un potentiel problème

Analysons la meilleure hypothèse h^* d'espace liée à nos données :

- Hypothèse qui réduit au mieux la variance.
- Répond au mieux par rapport à nos données (espace des hypothèses $\mathcal{H}$) ;
- Problème : liée au données $\rightarrow$ ne généralise peut-être pas / éloignée de f $\rightarrow$ risque réel.

Comment éviter le surapprentissage

Décomposer la base de données

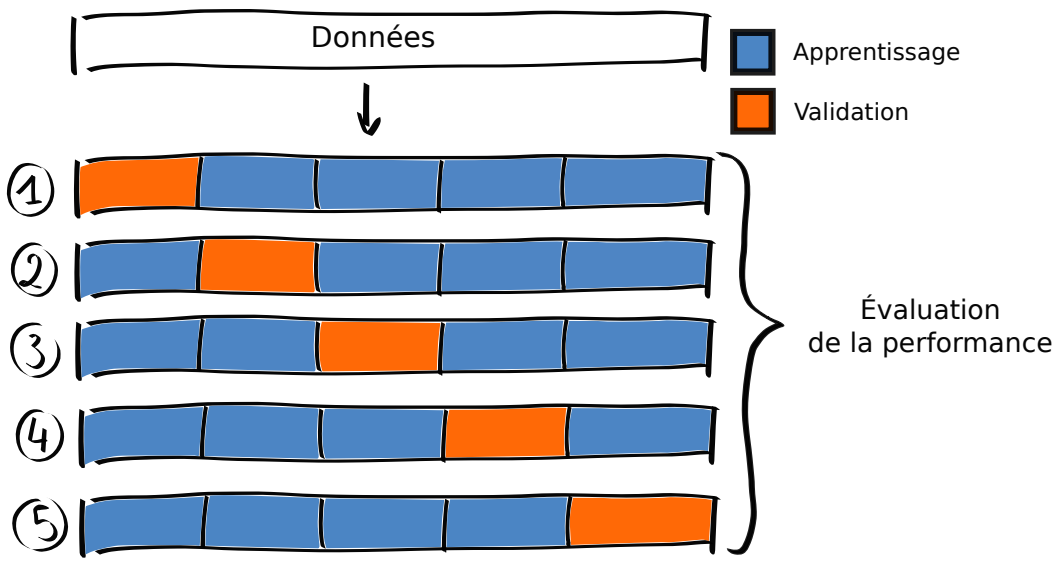
- Base d'apprentissage : sert pour l'entraînement du modèle (par exemple 70% des données) ;
- Base de test : permet d'évaluer le modèle sur de nouvelles données (par exemple 30% des données) ;
- Utiliser la fonction performance P (ou d'erreur) sur la base de test pour comparer les modèles.

Comment éviter le surapprentissage

Validation croisée

- Diviser en k sous-ensemble la base de données.
- Utiliser $k-1$ ensemble pour l'apprentissage, le reste pour valider.
- Réaliser k apprentissages.
- Calculer la moyenne et l'écart-type de scores P sur la base de validation de chacun apprentissage (estimer le biais et la variance).

Exemple avec $k = 5$



Ce qu'il est important de retenir à ce stade

- Il existe plusieurs classes de modèles : classification, régression, clustering...
- Un modèle apprend à reproduire (ou identifier des régularités $\rightarrow$ clustering) ce qui a pu être observé (données $\rightarrow$ échantillon).
- Plusieurs types d'apprentissage également : supervisé, non-supervisé, semi-supervisé, par renforcement...
- Il faut trouver au mieux l'hypothèse qui approche au mieux f ($h \approx f$), la fonction recherchée.
- Éviter le surapprentissage et plutôt chercher à généraliser.
- Utiliser une mesure de performance (ou d'erreur) pour identifier un modèle (comparaison).

Plan

Alpha GO (DeepMind, 2015)

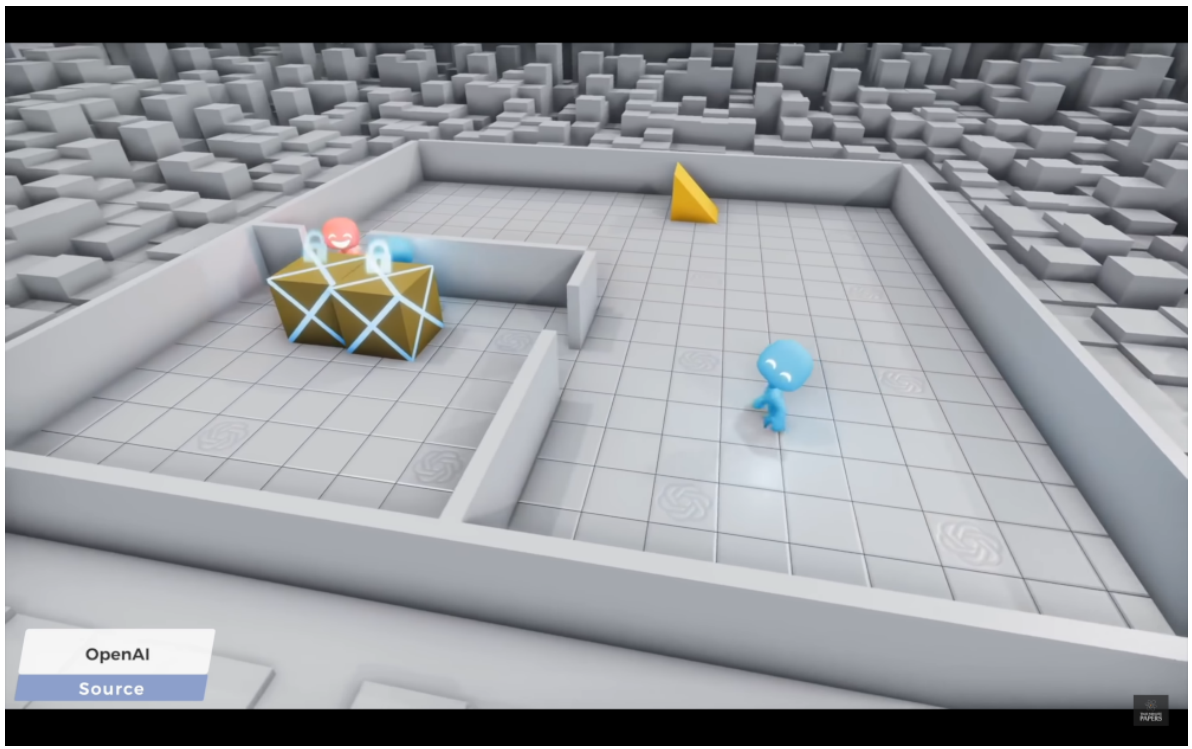
Au sein d'un environnement de jeu complexe...

identifier le meilleur coup qui maximise un certain gain (victoire)

Hide and Seek (OpenAI, 2019)

À partir d'un environnement inconnu...

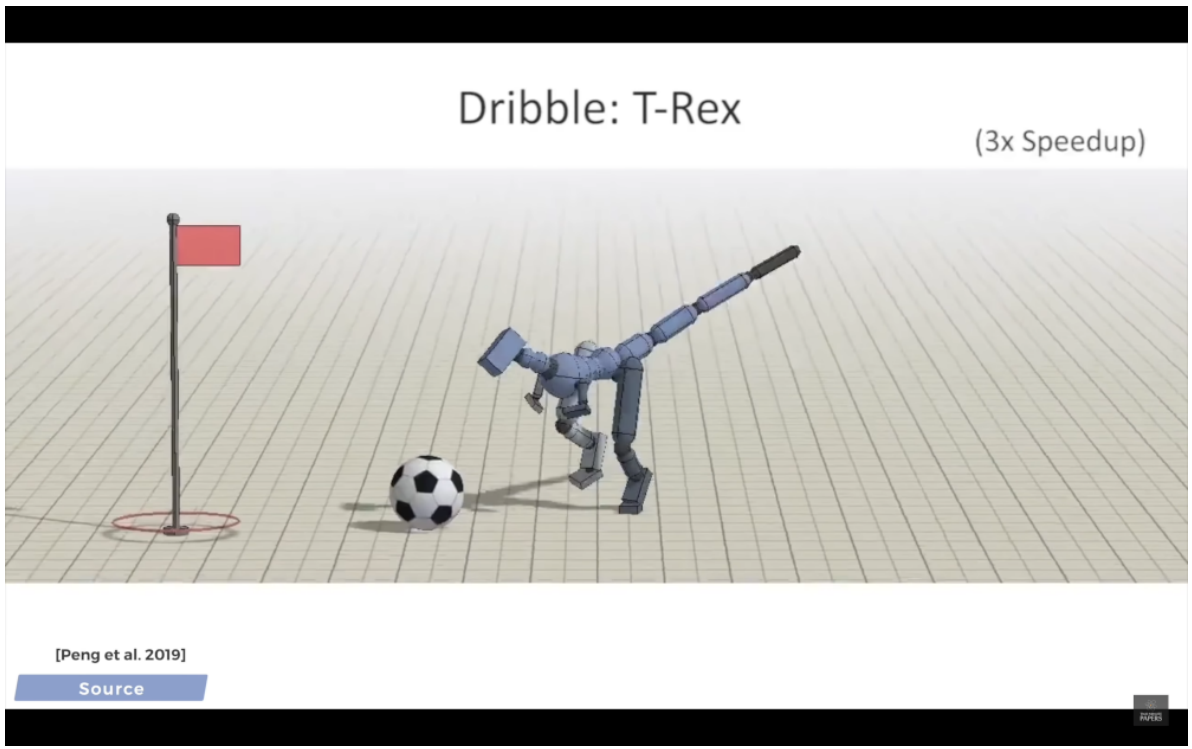
les agents doivent trouver un moyen de gagner en exploitant les actions possibles



Simulation d'agent (2019)

À partir d'un agent avec des particularités de mouvement

il est possible de lui donner l'objectif de dribbler jusqu'à un lieu



Face image manipulation (2021)

À partir d'un modèle permettant de générer des visages

il est possible avec ses paramètres pour éditer les visages

Jouer à Minecraft (2021)

À partir de l'environnement de Minecraft

L'agent apprend à réaliser certaines tâches

Génération d'images : Dall·E 2.0 (OpenAI, 2022)

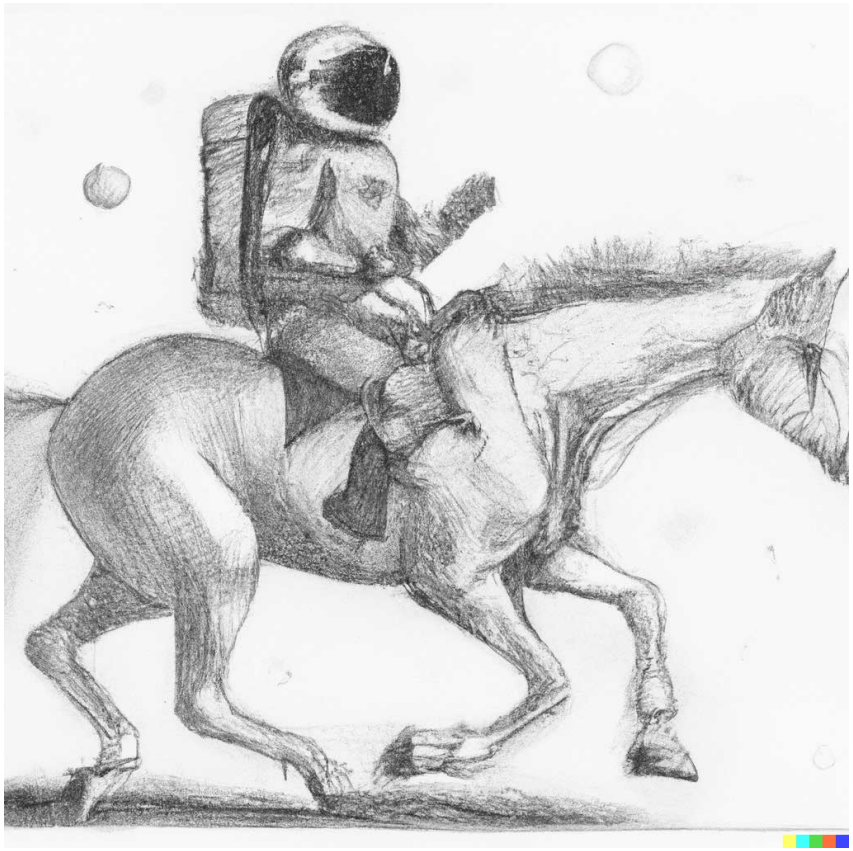
À partir d'un texte, le modèle génère une image

Teddy bears working on new AI research on the moon in the 1980s

Génération d'images : Dall·E 2.0 (OpenAI, 2022)

À partir d'un texte, le modèle génère une image

An astronaut riding a horse as a pencil drawing



Génération d'images : Dall·E 2.0 (OpenAI, 2022)

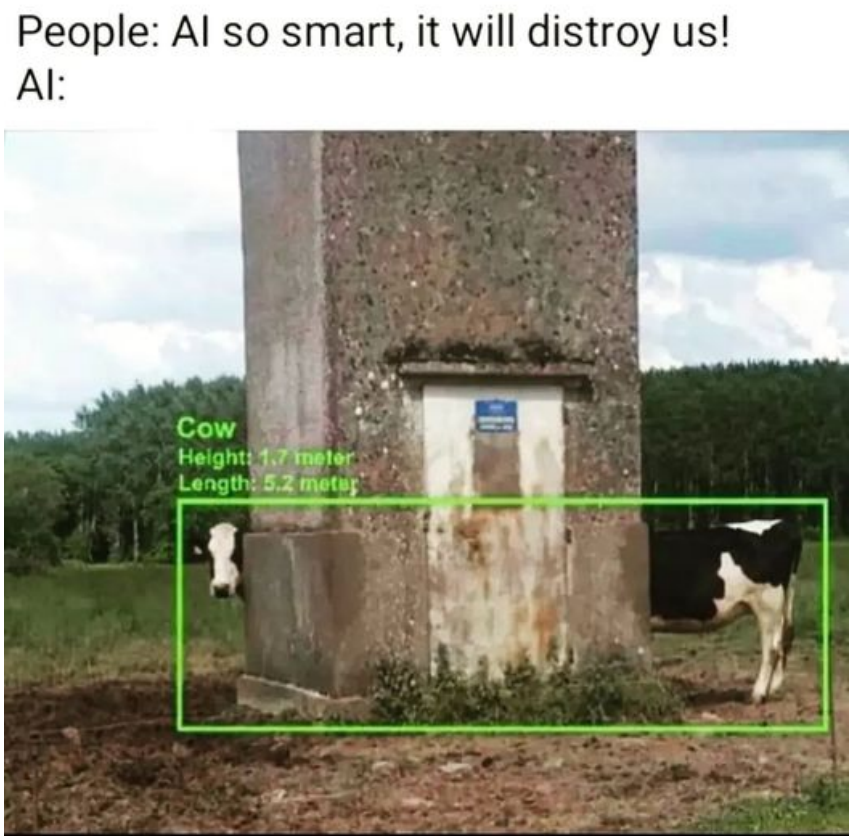
À partir d'un texte, le modèle génère une image

An astronaut lounging in a tropical resort in space in a vaporwave style

Mais aussi...

Tout ne se passe pas toujours comme prévu

Évaluer la robustesse d'un modèle face à l'inconnu reste compliqué, des biais peuvent être présents.



TP0 : Initiation à Python (mise en place de l'environnement de travail)

Plan

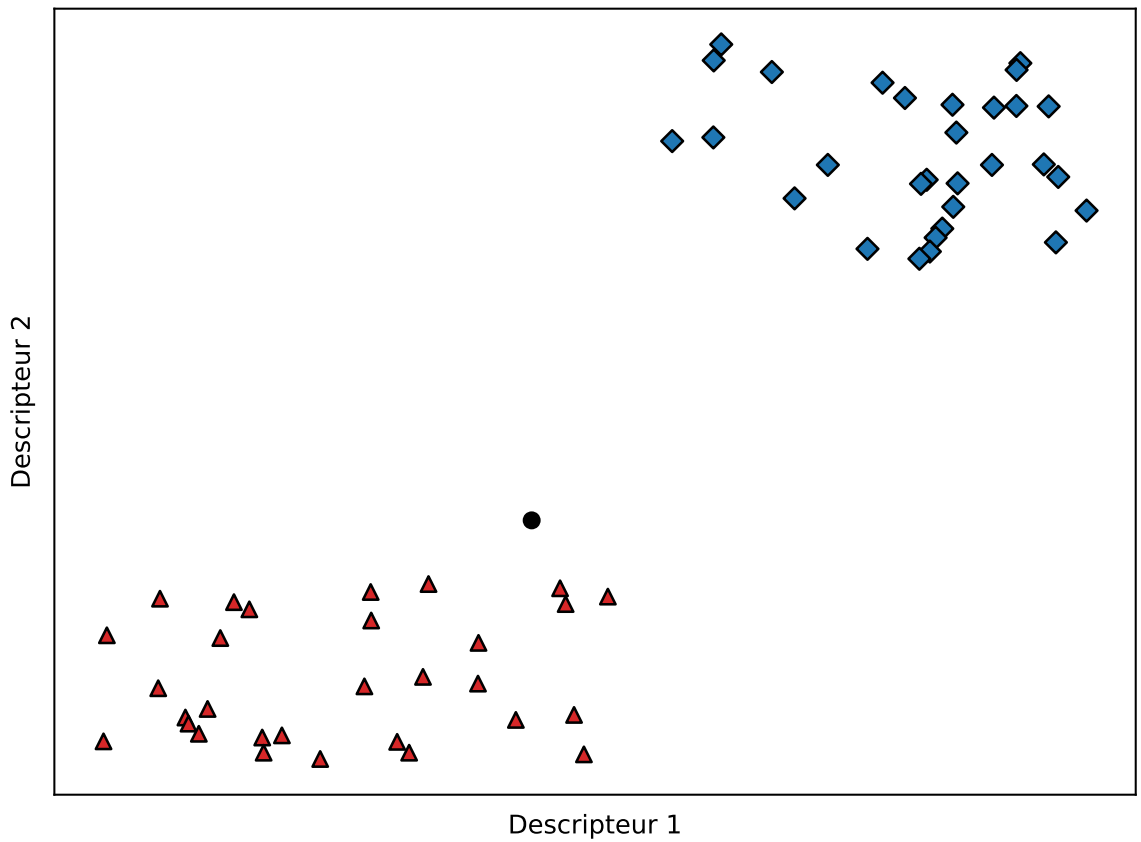
Plan

TP1 : Modèles de régression linéaire et polynomiale

Plan

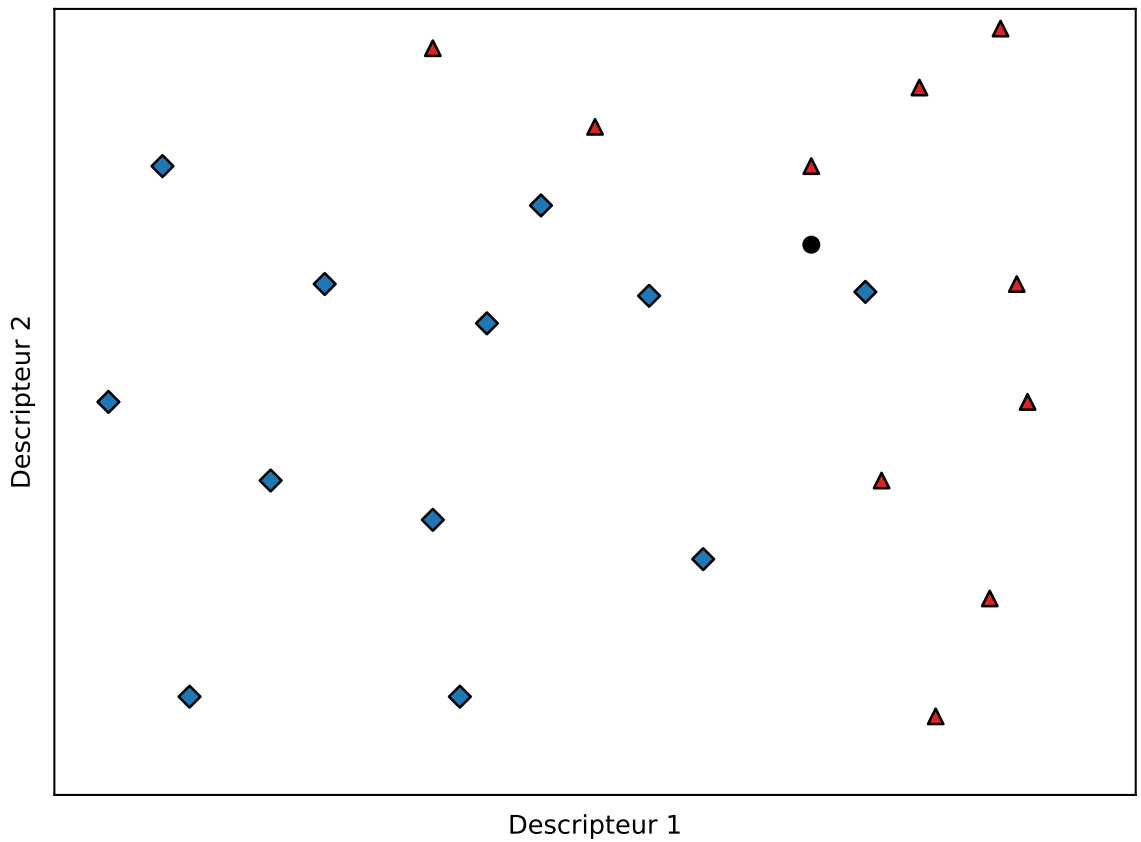
Classification d'un cas simple

Quel label pourrait-on affecter au point noir ?



Classification d'un cas plus complexe

Et maintenant... ?



Le principe du KNN

Principe de l'algorithme (*K-Nearest Neighbors*)

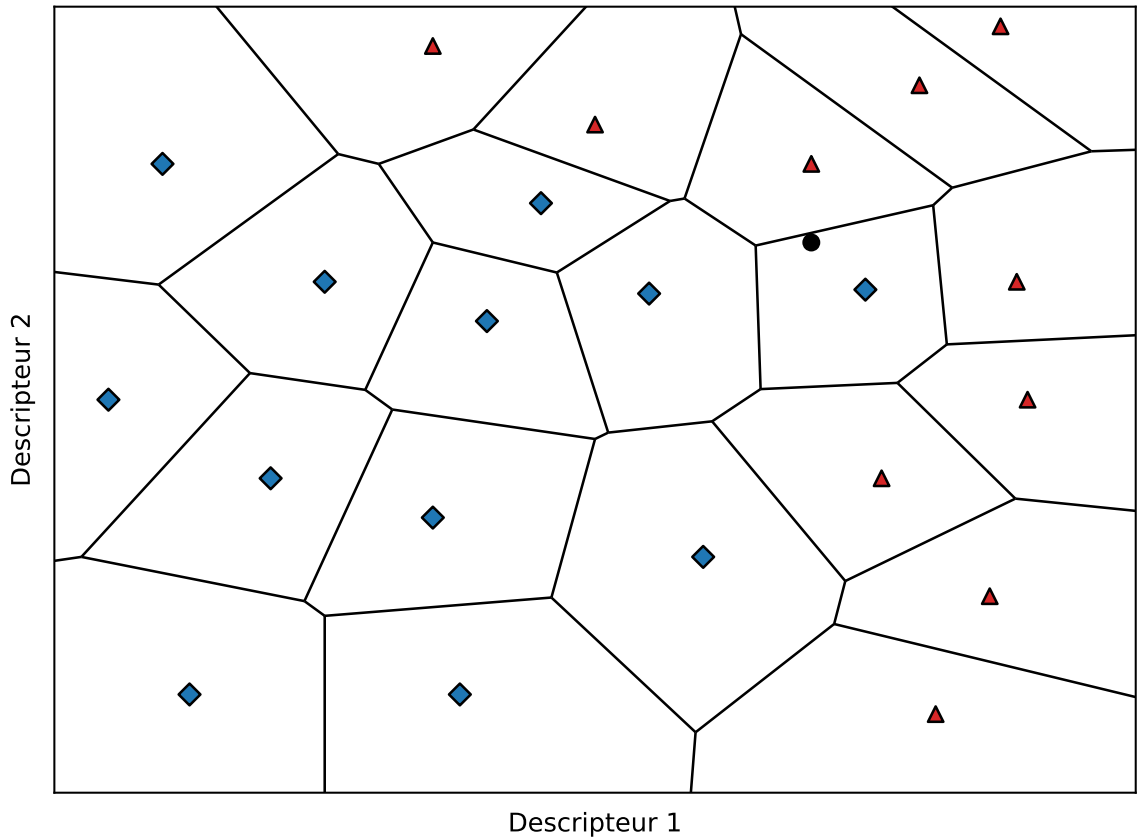
- Cherche le plus proche voisin.
- Affecte au nouveau point le label de la classe correspondante.

Comment ?

- Utilisation des données d'apprentissage pour créer diviser en cellules le plan (diagramme de Voronoï).
- Utilisation des cellules du plan pour sélectionner la classe.

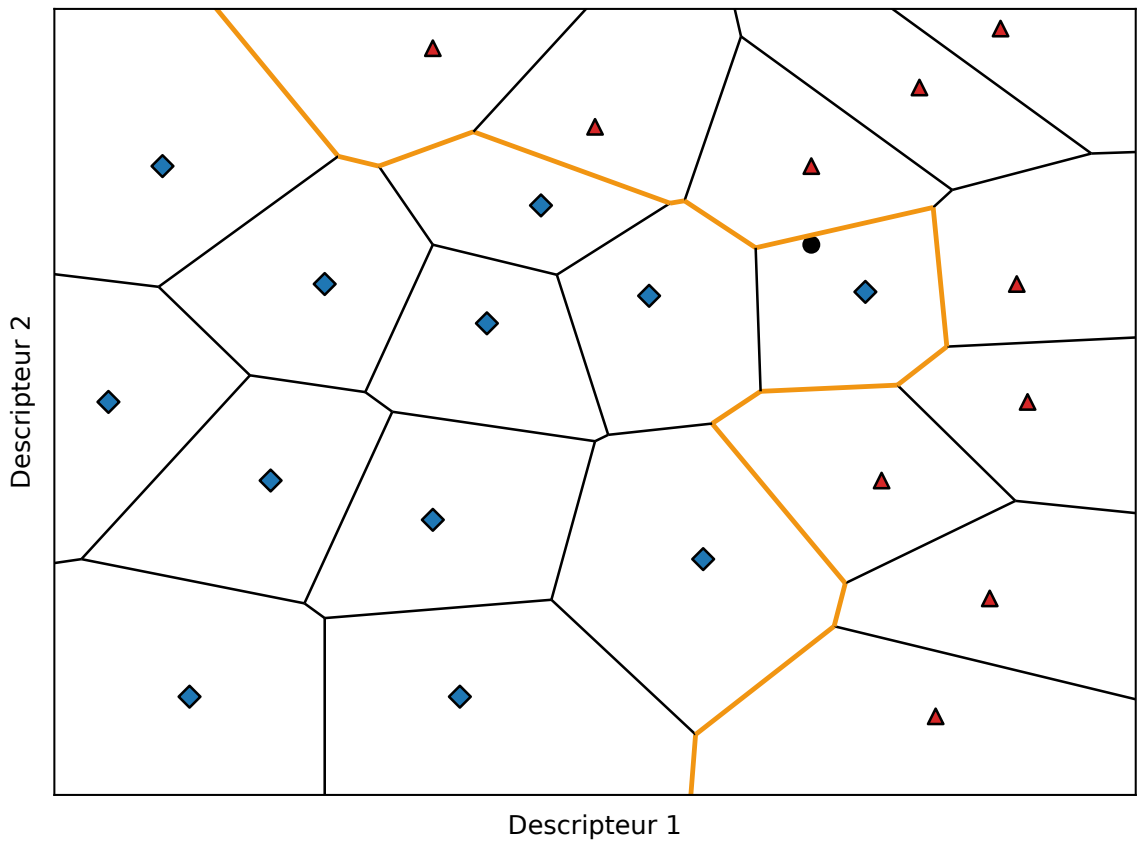
Classification d'un cas plus complexe : diagramme de Voronoï

Quel label pourrait-on affecter au point noir ?



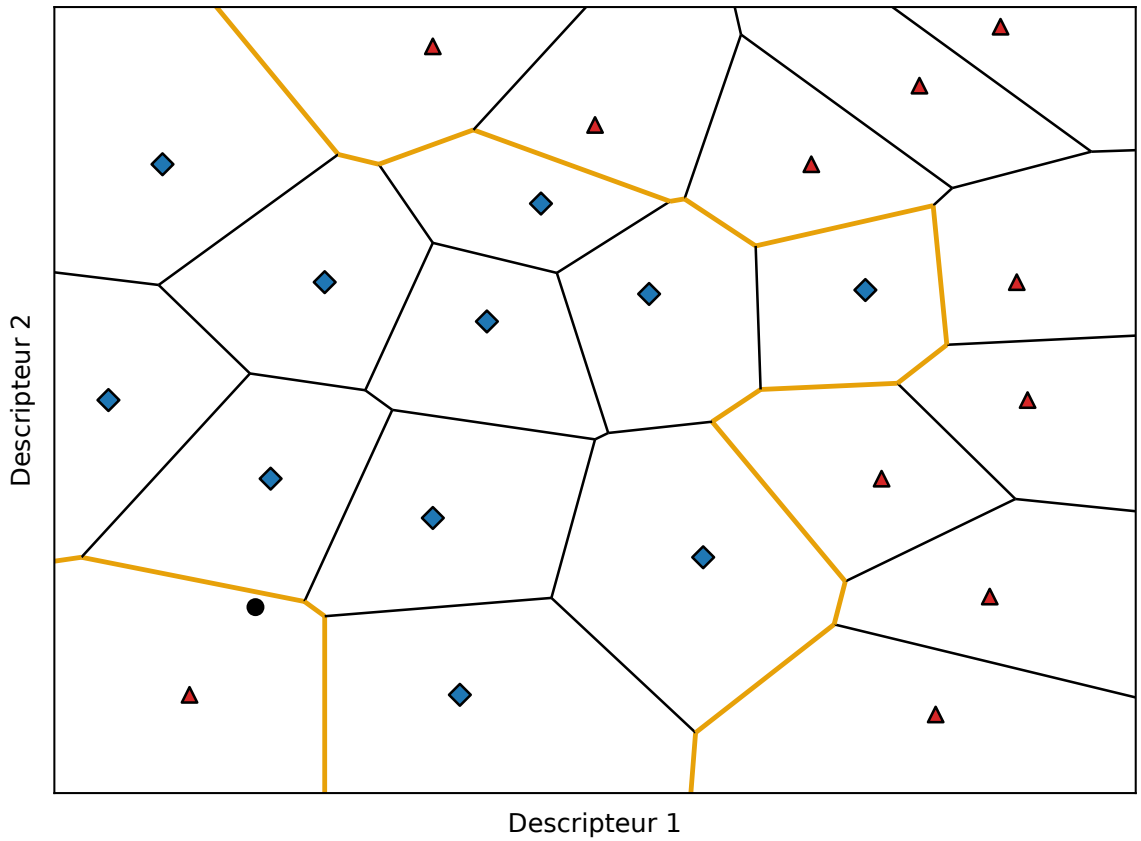
Classification d'un cas plus complexe : diagramme de Voronoï

Définition d'une frontière non linéaire entre les classes



Classification d'un cas plus complexe : diagramme de Voronoï

Quelle robustesse dans certains cas... ?



Robustesse de l'algorithme

Gestion des points aberrants

- Un tel point change la frontière de décision.
- $\rightarrow$ Comportement non souhaité.

Solution envisagée

- Utiliser davantage de voisins, plutôt qu'un seul.
- $\rightarrow$ principe des k plus proches voisins.

KNN pour la classification

Données

- Données d'apprentissage : couples (x_i, c_i).
- Un nouveau point à classer : x.

KNN pour la classification

Données

- Données d'apprentissage : couples (x_i, c_i).
- Un nouveau point à classer : x.

Utilisation

- On possède toutes les distances : $D(x_i, x)$.
- Sélection des k plus proches.
- Affectation de la classe majoritaire représentée parmi les plus proches.

KNN pour la classification : matrice de confusion

Identifier les problèmes de classifications (binaire)

- Vrais positifs (VP) : nombre d'échantillons qui sont correctement classés comme positifs, et leur étiquette réelle est positive.
- Faux positifs (FP) : nombre d'échantillons qui sont incorrectement classés comme positifs, alors que leur étiquette réelle est négative.
- Vrais négatifs (VN) : nombre d'échantillons qui sont correctement classés comme négatifs, alors que leur étiquette réelle est négative.
- Faux négatifs (FN) : nombre d'échantillons qui sont incorrectement classés comme négatifs, alors que leur étiquette réelle est positive.

VP	FN
FP	VN

Matrice de confusion

KNN pour la régression

Données

- Données d'apprentissage : couples (x_i, y_i).
- Un nouveau point : x.

KNN pour la régression

Données

- Données d'apprentissage : couples (x_i, y_i).
- Un nouveau point : x.

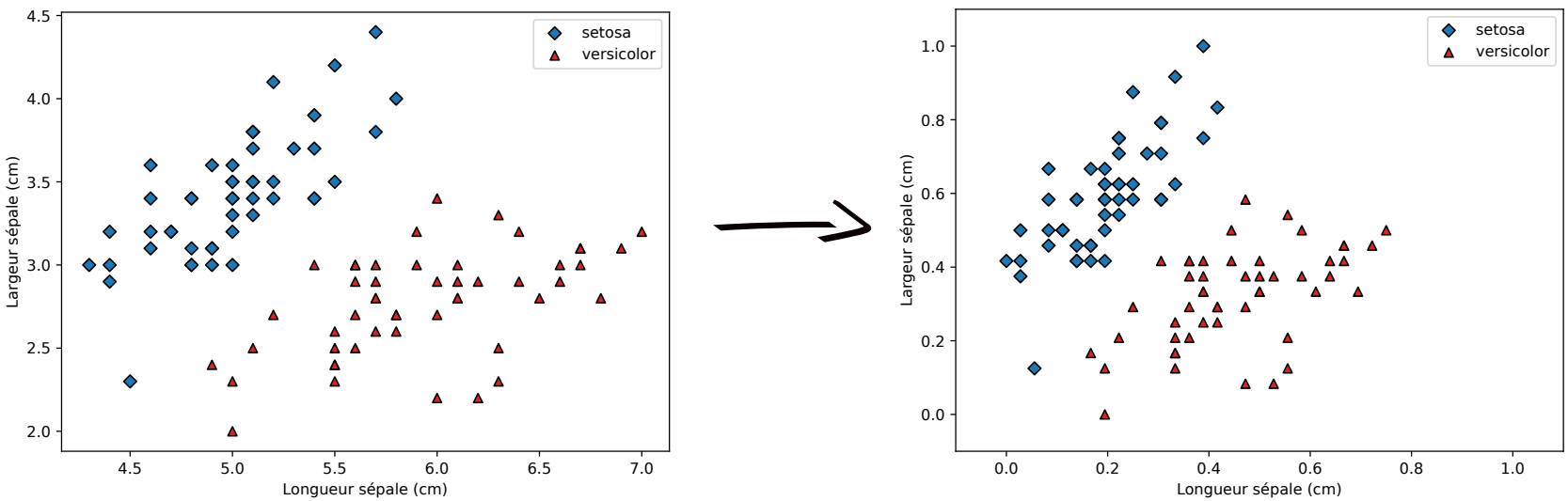
Utilisation

- On possède toutes les distances : $D(x_i, x)$.
- Sélection des k plus proches.
- On définit ainsi : $\hat{y} = \frac{1}{k} \sum_{i=1}^{k} y_i$

KNN : besoin de normalisation

L'importance de la mise à l'échelle

- Algorithme sensible aux données car calcul d'une distance.
- Les distances peuvent aussi être influencées par les unités de mesure.
- Nécessité de normaliser et mettre à l'échelle chaque descripteur.



TP2 : les k plus proches voisins

Plan

Les arbres de décision

Définition

- Technique de classification d'un objet fondée par une suite de tests sur les attributs qui le décrivent.
- La réponse d'un test indique le prochain test à réaliser ou le label affecté.
- Cette suite successive représente un arbre.
- Couramment employée en science naturelles (explicabilité).
- Peut-être adapté à la régression.

Les arbres de décision : un exemple

Données

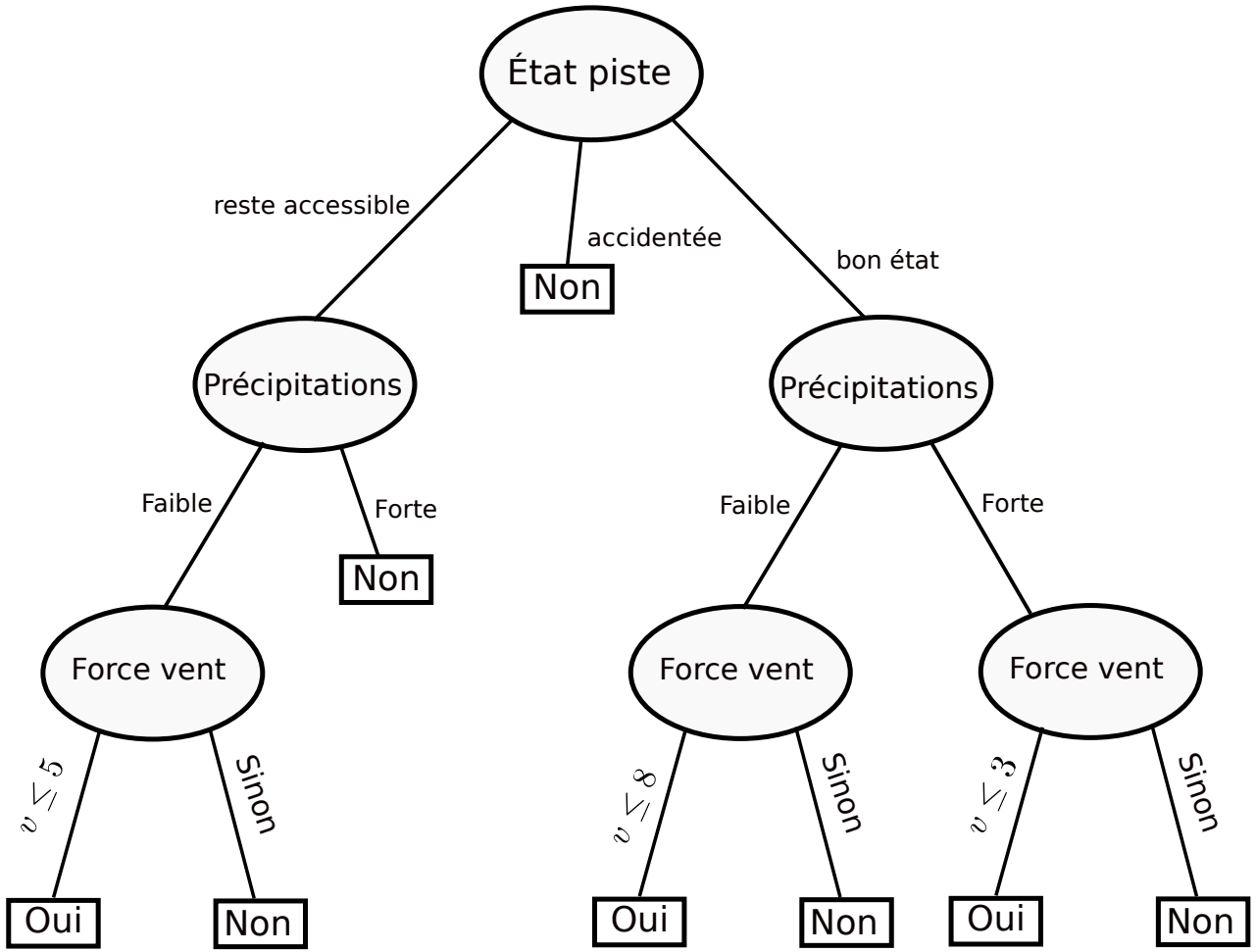
- La force du vent : attribut numérique $\{1, ..., 10\}$.
- L'intensité de précipitations : attribut nominal {faible, forte}.
- État sur la piste cyclable : attribut nominal {accidentée, reste accessible, bon état}

Question

Puis-je prendre mon vélo pour aller travailler sans risque ?

Les arbres de décision : un exemple

Question : Puis-je prendre mon vélo pour aller travailler sans risque ?



Les arbres de décision : les principes

Les avantages

- Nombre moyen de tests peut être réduit : si les d attributs sont binaires, alors on a d test.
- La structure de décision est globale : on peut traiter C classes.
- Interprétation facile

Les arbres de décision : les principes

Les avantages

- Nombre moyen de tests peut être réduit : si les d attributs sont binaires, alors on a d test.
- La structure de décision est globale : on peut traiter C classes.
- Interprétation facile

Comment construire un arbre ?

- Choisir une question qui porte sur un attribut
- Comment choisir le bon attribut ?
- Comment évaluer un arbre ? On pourrait à priori chercher à minimiser le nombre de tests (vers un arbre simple) ?
- Parcours exhaustif des arbres impossible : $\sum_{i=0}^{d-1}(d-i)^{v_i}$
  - d le nombre d'attributs.
  - v le nombre de valeurs par attribut.

Les arbres de décision : les principes

Algorithme 1 : Construction récursive d'un arbre de décision

```
1  Function Construire-arbre(noeud X) begin
2      if Tous les points de X appartiennent à la même classe then
3          Créer une feuille portant le nom de cette classe
4      else
5          Choisir le meilleur attribut pour créer un noeud
6          Le test associé à ce noeud sépare X en deux parties notées X_g et X_d
7          Construire-arbre(X_g)
8          Construire-arbre(X_d)
9      end
10 end
```

Le point important

- Choisir le meilleur attribut pour la séparation des données

Les arbres de décision : position du problème

Définition du problème

- On possède un ensemble d'apprentissage S de m exemples.
- Un exemple est décrit par d attributs : $\{x_i, i = 1, d\}$.
- Un exemple est décrit par une classe $u \in \mathcal{U} = \{u_1, \ldots, u_c\}$

Les arbres de décision : position du problème

Définition du problème

- On possède un ensemble d'apprentissage S de m exemples.
- Un exemple est décrit par d attributs : $\{x_i, i = 1, d\}$.
- Un exemple est décrit par une classe $u \in \mathcal{U} = \{u_1, \ldots, u_c\}$

Représentation à un niveau d'un noeud

- Un noeud est composé de n points de l'échantillon S :
  - Répartis en C classes u_j.
  - Chaque classe u_j possède n_j points.
- Soit a un attribut binaire quelconque ($a \in d$).
- a partage chaque n_j en deux sous parties, comportant :
  - l_j points pour $a = VRAI$.
  - r_j points pour $a = FAUX$.

Les arbres de décision : interprétation probabiliste

Vu comme un tirage aléatoire

- l_j/n est une estimation de $P(a = VRAI, u = u_j)$.
- r_j/n est une estimation de $P(a = FAUX, u = u_j)$.
- l/n est une estimation de $P(a = VRAI)$.
- r/n est une estimation de $P(a = FAUX)$.
- n_j/n est une estimation de $P(u = u_j)$.

Les arbres de décision : information mutuelle

Théorie de l'information

Une mesure naturelle de l'homogénéité entre deux distributions de probabilités à valeurs discrètes (entropie croisée). On parle aussi de dépendance statistique.

$$\mathcal{I}(u,a) = -\sum_{i,j \in \mathcal{D}_u \times \mathcal{D}_a} p(i,j) \log_2 \frac{p(i,j)}{p(i)p(j)}$$

Remarques

- $\mathcal{I}(u,a)$ possède un minimum (0) quand $p(i,j) = p(i)p(j)$: lorsque les deux distributions sont indépendantes.
- $\mathcal{I}(u,a)$ est maximale lorsque les deux distributions sont complètement corrélées.

Les arbres de décision : entropie

Propriétés

- La variable aléatoire u possède une entropie $H(u)$ définit par :

$$\mathrm{H}(u) = -\sum_{i \in \mathcal{D}_u} p(i) \log_2 \big(p(i)\big)$$

- L'entropie u conditionnée par a définit par :

$$\mathrm{H}(u|a) = -\sum_{i,j \in \mathcal{D}_u \times \mathcal{D}_a} p(i,j) \log_2 \big(p(i|j)\big)$$

- D'après la théorie de l'information on a :

$$\mathcal{I}(u,a) = \mathrm{H}(u) - \mathrm{H}(u|a)$$

Les arbres de décision : choix d'un attribut

On peut ainsi estimer :

$$\hat{\mathcal{I}}(u,a) = -\sum_{j=1}^{C} \frac{l_j}{n} \log_2 \frac{l_j/n}{(l/n)\times(n_j/n)} + \frac{r_j}{n} \log_2 \frac{r_j/n}{(r/n)\times(n_j/n)} \qquad (1)$$

$$\hat{\mathsf{H}}(u,a) = -\sum_{j=1}^{C} \frac{l_j}{n} \log_2 \frac{l_j}{l} + \frac{r_j}{n} \log_2 \frac{r_j}{r} \qquad (2)$$

Les arbres de décision : choix d'un attribut

Ce qui nous permet d'exprimer :

$$\hat{\mathsf{H}}(u|a) = \frac{l}{n} J(a = VRAI) + \frac{r}{n} J(a = FAUX) \tag{3}$$

avec :

- $J(a = VRAI) = -\sum_{j=1}^{C} \frac{l_j}{l} \log_2 \frac{l_j}{l}$
- $J(a = FAUX) = -\sum_{j=1}^{C} \frac{r_j}{r} \log_2 \frac{r_j}{r}$

Les arbres de décision : choix d'un attribut

Ce qui nous permet d'exprimer :

$$\hat{\mathsf{H}}(u|a) = \frac{l}{n} J(a = VRAI) + \frac{r}{n} J(a = FAUX) \tag{3}$$

avec :

- $J(a = VRAI) = -\sum_{j=1}^{C} \frac{l_j}{l} \log_2 \frac{l_j}{l}$
- $J(a = FAUX) = -\sum_{j=1}^{C} \frac{r_j}{r} \log_2 \frac{r_j}{r}$

On recherche finalement l'attribut qui

$$i^\star \arg\min_{i=1,d} \hat{\mathsf{H}}(u|a_i)$$

Les arbres de décision : un exemple

Problème
Un enfant peut-il aller jouer chez son voisin ou non ?

Données
Récoltées sur les 8 derniers jours en fonction de 4 descripteurs :

- Devoir finis (DF).
- Bonne humeur de sa mère (BH).
- Beau temps (BT).
- Goûter pris (GP).

Les arbres de décision : un exemple

	Devoirs finis (DF)	Mère de bonne humeur (BH)	Beau temps (BT)	Goûter Pris (GP)	**Décision**
1	vrai	faux	vrai	faux	oui
2	faux	vrai	faux	vrai	oui
3	vrai	vrai	vrai	faux	oui
4	vrai	faux	vrai	vrai	oui
5	faux	vrai	vrai	vrai	non
6	faux	vrai	faux	faux	non
7	vrai	faux	faux	vrai	non
8	vrai	vrai	faux	faux	non

Les arbres de décisions : un exemple

	Devoirs finis (DF)	Mère de bonne humeur (BH)	Beau temps (BT)	Goûter Pris (GP)	**Décision**
1	**vrai**	faux	vrai	faux	**oui**
2	**faux**	vrai	faux	vrai	**oui**
3	**vrai**	vrai	vrai	faux	**oui**
4	**vrai**	faux	vrai	vrai	**oui**
5	**faux**	vrai	vrai	vrai	**non**
6	**faux**	vrai	faux	faux	**non**
7	**vrai**	faux	faux	vrai	**non**
8	**vrai**	vrai	faux	faux	**non**

$$\mathrm{H}(u = oui|DF) = \frac{5}{8}J(DF = vrai) + \frac{3}{8}J(DF = faux)$$

avec :

- $J(DF = vrai) = -\frac{3}{5}\log_2\left(\frac{3}{5}\right) - \frac{2}{5}\log_2\left(\frac{2}{5}\right)$
- $J(DF = faux) = -\frac{1}{3}\log_2\left(\frac{1}{3}\right) - \frac{2}{3}\log_2\left(\frac{2}{3}\right)$

Soit : $\mathrm{H}(oui|DF) \approx 0.95$

Les arbres de décisions : un exemple

De la même manière, on obtient :

- $\mathrm{H}(oui|DF) \approx 0.95$
- $\mathrm{H}(oui|BH) \approx 0.95$
- $\mathrm{H}(oui|BT) \approx 0.81$
- $\mathrm{H}(oui|GT) \approx 1$

On choisit donc pour racine : **Beau Temps**

Les arbres de décisions : un exemple

Données pour la branche de gauche :

	Devoirs finis (DF)	Mère de bonne humeur (BH)	Goûter Pris (GP)	**Décision**
1	vrai	faux	faux	oui
3	vrai	vrai	faux	oui
4	vrai	faux	vrai	oui
5	faux	vrai	vrai	non

Les arbres de décisions : un exemple

Données pour la branche de droite :

	Devoirs finis (DF)	Mère de bonne humeur (BH)	Goûter Pris (GP)	**Décision**
2	faux	vrai	vrai	oui
6	faux	vrai	faux	non
7	vrai	faux	vrai	non
8	vrai	vrai	faux	non

Les arbres de décisions : attributs non binaires

- **Cas nominal :** généralisable facilement en plusieurs attributs binaires.

  $\rightarrow$ attribut `couleur` composé de trois couleurs, peut être transformé en 3 attributs binaires de présence ou non de la couleur.

- **Cas continu :** nombre de données d'apprentissage est fini $\Longrightarrow$ nombre de valeurs que prend cet attribut sur les exemples est aussi fini.

  $\rightarrow$ un seuil $s(a)$ peut-être trouvé qui minimise l'entropie.

Les arbres de décisions : un exemple

TD1 [partie 1] : calculer l'arbre de décision complet relatif aux données

Les arbres de décision : principe d'élagage

Pourquoi élaguer ?

- L'arbre actuel construit est dit T_{max}.
- L'arbre est composé de feuilles *pures* : exemples de la même classe.
- Taux d'erreur actuel sur les données d'apprentissage est de 0.
- **Risque :** mauvaise estimation sur de nouvelles données.

Les arbres de décision : principe d'élagage

Pourquoi élaguer ?

- L'arbre actuel construit est dit T_{max}.
- L'arbre est composé de feuilles *pures* : exemples de la même classe.
- Taux d'erreur actuel sur les données d'apprentissage est de 0.
- **Risque :** mauvaise estimation sur de nouvelles données.

Élaguer un arbre

Méthode de régularisation ou de sélection de modèle. Il en existe deux méthodes :

- le pré-élagage
- le post-élagage

Les arbres de décision : pré-élagage

Fonctionnement du pré-élagage

Une fois l'attribut sélectionné pour former un noeud, on utilise un seuil sur l'entropie pour savoir si l'on continue ou non.

- $\rightarrow$ Ce qui revient à admettre que s'il existe une classe suffisamment majoritaire sous un noeud, on peut considérer que c'est une feuille.
- $\rightarrow$ L'inconvénient principal de cette méthode et qu'elle ne prend en compte qu'un critère **local** (feuille examinée).

On préfère souvent des méthodes d'élagage a posteriori du fait de cet inconvénient principal.

Les arbres de décision : post-élagage

Principe du post-élagage

On construit d'abord l'arbre complètement, puis on cherche à l'élaguer (le simplifier) progressivement en remontant des feuilles vers la racine.

- Utilisation d'un critère de qualité de l'arbre comme arrêt.
- On utilise une base de validation pour mesurer cette qualité.

Les arbres de décision : post-élagage

Fonctionnement du post-élagage

- On note T_{max} l'arbre complet composée de feuilles pures.
- On note $S = (T_{max}, T_1, \ldots, T_k, ..., T_n)$.
- Pour passer de T_k à T_{k+1}, on transforme un noeud dans T_k en feuille.

Évaluation de T_{k+1}

- Comparer le « coût » de l'arbre élagué et celui de l'arbre non élagué.
- Arrêter l'élagage quand le coût de T_{k+1} dépasse le coût de T_k.
- Le coût est généralement une mesure multi-critères : mesure de l'erreur commise et mesure de la complexité de l'arbre.

Les arbres de décision : post-élagage

Critère de choix du noeud v qui minimise :

$$\bar{w}(T_k, v) = \frac{MC(v,k) - MCT(v,k)}{n(k).(nt(v,k) - 1)}$$

où :

- $MC(v,k)$ est le nombre d'exemples de l'ensemble d'apprentissage mal classés par le noeud v de T_k dans l'arbre élagué à v.
- $MCT(v,k)$ est le nombre d'exemple de l'ensemble d'apprentissage mal classés sous le noeud v dans l'arbre
- $n(k)$ est le nombre de feuilles de T_k
- $nt(v,k)$ est le nombre de feuilles du sous-arbre de T_k situé sous le noeud v

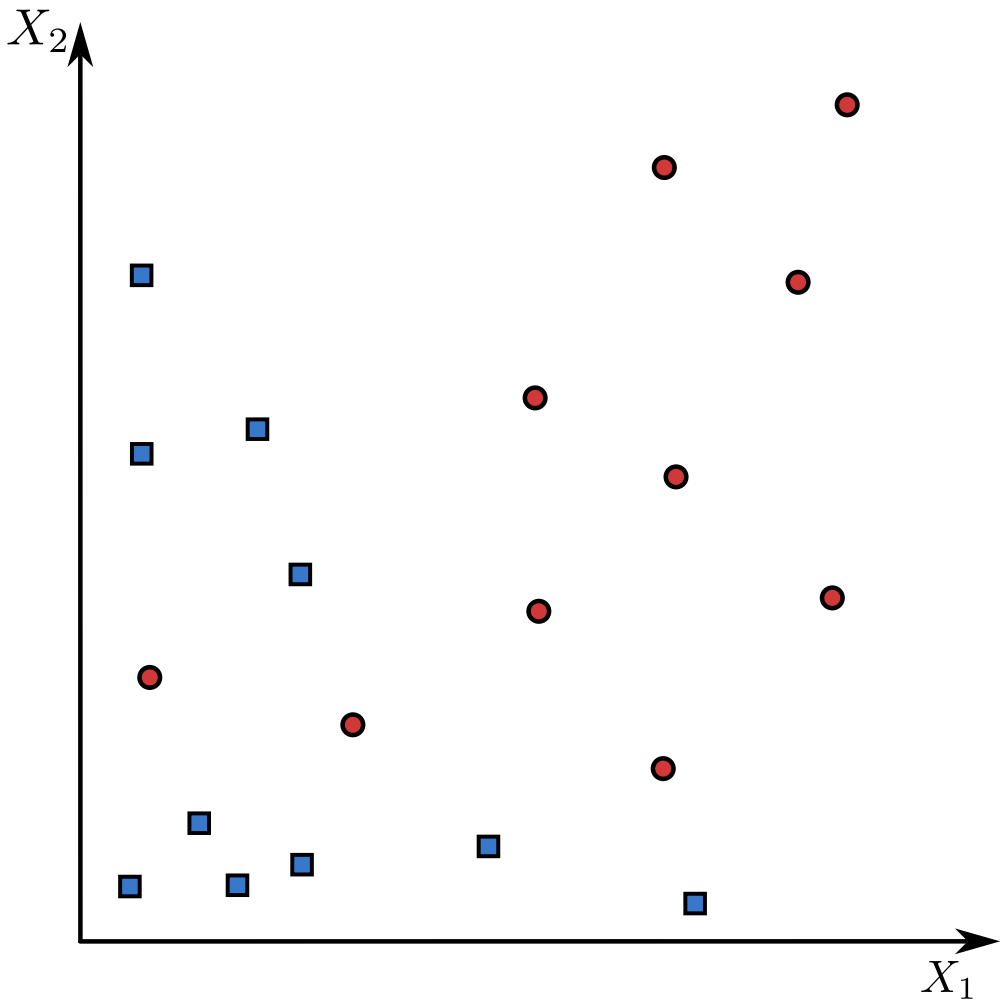
Les arbres de décisions : post-élagage

Algorithme 2 : Élagage d'un arbre de décision

Function élaguer(T_{max})
begin
  $k \leftarrow 0$
  $T_k \leftarrow T_{max}$
  while T_k *a plus d'un noeud* **do**
    for *noeud* v *de* T_k **do**
      calculer le critère $\bar{w}(T_k, v)$ sur l'ensemble d'apprentissage
    end
    choisir le noeud v_m pour lequel le critère est minimum
    T_{k+1} se déduit de T_k en y remplaçant v_m par une feuille
    $k \leftarrow k + 1$
  end
  Dans l'ensemble des arbres $\{T_{max}, T_1, \ldots, T_k, \ldots, T_n\}$, choisir celui qui a la plus petite erreur de classification sur l'ensemble de validation.
end

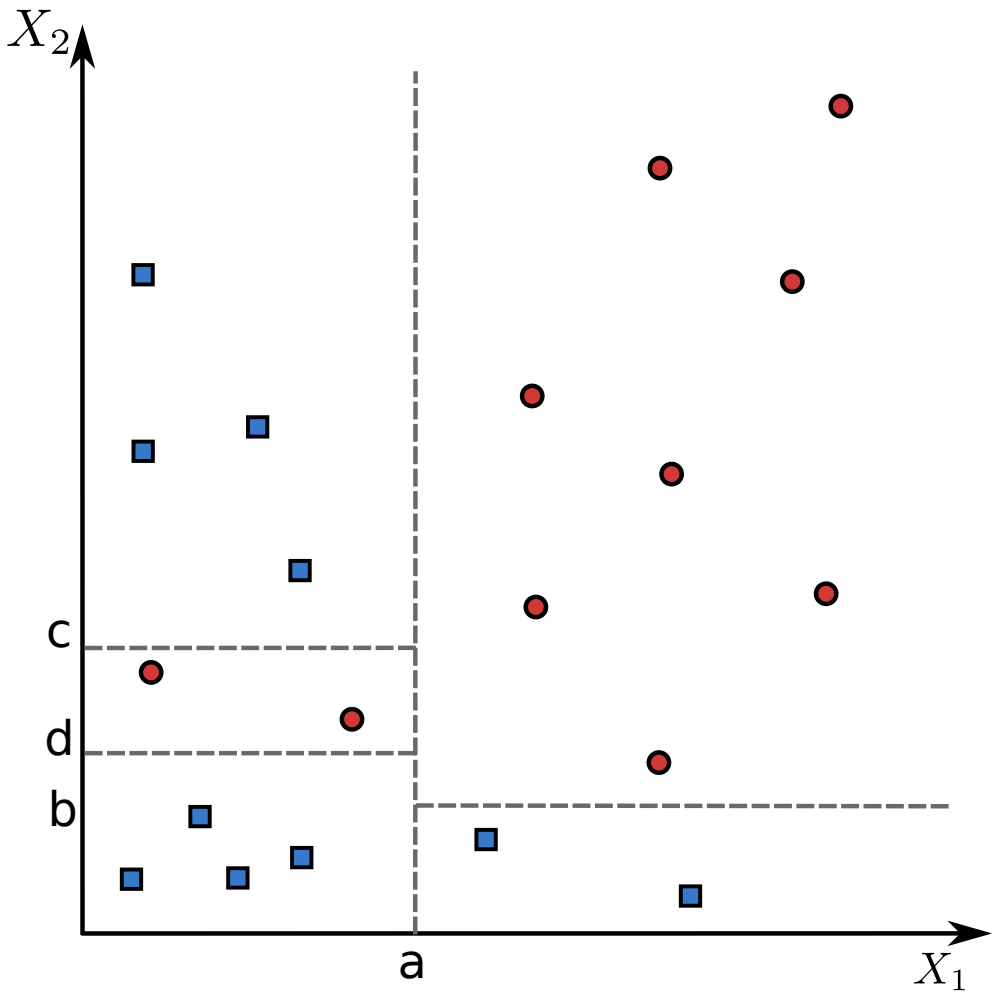
Arbre de décision : exemple post-élagage

Prenons comme exemple, cette base d'apprentissage :



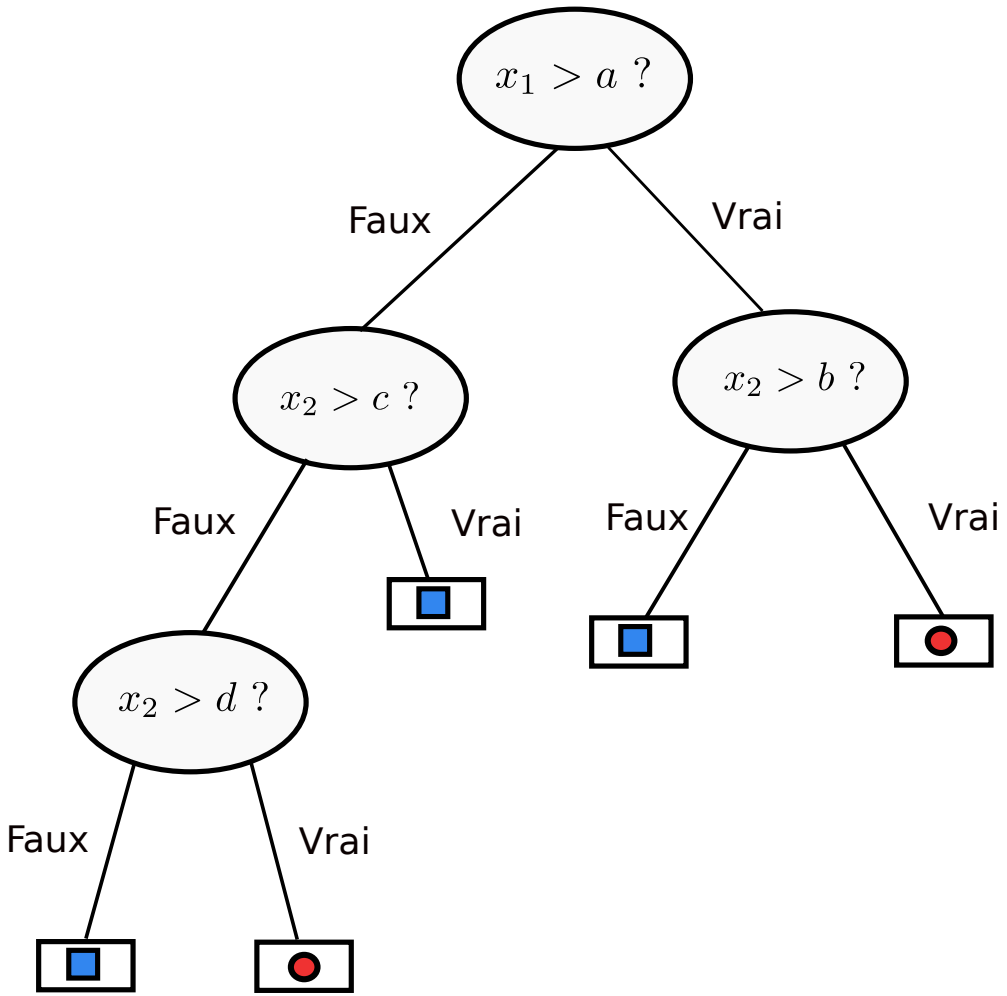
Arbre de décision : exemple post-élagage

L'arbre géométrique correspondant est :



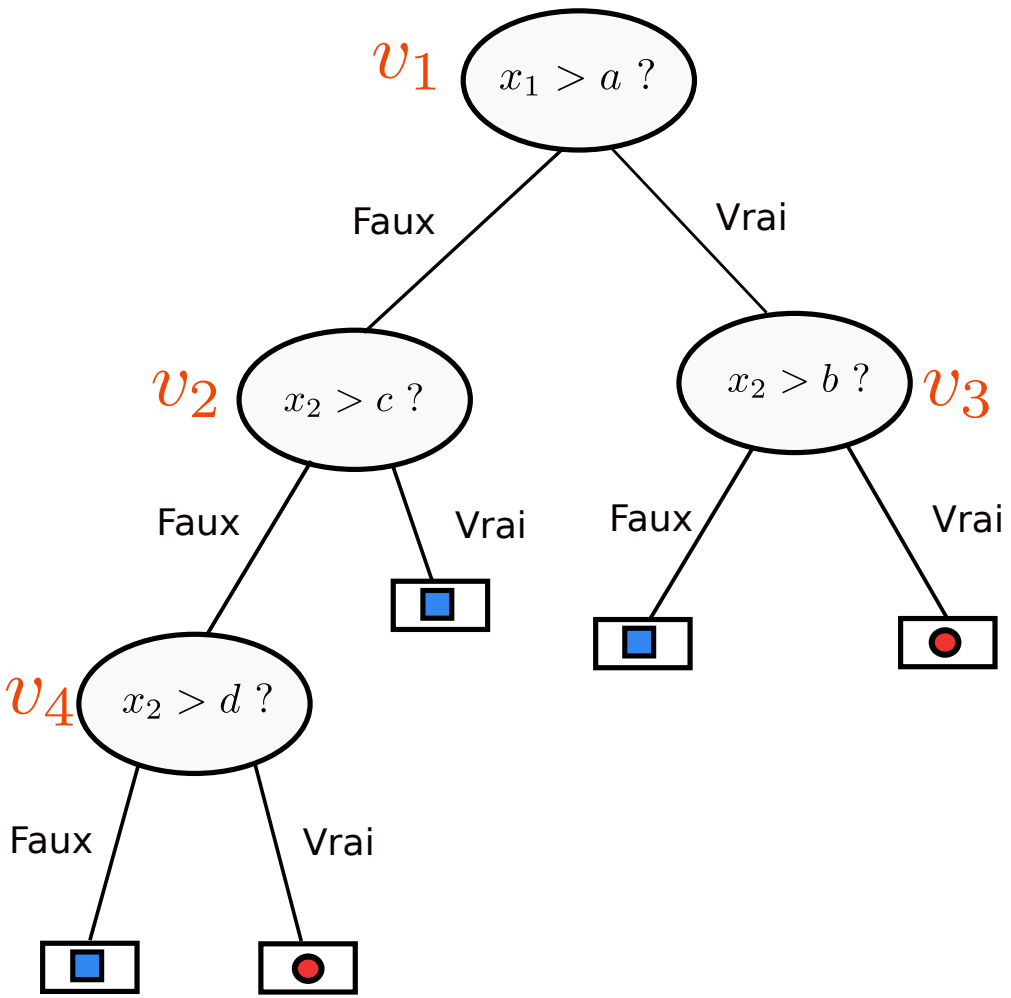
Arbre de décision : exemple post-élagage

Avec l'arbre de décision correspondant :

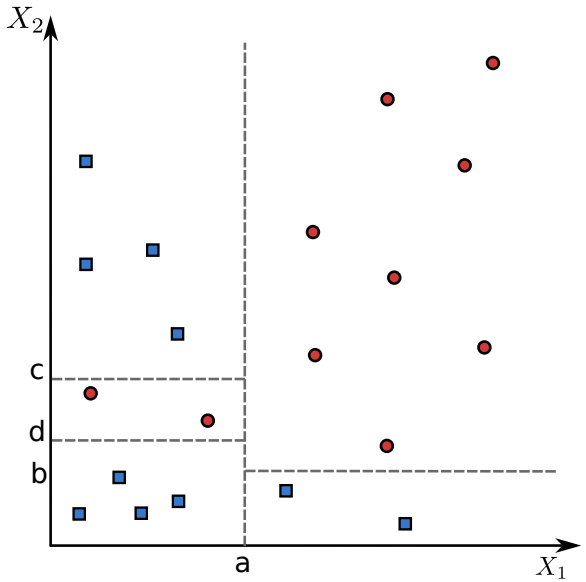


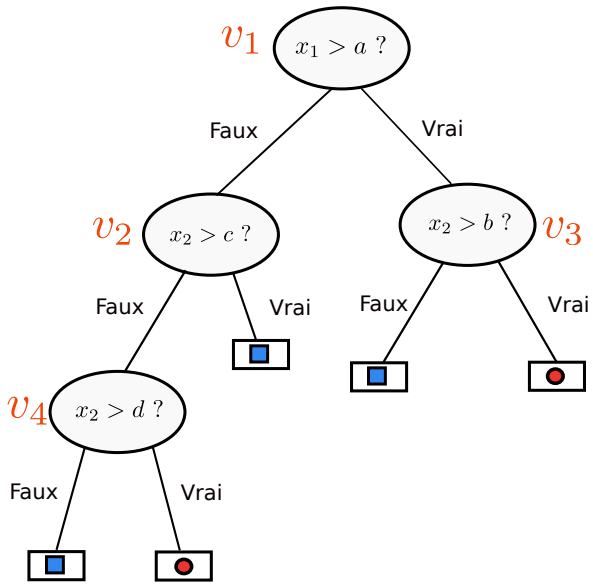
Arbre de décision : exemple post-élagage

Pour élaguer, on doit calculer $\bar{w}(T_k, v_i)$:



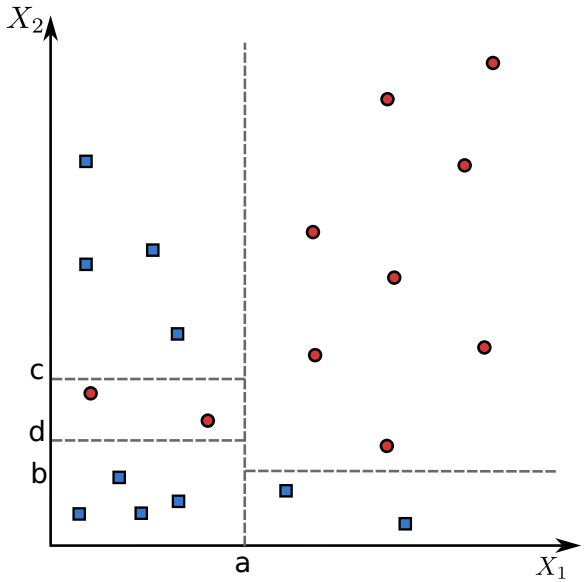
Arbre de décision : exemple post-élagage

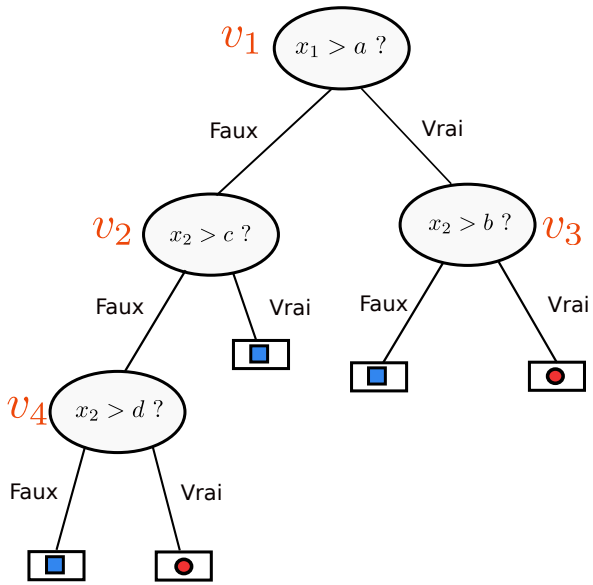




- $\bar{w}(T_{max}, v_1) = \frac{MC(v_1,k) - MCT(v_1,k)}{n(k).(nt(v_1,k)-1)} = \frac{10-0}{5(5-1)} = \frac{1}{2}$

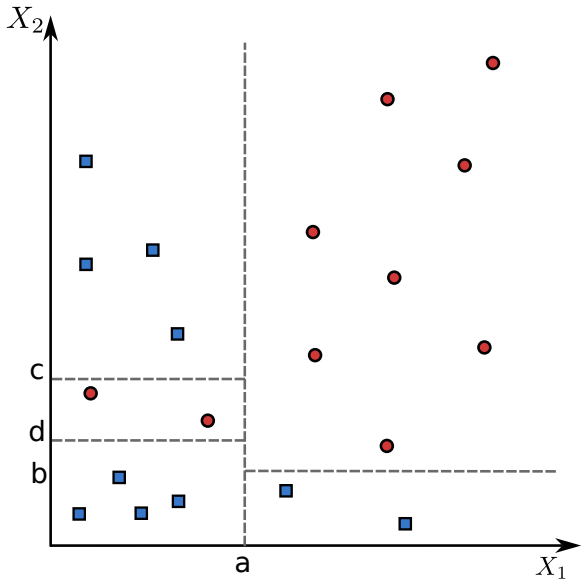
Arbre de décision : exemple post-élagage

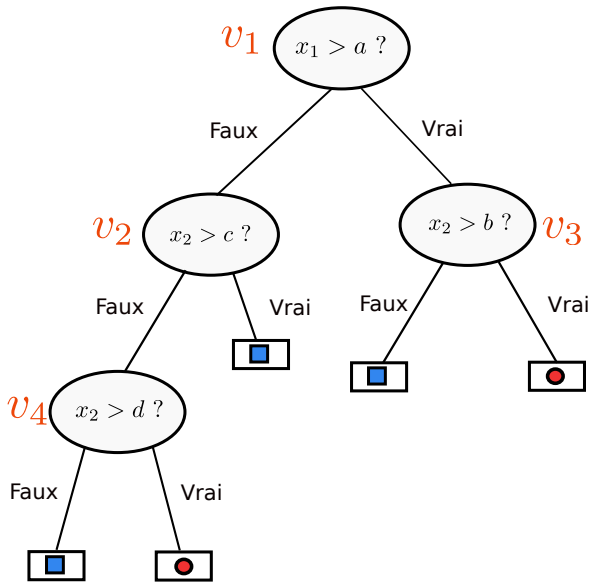




- $\bar{w}(T_{max}, v_1) = \frac{MC(v_1,k) - MCT(v_1,k)}{n(k).(nt(v_1,k)-1)} = \frac{10-0}{5(5-1)} = \frac{1}{2}$
- $\bar{w}(T_{max}, v_2) = \frac{2-0}{5(3-1)} = \frac{2}{10}$

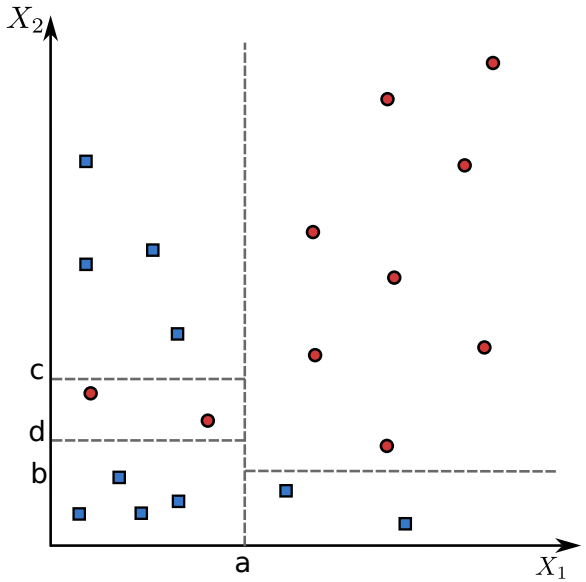
Arbre de décision : exemple post-élagage

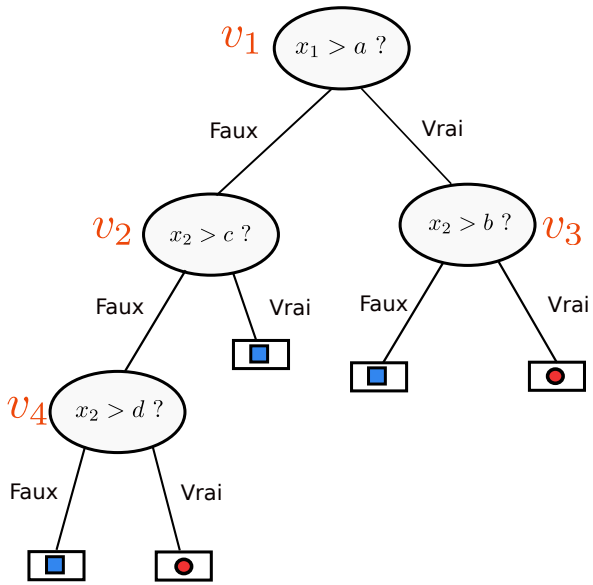




- $\bar{w}(T_{max}, v_1) = \frac{MC(v_1,k) - MCT(v_1,k)}{n(k).(nt(v_1,k)-1)} = \frac{10-0}{5(5-1)} = \frac{1}{2}$
- $\bar{w}(T_{max}, v_2) = \frac{2-0}{5(3-1)} = \frac{2}{10}$
- $\bar{w}(T_{max}, v_3) = \frac{2-0}{5(2-1)} = \frac{2}{5}$

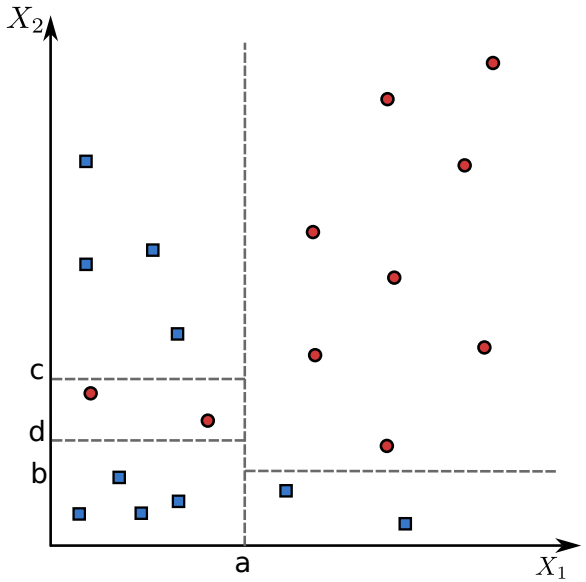
Arbre de décision : exemple post-élagage

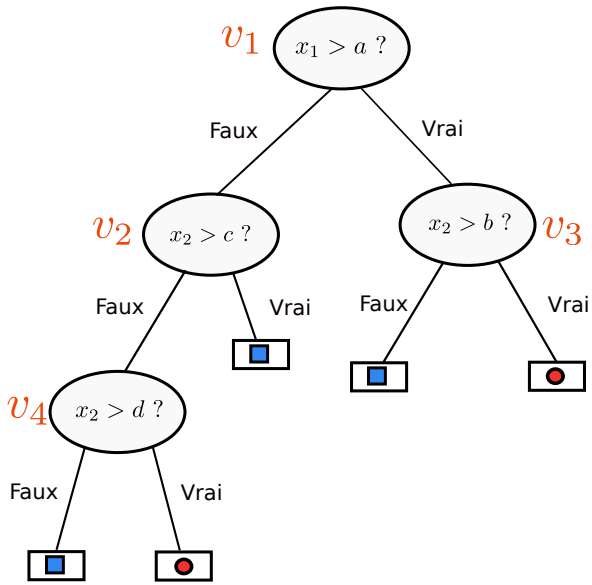




- $\bar{w}(T_{max}, v_1) = \frac{MC(v_1,k) - MCT(v_1,k)}{n(k).(nt(v_1,k)-1)} = \frac{10-0}{5(5-1)} = \frac{1}{2}$
- $\bar{w}(T_{max}, v_2) = \frac{2-0}{5(3-1)} = \frac{2}{10}$
- $\bar{w}(T_{max}, v_3) = \frac{2-0}{5(2-1)} = \frac{2}{5}$
- $\bar{w}(T_{max}, v_4) = \frac{2-0}{5(2-1)} = \frac{2}{5}$

Arbre de décision : exemple post-élagage

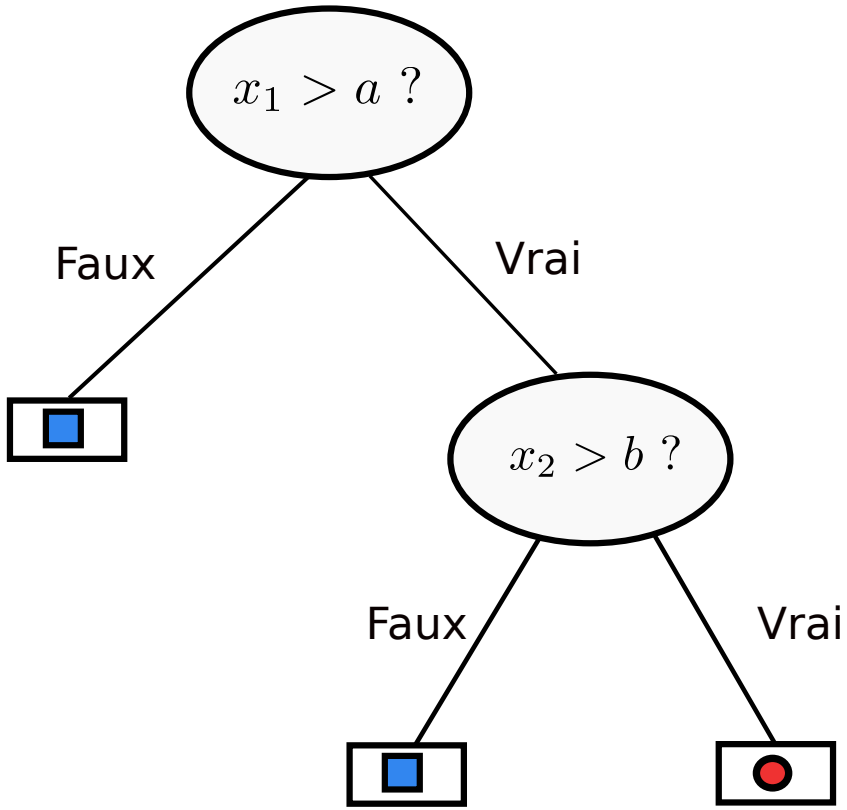




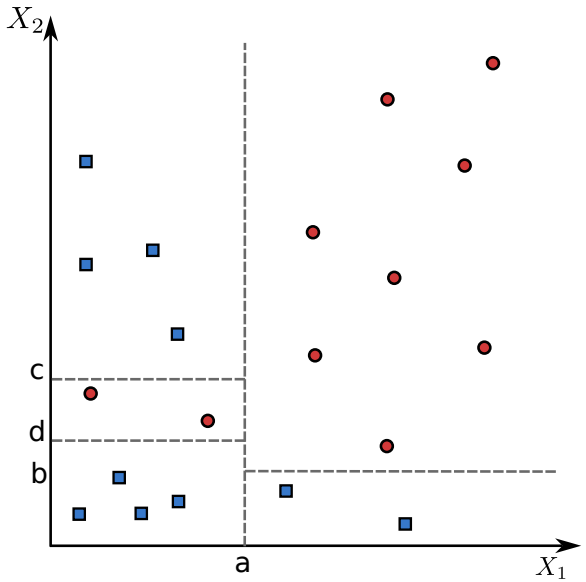
- $\bar{w}(T_{max}, v_1) = \frac{MC(v_1,k) - MCT(v_1,k)}{n(k).(nt(v_1,k)-1)} = \frac{10-0}{5(5-1)} = \frac{1}{2}$
- $\bar{w}(T_{max}, v_2) = \frac{2-0}{5(3-1)} = \frac{2}{10}$
- $\bar{w}(T_{max}, v_3) = \frac{2-0}{5(2-1)} = \frac{2}{5}$
- $\bar{w}(T_{max}, v_4) = \frac{2-0}{5(2-1)} = \frac{2}{5}$

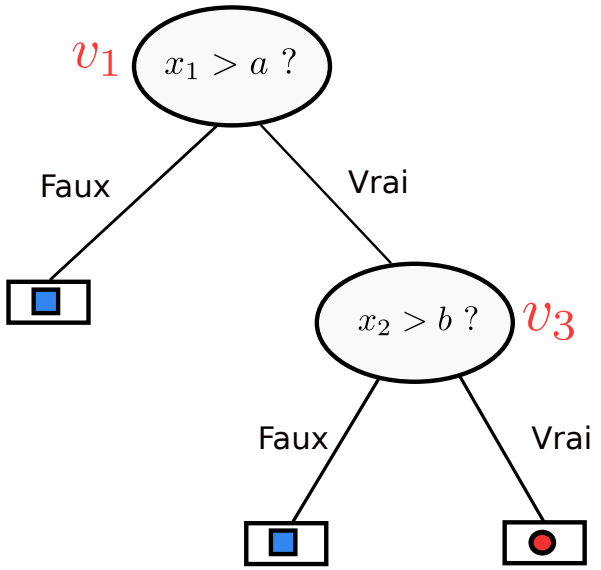
Arbre de décision : exemple post-élagage

On obtient ainsi T_1 :



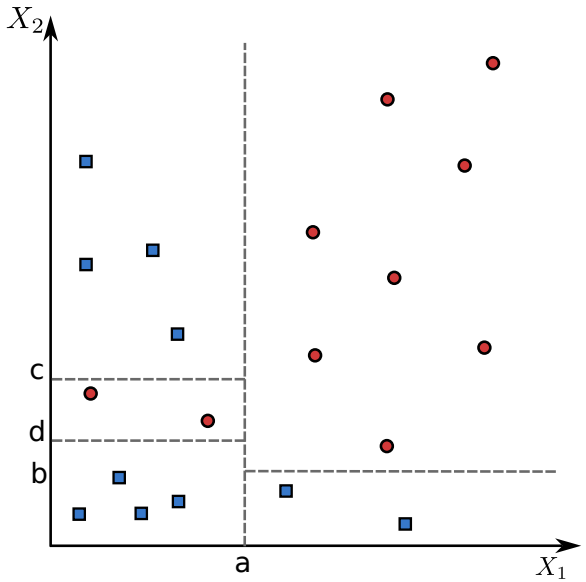
Arbre de décision : exemple post-élagage

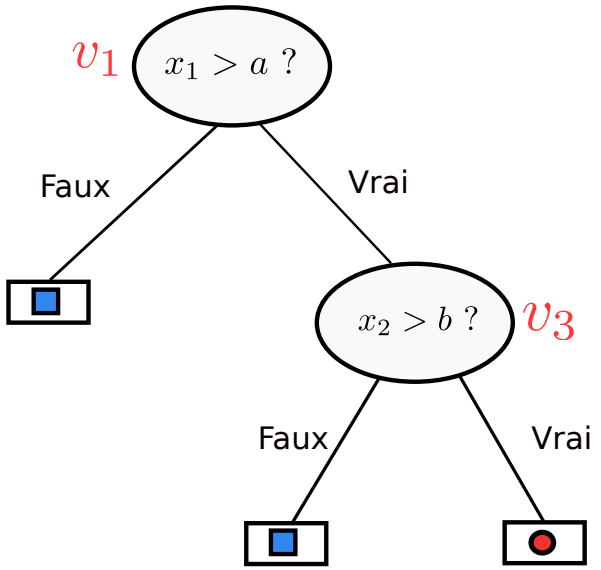




- $\bar{w}(T_1, v_1) = \frac{10-2}{3(3-1)} = \frac{4}{3}$

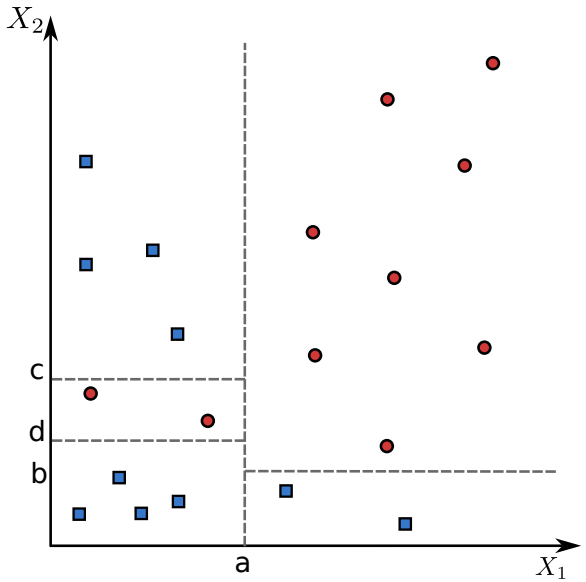
Arbre de décision : exemple post-élagage

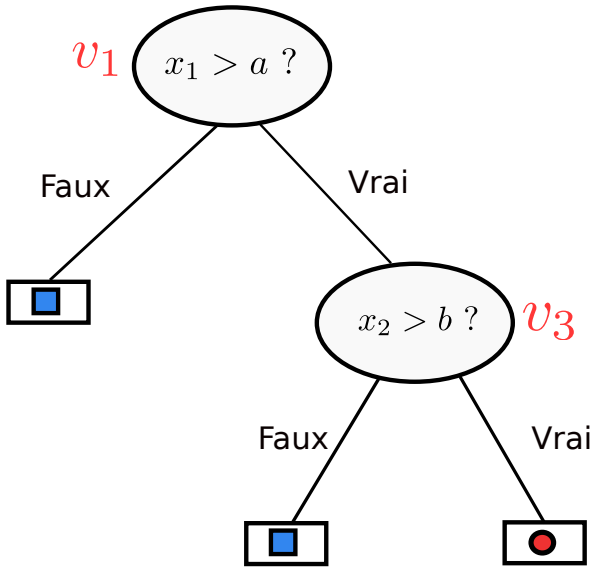




- $\bar{w}(T_1, v_1) = \frac{10-2}{3(3-1)} = \frac{4}{3}$
- $\bar{w}(T_1, v_3) = \frac{4-2}{3(2-1)} = \frac{2}{3}$

Arbre de décision : exemple post-élagage

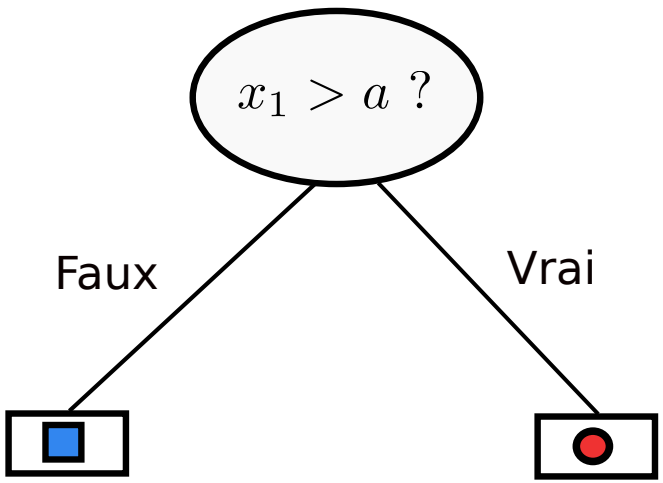




- $\bar{w}(T_1, v_1) = \frac{10-2}{3(3-1)} = \frac{4}{3}$
- $\bar{w}(T_1, v_3) = \frac{4-2}{3(2-1)} = \frac{2}{3}$

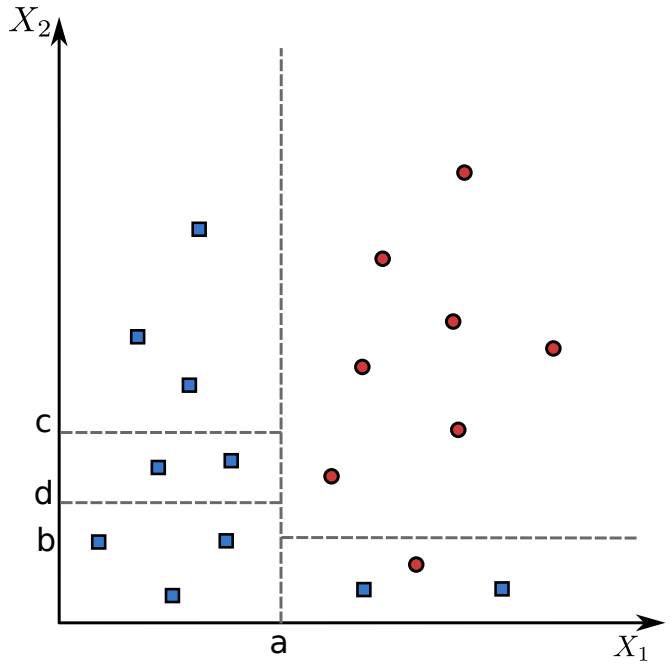
Arbre de décision : exemple post-élagage

On obtient ainsi T_2 :



Arbre de décision : exemple post-élagage

Validation sur un ensemble de validation :



- Erreur $T_{max} = \frac{3}{17}$
- Erreur $T_1 = \frac{1}{17}$
- Erreur $T_2 = \frac{2}{17}$

TD1 [partie 2] : Arbre de décision avec scikit-learn

Plan

Plan

Apprentissage non supervisé

Objectifs

- Données à disposition sans label.
- On cherche à trouver une partition π de ces données en classes dites « naturelles ».
  $\rightarrow$ Chaque donnée se voit attribuer une classe et une seule.
- On parle aussi de « classification automatique ».

Définition

Soit un ensemble fini S. Une partition π de S est un ensemble de parties de S, non vides et disjointes deux à deux, dont l'union est S. Si s désigne un élément de S, il existe donc un unique élément, ou bloc, de π comprenant s.

Apprentissage non supervisé

Comment partitionner ?

- Déterminer des « clusters ».
- Utilisation des attributs (descripteurs de chaque point).
- Une méthode connue : fondée sur la distance entre points.
  - $\rightarrow$ Exemple : distance euclidienne

Apprentissage non supervisé

Comment partitionner ?

- Déterminer des « clusters ».
- Utilisation des attributs (descripteurs de chaque point).
- Une méthode connue : fondée sur la distance entre points.
  - $\rightarrow$ Exemple : distance euclidienne

La p-distance entre deux points composés de d descripteurs :

- $\mathbf{x} = (x_1, \ldots, x_d)$
- $\mathbf{y} = (y_1, \ldots, y_d)$

est définie par :

$$\Delta_p(\mathbf{x}, \mathbf{y}) = \sqrt[p]{\sum_{i=1}^{d} |x_i - y_i|^p}$$

Apprentissage non supervisé

Centre de gravité d'un ensemble de données :

Pour chaque m point composées de d descripteurs :

- $\mathbf{x_1} = (x_{11}, \ldots, x_{1d})$
- $\mathbf{x_m} = (x_{m1}, \ldots, y_{md})$

On peut définir le centre de gravité liées aux données par :

$$\mu = \left(\frac{1}{m} \sum_{i=1}^{m} x_{i1}, \ldots, \frac{1}{m} \sum_{i=1}^{m} x_{id} \right)$$

Plan

Les K-means

Objectifs de l'algorithme

- Cherche à trouver une partition π_i à k clusters (C classes).
- Processus itératif d'affectation des centres de gravités $\mu_1, \ldots, \mu_C$ des C classes.

Comment ?

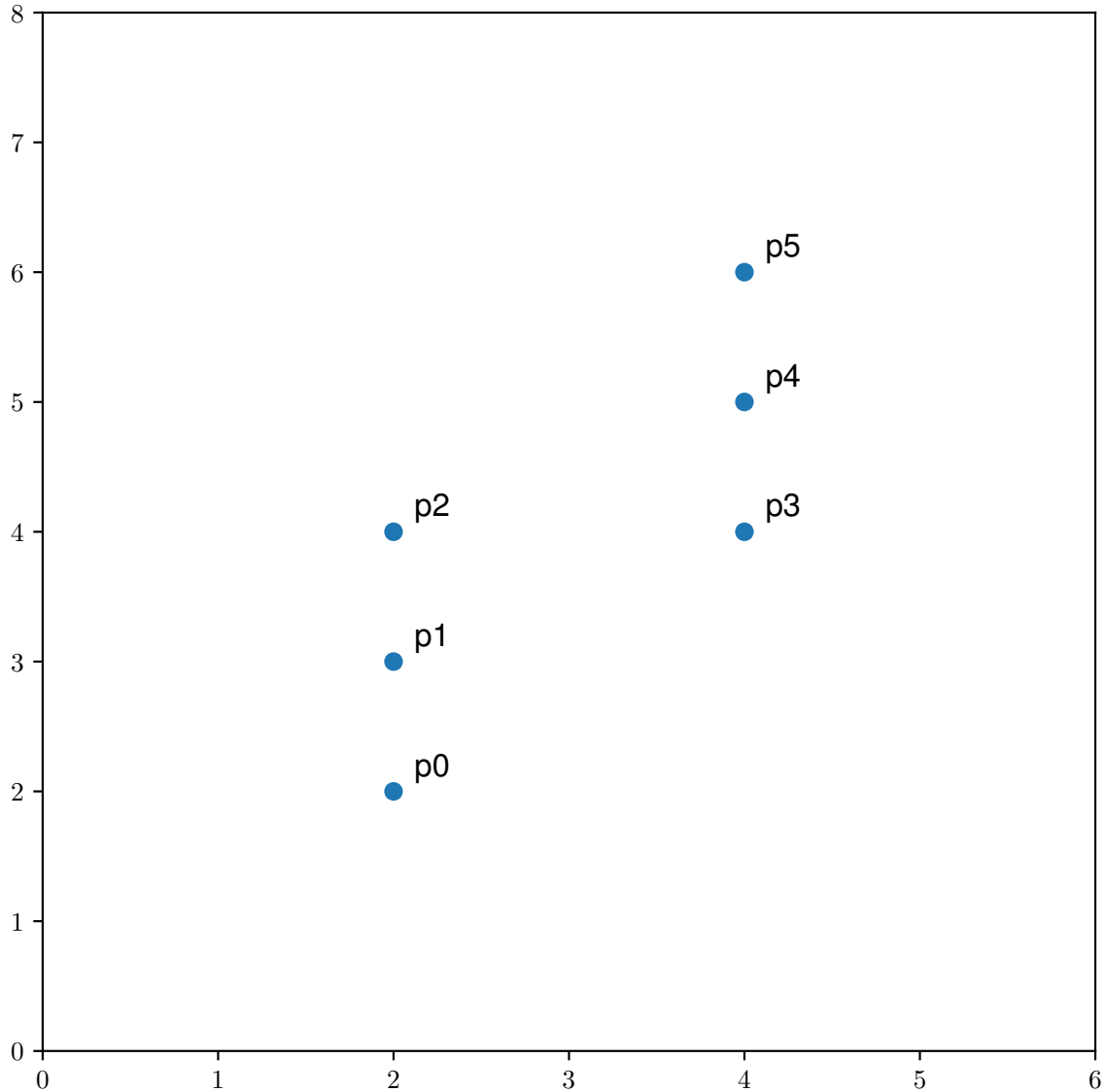
En minimisant la somme des variances intra-classes (variance de la partition π_i étudiée) :

$$T = \frac{1}{m} \sum_{j=1}^{C} \sum_{i=1}^{m} \delta_i^j (\mathbf{x}_i - \mu_j)^2$$

avec :

- C le nombre de classes.
- m le nombre de points.
- $\delta_i^j = 1$ si $i = j$ et 0 sinon.

Les K-means : un exemple



Les K-means : un exemple

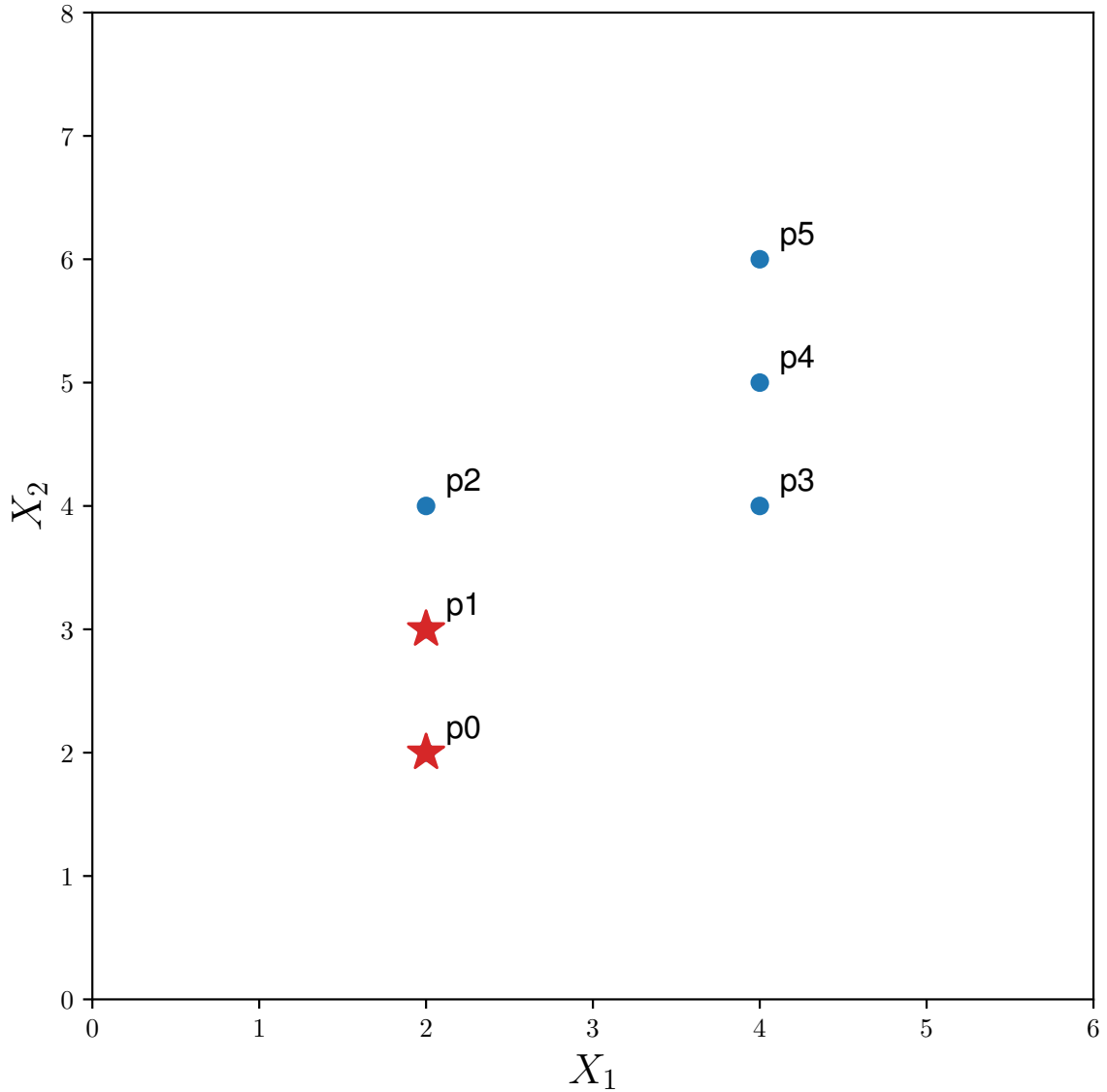
Initialisation

- On sélectionne le nombre de classes souhaitées, ici 2.
- On définit deux points comme étant les centroïdes de départ.

Choix des centroïdes

- $p0(2, 2)$
- $p1(2, 3)$

Les K-means : un exemple



Les K-means : un exemple

Affectation

- Pour chaque point, on affecte la classe (centroïde le plus proche).

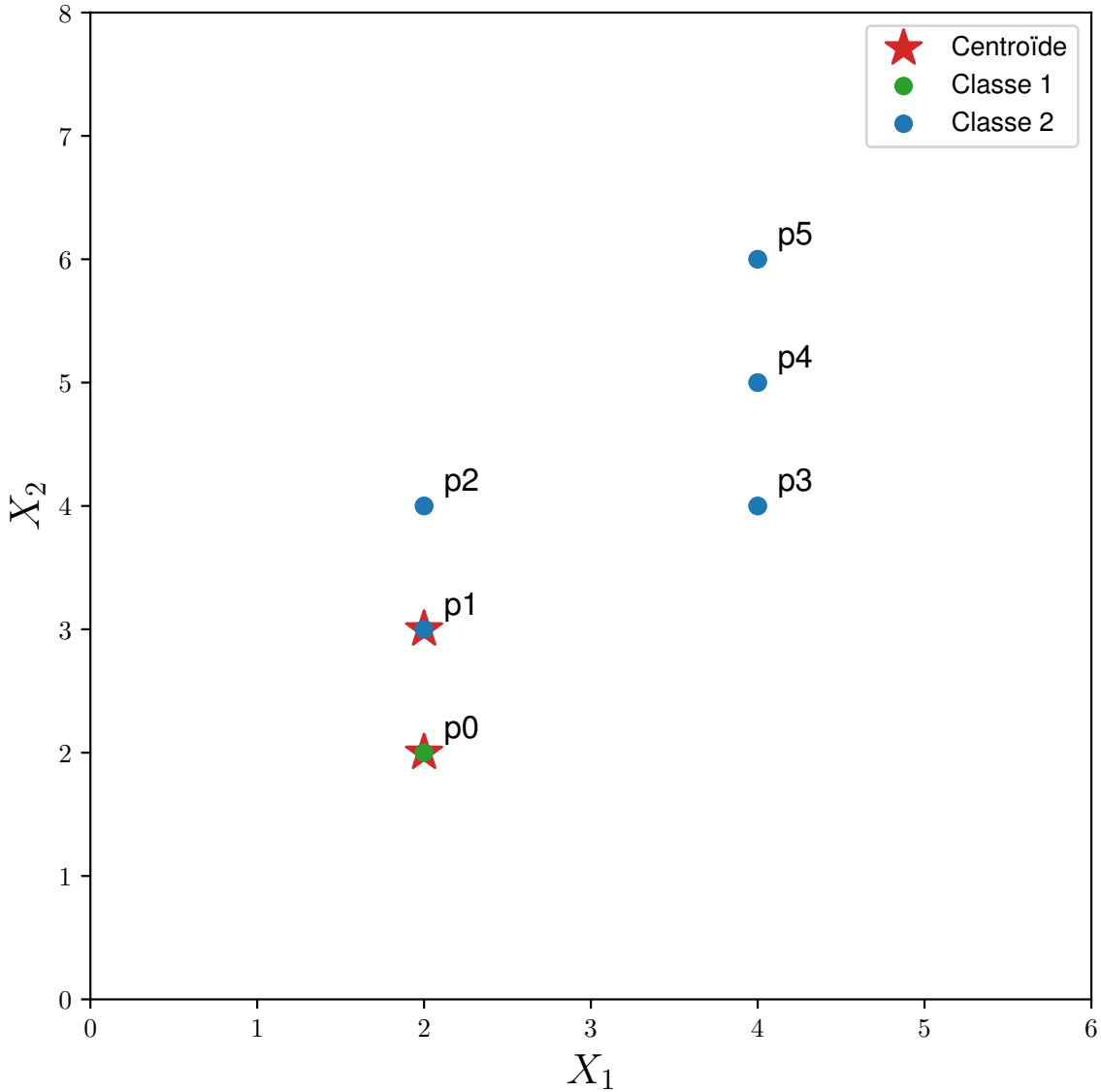
Affectation des classes

- $\{p0\} \in C_1$
- $\{p1, p2, p3, p4, p5\} \in C_2$

Estimation de l'erreur

- $T = 27$

Les K-means : un exemple



Les K-means : un exemple

Mis à jour des centroïdes

- On calcule de nouveau les centres de gravité de chaque classe.
- Puis on affecte la nouvelle classe (si modifiée).

Nouveaux centroïdes

- $\mu_1 = (2, 2)$
- $\mu_2 = (3.2, 4.4)$

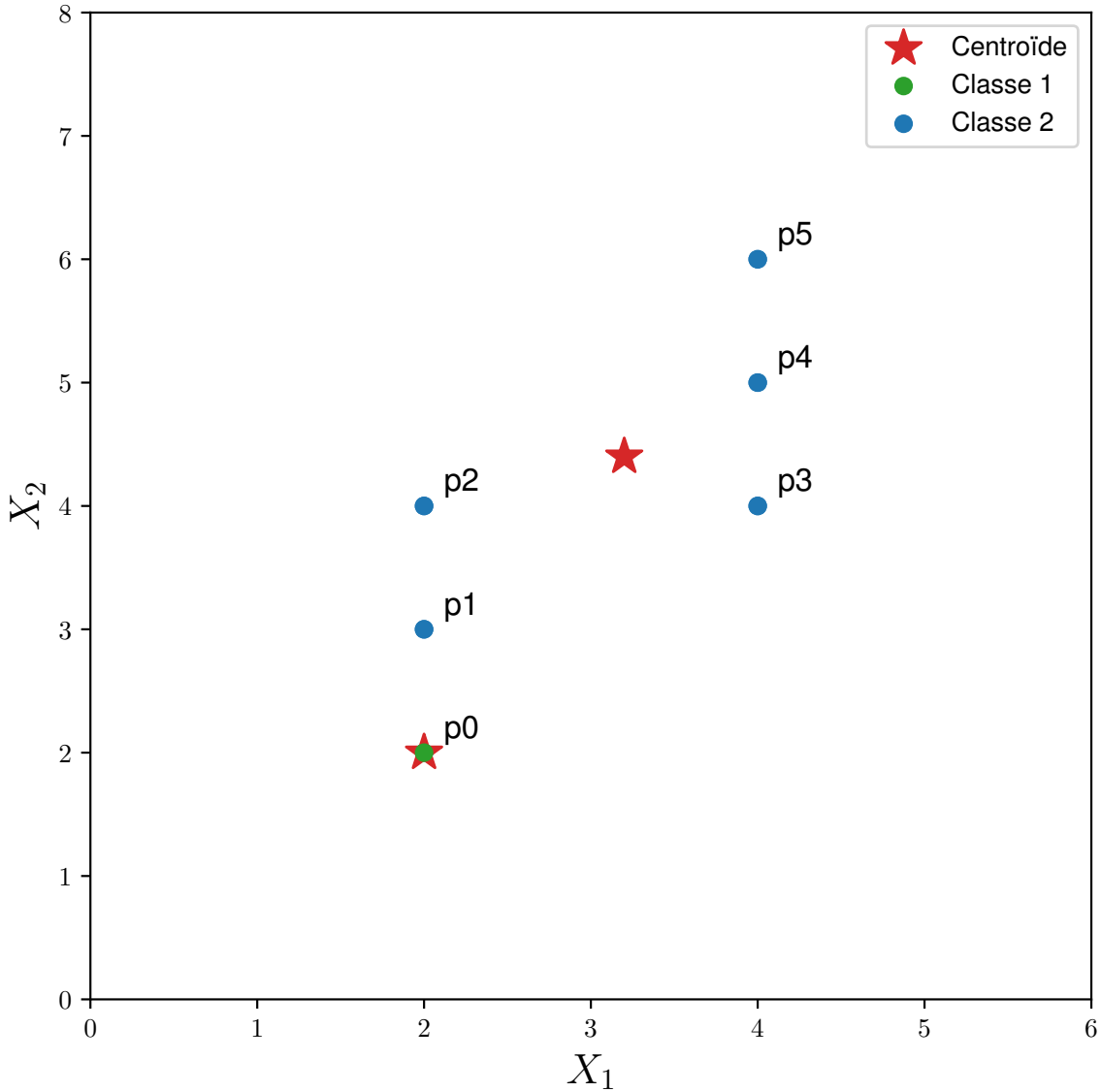
Affectation des classes

- $\{p0\} \in C_1$
- $\{p1, p2, p3, p4, p5\} \in C_2$

Estimation de l'erreur

- $T = 10$

Les K-means : un exemple



Les K-means : un exemple

Nouvelle itération :

Nouveaux centroïdes

- $\mu_1 = (2, 2.5)$
- $\mu_2 = (3.5, 4.75)$

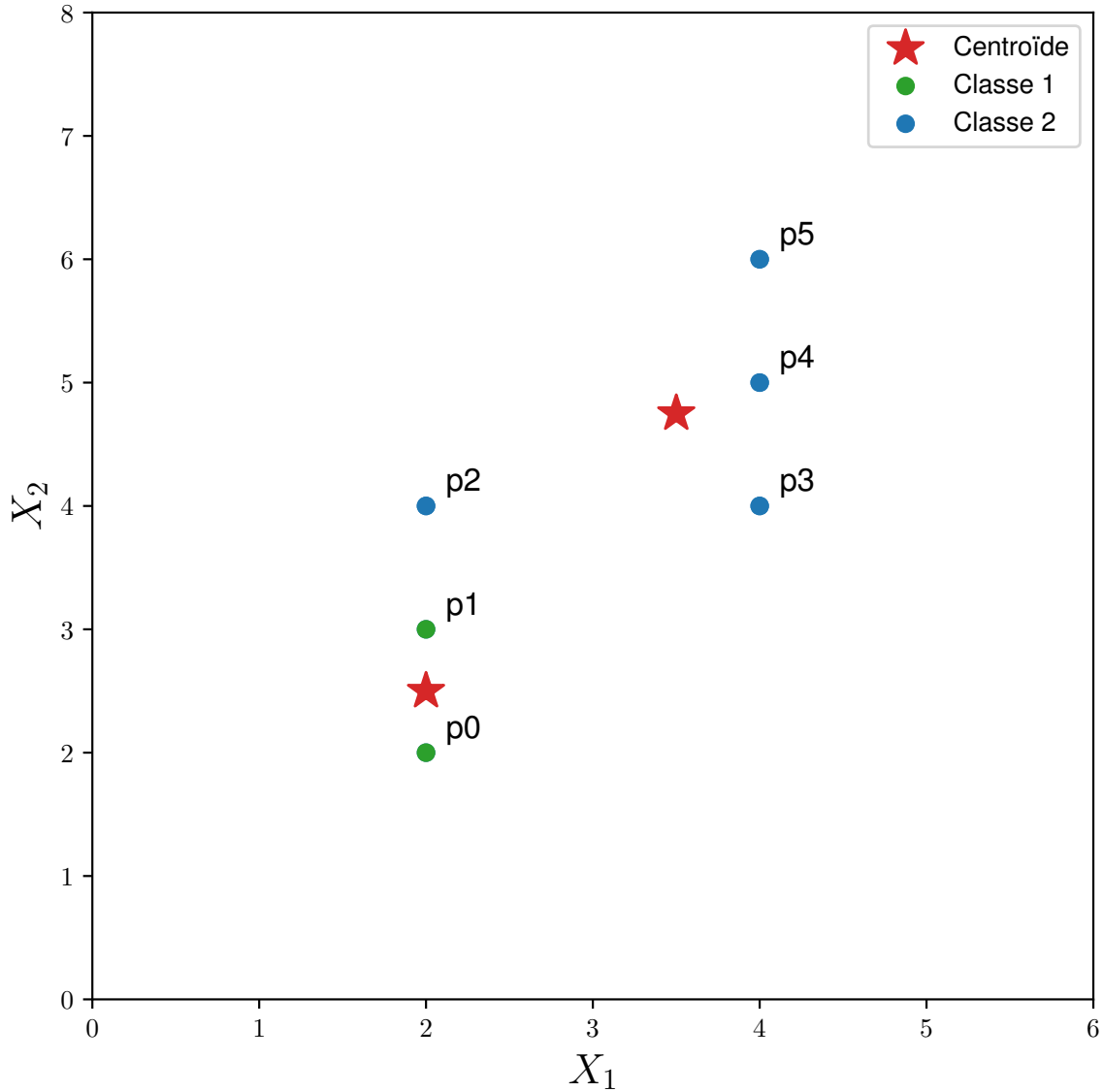
Affectation des classes

- $\{p0, p1\} \in C_1$
- $\{p2, p3, p4, p5\} \in C_2$

Estimation de l'erreur

- $T = 6.25$

Les K-means : un exemple



Les K-means : un exemple

Nouvelle itération :

Nouveaux centroïdes

- $\mu_1 = (2, 3)$
- $\mu_2 = (4, 5)$

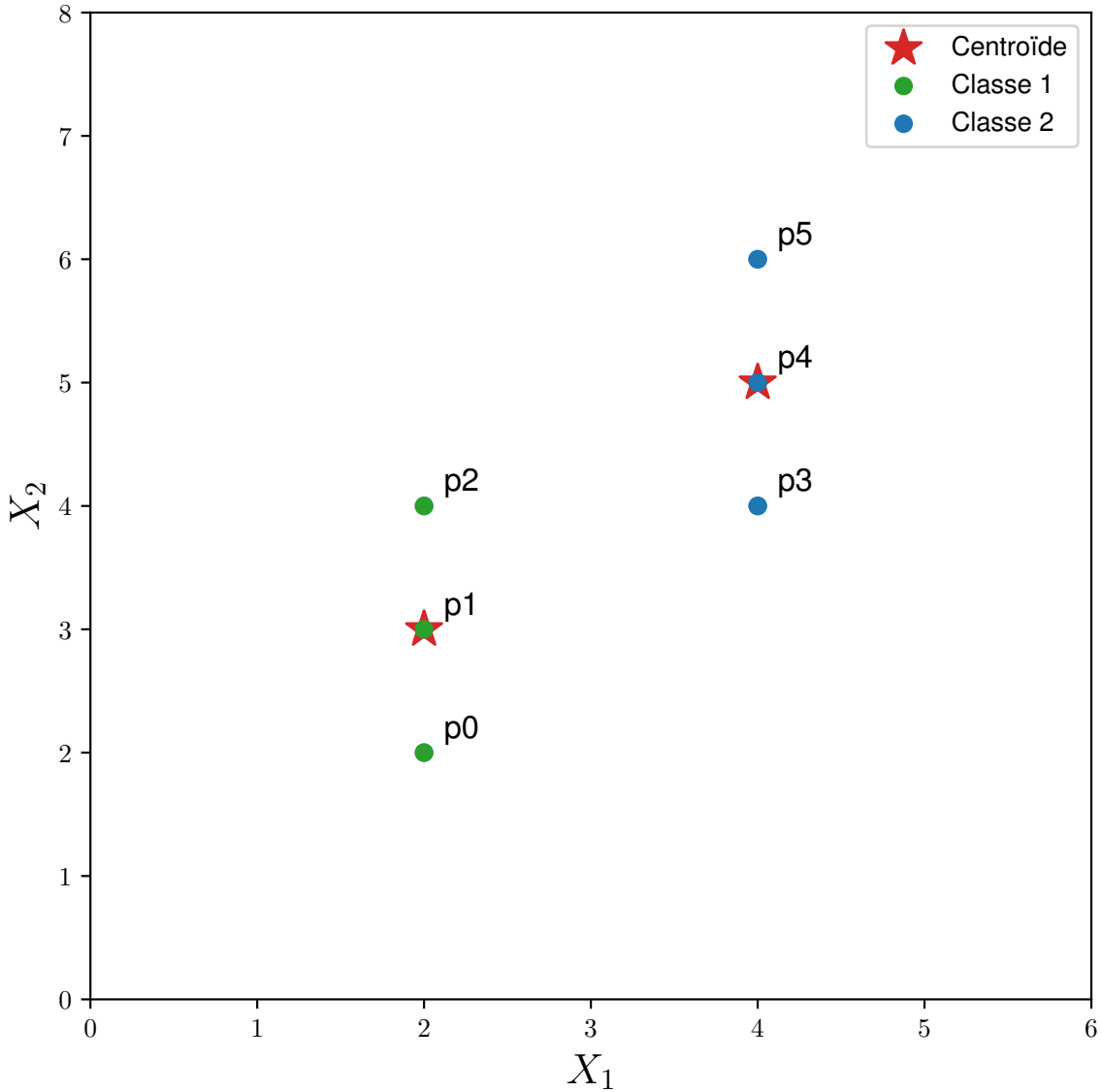
Affectation des classes

- $\{p0, p1, p2\} \in C_1$
- $\{p3, p4, p5\} \in C_2$

Estimation de l'erreur

- $T = 4$

Les K-means : un exemple



TP3 : K-means

Plan

Plan

Concepts et définitions

Principe du renforcement

- Apprentissage en ligne
- Un agent interagit avec un environnement
- Système de récompenses obligatoire
- L'agent doit apprendre à recevoir des récompenses positives
- **Applications :** jeux, finance, robotique, audiovisuel...

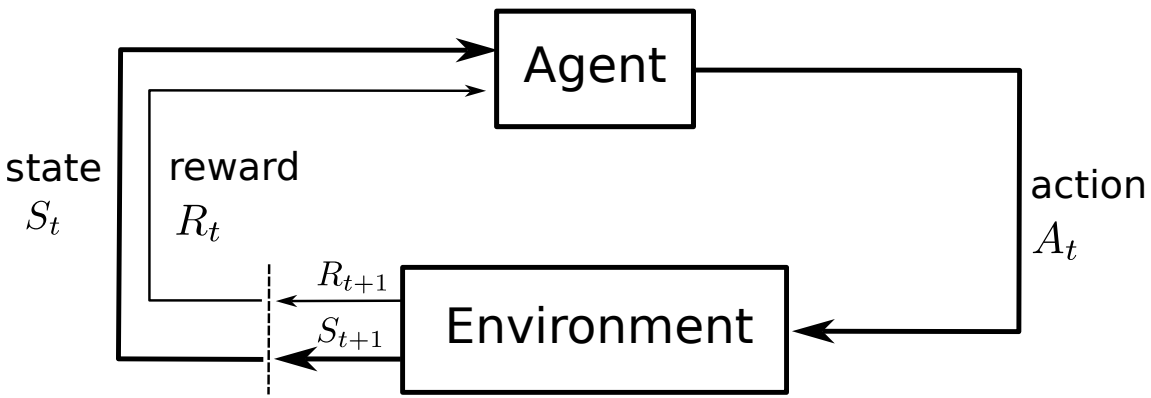
Concepts et définitions

Les notations

- $\mathcal{E}$ l'ensemble des états
- $\mathcal{A}$ l'ensemble des actions
- s un état
- a une action
- $\pi(s, a)$ politique de choix état-action (probabilité de choisir l'action a à l'état s)
- $Q(s, a)$ l'espérance de gain quand l'action a est prise dans l'état s
- r_t la récompense perçue par l'agent à l'instant t
- R_t gain cumulé par l'agent à l'instant t
- $\mathcal{E}_\pi$ l'esperance de gain en suivant la politique π

Concepts et définitions

Principe d'interaction Agent/Environnement :



Concepts et définitions

Problème à traiter

- L'agent ne connaît pas son environnement :
  - $\rightarrow$ il ne connaît pas les récompenses (signaux) à chaque état.
  - $\rightarrow$ il ne connaît pas la topologie de l'espace des états $\mathcal{A}$ (les états accessibles à partir d'un état donné).
- À l'état s_t, l'agent ne sait pas l'effet de son action, **mais peut chercher à l'estimer**.

Concepts et définitions

L'objectif de l'agent : maximiser son gain.

La notion de gain

- Gain cumulé horizon fini :

$$R_T = r_0 + r_1 + r_2 + \cdots + r_{T-1}|s_0 = \sum_{t=0}^{T-1} r_t|s_0$$

- Gain cumulé avec intérêt et horizon infini avec $0 \leq \gamma \leq 1$:

$$R_T = r_0 + \gamma r_1 + \gamma^2 r_2 \ldots |s_0 = \sum_{t=0}^{\infty} \gamma^t r_t|s_0$$

- Gain en moyenne :

$$R_T = \frac{1}{T-1} \sum_{t=0}^{T-1} r_t|s_0$$

Plan

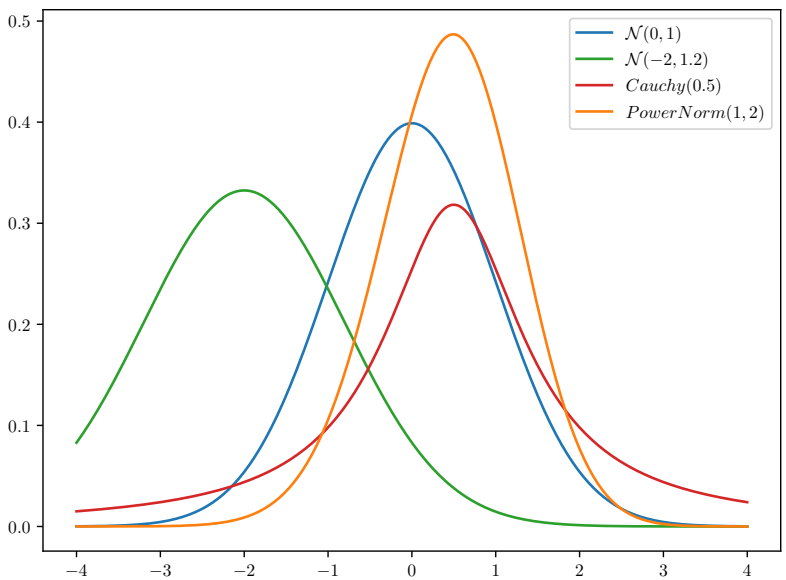
Problème du *Multi-Armed Bandit*

Problème : quelle machine choisir à l'instant t ?

Problème du *Multi-Armed Bandit*

Problème : quelle machine choisir à l'instant t ?

Distribution des gains potentiels inconnus



Exploitation vs Exploration

Si l'on essaie chacun des n bras disponibles, on peut suivre une stratégie :

- **Exploitation :** tirer le bras qui nous a ramené le meilleur gain.
- **Exploration :** tirer le prochain bras aléatoirement (essayer aléatoirement tous les bras un même nombre de fois).

Une bonne solution (voire optimale) : trouver un compromis.

Upper Confidence Bound

UCB1 [Auer 2002] est une formule permettant ce compromis

- On définit la moyenne des récompenses obtenues : $\hat{x}_i = R_t^i / n_i$ pour chaque k bras.
- n le nombre total d'actions réalisées, tel que $n = \sum_{i=1}^{k} n_i$

Pour chaque bras on calcule :

$$UCB1 = \hat{x}_i + c \times \sqrt{\frac{2log(n)}{n_i}}$$

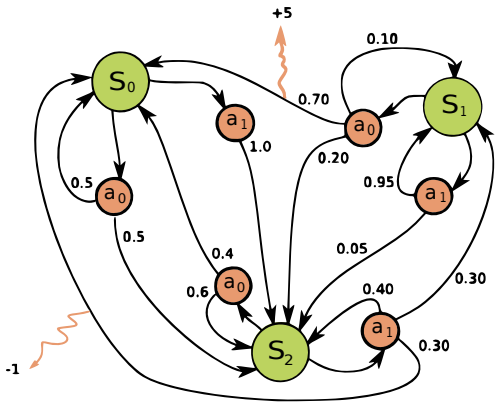
TP4 : Upper Confidence Bound (UCB)

Plan

Méthodes des différences temporelles

Contexte d'apprentissage

- Processus de décision Markovien (MDP)
  $\rightarrow$ informations cachées
- Q-function (état, action), stockée sous la forme du Q-table
- Différences temporelles pour déterminer $Q^\star(s, a)$



Plusieurs méthodes existes

- TD-λ
- SARSA (*State-Action-Reward-State-Action*)
- Expected-SARSA
- Q-learning

Méthode SARSA : orientée politique

SARSA : State-Action-Reward-State-Action

- Méthode orientée politique (par exemple ϵ-greedy)
  - 10% une action aléatoire
  - 90% la meilleure action
- La valeur de la différence de temps est calculée en utilisant la combinaison état-action actuelle et la combinaison état-action suivante.

$$Q^{\pi}(s,a) \leftarrow Q^{\pi}(s,a) + \alpha[r + \gamma Q^{\pi}(s',a') - Q^{\pi}(s,a)]$$

Pourquoi « on-policy » ?

- Chaque action est choisie à partir de la politique π
- L'état suivant est obtenue à partir de cette action

L'algorithme du Q-learning

Une méthode d'amélioration hors politique

- L'objectif est d'obtenir $Q^\star(s,a)$, l'espérance de gain avec l'action a sachant l'état s
- On choisit toujours la meilleure action de s' pour la mise à jour de la Q-value

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a' \in \mathcal{A}} Q(s',a') - Q(s,a)]$$

L'algorithme du Q-learning

Une méthode d'amélioration hors politique

- L'objectif est d'obtenir $Q^\star(s,a)$, l'espérance de gain avec l'action a sachant l'état s
- On choisit toujours la meilleure action de s' pour la mise à jour de la Q-value

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a' \in \mathcal{A}} Q(s',a') - Q(s,a)]$$

Pourquoi « off-policy » ?

- Convergence assurée quelle que soit la manière dont les actions sont sélectionnées (politique) à chaque instant
- Nécessite que chaque état soit visité infiniment souvent pour obtenir la valeur $Q^\star(s,a)$
- La convergence est plus lente que d'autres méthodes basées politiques

SARSA : algorithme

Algorithme 3 : Algorithme SARSA

Initialiser s et Q
Choisir l'action a depuis l'état s à partir de la politique π
begin
    while *Critère d'arrêt non respecté* **do**
        for *chaque étape de l'épisode* **do**
            Effectuer l'action a et observer r et s'
            Choisir l'action a' depuis l'état s' à partir de la politique π
            $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
            $s \leftarrow s'$
            $a \leftarrow a'$
        end
    end
    Retourner Q
end

Q-learning : algorithme

Algorithme 4 : Algorithme Q-learning

Initialiser s et Q
begin
  while *Critère d'arrêt non respecté* **do**
    for *chaque étape de l'épisode* **do**
      Choisir l'action a depuis l'état s à partir de la politique π
      Effectuer l'action a et observer r et s'
      $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a' \in \mathcal{A}} Q(s',a') - Q(s,a)]$
      $s \leftarrow s'$
    end
  end
  Retourner Q
end

Méthodes des différences temporelles

Paramètres d'apprentissage

- α : le taux d'apprentissage (peut être adaptatif)
- γ : le facteur d'actualisation (importance des informations futures)

Les points importants

- Bien définir son état
  - État trop complexe : convergence longue voire non possible
  - État trop simplifié : l'agent n'arrive pas à apprendre
- Avoir une récompense bien adaptée

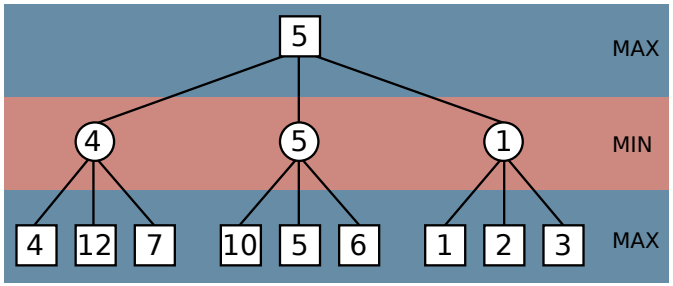
TP5 : Q-learning

Plan

Théorie du jeu : limites d'Alpha-beta prunning

Contexte : jeux 2 joueurs

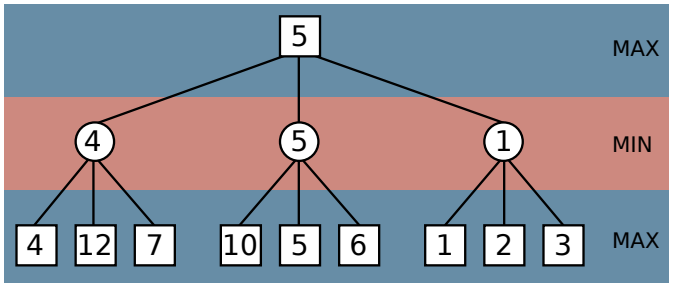
- Évaluer la meilleur prochaine action de jeu
- Avec l'utilisation de l'algorithm minimax :
  - Maximiser le gain du coup du joueur
  - Minimiser le gain du coup de l'adversaire
- $\alpha\beta$ prunning : minimax amélioré (parcours non exhaustif de l'arbre)



Théorie du jeu : limites d'Alpha-beta prunning

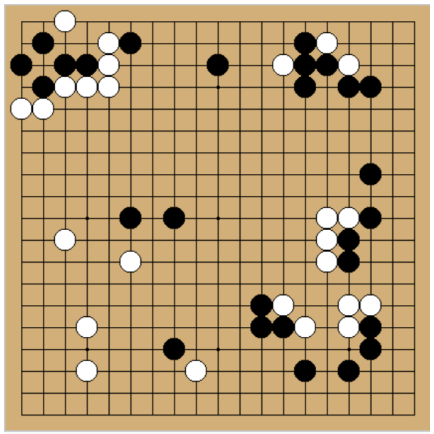
Contexte : jeux 2 joueurs

- Évaluer la meilleur prochaine action de jeu
- Avec l'utilisation de l'algorithm minimax :
  - Maximiser le gain du coup du joueur
  - Minimiser le gain du coup de l'adversaire
- $\alpha\beta$ prunning : minimax amélioré (parcours non exhaustif de l'arbre)



Limitations : jeu de Go

- Espace des actions important, profondeur / largeur de l'arbre de jeu
- Un état du jeu : 250 mouvements en moyenne
- Une partie : > 250 mouvements en moyenne
- Difficulté de posséder une bonne *fonction d'évaluation*



Qu'est-ce que MCTS ?

L'algorithme de Monte Carlo Tree Search

- Processus de décisions : état s avec un ensemble d'actions $\mathcal{A}$
- Principalement connu et utilisé pour résoudre des jeu (l'arbre de jeu)
- Simule une partie aléatoirement jusqu'à la fin (pas de fonction d'évaluation)

Qu'est-ce que MCTS ?

L'algorithme de Monte Carlo Tree Search

- Processus de décisions : état s avec un ensemble d'actions $\mathcal{A}$
- Principalement connu et utilisé pour résoudre des jeu (l'arbre de jeu)
- Simule une partie aléatoirement jusqu'à la fin (pas de fonction d'évaluation)

Objectif principal

- Évaluer la qualité de l'action a à l'état s
- Sélectionner la meilleur action à l'état s

Qu'est-ce que MCTS ?

L'algorithme de Monte Carlo Tree Search

- Processus de décisions : état s avec un ensemble d'actions $\mathcal{A}$
- Principalement connu et utilisé pour résoudre des jeu (l'arbre de jeu)
- Simule une partie aléatoirement jusqu'à la fin (pas de fonction d'évaluation)

Objectif principal

- Évaluer la qualité de l'action a à l'état s
- Sélectionner la meilleur action à l'état s

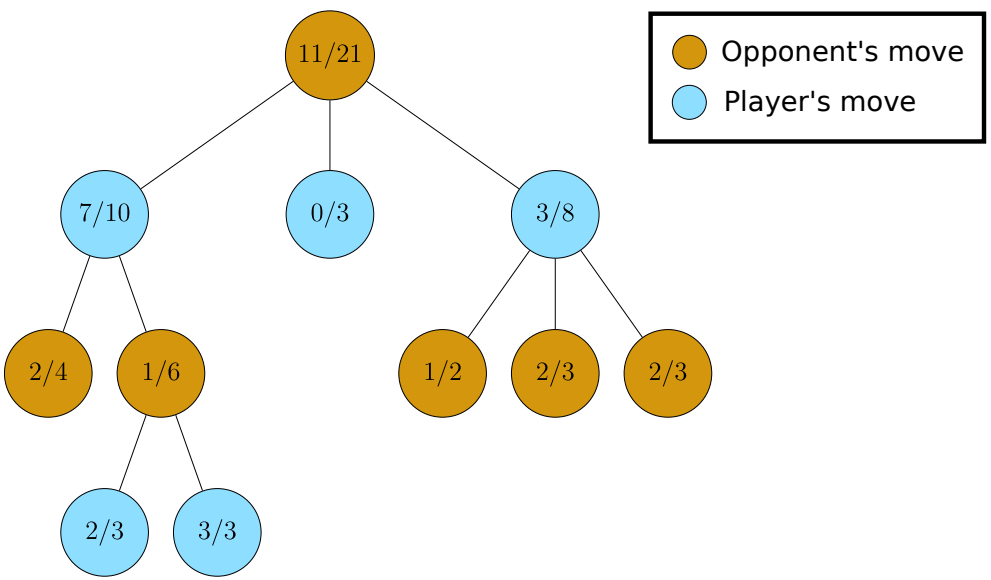
Comment ?

- Calculer la Q-value à partir des simulations à l'état s en collection des récompenses
- $Q(s,a) = \frac{1}{n_a} \sum_{i=1}^{n_a} r_i^a$
- Basé sur le principe de Monte-Carlo : grand nombre de simulations

MCTS : représentation par l'arbre

État spécifique d'un jeu

- $||\mathcal{A}|| = 3$ à l'état s_t
- Jeu à 2 joueurs
- La valeur d'une action : nombre de victoires / nombre de parties
- Un des états suivant est une défaire (état terminal)



MCTS : Concepts et algorithme

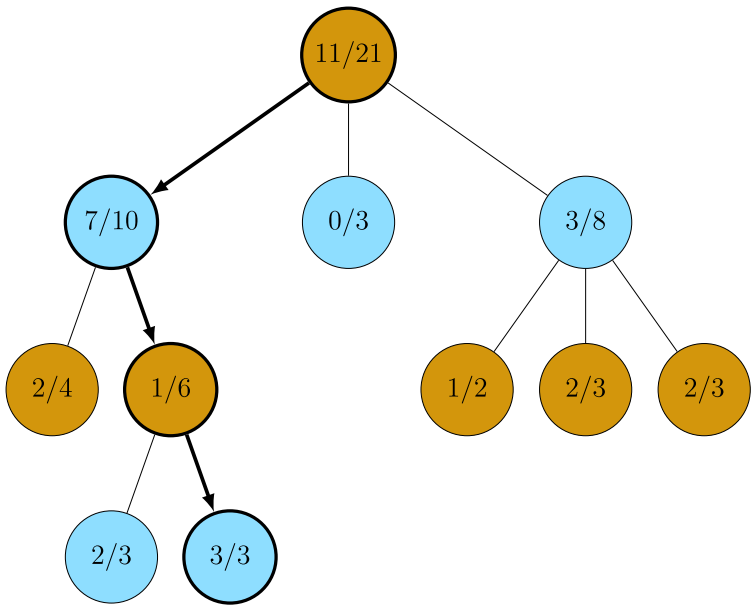
Le framework MCTS

- Construction de l'arbre à partir de simulations
- Les états qui ont été évalués sont stockés dans un arbre de recherche
- Basé sur 4 grandes étapes : `select`, `expand`, `simulate` et `backpropagate`.

MCTS : Concepts et algorithme

L'étape `select`

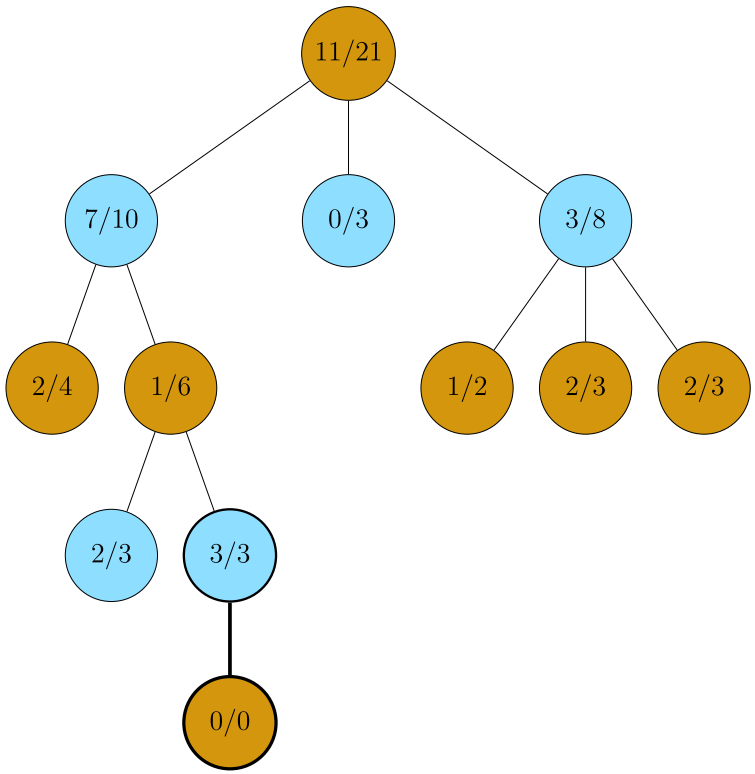
Sélectionnez un seul nœud dans l'arbre qui n'est pas entièrement développé (au moins un de ses enfants n'est pas encore exploré).



MCTS : Concepts et algorithme

L'étape `expand`

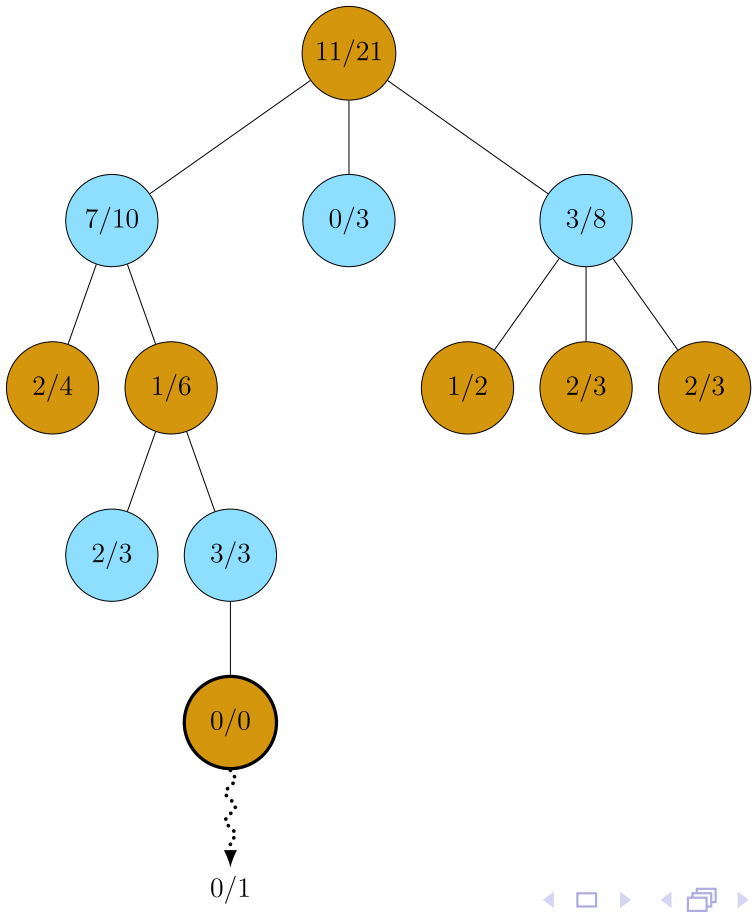
Expansion du nœud en appliquant une action disponible du nœud.



MCTS : Concepts et algorithme

L'étape `simulation`

À partir de l'un des résultats de l'expansion, effectuez une simulation aléatoire complète jusqu'à un état final afin d'obtenir le résultat.

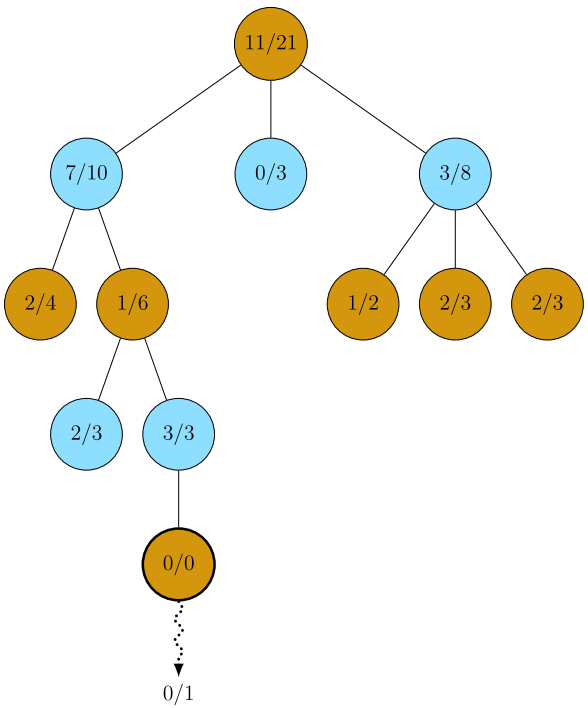


MCTS : Concepts et algorithme

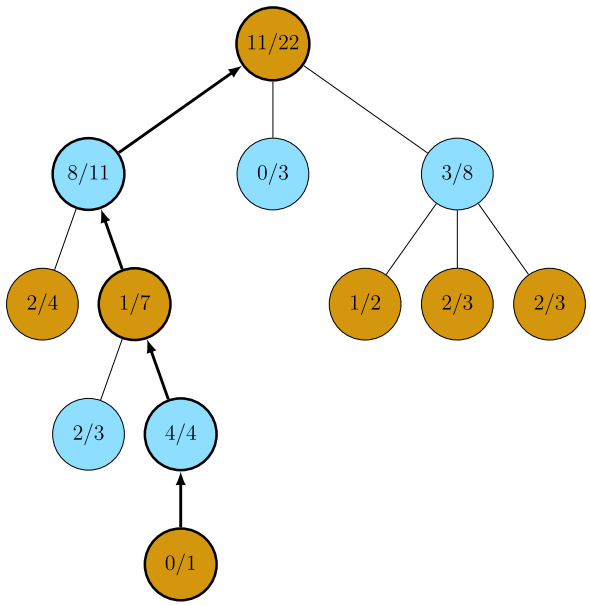
L'étape backpropagation

Enfin, la valeur du nœud est rétro-propagée jusqu'au nœud racine, en mettant à jour la valeur de chaque nœud *ancêtre* sur le chemin en utilisant la valeur attendue.

Simulation



Backpropagation



MCTS : Concepts et algorithme

Avantages

- Algorithme simple à implémenter
- Algorithme basé sur l'heuristique et pouvant fonctionner efficacement sans aucune connaissance du domaine en particulier
- Peut être sauvegardé dans n'importe quel état intermédiaire et cet état peut être utilisé dans des cas d'utilisation futurs si nécessaire
- Peut être parallélisé (Feuille / Racine / Arbre)

MCTS : Concepts et algorithme

Avantages

- Algorithme simple à implémenter
- Algorithme basé sur l'heuristique et pouvant fonctionner efficacement sans aucune connaissance du domaine en particulier
- Peut être sauvegardé dans n'importe quel état intermédiaire et cet état peut être utilisé dans des cas d'utilisation futurs si nécessaire
- Peut être parallélisé (Feuille / Racine / Arbre)

Inconvénients

- Peut nécessiter une quantité énorme de mémoire (en raison de la croissance rapide de l'arbre)
- Peut avoir un problème de fiabilité dans le cas d'un grand nombre de combinaisons (chacun des nœuds peut ne pas être visité suffisamment de fois)
- Nécessite un grand nombre d'itérations pour pouvoir déterminer efficacement le chemin le plus efficace

MCTS : politique de choix

MCTS construction et utilisation

- (Hors ligne et/ou en ligne) Construire l'arbre de recherche et obtenir une approximation des résultats
- (En ligne) Ensuite, utilisez la recherche par arbre construit et toujours sélectionner $\arg\max Q(s, a)$ à l'état s

MCTS : politique de choix

MCTS construction et utilisation

- (Hors ligne et/ou en ligne) Construire l'arbre de recherche et obtenir une approximation des résultats
- (En ligne) Ensuite, utilisez la recherche par arbre construit et toujours sélectionner $\arg\max Q(s,a)$ à l'état s

Monte Carlo Tree Search Pur

- Peut être appliqué à tout jeu dont les positions ont nécessairement un nombre fini de coups et une longueur finie
- Tous les coups réalisables sont déterminés : k parties aléatoires sont jouées jusqu'à la fin, et les scores sont enregistrés
- Le coup conduisant au meilleur score est choisi par :

$$\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$$

MCTS : politique de choix

Compromis exploration vs exploitation

- Explorer k fois chaque action implique une perte d'efficacité
- Mieux vaut explorer les bons coups en profondeur dans l'arbre
- L'exploitation seule peut fournir un minimum local

Utiliser l'algorithme UCB (Kocsis et Szepesvári 2006)

A chaque niveau, la sélection des noeuds se fait par noeud :

$$\pi(s) = \arg\max_{a \in \mathcal{A}} \left[Q(s,a) + c \sqrt{\frac{2 \times log(n)}{n_a}} \right]$$

avec un compromis paramétré par c (théoriquement égal à $\sqrt{2}$) :

- Maintenir un certain équilibre entre l'exploitation des variantes profondes après les coups ayant un taux de victoire moyen élevé.
- L'exploration de mouvements avec peu de simulations

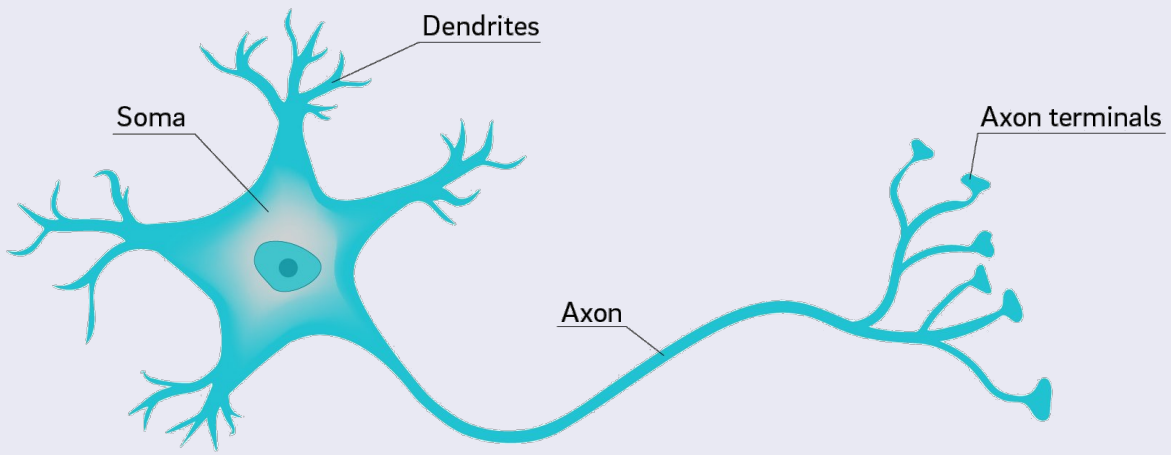
$\Rightarrow$ **Finalement :** Upper Confidence Tree $= MCTS + UCB$

Plan

Plan

Réseaux de neurones : principe

Processus biologique : circuit neuronal [1]



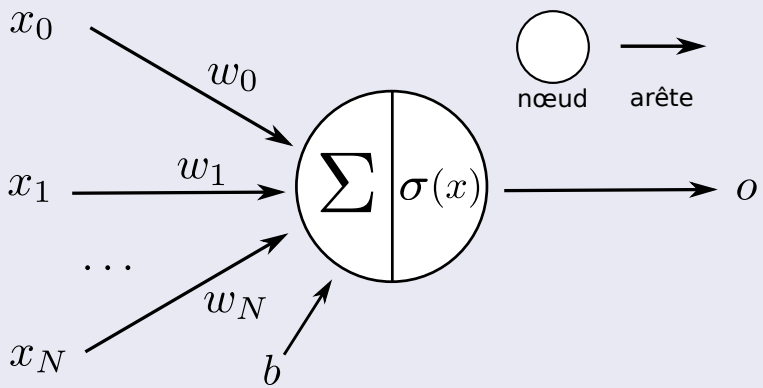
- Un neurone permet la transition d'informations
- Axon terminals $\rightarrow$ synapses
- Composé d'une entrée et d'une sortie
- Inspire le mouvement de pensée du connexionnisme

1. Source : https://www.upgrad.com

Réseaux de neurones : perceptron

Perceptron : un neurone articiel

- Rosenblatt en 1958
- Somme pondérée des entrées
- Fonction d'activation $f(x)$ (synapse)
- Ajout d'un biais [1] $(y = ax + b)$

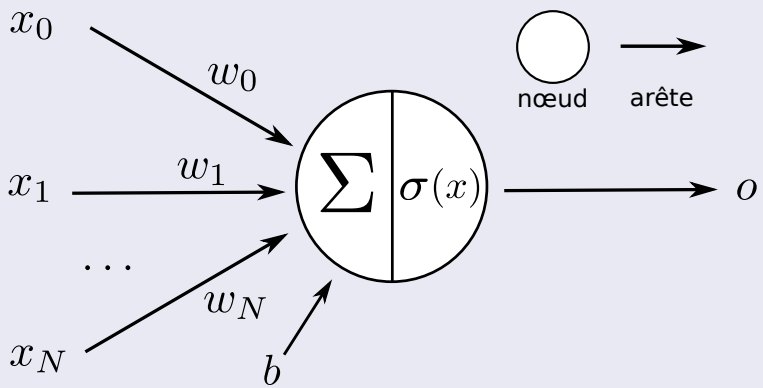


1. Ce paramètre est également appelé seuil

Réseaux de neurones : perceptron

Perceptron : un neurone articiel

- Rosenblatt en 1958
- Somme pondérée des entrées
- Fonction d'activation $f(x)$ (synapse)
- Ajout d'un biais [1] $(y = ax + b)$



Fonction d'activation

$$o = \sigma(x) = \begin{cases} 1 & \text{si} \quad \sum_{i=1}^{n} w_i x_i + b > \theta \\ 0 & \text{sinon} \end{cases}$$

1. Ce paramètre est également appelé seuil

Réseaux de neurone : perceptron

Backprogation : mise à jour de l'erreur (cas simple)

Pour chaque label y, on compare la sortie $f(x)$, pour propager l'erreur (si existante) :

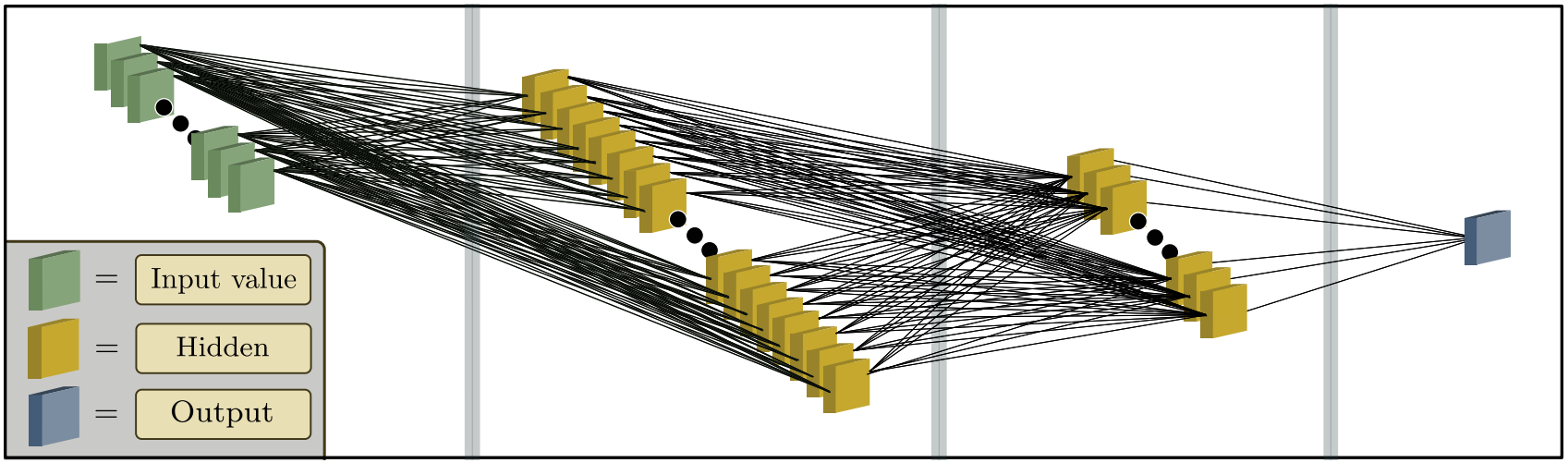
$$w_i = w_i + \alpha(y - f(x))x_i$$

- α le taux d'apprentissage

Objectif

- Mettre à jour tous les poids du neurone
- Afin qu'il réponde au mieux aux sorties attendues

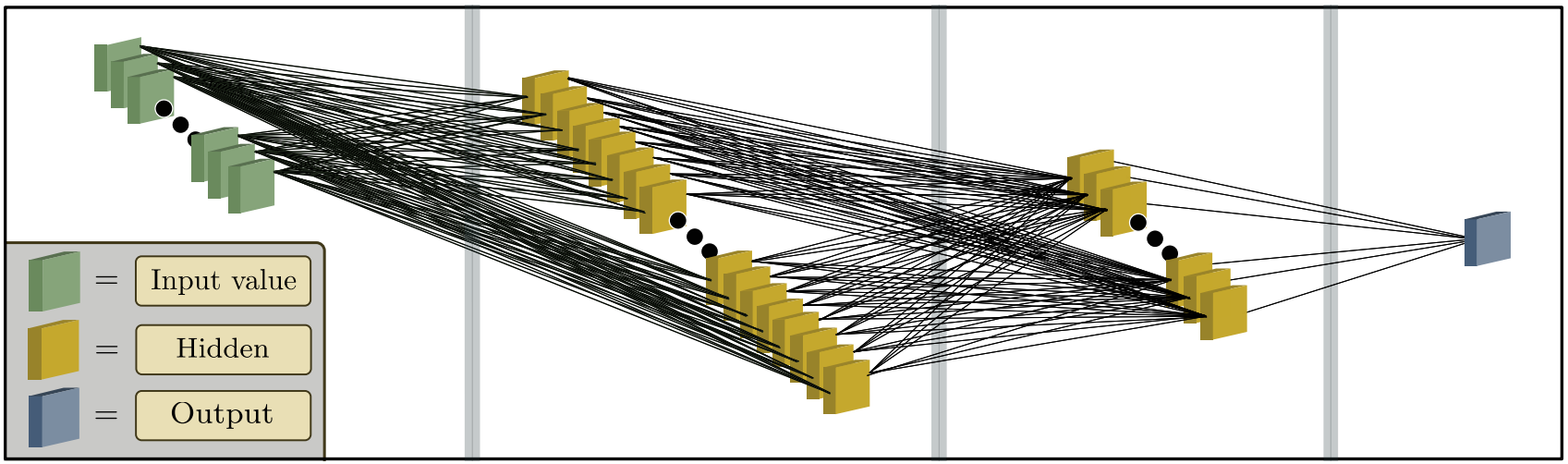
Perceptron Multi-couches : réseaux de neurones



Réseau de neurones profond

- Plusieurs couches de neurones interconnectés
- Chaque neurone d'une couche prend en entrée une somme pondérée des sorties neurones précédents
- Une couche d'entrée / une couche de sortie (dépend de la tâche souhaitée)
- Autant de couches cachées que souhaité
- Pour chaque couche un **biais** est toujours ajouté

Perceptron Multi-couches : réseaux de neurones



Réseau de neurones profond : paramètres

- Le nombre de poids à apprendre est de $n \times h + h$ avec :
  - n le nombre de données d'entrée
  - h le nombre de neurones dans la couche suivante
  - On ajoute h poids pour le biais pour cette couche
- Plus généralement on a pour l couches : $\sum_{i=0}^{l-1} h_i \times h_{i+1} + h_{i+1}$

Réseau de neurones profond : fonctions d'activations

Une fonction d'activation

- Fonction qui décide si un neurone doit être « activé » ou non
- À partir de la somme pondérée de ses entrées et du biais
- Objectif d'introduire une non-linéarité dans la sortie d'un neurone

Réseau de neurones profond : fonctions d'activations

Une fonction d'activation

- Fonction qui décide si un neurone doit être « activé » ou non
- À partir de la somme pondérée de ses entrées et du biais
- Objectif d'introduire une non-linéarité dans la sortie d'un neurone

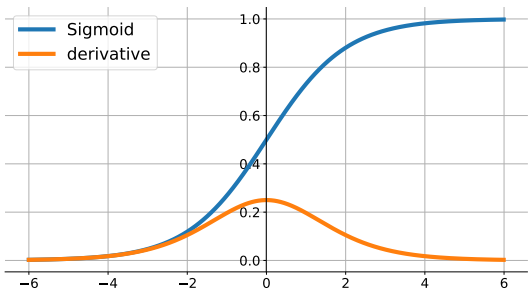
Pourquoi a-t-on besoin de fonctions d'activations ?

- Tout comme pour le perceptron, l'erreur obtenue sera propagée pour l'apprentissage. Toutefois, cela sera fait de couche en couche, on parle alors de **rétropropagation** de l'erreur (*backprogation*).
- Les fonctions d'activation rendent la rétropropagation possible puisque les **gradients** sont fournis avec l'erreur pour mettre à jour les poids et les biais.
- Comment ? Les fonctions sont **dérivables**

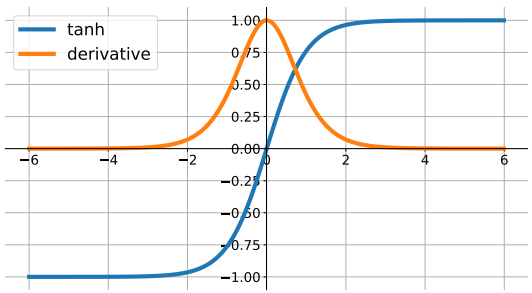
Réseau de neurones profond : fonctions d'activation

Quelques fonctions d'activation

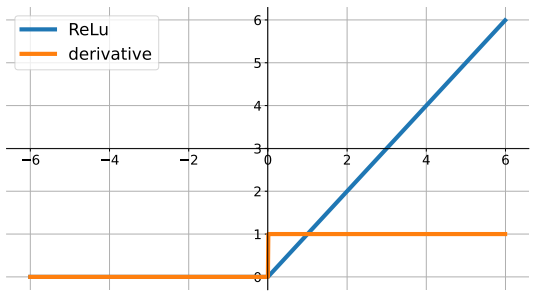
- **Sigmoïde :** utilisée pour retourner une probabilité (bornée $\in [0, 1]$)
- **Softmax :** en sortie d'une couche retourne un vecteur de probabilité dont la somme vaut 1
- **ReLu :** pour *rectified linear unit*, elle est généralement utilisée pour les couches cachées
- Et bien d'autres...



Sigmoïde



Tangente hyperbolique



ReLu

Réseau de neurones profond : propagation de l'erreur

Vers la notion de gradient

- Le gradient est calculé à partir des dérivées des fonctions d'activation
- Il permet de minimiser l'erreur du réseau et de mettre à jour les poids en conséquence par la **backprogation**
- On parle alors de descente de gradient

Fonction de perte : `Loss function`

- Fonction dérivable spécifique à la tâche du réseau de neurones
- Permet de calculer une erreur (résidu) pour chaque prédiction donnée
- L'erreur sera propagée de couche en couche pour mise à jour des poids

Réseaux de neurones profond : notion de gradient

Gradient du risque empirique

Pour un modèle de régression prenant en entrée $x_i = (x_i^1, x_i^2, \ldots, x_i^p)$:

$$h_\Theta(x_i) = w_0 + \sum_{j=1}^{p} w_j x_i^j$$

On cherche à optimiser les paramètres, tel que :

$$\hat{\Theta} = \underset{\Theta=\{w_0,\ldots,w_p\}}{\arg min} \frac{1}{n} \sum_{i=1}^{n} loss(h_\Theta(x_i), y_i)$$

- Θ les paramètres du modèle h
- $loss$ l'erreur résiduelle, par exemple une erreur quadratique
- n le nombre d'échantillons

Réseaux de neurones profond : notion de gradient

Gradient du risque empirique

Risque empirique avec une erreur quadratique (coût) :

$$C_\Theta(x, y) = \frac{1}{n} \sum_{i=1}^{n} (h_\Theta(x_i) - y_i)^2$$

Calcul du gradient :

$$\nabla C_\Theta(x, y) = \left(\frac{\partial C_\Theta(\dots)}{\partial w_0}, \frac{\partial C_\Theta(\dots)}{\partial w_1}, \dots, \frac{\partial C_\Theta(\dots)}{\partial w_p} \right)^T$$

$$= \frac{1}{n} \sum_{i=1}^{n} 2\big(h_\Theta(x_i) - y_i\big)\big(1, x_i^0, x_i^1, \dots, x_i^p\big)^T$$

Mise à jour des paramètres :

$$\Theta_t = \Theta_{t-1} - \lambda \nabla C_\Theta(x, y)$$

- λ le pas d'apprentissage

Réseaux de neurones profond : calcul du gradient

Comment calculer le gradient au sein d'un réseau de neurones ?

Calculer l'impact d'un poids sur un neurone de sortie :

$$\frac{\partial C}{\partial w_{jk}^L} = \frac{\partial z_j^L}{\partial w_{jk}^L} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial C}{\partial a_j^L}$$

- C le coût d'erreur de la donnée $h(x_i)$ par rapport à y_i
- L le nombre de couches
- w_{jk}^L un poids de connexion entre le neurone j la couche L et le neurone k de la couche $L-1$
- $z_j^L = \cdots + w_{jk}^L a_k^L + \ldots$
- $a_j^L = \sigma(z_j^L)$, avec σ une fonction d'activation dérivable

Utilisation de la `chain rule` (théorème des fonctions composées)

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z} \frac{\partial z}{\partial x} \Rightarrow \frac{\partial f(g(x))}{\partial x} = g'(x) f'(g(x))$$

Réseaux de neurones profond : calcul du gradient

Comment calculer le gradient au sein d'un réseau de neurones ?

Calculer l'impact d'un poids sur un neurone de sortie (après simplification) :

$$\frac{\partial C}{\partial w_{jk}^L} = \frac{\partial z_j^L}{\partial w_{jk}^L} \frac{\partial a_j^L}{\partial z_{jk}^L} \frac{\partial C}{\partial a_j^L}$$

- $\frac{\partial C}{\partial a_j^L} = 2(a_j^L - y_i)$ (car couche de sortie)
- $\frac{\partial a_j^L}{\partial z_{jk}^L} = \sigma'(z_k^L)$
- $\frac{\partial z_j^L}{\partial w_{jk}^L} = a_k^{L-1}$

Réseaux de neurones profond : calcul du gradient

Comment calculer le gradient au sein d'un réseau de neurones ?

Calculer l'impact d'un poids sur un neurone de sortie (après simplification) :

$$\frac{\partial C}{\partial w_{jk}^L} = \frac{\partial z_j^L}{\partial w_{jk}^L} \frac{\partial a_j^L}{\partial z_{jk}^L} \frac{\partial C}{\partial a_j^L}$$

- $\frac{\partial C}{\partial a_j^L} = 2(a_j^L - y_i)$ (car couche de sortie)
- $\frac{\partial a_j^L}{\partial z_{jk}^L} = \sigma'(z_k^L)$
- $\frac{\partial z_j^L}{\partial w_{jk}^L} = a_k^{L-1}$

Comment faire pour un poids non associé à une couche de sortie ?

- Cas actuellement traité : perceptron (couche d'entrée + couche de sortie)
- Pour un réseau plus dense : nécessité de calculer différemment le terme $\frac{\partial C}{\partial a_j^l}$

Réseaux de neurones profond : calcul du gradient

Comment calculer le gradient au sein d'un réseau de neurones ?

Calculer l'impact de la sortie d'un neurone après activation :

$$\frac{\partial C}{\partial a_k^{L-1}} = \sum_{j=0}^{n_L-1} \frac{\partial z_j^L}{\partial a_k^{L-1}} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial C}{\partial a_j^L}$$

- C le coût d'erreur de la donnée $h(x_i)$ par rapport à y_i
- L le nombre de couches
- a_k^{L-1} activation de la couche précédente
- a_j^{L-1} activation de la couche suivante
- $z_j^L = \cdots + w_{jk}^L a_k^L + \ldots$

Réseaux de neurones profond : calcul du gradient

Comment calculer le gradient au sein d'un réseau de neurones ?

Calculer l'impact de la sortie d'un neurone après activation (simplification) :

$$\frac{\partial C}{\partial a_k^{L-1}} = \sum_{j=0}^{n_L-1} \frac{\partial z_j^L}{\partial a_k^{L-1}} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial C}{\partial a_j^L}$$

- $\frac{\partial z_j^L}{\partial a_k^{L-1}} = w_{jk}^{(l+1)}$
- $\frac{\partial a_j^L}{\partial z_j^L} = \sigma'(z_k^{(l+1)})$

Réseaux de neurones profond : backprogation du gradient

Comment propager le gradient au sein d'un réseau de neurones ?

Répéter ce processus pour l'ensemble des couches, de la sortie à celle d'entrée :

$$\nabla C \Leftarrow \begin{cases} \dfrac{\partial C}{\partial w_{jk}^{l}} = a_k^{l-1} \sigma'(z_j^L) \dfrac{\partial C}{\partial a_j^L} \end{cases}$$

avec :

- Si couche de sortie : $\frac{\partial C}{\partial a_j^L} = 2(a_j^L - y_i)$ (si erreur quadratique)
- Sinon : $\frac{\partial C}{\partial a_j^L} = \sum_{j=0}^{n_{l+1}-1} w_{jk}^{(l+1)} \sigma'(z_j^{(l+1)}) \frac{\partial C}{\partial a_j^{l+1}}$

Réseaux de neurones profond : backprogation du gradient

Comment propager le gradient au sein d'un réseau de neurones ?

Répéter ce processus pour l'ensemble des couches, de la sortie à celle d'entrée :

$$\nabla C \Leftarrow \begin{cases} \dfrac{\partial C}{\partial w_{jk}^{l}} = a_k^{l-1} \sigma'(z_j^L) \dfrac{\partial C}{\partial a_j^L} \end{cases}$$

avec :

- Si couche de sortie : $\frac{\partial C}{\partial a_j^L} = 2(a_j^L - y_i)$ (si erreur quadratique)
- Sinon : $\frac{\partial C}{\partial a_j^L} = \sum_{j=0}^{n_{l+1}-1} w_{jk}^{(l+1)} \sigma'(z_j^{(l+1)}) \frac{\partial C}{\partial a_j^{l+1}}$

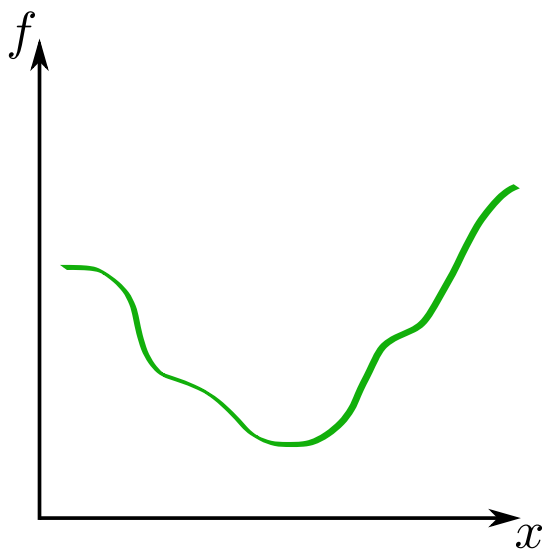
La backprogation : avantages / inconvénients

- Calcul rapide pour chaque échantillon
- Le gradient final se base sur la moyenne des gradients des échantillons
- Processus parallélisable sur GPU/TPU
- **Attention :** nécessite toutefois du stockage d'informations intermédiaires

Réseau de neurones profond : descente de gradient

Objectif de la descente de gradient

- Itérativement améliorer une solution x trouvée dans un espace de solutions
- Objectif : trouver la meilleure solution
- Utilisation du gradient pour minimiser l'erreur (`Loss` : f)

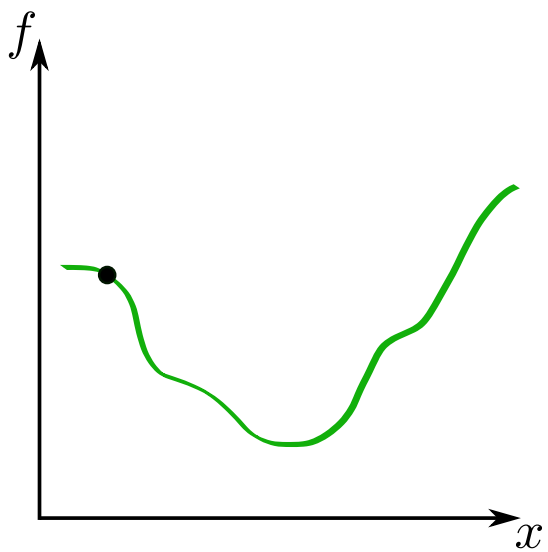


Espace de solutions $\mathcal{X}$

Réseau de neurones profond : descente de gradient

Objectif de la descente de gradient

- Itérativement améliorer une solution x trouvée dans un espace de solutions
- Objectif : trouver la meilleure solution
- Utilisation du gradient pour minimiser l'erreur (`Loss` : f)

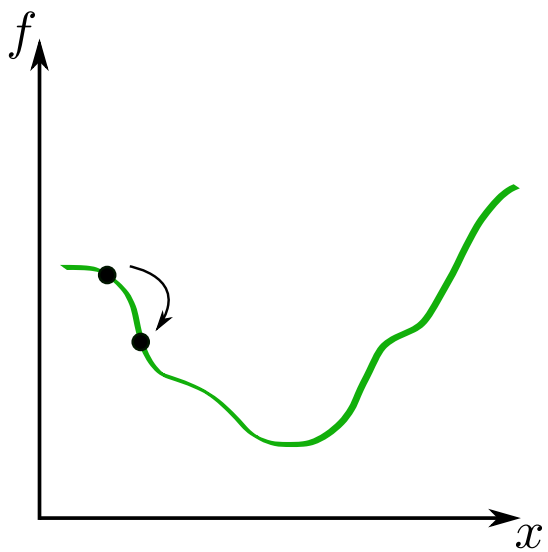


Itération 1

Réseau de neurones profond : descente de gradient

Objectif de la descente de gradient

- Itérativement améliorer une solution x trouvée dans un espace de solutions
- Objectif : trouver la meilleure solution
- Utilisation du gradient pour minimiser l'erreur (`Loss` : f)

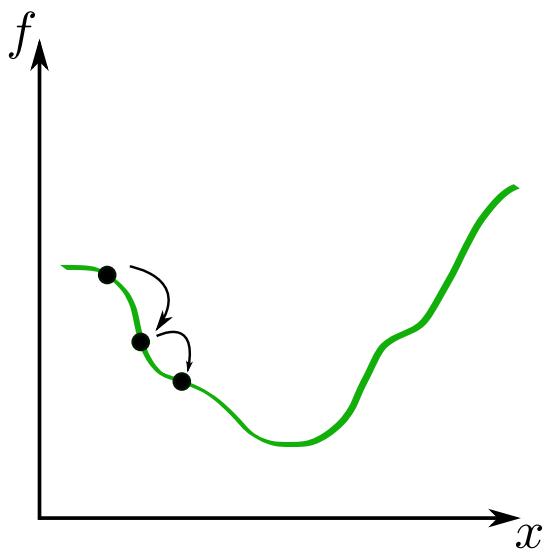


Itération 2

Réseau de neurones profond : descente de gradient

Objectif de la descente de gradient

- Itérativement améliorer une solution x trouvée dans un espace de solutions
- Objectif : trouver la meilleure solution
- Utilisation du gradient pour minimiser l'erreur (`Loss` : f)

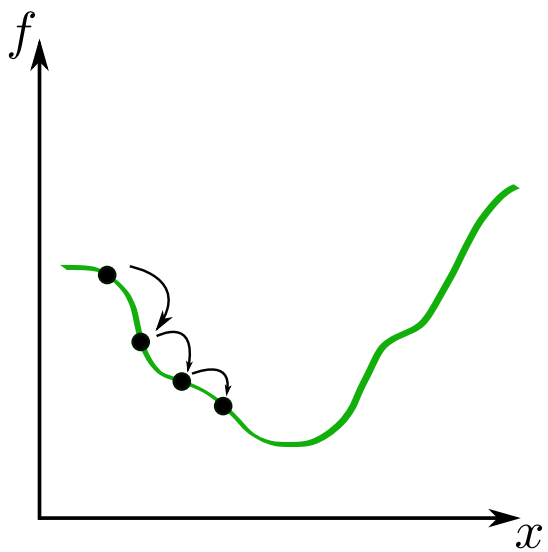


Itération 3

Réseau de neurones profond : descente de gradient

Objectif de la descente de gradient

- Itérativement améliorer une solution x trouvée dans un espace de solutions
- Objectif : trouver la meilleure solution
- Utilisation du gradient pour minimiser l'erreur ($\texttt{Loss} : f$)

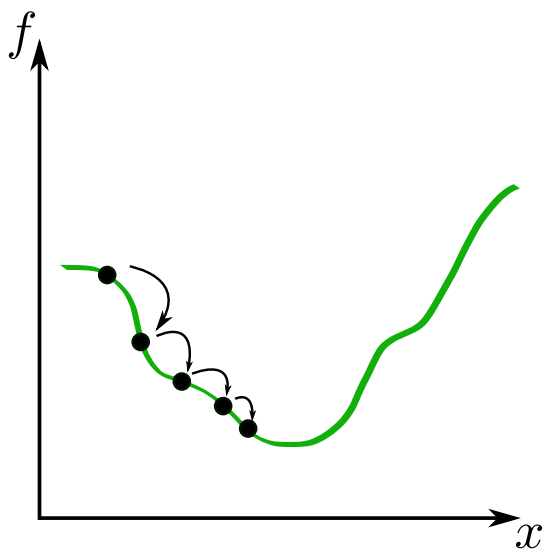


Itération 4

Réseau de neurones profond : descente de gradient

Objectif de la descente de gradient

- Itérativement améliorer une solution x trouvée dans un espace de solutions
- Objectif : trouver la meilleure solution
- Utilisation du gradient pour minimiser l'erreur (`Loss` : f)

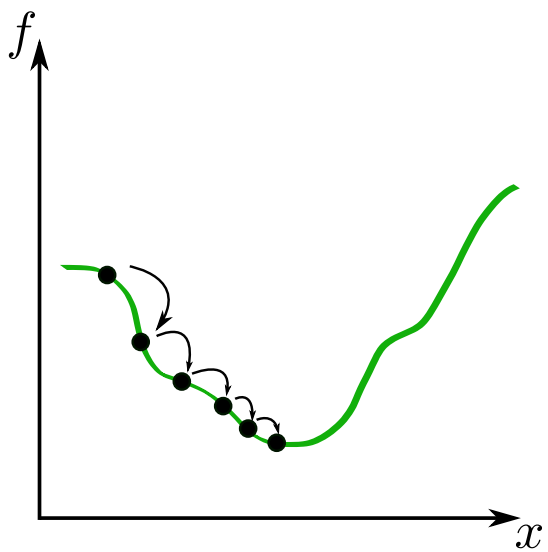


Itération 5

Réseau de neurones profond : descente de gradient

Objectif de la descente de gradient

- Itérativement améliorer une solution x trouvée dans un espace de solutions
- Objectif : trouver la meilleure solution
- Utilisation du gradient pour minimiser l'erreur (`Loss` : f)

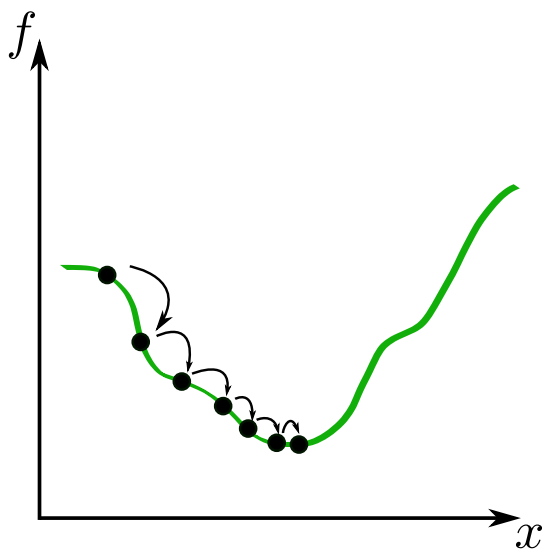


Itération 6

Réseau de neurones profond : descente de gradient

Objectif de la descente de gradient

- Itérativement améliorer une solution x trouvée dans un espace de solutions
- Objectif : trouver la meilleure solution
- Utilisation du gradient pour minimiser l'erreur ($\texttt{Loss} : f$)

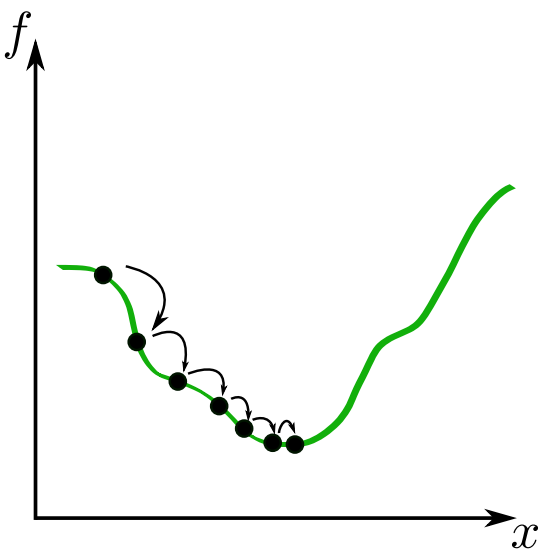


Itération 7

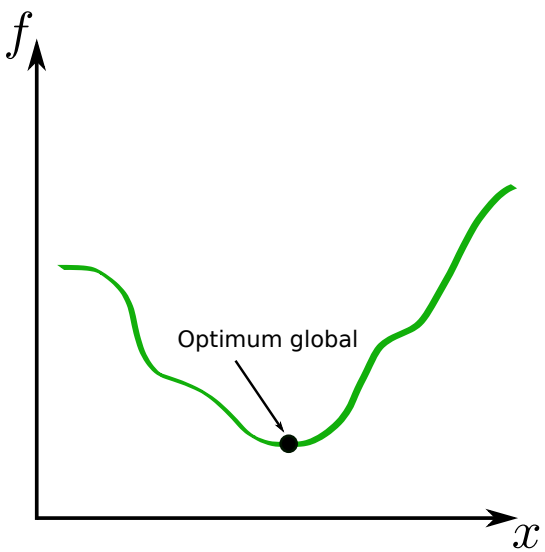
Réseau de neurones profond : descente de gradient

Objectif de la descente de gradient

- Itérativement améliorer une solution x trouvée dans un espace de solutions
- Objectif : trouver la meilleure solution
- Utilisation du gradient pour minimiser l'erreur (`Loss` : f)



Itération 7

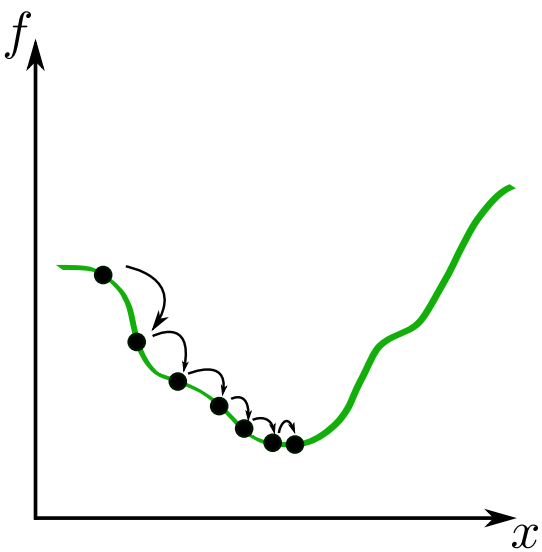


Cas convexe

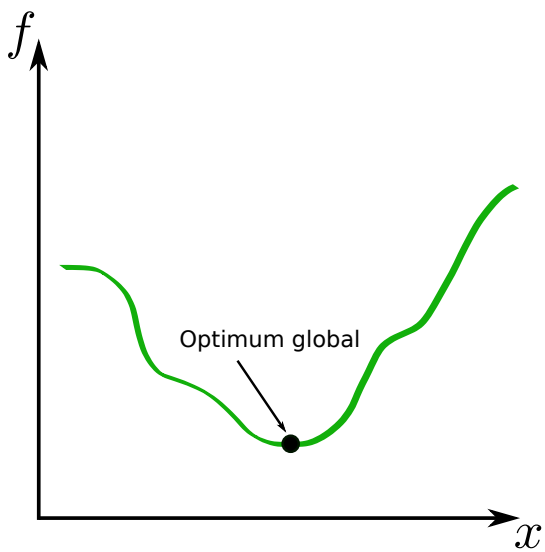
Réseau de neurones profond : descente de gradient

Objectif de la descente de gradient

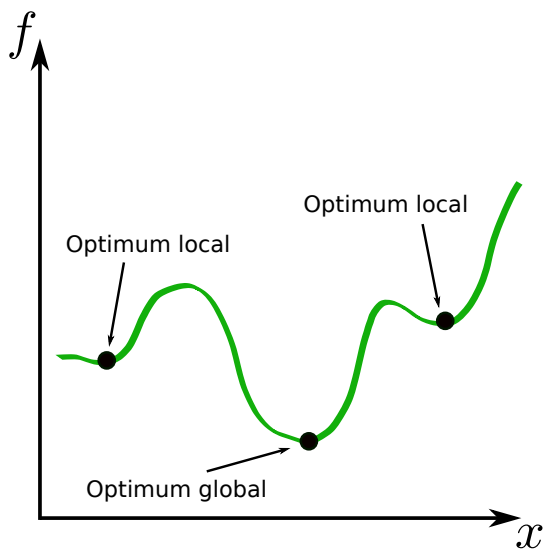
- Itérativement améliorer une solution x trouvée dans un espace de solutions
- Objectif : trouver la meilleure solution
- Utilisation du gradient pour minimiser l'erreur ($\texttt{Loss} : f$)



Itération 7



Cas convexe



Cas non convexe

Réseau de neurones profond : les batchs

La notion de batch

- On parle aussi le **lot** de données
- Un sous-ensemble (généralement petit) de la base d'apprentissage

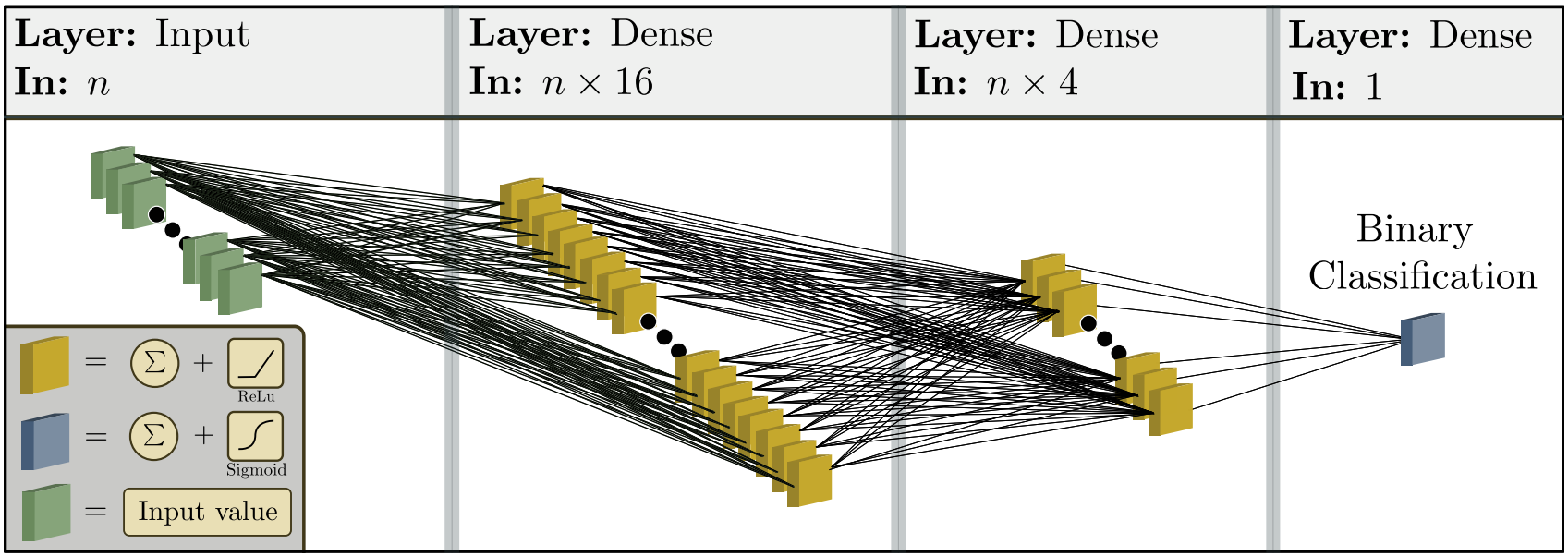
Les avantages

- Utilisé comme itération dans la descente de gradient
- Permet de fournir un ensemble réduit au modèle et de calculer les gradients (coût mémoire réduit) $\rightarrow$ descente de gradient dites **stochastique**
- Permet un compromis biais/variance dans la descente de gradient :
  - batch = base d'apprentissage : biais fort / variance réduite (surapprentissage)
  - batch = 1 : biais faible / variance élevée

Réseau de neurones profond : classification binaire

Exemple d'un réseau de neurones : classification binaire

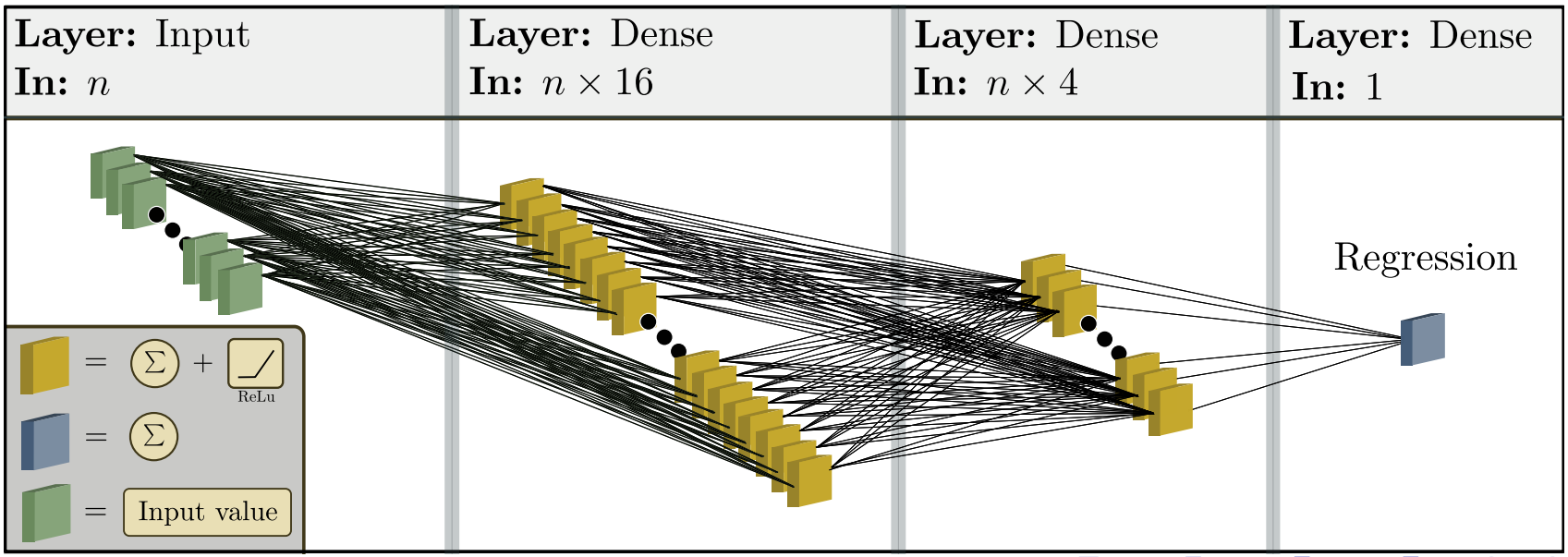
- **Entrée :** un vecteur x de taille n
- **Couches cachées :** 2 complètement connectées
- **Sortie :** une probabilité $\in [0, 1]$ (fonction d'activation Sigmoide)
- **Loss :** entropie croisée binaire, ...



Réseau de neurones profond : régression

Exemple d'un réseau de neurones : régression

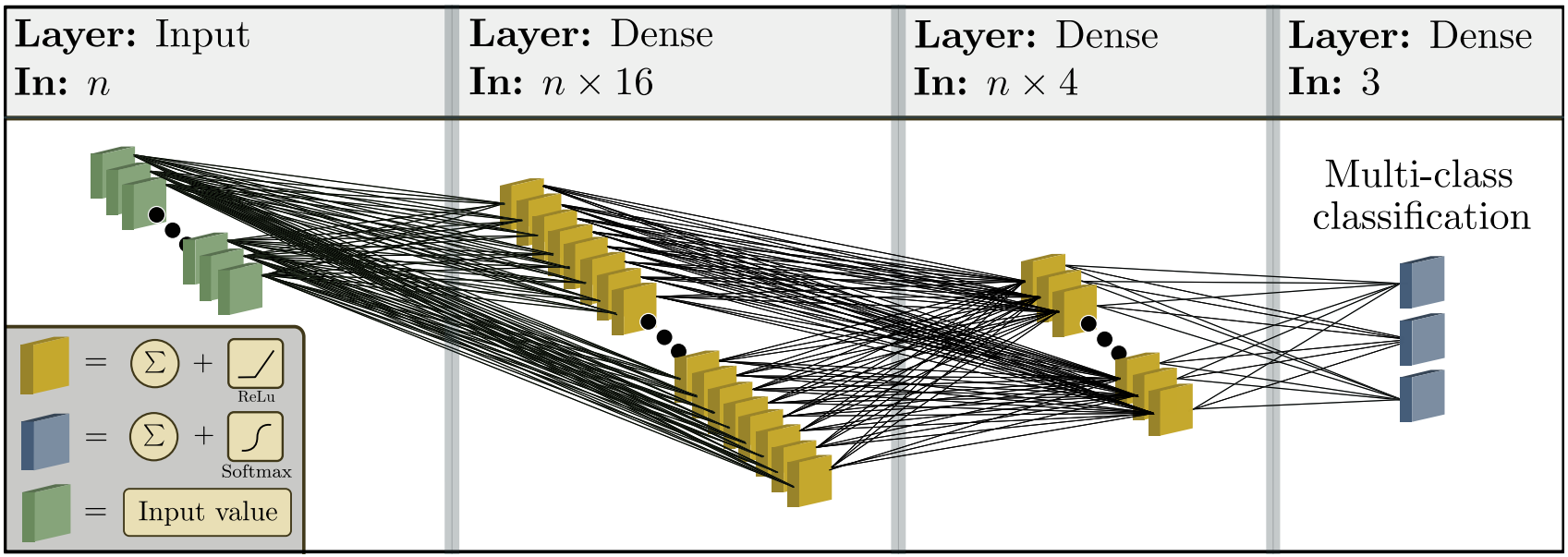
- **Entrée :** un vecteur x de taille n
- **Couches cachées :** 2 complètement connectées
- **Sortie :** un scalaire (pas de fonction d'activation)
- **Loss :** Erreur absolue moyenne (MAE), erreur quadratique moyenne (MSE), ...



Réseau de neurones profond : classification multi-classe

Exemple d'un réseau de neurones : classification multi-classe

- **Couche d'entrée :** un vecteur x de taille n
- **Couches cachées :** 2 complètement connectées
- **Couche de sortie :** un vecteur y de probabilité d'appartenance de classe $(\sum y_i = 1)$
- **Loss :** entropie croisée catégorielle



Réseaux de neurones : exemple en Python

Les principales librairies

- Tensorflow : API de bas niveau développée par Google
- Pytorch : API de bas niveau développée par Facebook
- Keras : API de plus haut niveau intégrée au sein de Tensorflow (prototypage rapide).

```
from tensorflow import keras

input_size = (20, )
model = keras.Sequential([
    keras.layers.Input(input_size, name="InputLayer"),
    keras.layers.Dense(128, activation='relu', name="DenseLayer1"),
    keras.layers.Dense(64, activation='relu', name="DenseLayer2"),
    keras.layers.Dense(1, name="OutputLayer1")
])
```

Réseaux de neurones : exemple en Python

Autres paramètres

- **Epoch :** le nombre de fois que l'on va parcourir les données d'apprentissage
- **Optimizer :** choix de la manière dont on va réaliser une descente de gradient. Généralement `rmsprop` ou `adam` (le choix du taux d'apprentissage est important)
- **Couche dropout :** pourcentage de neurones non mis à jour lors da la backpropagation dans la couche `dense` qui précède.
- **Couche Batch Normalization** : convertit les sorties inter-couches en un format standard (normalisation). [1]

```
input_size = (20, )
model = keras.Sequential([
    ...
])
model.compile(optimizer='rmsprop', loss='mse', metrics=['mae', 'mse'])
model.fit(x_train, y_train, batch_size=64, epochs=20)
```

1. Cette normalisation garantit qu'aucune valeur d'activation n'est trop élevée ou trop faible ce qui permet un apprentissage plus rapide.

TP6 : Réseaux de neurones profonds

Plan

Réseaux de neurones convolutifs : pourquoi ?

La limite des réseaux de neurones multi-couches simples

- Pour une image de taille $24 \times 24 = 576$ neurones d'entrée
- Pour une plus grande résolution : $512 \times 512 = 262144$ neurones d'entrée
- Mais on ne compte ici que la couche d'entrée...

Réseaux de neurones convolutifs : pourquoi ?

La limite des réseaux de neurones multi-couches simples

- Pour une image de taille $24 \times 24 = 576$ neurones d'entrée
- Pour une plus grande résolution : $512 \times 512 = 262144$ neurones d'entrée
- Mais on ne compte ici que la couche d'entrée...

Avec un structure simple

Si l'on compte en nombre de paramètres pour un réseau classique avec s la taille des données d'entrée ($h \times w$ d'une image) :

- $I_s \rightarrow H_{\frac{s}{2}} \rightarrow H_{\frac{s}{4}} \rightarrow O_1$

Réseaux de neurones convolutifs : pourquoi ?

La limite des réseaux de neurones multi-couches simples

- Pour une image de taille $24 \times 24 = 576$ neurones d'entrée
- Pour une plus grande résolution : $512 \times 512 = 262144$ neurones d'entrée
- Mais on ne compte ici que la couche d'entrée...

Avec un structure simple

Si l'on compte en nombre de paramètres pour un réseau classique avec s la taille des données d'entrée ($h \times w$ d'une image) :

- $I_s \rightarrow H_{\frac{s}{2}} \rightarrow H_{\frac{s}{4}} \rightarrow O_1$
- $s = 24 \times 24$: 207 937 paramètres
- $s = 512 \times 512$: 42 949 935 105 paramètres (outch...)

Réseaux de neurones convolutifs : contexte

Une solution : couche par convolution

- Utilisation d'une fenêtre de poids (kernel)
- La fenêtre stocke les paramètres à entraîner pour le modèle
- Réduction drastique du nombre de paramètres à entraîner/stocker
  $\rightarrow$ une couche classique entièrement connectée peut-être utilisée à la fin
- Chaque fenêtre entraînée offre un plan filtré de l'image (par convolution)

Réseaux de neurones convolutifs : contexte

Une solution : couche par convolution

- Utilisation d'une fenêtre de poids (kernel)
- La fenêtre stocke les paramètres à entraîner pour le modèle
- Réduction drastique du nombre de paramètres à entraîner/stocker
  $\rightarrow$ une couche classique entièrement connectée peut-être utilisée à la fin
- Chaque fenêtre entraînée offre un plan filtré de l'image (par convolution)

Une idée pas si récente

LeNet [LeCun et al. 1989] *Backpropagation Applied to Handwritten Zip Code Recognition*

- Simplement limitée par la puissance de calcul pour de plus grandes images

Réseaux de neurones convolutifs : principe

Principe de convolution

- Utilisation d'un kernel de taille $kx \times ky$ (si 2-D)
- On parcourt l'image d'entrée en appliquant ce kernel [1]
- On obtient en sortie une image filtrée
- Un kernel peut-être n-D

Input

4	0	4	3	0
1	2	7	5	2
4	8	1	6	5
1	5	0	7	8
8	2	6	3	1

1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

Réseaux de neurones convolutifs : principe

Principe de convolution

- Utilisation d'un kernel de taille $kx \times ky$ (si 2-D)
- On parcourt l'image d'entrée en appliquant ce kernel [1]
- On obtient en sortie une image filtrée
- Un kernel peut-être n-D

Input

4	0	4	3	0
1	2	7	5	2
4	8	1	6	5
1	5	0	7	8
8	2	6	3	1

Kernel

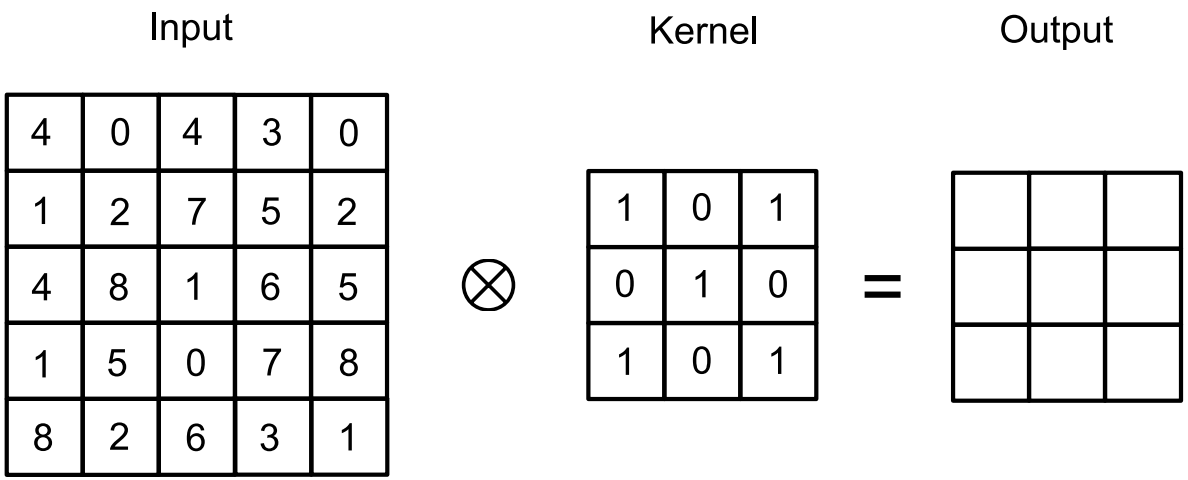
1	0	1
0	1	0
1	0	1

1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

Réseaux de neurones convolutifs : principe

Principe de convolution

- Utilisation d'un kernel de taille $kx \times ky$ (si 2-D)
- On parcourt l'image d'entrée en appliquant ce kernel [1]
- On obtient en sortie une image filtrée
- Un kernel peut-être n-D

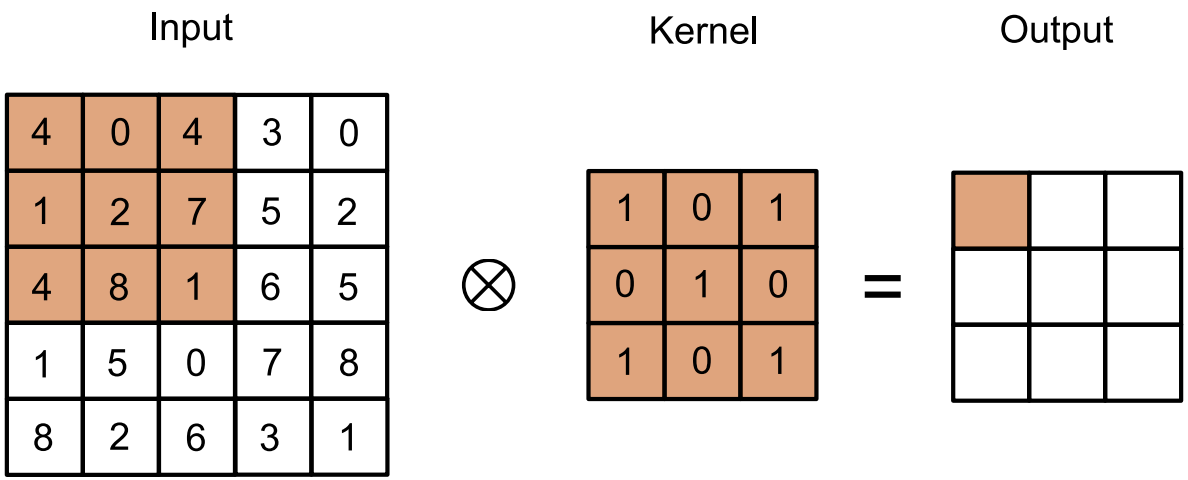


1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

Réseaux de neurones convolutifs : principe

Principe de convolution

- Utilisation d'un kernel de taille $kx \times ky$ (si 2-D)
- On parcourt l'image d'entrée en appliquant ce kernel [1]
- On obtient en sortie une image filtrée
- Un kernel peut-être n-D



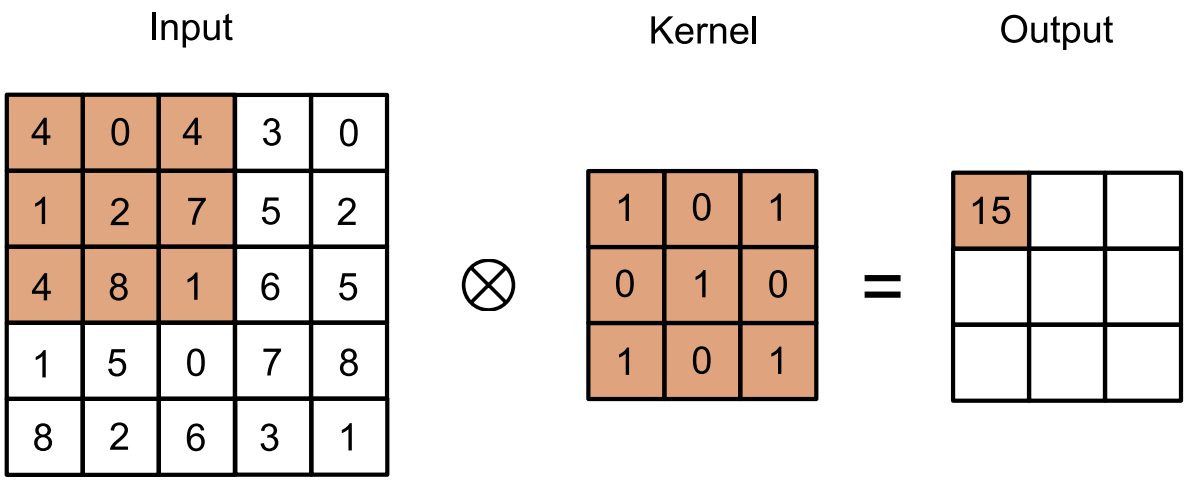
$$y_{00} = (4 \times 1 + 0 \times 0 + 4 \times 1 + 1 \times 0 + 2 \times 1 + 7 \times 0 + 4 \times 1 + 8 \times 0 + 1 \times 1)$$
$$= 15$$

1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

Réseaux de neurones convolutifs : principe

Principe de convolution

- Utilisation d'un kernel de taille $kx \times ky$ (si 2-D)
- On parcourt l'image d'entrée en appliquant ce kernel [1]
- On obtient en sortie une image filtrée
- Un kernel peut-être n-D

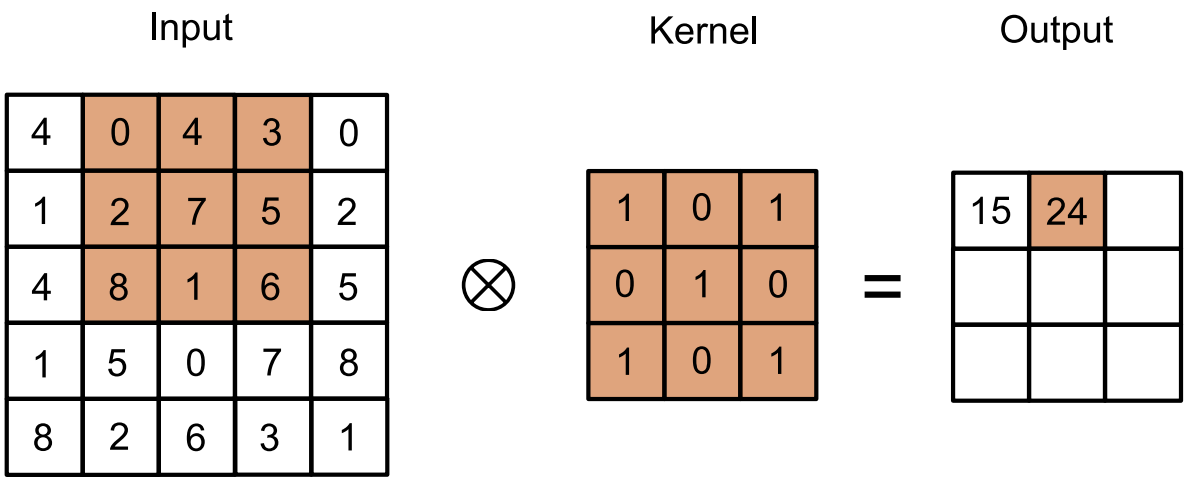


1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

Réseaux de neurones convolutifs : principe

Principe de convolution

- Utilisation d'un kernel de taille $kx \times ky$ (si 2-D)
- On parcourt l'image d'entrée en appliquant ce kernel [1]
- On obtient en sortie une image filtrée
- Un kernel peut-être n-D

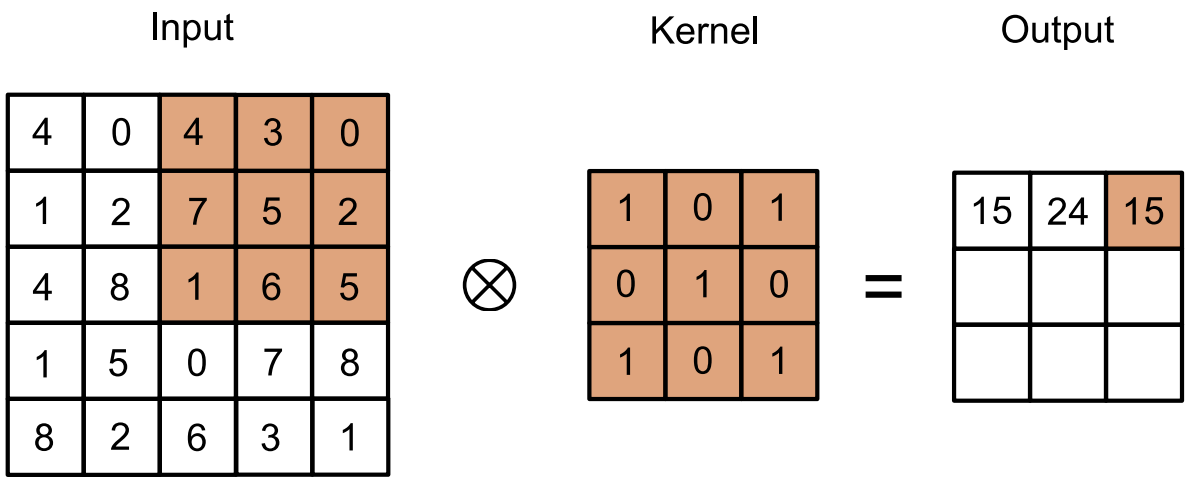


1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

Réseaux de neurones convolutifs : principe

Principe de convolution

- Utilisation d'un kernel de taille $kx \times ky$ (si 2-D)
- On parcourt l'image d'entrée en appliquant ce kernel [1]
- On obtient en sortie une image filtrée
- Un kernel peut-être n-D

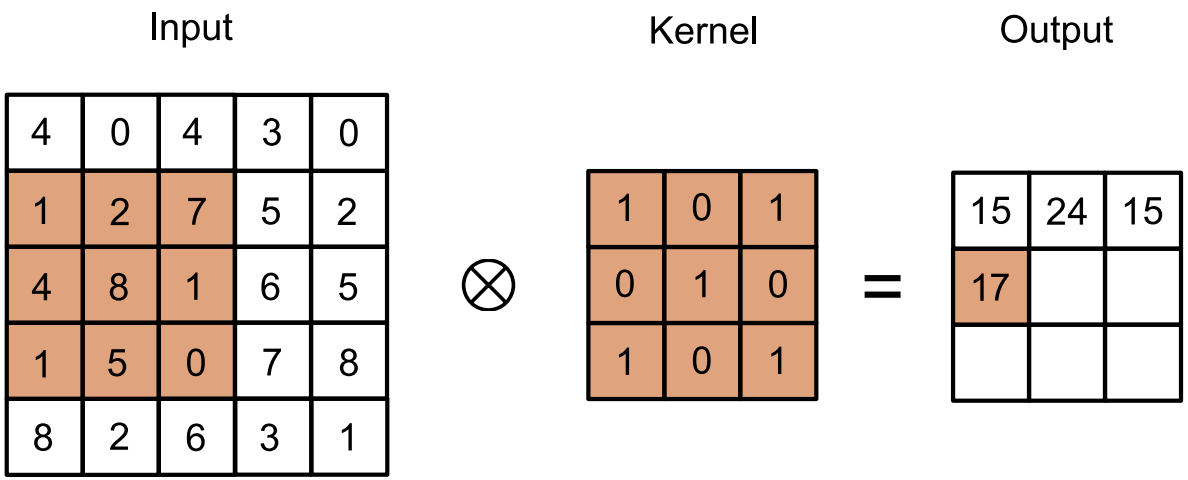


1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

Réseaux de neurones convolutifs : principe

Principe de convolution

- Utilisation d'un kernel de taille $kx \times ky$ (si 2-D)
- On parcourt l'image d'entrée en appliquant ce kernel [1]
- On obtient en sortie une image filtrée
- Un kernel peut-être n-D

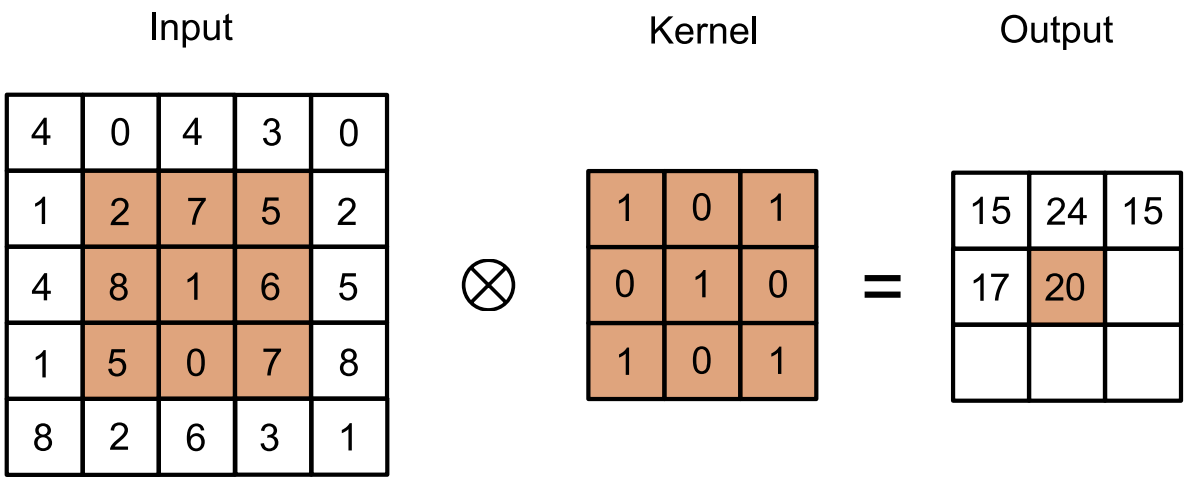


1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

Réseaux de neurones convolutifs : principe

Principe de convolution

- Utilisation d'un kernel de taille $kx \times ky$ (si 2-D)
- On parcourt l'image d'entrée en appliquant ce kernel [1]
- On obtient en sortie une image filtrée
- Un kernel peut-être n-D

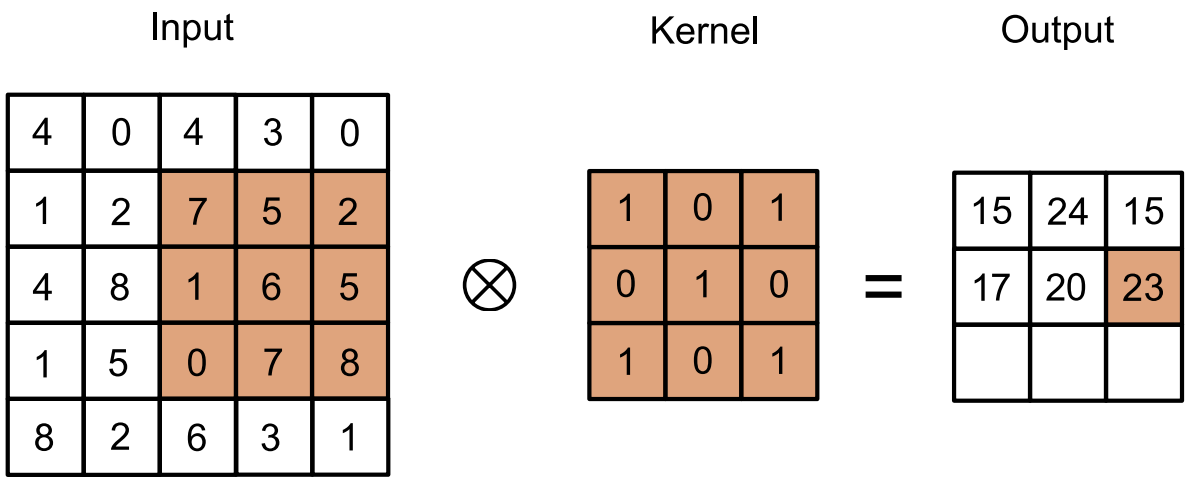


1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

Réseaux de neurones convolutifs : principe

Principe de convolution

- Utilisation d'un kernel de taille $kx \times ky$ (si 2-D)
- On parcourt l'image d'entrée en appliquant ce kernel [1]
- On obtient en sortie une image filtrée
- Un kernel peut-être n-D

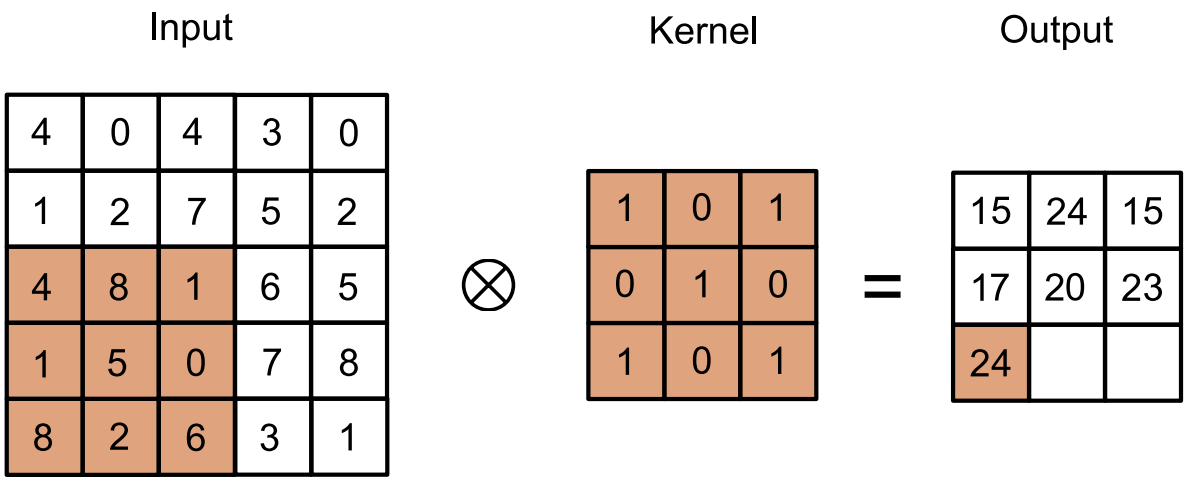


1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

Réseaux de neurones convolutifs : principe

Principe de convolution

- Utilisation d'un kernel de taille $kx \times ky$ (si 2-D)
- On parcourt l'image d'entrée en appliquant ce kernel [1]
- On obtient en sortie une image filtrée
- Un kernel peut-être n-D

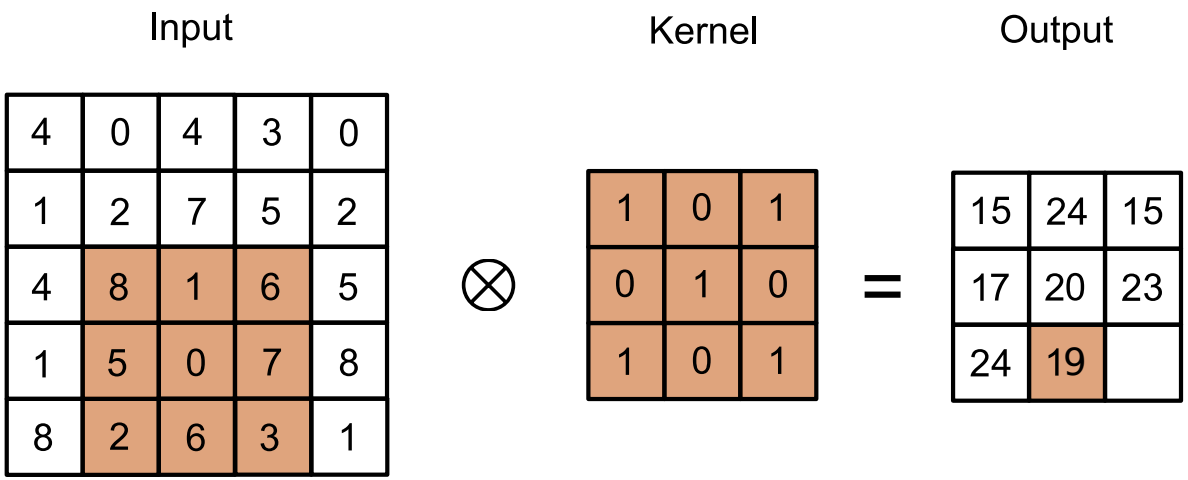


1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

Réseaux de neurones convolutifs : principe

Principe de convolution

- Utilisation d'un kernel de taille $kx \times ky$ (si 2-D)
- On parcourt l'image d'entrée en appliquant ce kernel [1]
- On obtient en sortie une image filtrée
- Un kernel peut-être n-D

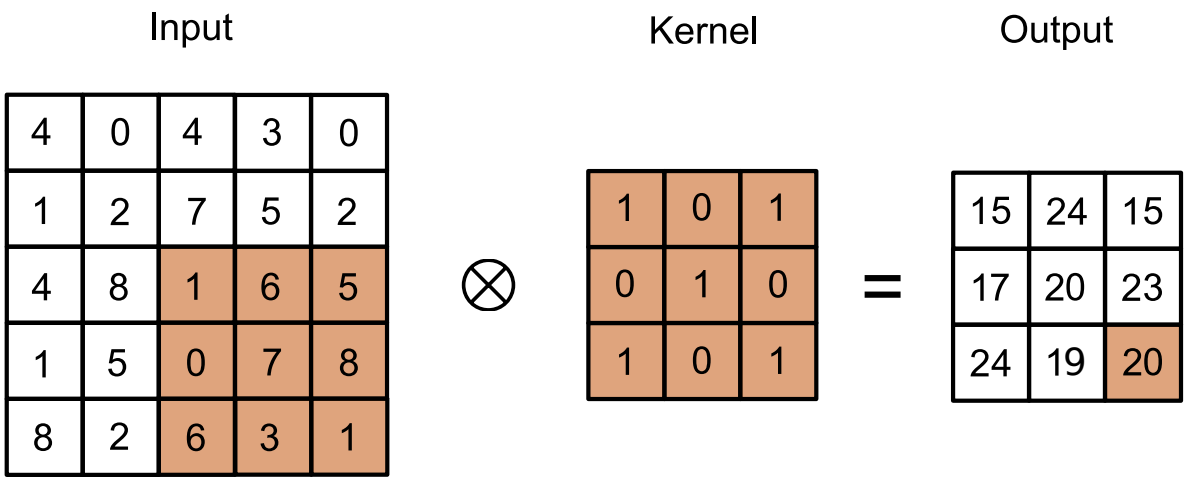


1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

Réseaux de neurones convolutifs : principe

Principe de convolution

- Utilisation d'un kernel de taille $kx \times ky$ (si 2-D)
- On parcourt l'image d'entrée en appliquant ce kernel [1]
- On obtient en sortie une image filtrée
- Un kernel peut-être n-D



1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

Réseaux de neurones convolutifs : paramètres

Les paramètres importants [1]

- Taille du kernel
- Nombre de plans obtenues en sortie après application de convolution (nombre de kernels)
- **Padding** (« rembourrage ») / **Stride** (pas d'enjambement)

1. Il existe aussi le paramètre `dilation` que nous n'aborderons pas ici.

Réseaux de neurones convolutifs : paramètres

Les paramètres importants [1]

- Taille du kernel
- Nombre de plans obtenues en sortie après application de convolution (nombre de kernels)
- **Padding** (« rembourrage ») / **Stride** (pas d'enjambement)

Les couches intermédiaires

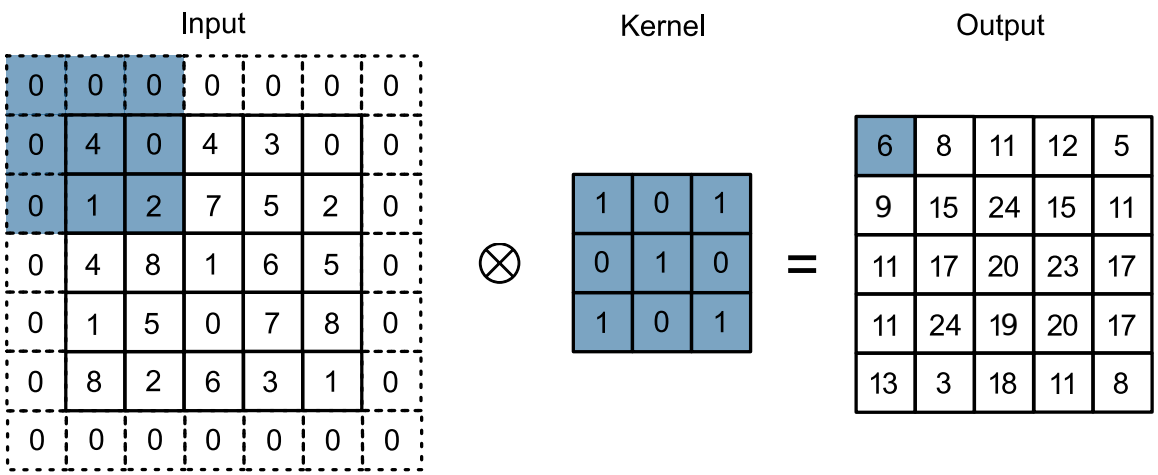
- Couche d'activation (toujours) : application de la fonction d'activation sur chaque nouveau « pixel » obtenu du plan
- **Pooling** (mise en commun) : réduction rapide et filtrage des données (aucun paramètre)
- **Dropout** : possible également mais rarement utilisé pour la convolution (perte de cohérence spatiale des informations filtrées). Peut toutefois être utilisée après le Pooling.
- **Batch Normalization** : peut-être utilisée en sortie d'un layer convolutif

1. Il existe aussi le paramètre `dilation` que nous n'aborderons pas ici.

Réseaux de neurones convolutifs : padding

Le paramètre padding

- Ajout de bord à l'image
- Permet davantage de prendre en compte les informations de bord de l'image
- Peut permettre de conserver la taille de l'image

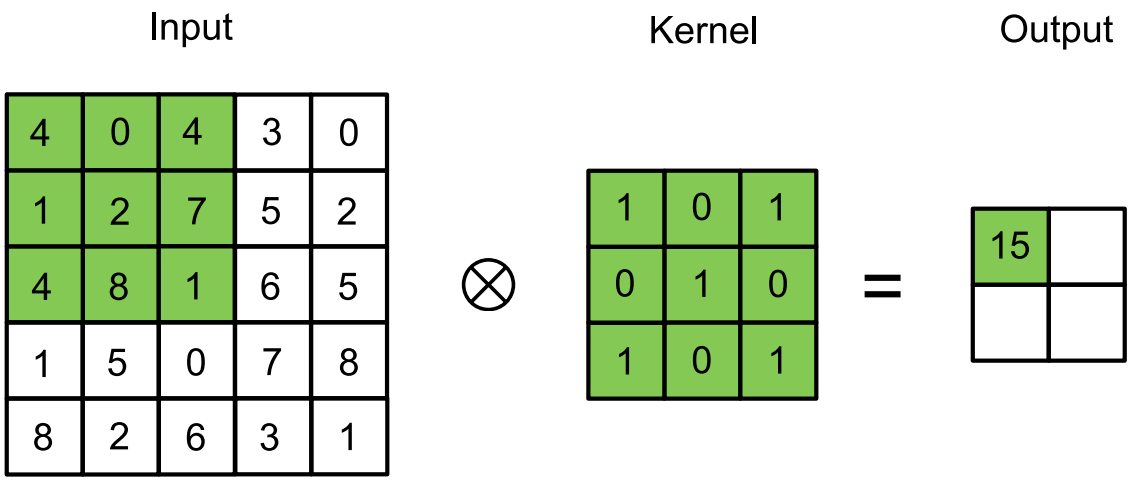


Couche convolutive avec un padding $= 1$

Réseaux de neurones convolutifs : stride

Le paramètre `stride`

- Paramètre du pas de décalage du kernel (par défaut 1)
- Réduit plus considérablement la sortie après convolution si > 1

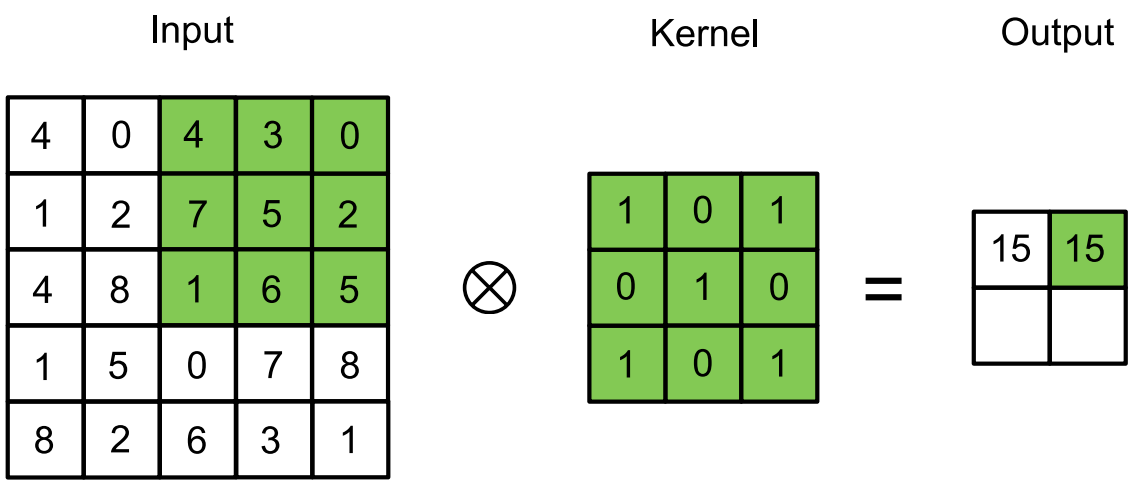


Convolution (stride $= 2$ et padding $= 0$)

Réseaux de neurones convolutifs : stride

Le paramètre `stride`

- Paramètre du pas de décalage du kernel (par défaut 1)
- Réduit plus considérablement la sortie après convolution si > 1

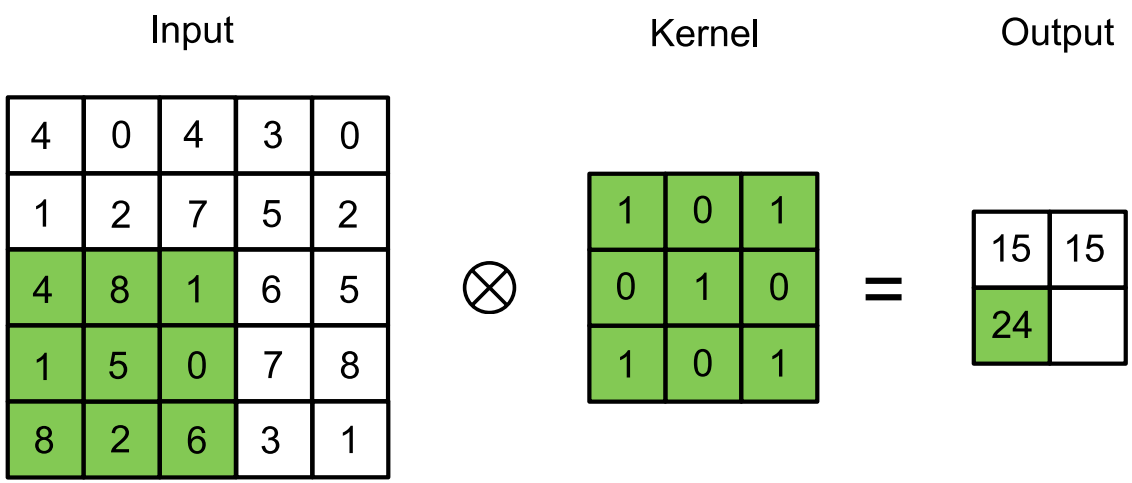


Convolution (stride $= 2$ et padding $= 0$)

Réseaux de neurones convolutifs : stride

Le paramètre `stride`

- Paramètre du pas de décalage du kernel (par défaut 1)
- Réduit plus considérablement la sortie après convolution si > 1

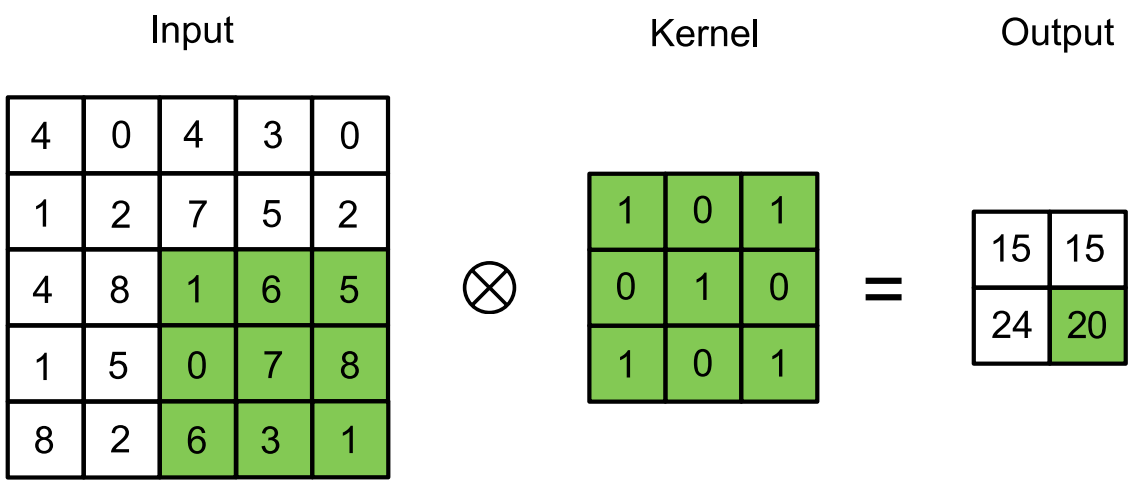


Convolution (stride $= 2$ et padding $= 0$)

Réseaux de neurones convolutifs : stride

Le paramètre `stride`

- Paramètre du pas de décalage du kernel (par défaut 1)
- Réduit plus considérablement la sortie après convolution si > 1

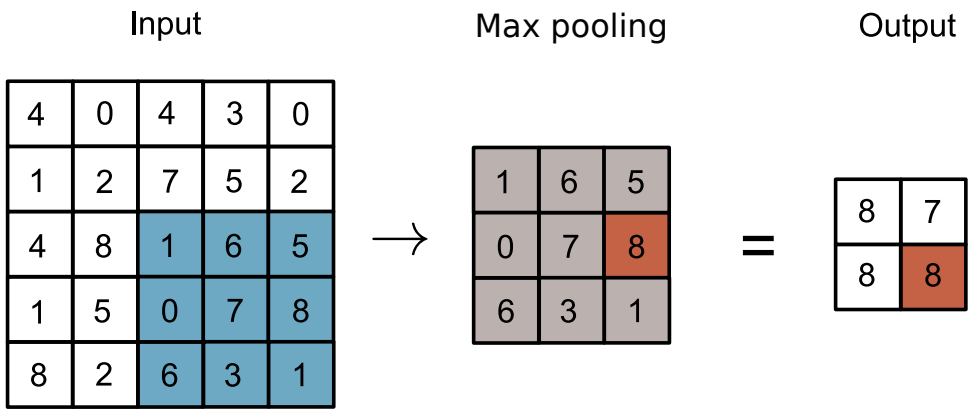


Convolution (stride $= 2$ et padding $= 0$)

Réseaux de neurones convolutifs : pooling

La couche `pooling`

- Permet le regroupement d'informations à partir d'une taille de kernel
- On peut aussi définir le **padding** et le **stride**
- Ne nécessite aucun paramètre d'apprentissage
- Peut-être le max, le min, la moyenne...



Max-Pooling (stride $= 2$ et padding $= 0$)

Réseaux de neurones convolutifs : réduction de complexité

Le nombre de paramètres à entraîner

Admettons que nous avons :

- Un kernel de taille $kx \times ky$
- I le nombre de plans d'entrée (nombre de canals si image, par exemple 3 pour RGB)
- F le nombre de plan de sortie (généralement appelé « feature maps »)

On obtient alors un nombre P de paramètres pour notre couche convolutive de l'ordre de :

$$P = (kx \times ky \times I + 1) \times F$$

Réseaux de neurones convolutifs : sortie

Taille des données de sortie

Après application d'une couche convolutive sur des données d'entrée de dimension $H \times W$ (exemple 2-D), la taille de sortie sera définie par :

$$W_{out} = \frac{W_{in} - kx + 2 \times p}{s} + 1$$

$$H_{out} = \frac{H_{in} - ky + 2 \times p}{s} + 1$$

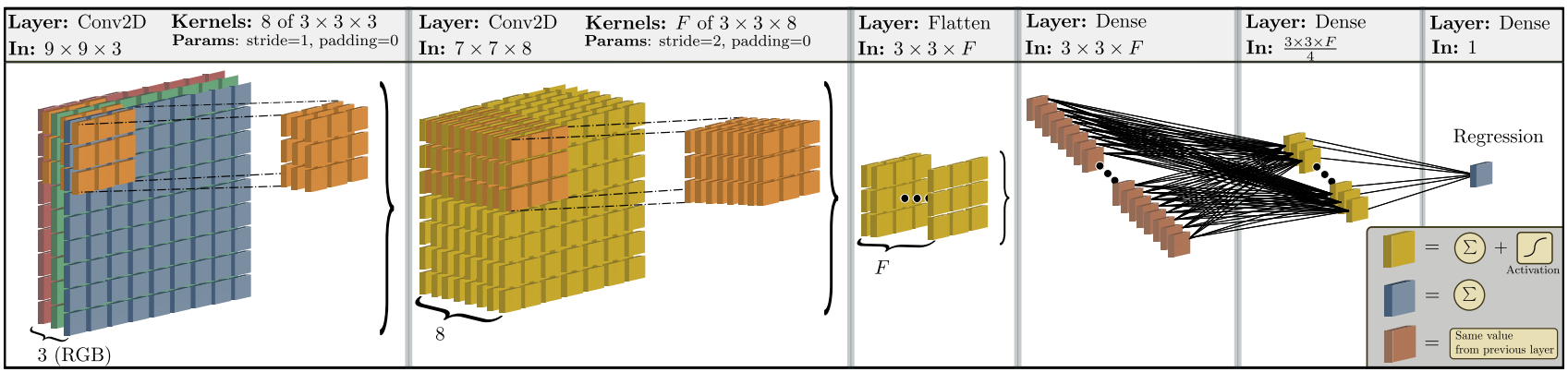
Avec :

- Un kernel de dimension $kx \times ky$
- s le stride
- p le padding

Réseaux de neurones convolutifs : un exemple

Exemple d'architecture : régression

- **Entrée :** une image RGB de taille 9×9
- **Sortie :** un scalaire (pas de fonction d'activation dans ce cas)



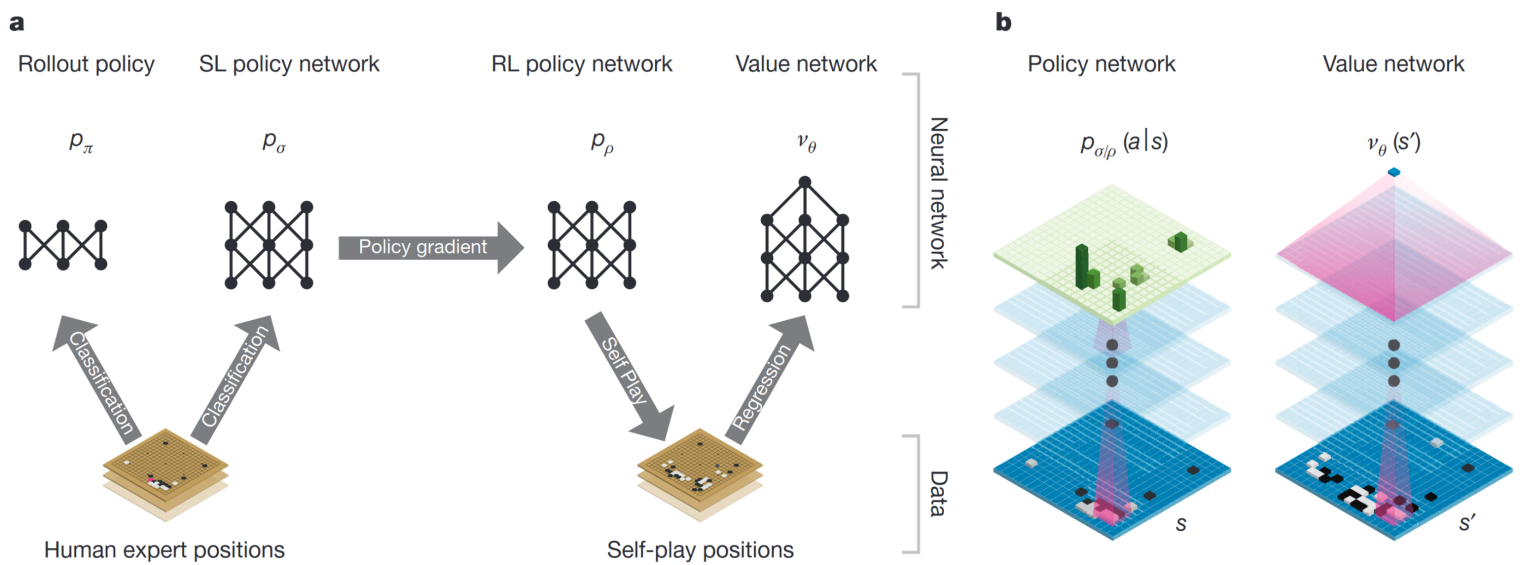
TP7 : Réseaux de neurones convolutifs

Plan

MCTS : utilisation du Deep Learning

MCTS amélioré

- SL Policy network : formés à partir de données humaines
- Rollout policy (réseau simplifié) : politique rapide de choix de mouvements (expert) pour déplacer quand on simule
- RL Policy network : renforcée en jouant contre elle-même
- Value network : score estimé de $\mathcal{V}_\theta$ (probabilité de gagner la partie)

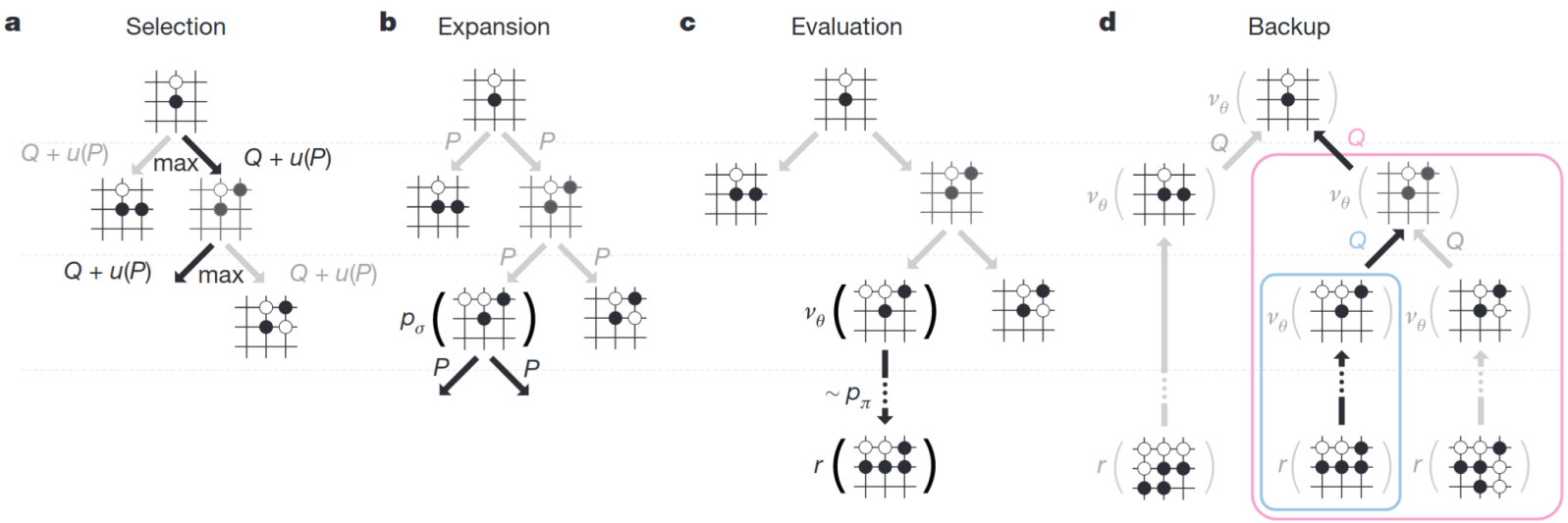


Principe d'AlphaGo (Silver et al. 2016)

MCTS : utilisation du Deep Learning

MCTS amélioré : phase d'apprentissage

- Sélection : $u(P)$ partie exploration
- Expansion : politique p_σ (afin d'améliorer la politique RL et le réseau de politique)
- Évaluation : $V(s) = (1 - \lambda)\mathcal{V}_\theta(s) + \lambda z$ (z le résultat de la fonction r)
- Backpropagation : $V(s)$ est rétro-propagé afin de calculer $Q(s, a)$



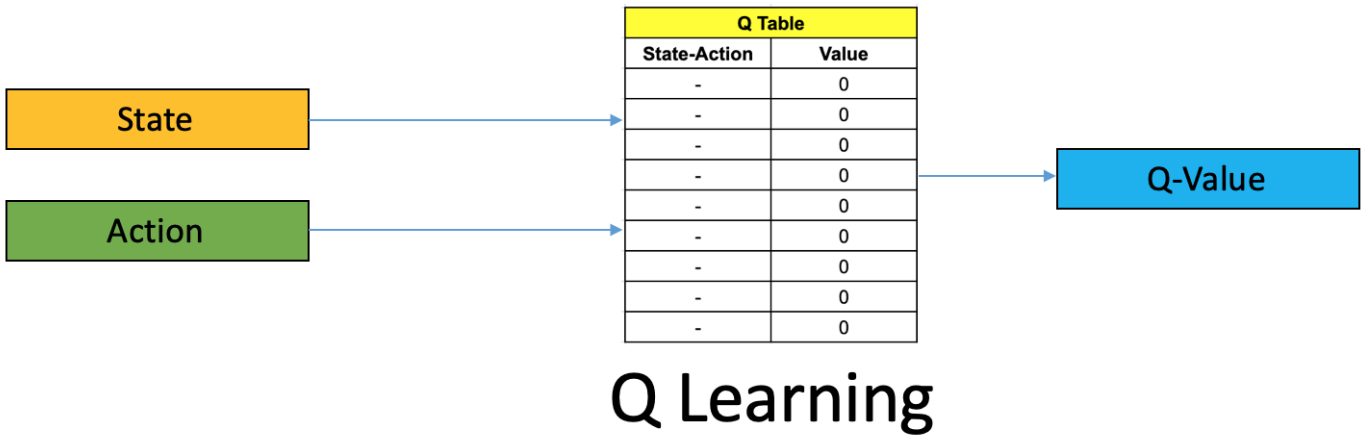
AlphaGo MCTS (Silver et al. 2016)

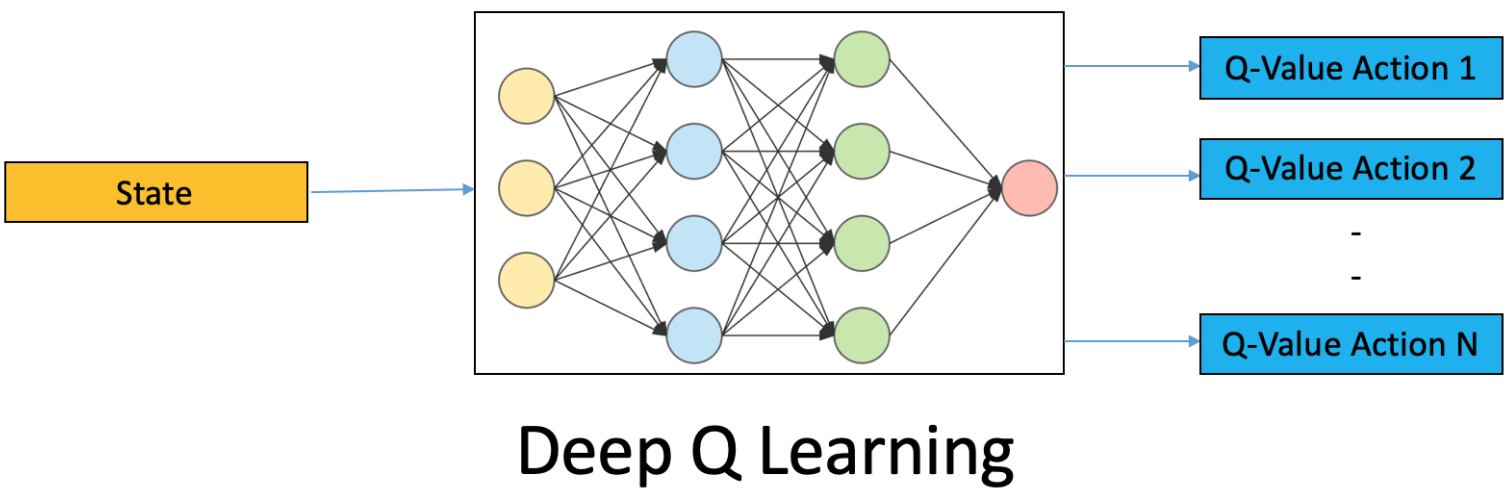
MCTS : utilisation du Deep Learning

Pour aller plus loin : AlphaZero

- Ne requiert aucune données humaines (experts)
- Apprend en jouant contre lui-même de zéro
- Politique de choix améliorée

Deep Q-learning





Source : https://cdn.analyticsvidhya.com

Deep Q-learning

L'équation de Bellman

$$Q(s,a) = r + \gamma \max_a Q(s',a')$$

- Pas d'utilisation de la différence temporelle pour la mise à jour dans cette forme d'équation
- Récompense actuelle + celles attendues

Deep Q-learning

L'équation de Bellman

$$Q(s,a) = r + \gamma \max_a Q(s',a')$$

- Pas d'utilisation de la différence temporelle pour la mise à jour dans cette forme d'équation
- Récompense actuelle + celles attendues

Estimation de l'erreur : Deep Q-learning

$$L = \left(r + \gamma \max_a Q(s',a') - Q(s,a)\right)^2$$

- $r + \gamma \max_a Q(s',a') \rightarrow$ **valeur cible**
- $Q(s,a) \rightarrow$ **sortie actuelle du réseau neuronal**

Deep Q-learning

Estimation de l'erreur : Deep Q-learning

$$L = \left(r + \gamma \max_a Q(s', a') - Q(s, a)\right)^2$$

- $r + \gamma \max_a Q(s', a') \rightarrow$ **valeur cible**
- $Q(s, a) \rightarrow$ **sortie actuelle du réseau neuronal**

Comment mettre à jour ?

1. Simuler l'étape pendant l'épisode (simulation) jusqu'à la fin
   - *(old_state, action, reward, new_state, done)*
2. Stocker les mouvements précédents dans les données de relecture d'actions (éviter le problème du gradient : étapes successives du même épisode).
3. Échantillonner une ou plusieurs actions (batch) :
   - Permet de calculer (hors ligne) : $r + \gamma \max_a Q(s', a')$
   - Puis propagation de la perte dans le NN

Références

[1] Peter Auer. « Using confidence bounds for exploitation-exploration trade-offs ». In : *Journal of Machine Learning Research* 3.Nov (2002), p. 397-422.

[2] Levente Kocsis et Csaba Szepesvári. « Bandit based monte-carlo planning ». In : *European conference on machine learning*. Springer. 2006, p. 282-293.

[3] David Silver et al. « Mastering the game of Go with deep neural networks and tree search ». In : *nature* 529.7587 (2016), p. 484-489.