Apprentissage profond (Deep learning)

BUT3 : Université du Littoral Côte d'Opale

Jérôme Buisine

Team: IMAP (Images et Apprentissage)

11 décembre 2023

Objectifs du module

Les différents objectifs

- Comprendre la notion d'apprentissage profond.
- Comprendre le fonctionnement / mécanismes et les limites des réseaux de neurones.
- Identifier l'architecture / hyperparamètres adéquat aux problèmes.
- Compréhension / aperçu d'architectures plus complexes.

Objectifs du module

Les différents objectifs

- Comprendre la notion d'apprentissage profond.
- Comprendre le fonctionnement / mécanismes et les limites des réseaux de neurones.
- Identifier l'architecture / hyperparamètres adéquat aux problèmes.
- Compréhension / aperçu d'architectures plus complexes.

Remarques importantes

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

Déroulement du module

Organisation du module

- Partie cours à chaque séance si nouvelle notion abordée
- **chaque TD/TP** :
  - un commit à fournir comme rendu.
  - une note.
- 1 projet de mise en situation

Note finale (si examen)

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

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.

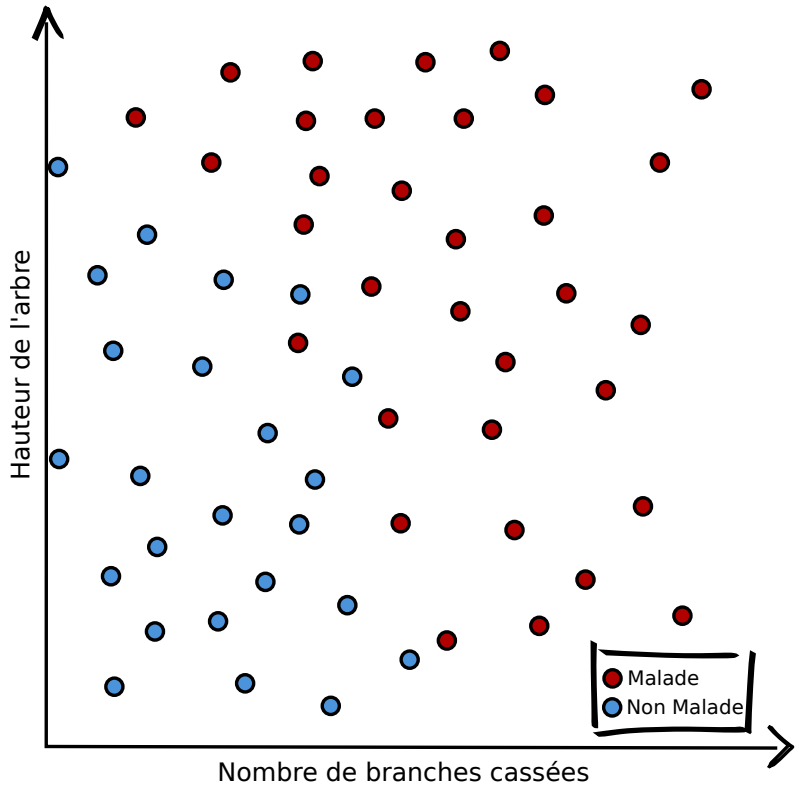
Afin de relever le pourcentage d'arbre malade pour cette année et les années suivantes, ils décident d'identifier deux critères : le nombre de branches cassées, la hauteur de l'arbre.

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

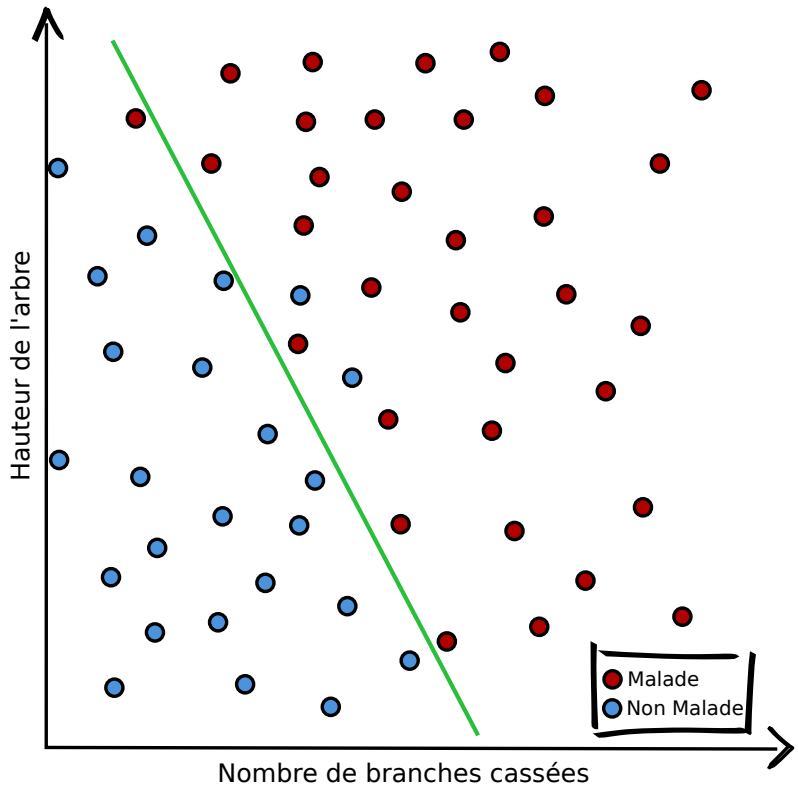
Maladie d'un arbre : représentation des données



Remarque

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

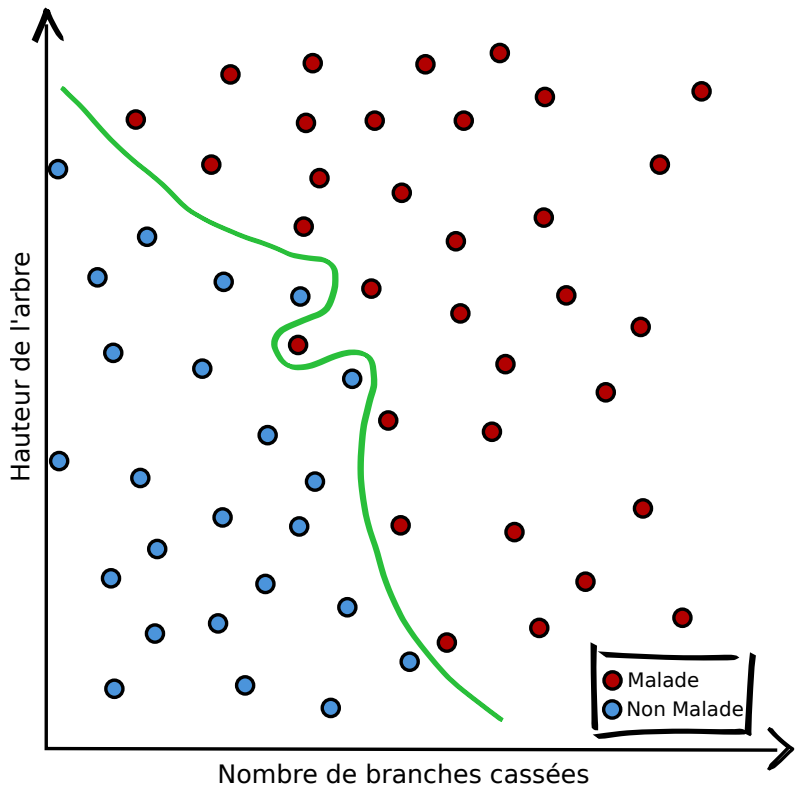
Maladie d'un arbre : modèle de séparation



Remarque

La ligne séparatrice (modèle) des données implique quelques erreurs

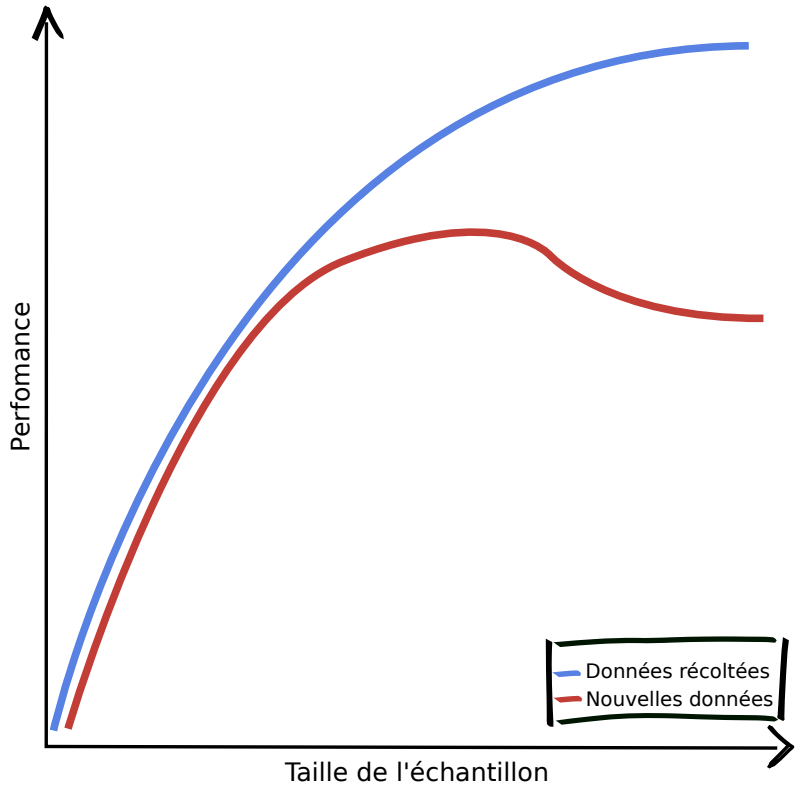
Maladie d'un arbre : modèle de séparation



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...

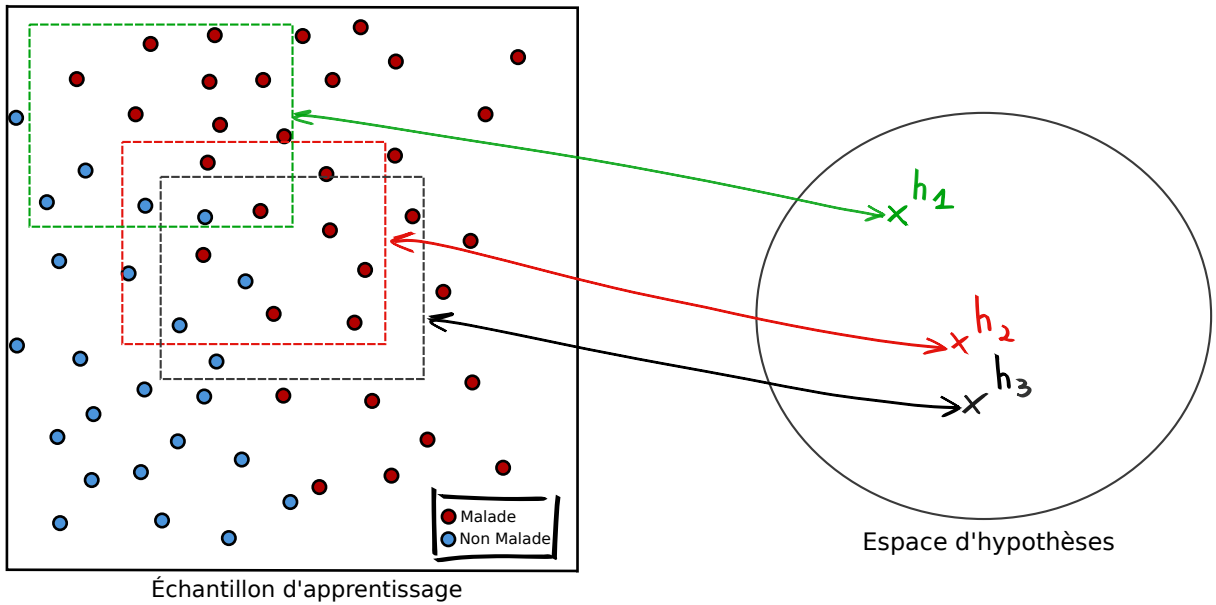
Apprendre un comportement étant donné une observation pour choisir la meilleure action dans le futur étant donné l'état de l'environnement.

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.
- $y_i = f(x_i)$

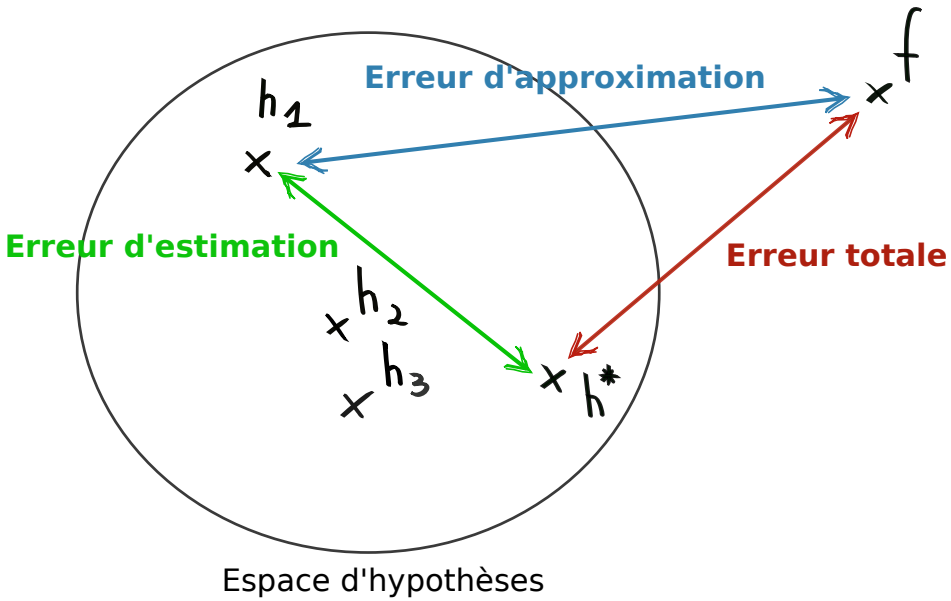
Risque réel

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.

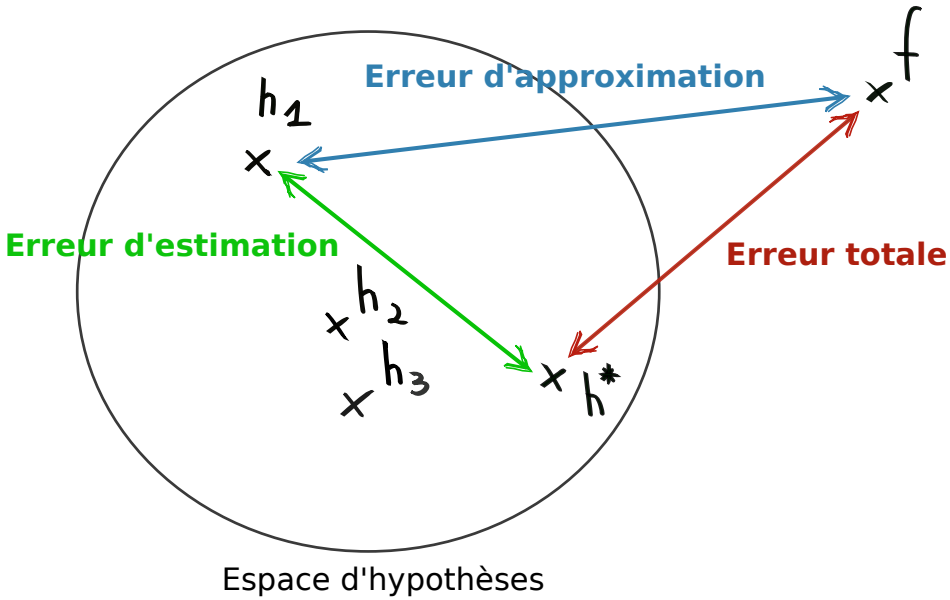
Biais et variance



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

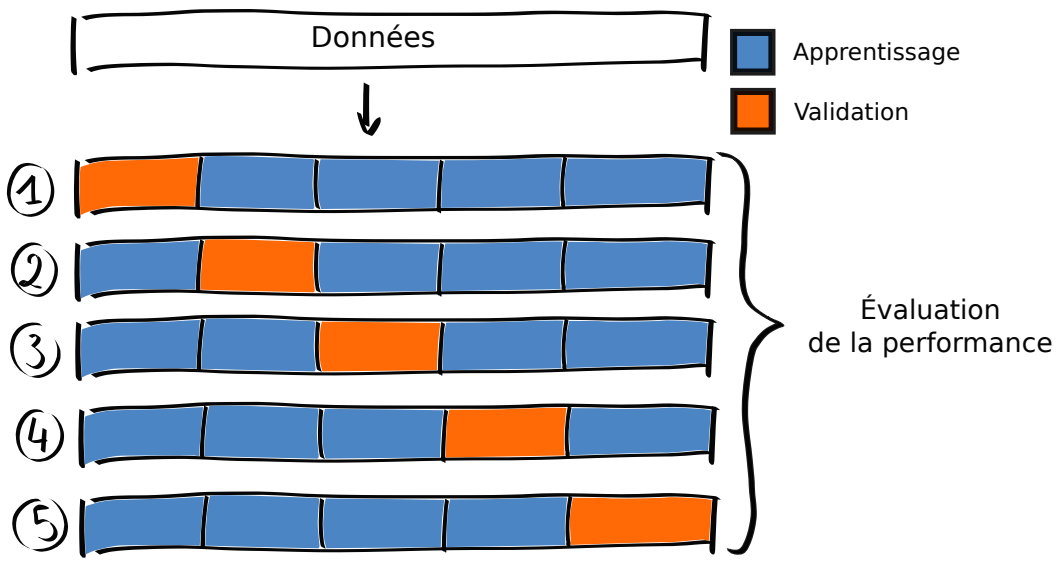
- Base d'apprentissage : sert pour l'entraînement du modèle (par exemple 70% des données) ;
- Base de test : permet d'évaluer le modèle sur de nouvelles données (par exemple 30% des données) ;
- Utiliser la fonction performance P (ou d'erreur) sur la base de test pour comparer les modèles.

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)

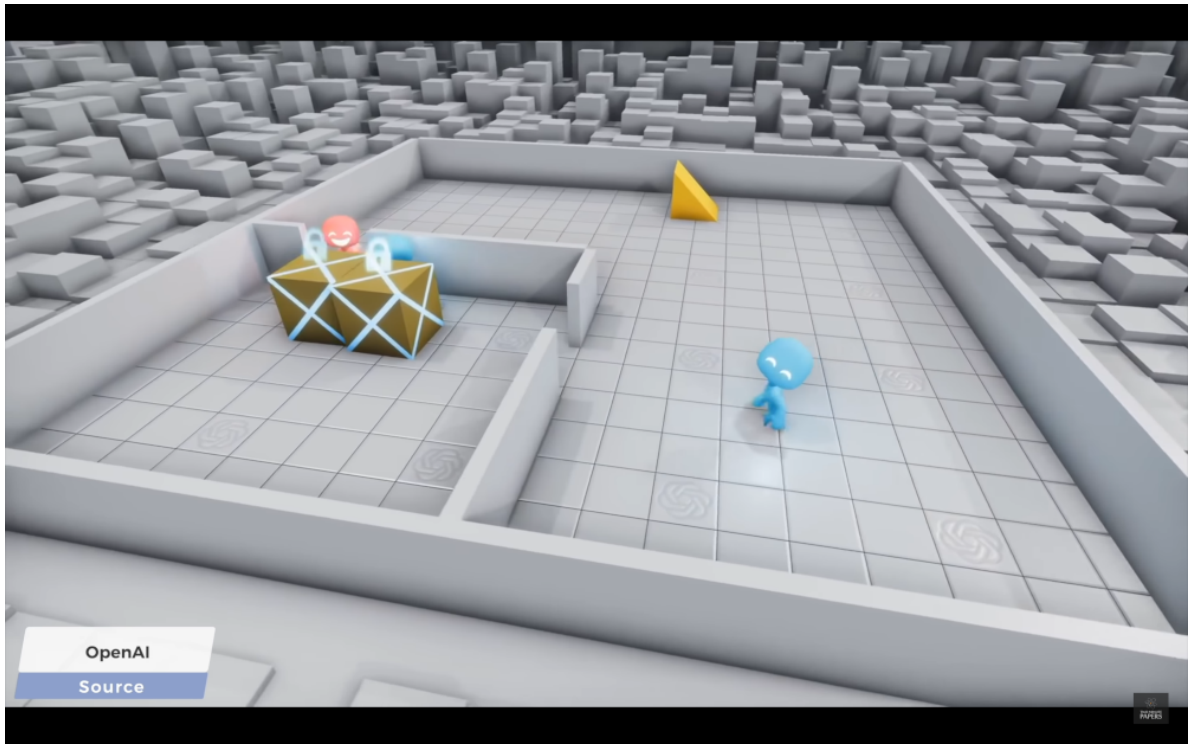
Au sein d'un environnement de jeu complexe...

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

Hide and Seek (OpenAI, 2019)

À partir d'un environnement inconnu...

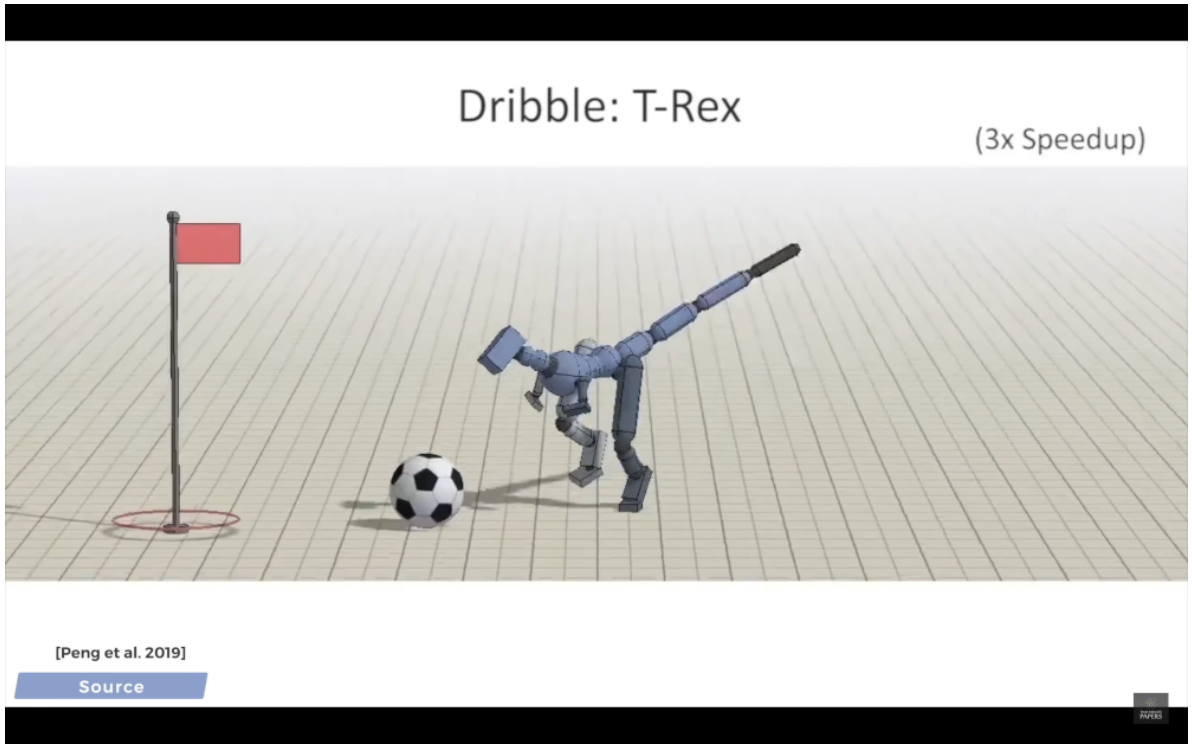
les agents doivent trouver un moyen de gagner en exploitant les actions possibles



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)

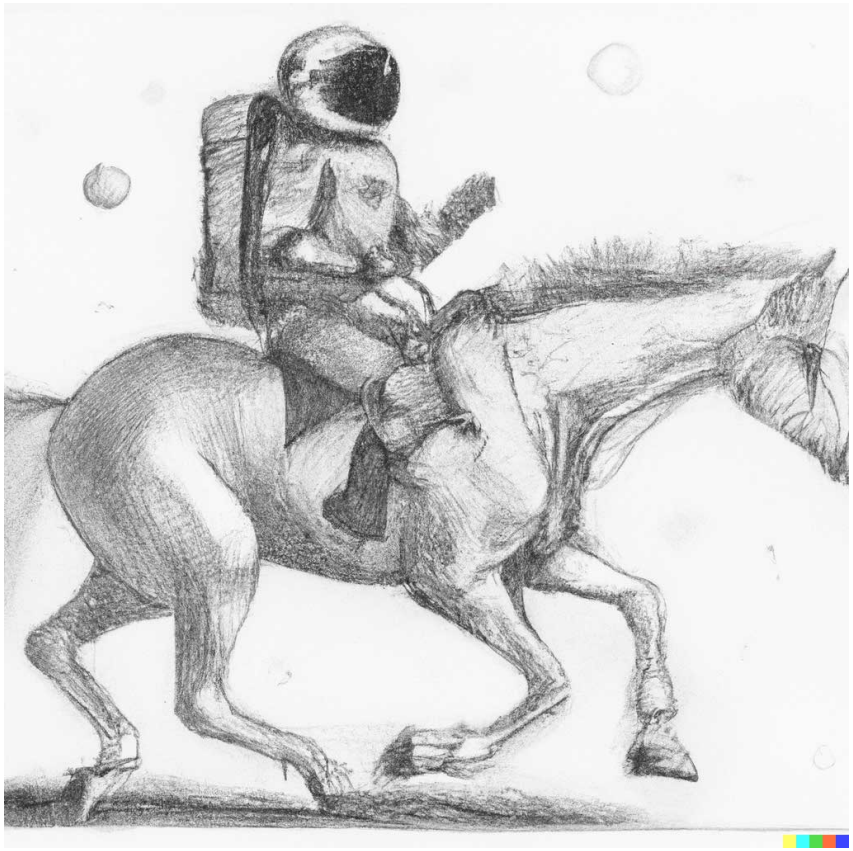
À partir d'un texte, le modèle génère une image

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

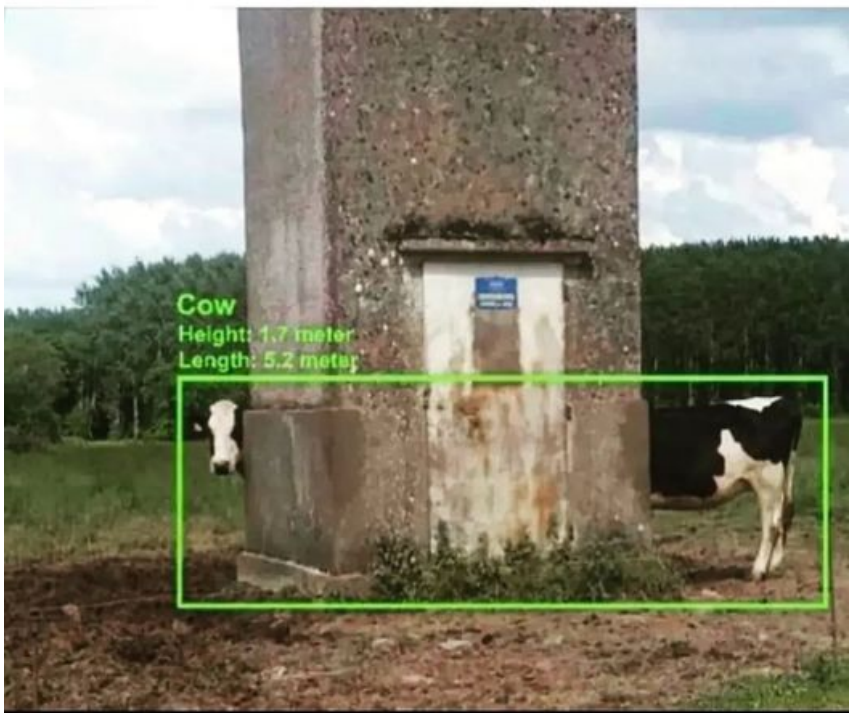
An astronaut lounging in a tropical resort in space in a vaporwave style

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.

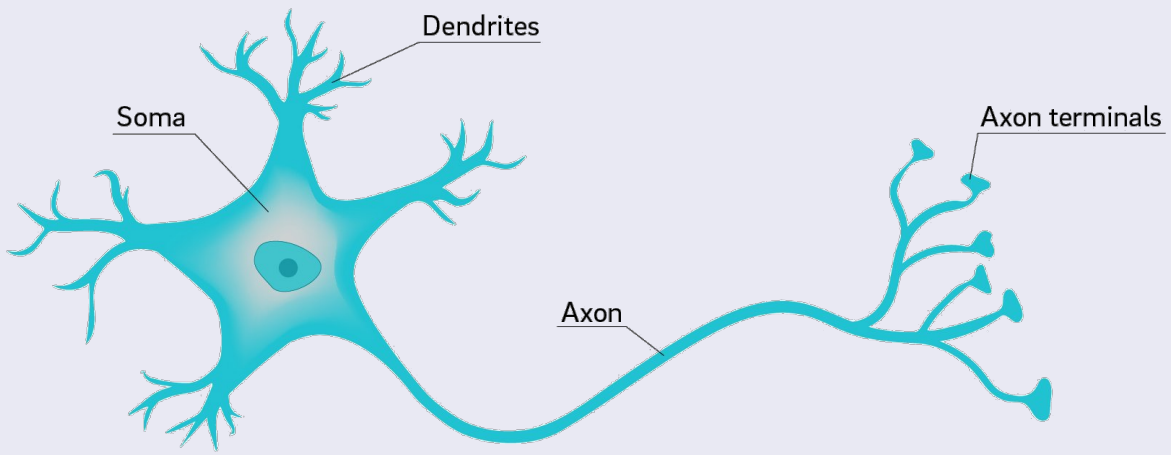
People: AI so smart, it will distroy us!
AI:



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

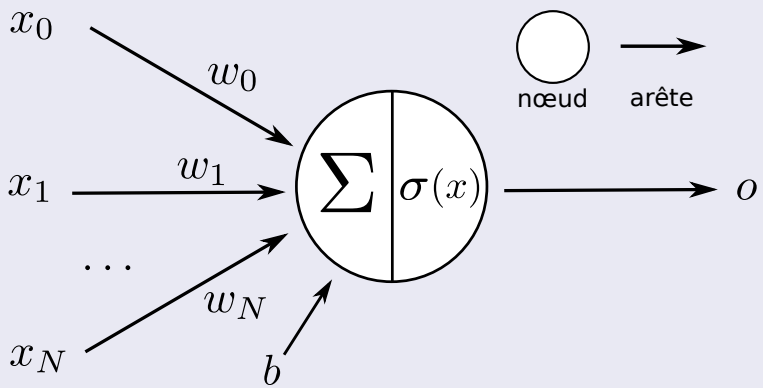
1. Source : https://www.upgrad.com

Plan

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] $(y = ax + b)$

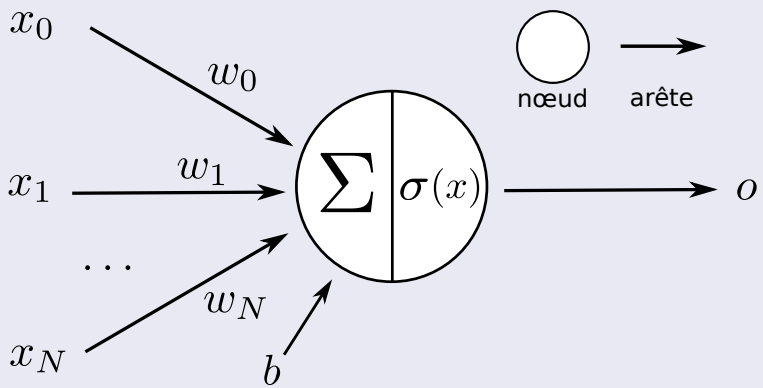


1. Ce paramètre est également appelé seuil

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) :

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

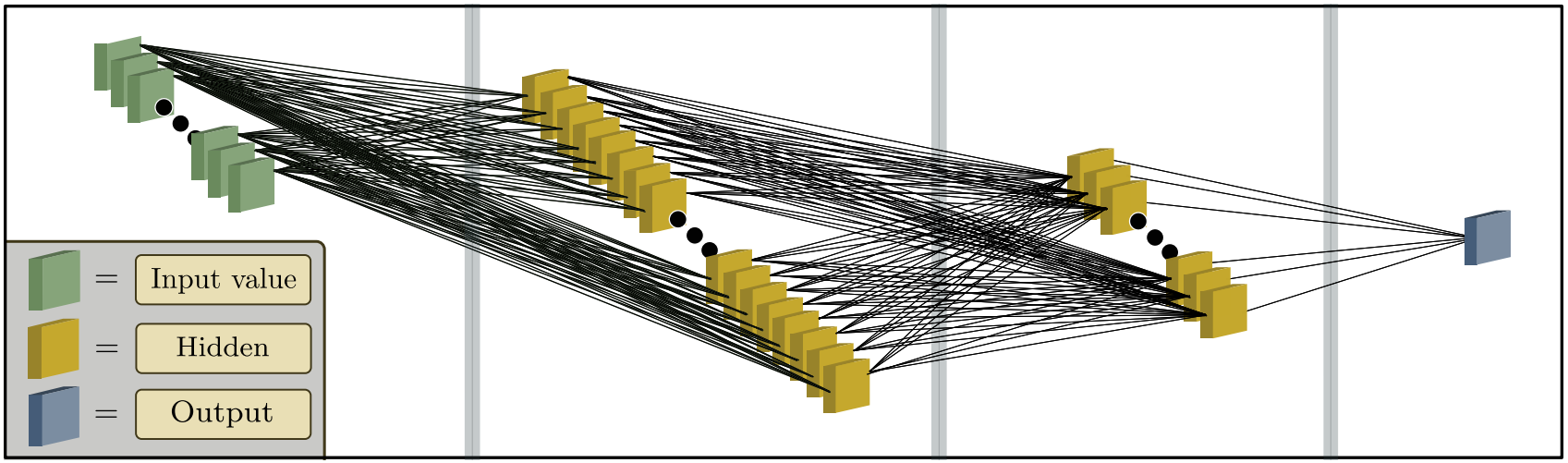
- α le taux d'apprentissage

Objectif

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

Plan

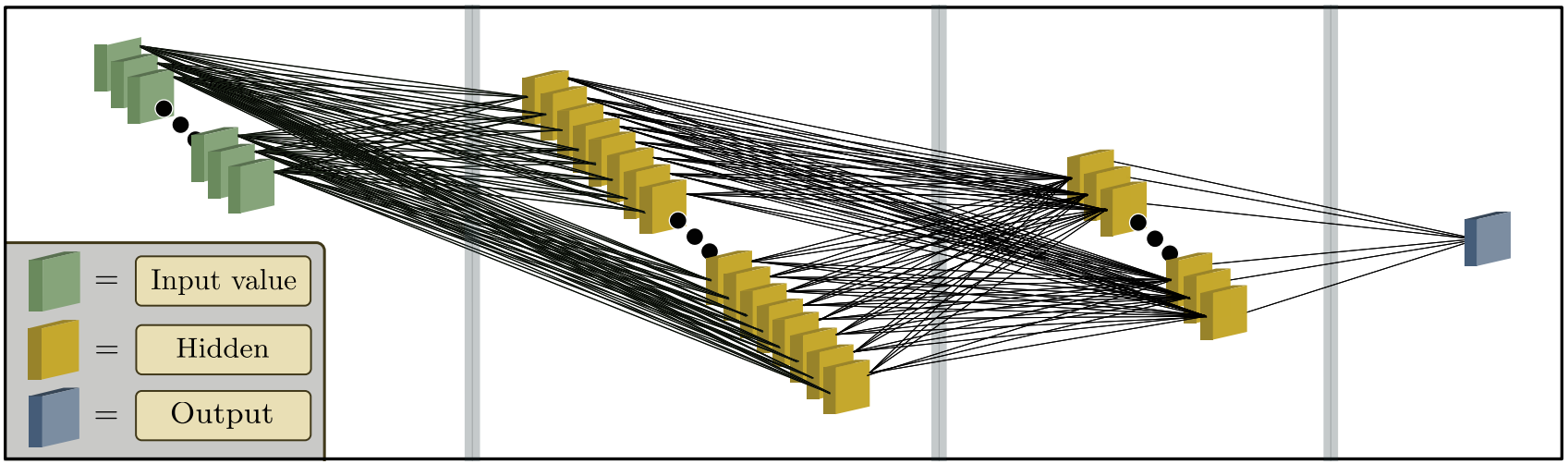
Perceptron Multi-couches : réseaux de neurones



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

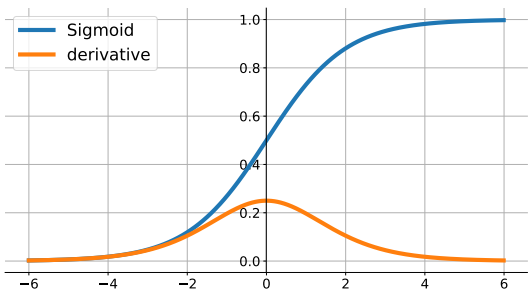
Pourquoi a-t-on besoin de fonctions d'activations ?

- 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**

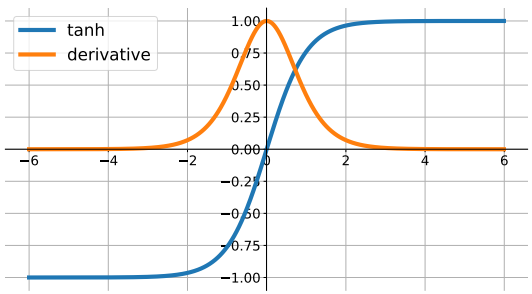
Réseau de neurones profond : fonctions d'activation

Quelques fonctions d'activation

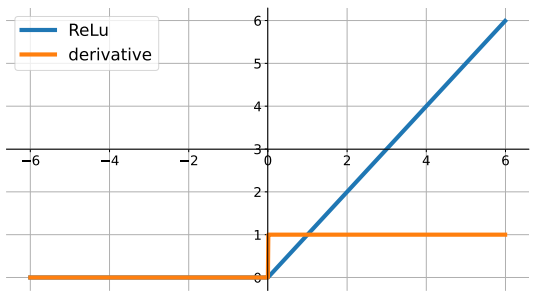
- **Sigmoïde :** utilisée pour retourner une probabilité (bornée $\in [0, 1]$)
- **Softmax :** en sortie d'une couche retourne un vecteur de probabilité dont la somme vaut 1
- **ReLu :** pour *rectified linear unit*, elle est généralement utilisée pour les couches cachées
- Et bien d'autres...



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 : $\frac{\partial C}{\partial a_j^L} = \sum_{j=0}^{n_{l+1}-1} w_{jk}^{(l+1)} \sigma'(z_j^{(l+1)}) \frac{\partial C}{\partial a_j^{l+1}}$

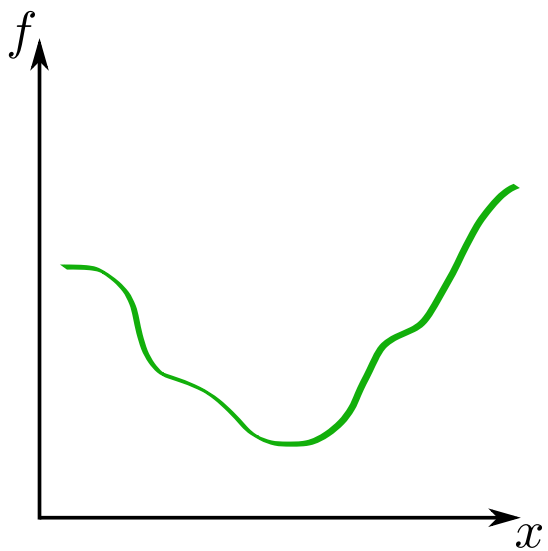
La backprogation : avantages / inconvénients

- 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 ($\texttt{Loss} : f$)

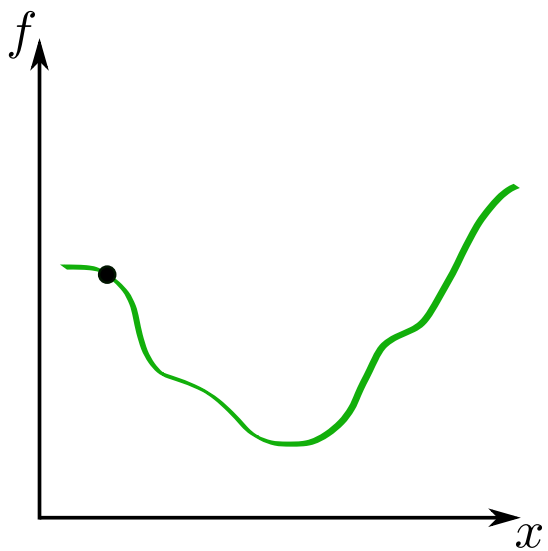


Espace de solutions $\mathcal{X}$

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)

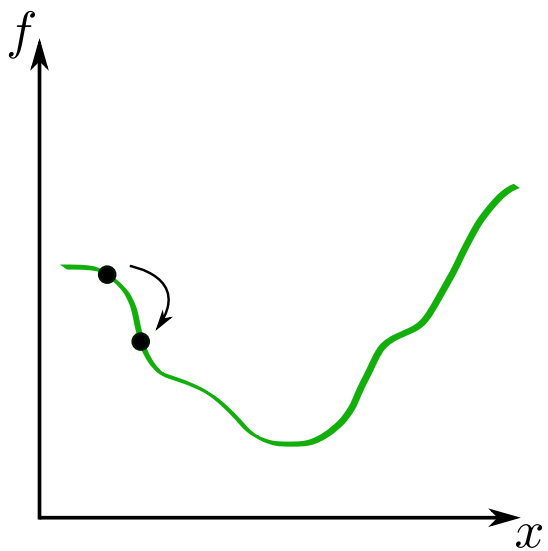


Itération 1

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 ($\texttt{Loss} : f$)

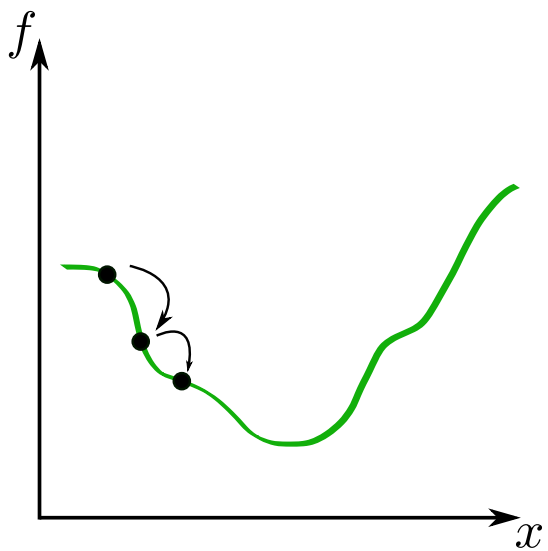


Itération 2

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)

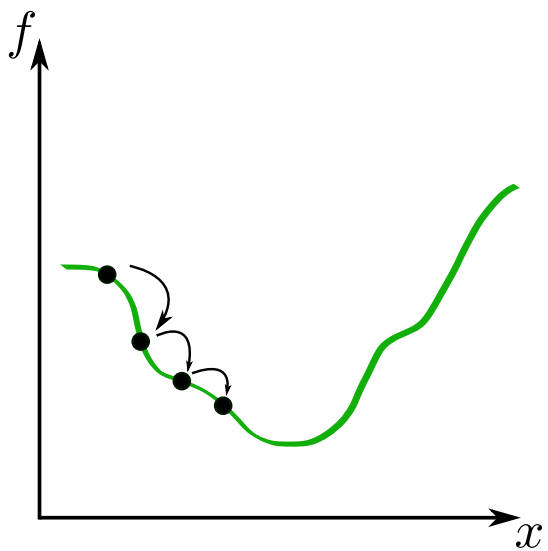


Itération 3

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 (`Loss` : f)

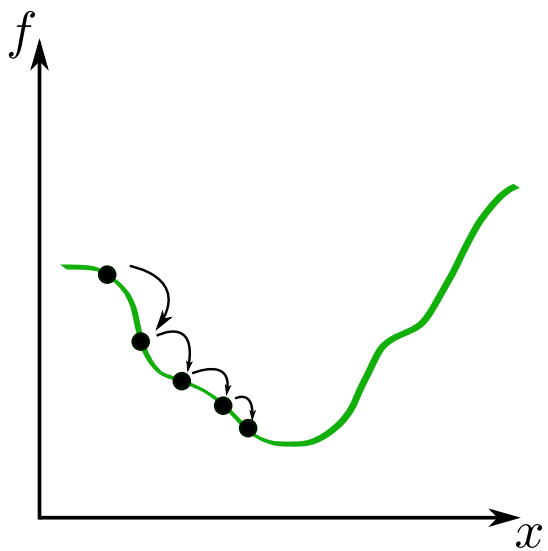


Itération 4

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)

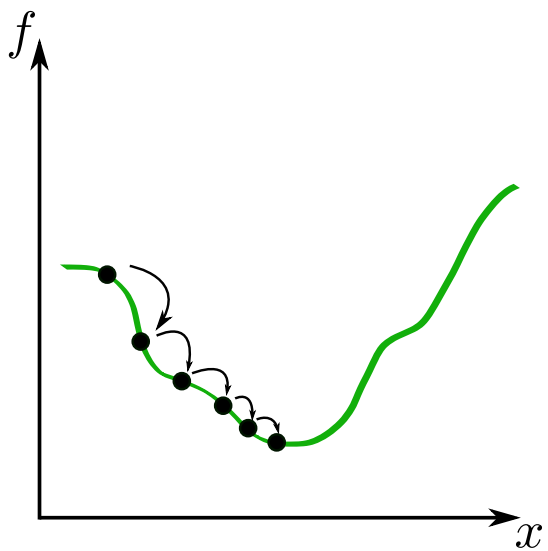


Itération 5

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)

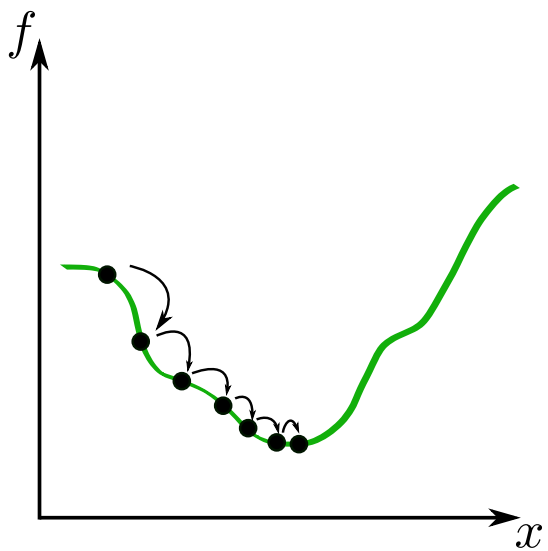


Itération 6

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 (`Loss` : f)

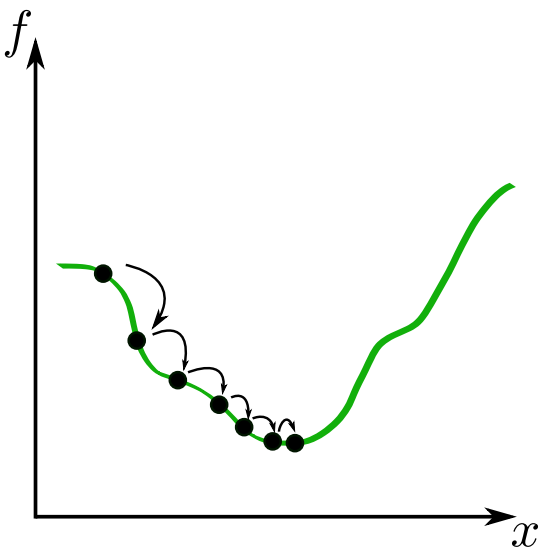


Itération 7

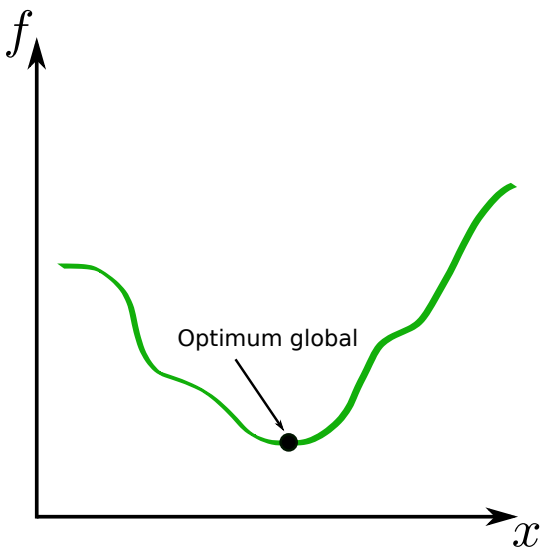
Réseau de neurones profond : descente de gradient

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

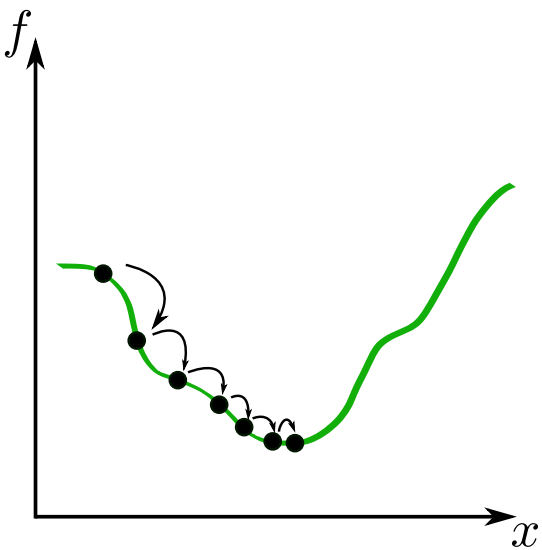


Cas convexe

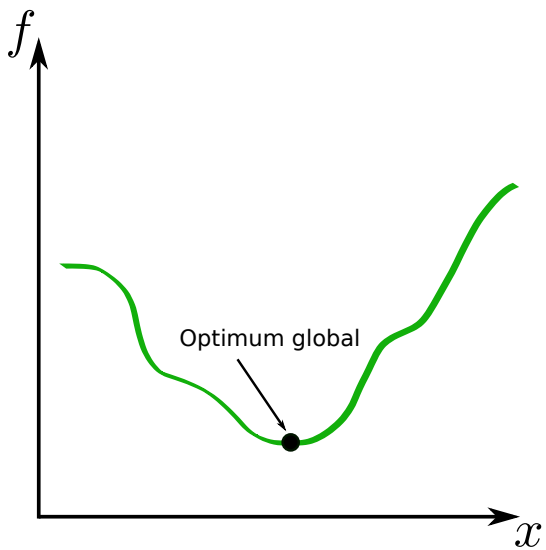
Réseau de neurones profond : descente de gradient

Objectif de la descente de gradient

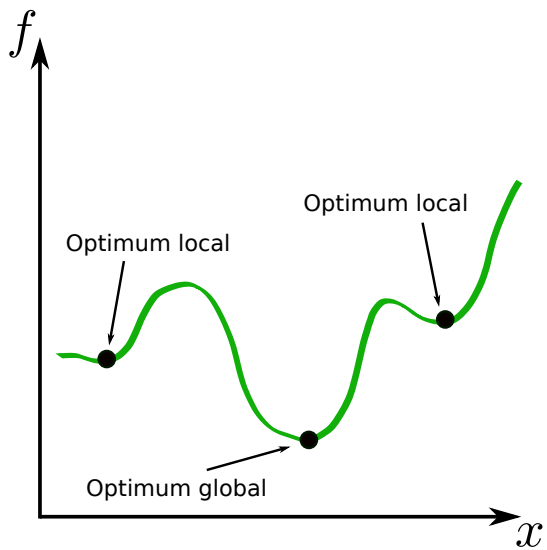
- Itérativement améliorer une solution x trouvée dans un espace de solutions
- Objectif : trouver la meilleure solution
- Utilisation du gradient pour minimiser l'erreur (`Loss` : f)



Itération 7



Cas convexe



Cas non convexe

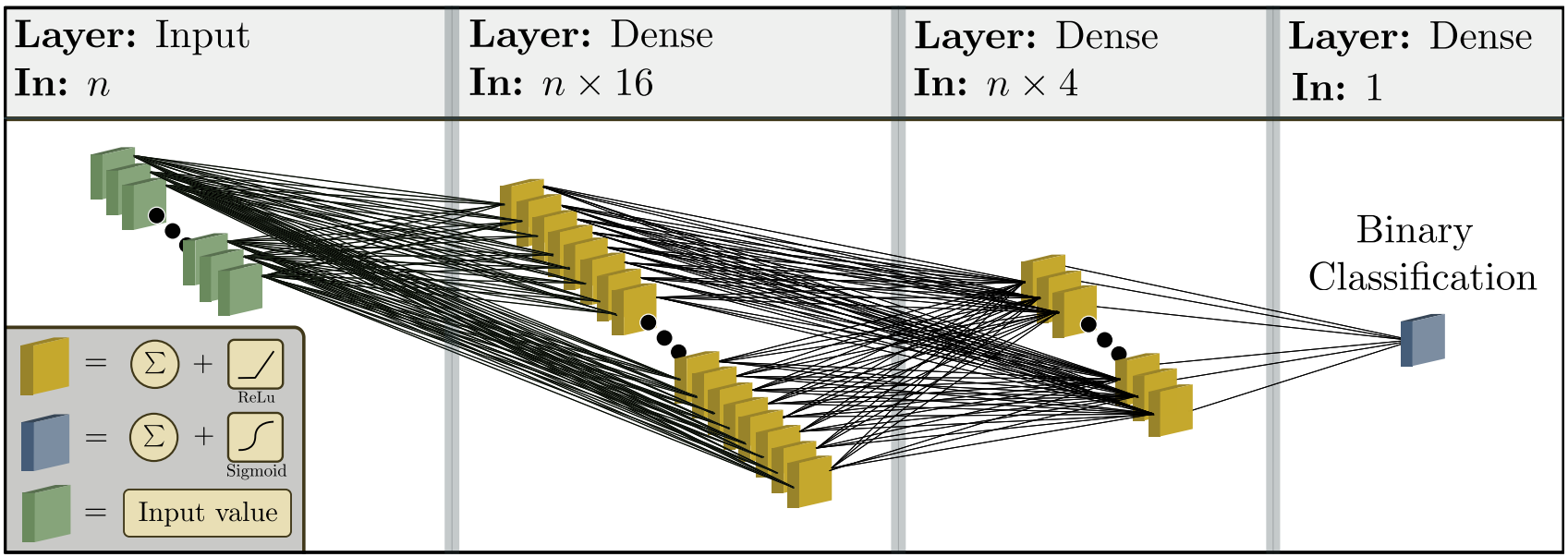
TP1 : Réseaux de neurones et descente de gradient

Plan

Réseau de neurones profond : classification binaire

Exemple d'un réseau de neurones : classification binaire

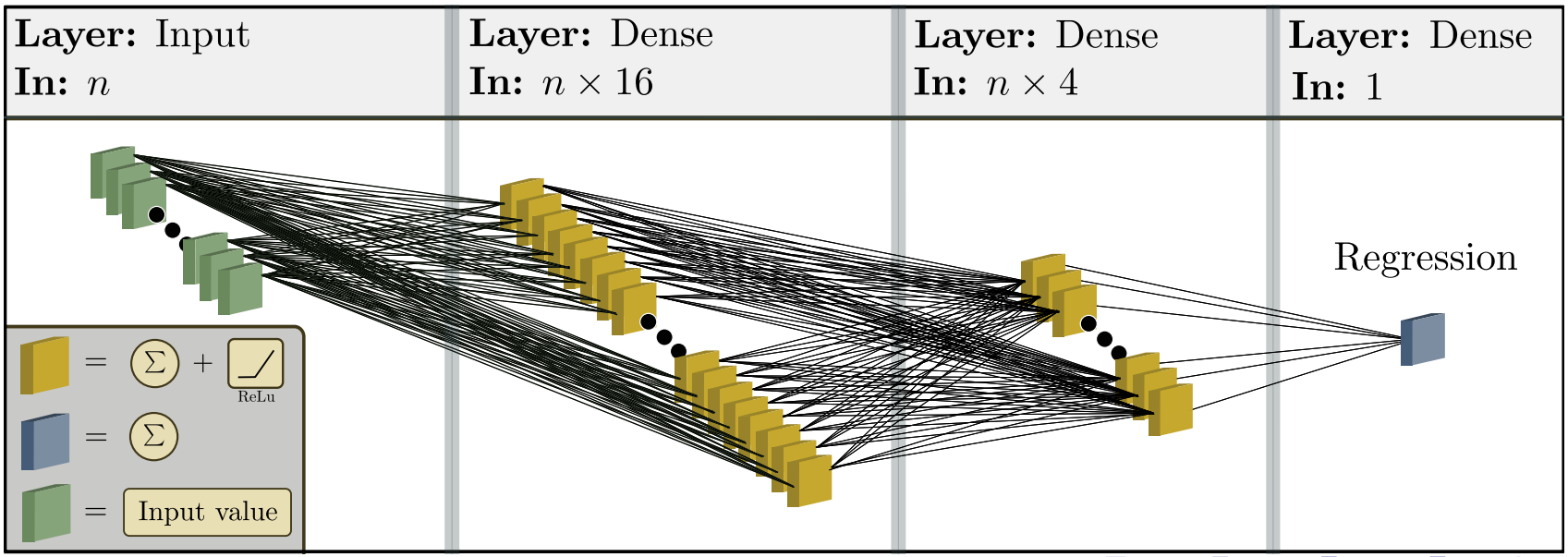
- **Entrée :** un vecteur x de taille n
- **Couches cachées :** 2 complètement connectées
- **Sortie :** une probabilité $\in [0, 1]$ (fonction d'activation Sigmoide)
- **Loss :** entropie croisée binaire, ...



Réseau de neurones profond : régression

Exemple d'un réseau de neurones : régression

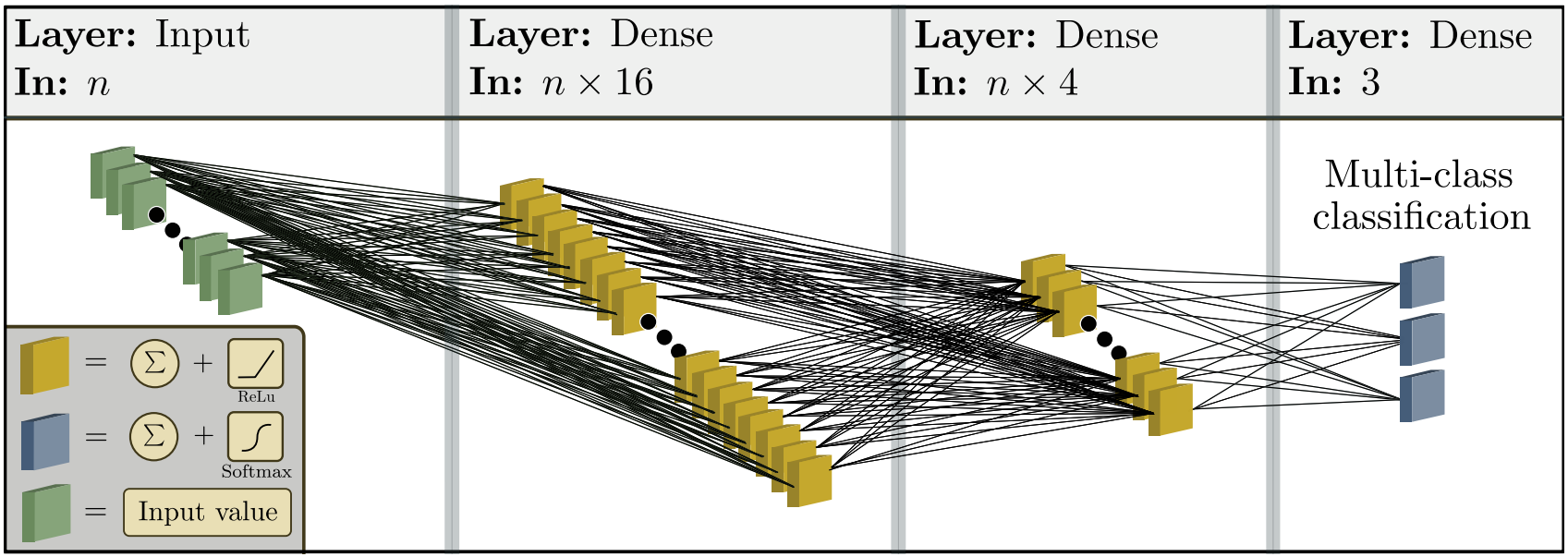
- **Entrée :** un vecteur x de taille n
- **Couches cachées :** 2 complètement connectées
- **Sortie :** un scalaire (pas de fonction d'activation)
- **Loss :** Erreur absolue moyenne (MAE), erreur quadratique moyenne (MSE), ...



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.

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

TP2 : Réseaux de neurones profonds

Plan

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

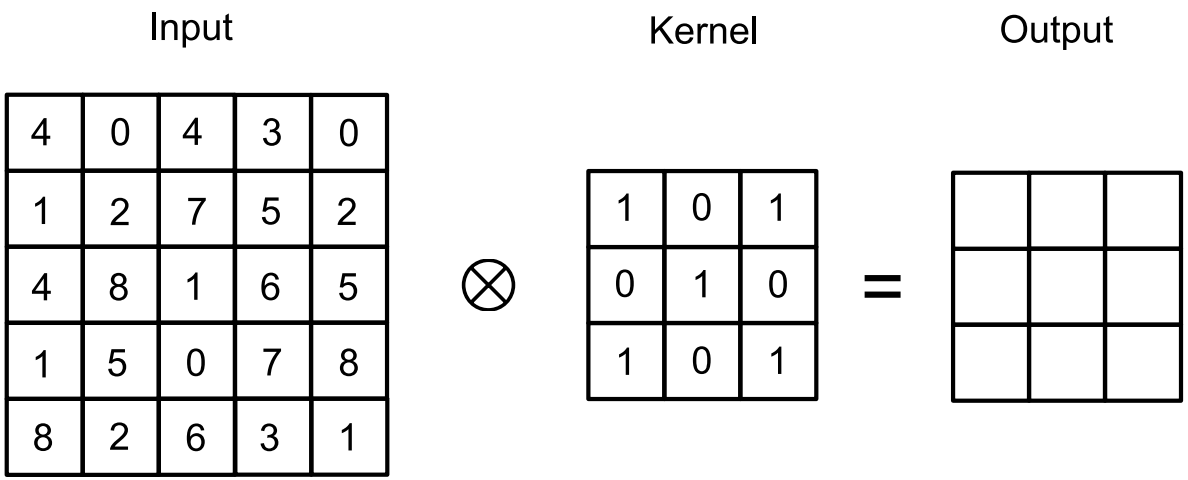
1	0	1
0	1	0
1	0	1

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

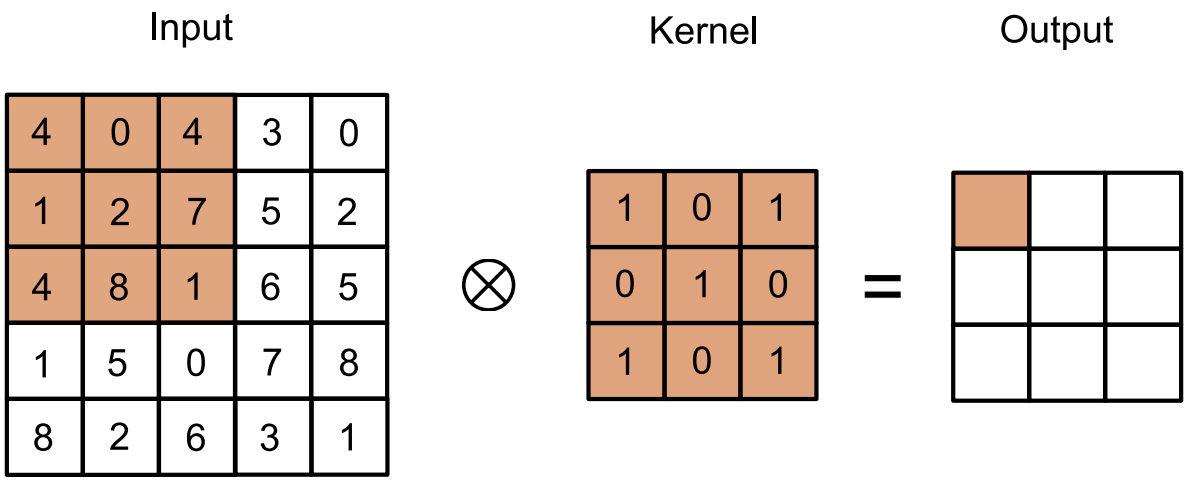


1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

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



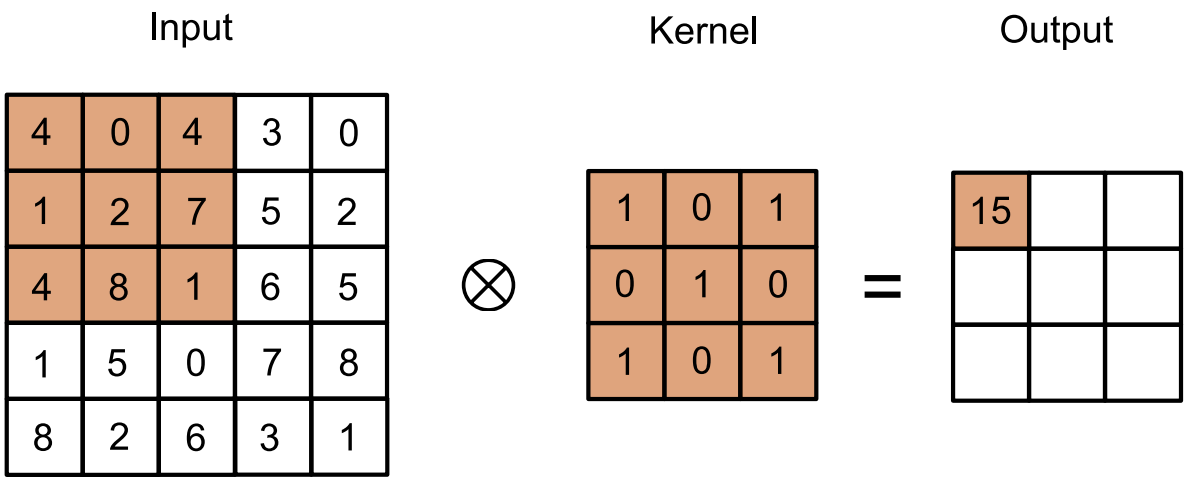
$$y_{00} = (4 \times 1 + 0 \times 0 + 4 \times 1 + 1 \times 0 + 2 \times 1 + 7 \times 0 + 4 \times 1 + 8 \times 0 + 1 \times 1)$$
$$= 15$$

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

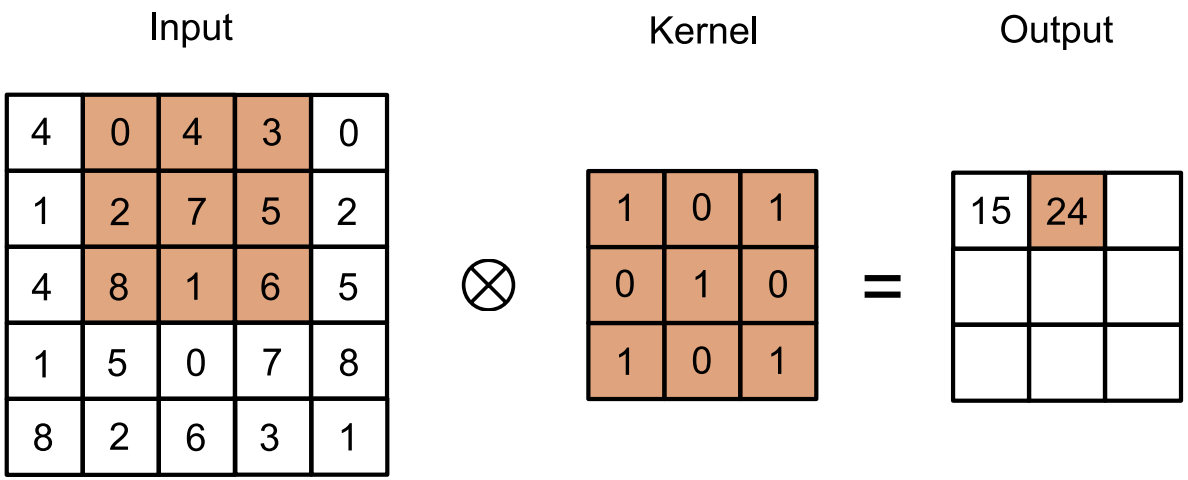


1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

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

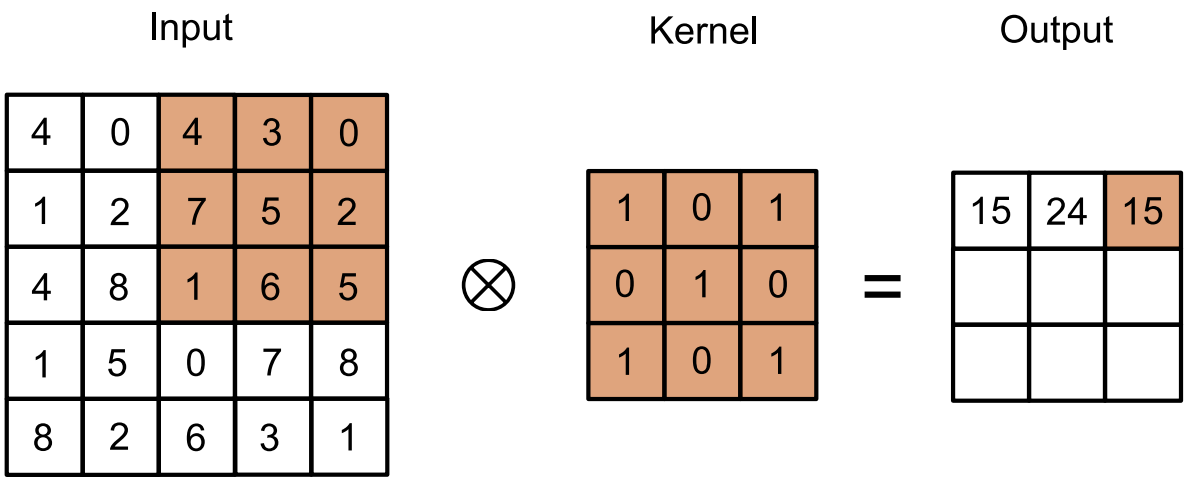


1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

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

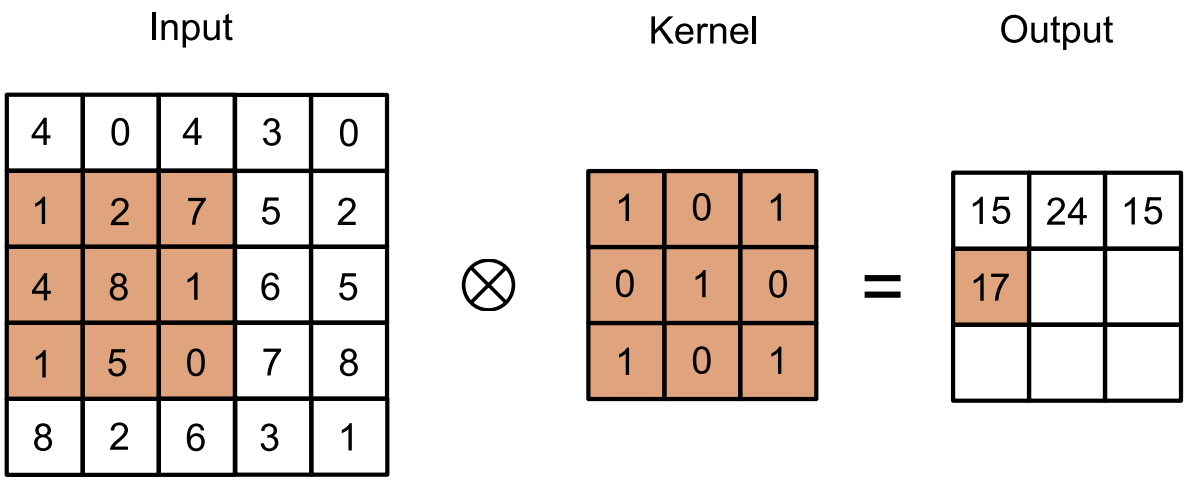


1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

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

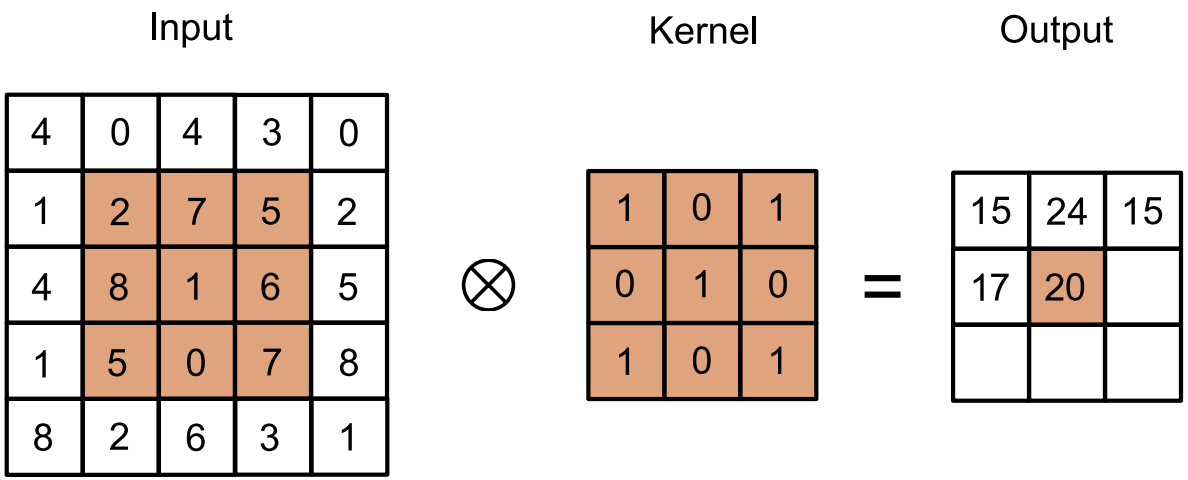


1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

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

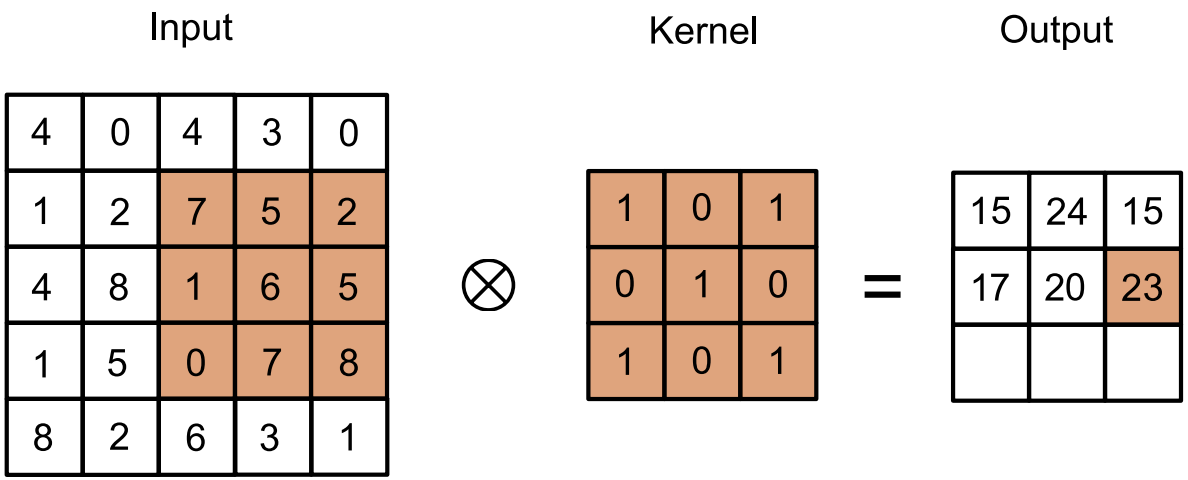


1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

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

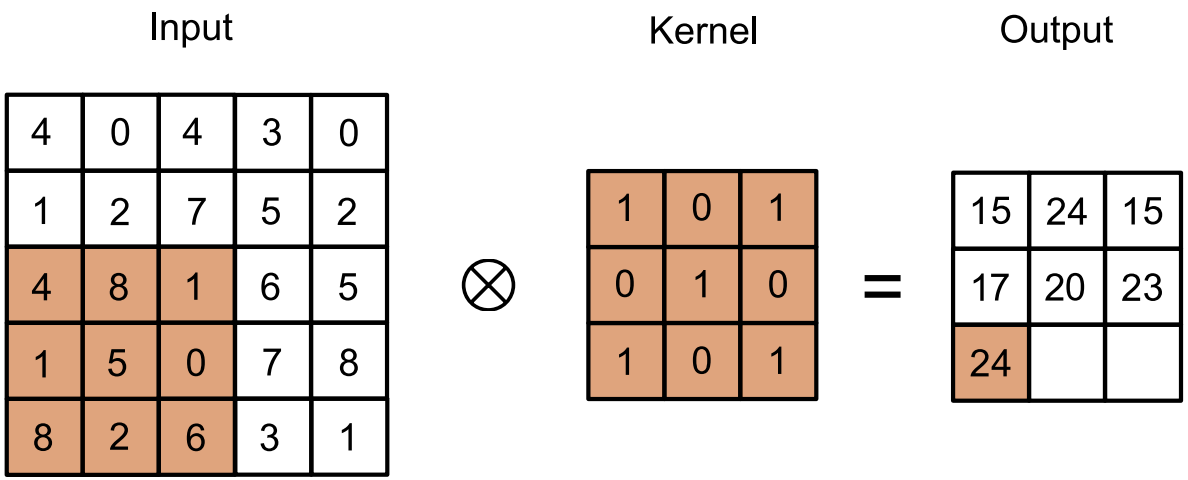


1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

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

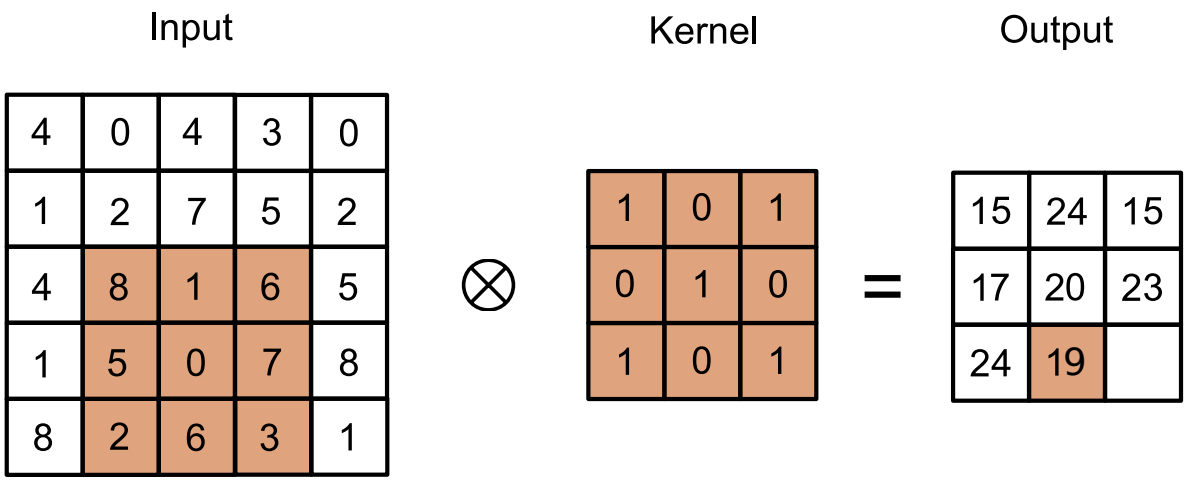


1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

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

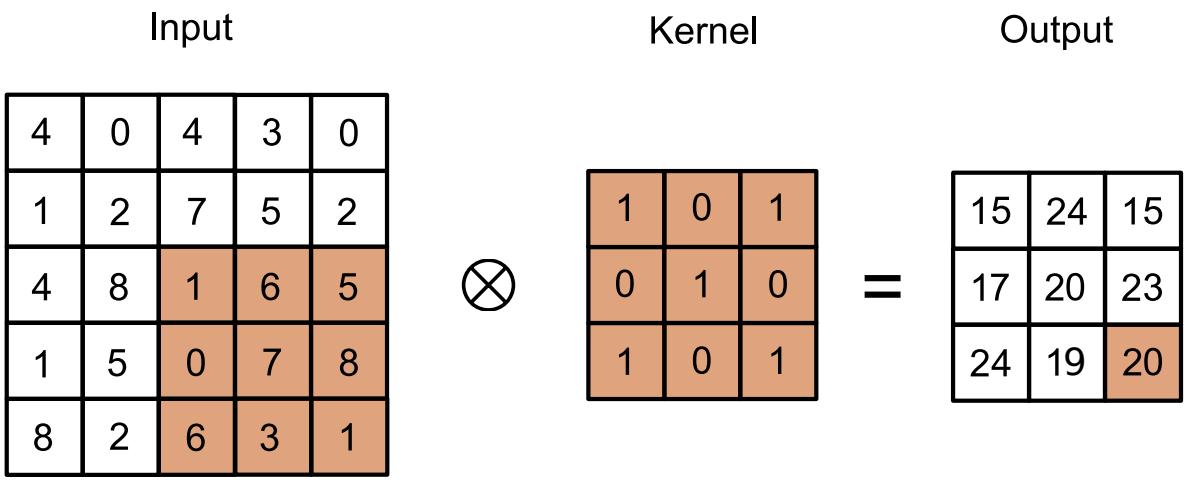


1. Le symbole $\otimes$ correspond au produit d'Hadamard (produit tensoriel)

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)

Plan

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

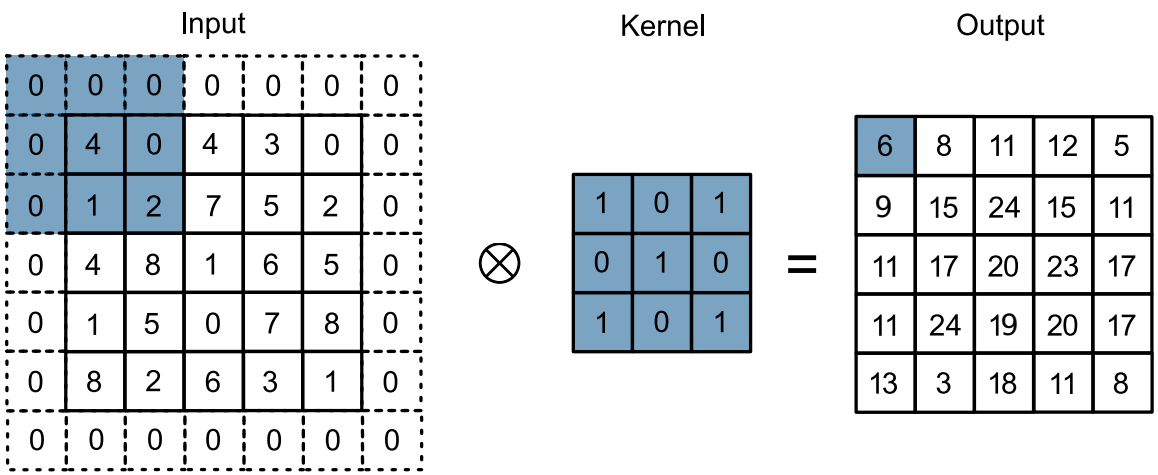
- Couche d'activation (toujours) : application de la fonction d'activation sur chaque nouveau « pixel » obtenu du plan
- **Pooling** (mise en commun) : réduction rapide et filtrage des données (aucun paramètre)
- **Dropout** : possible également mais rarement utilisé pour la convolution (perte de cohérence 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

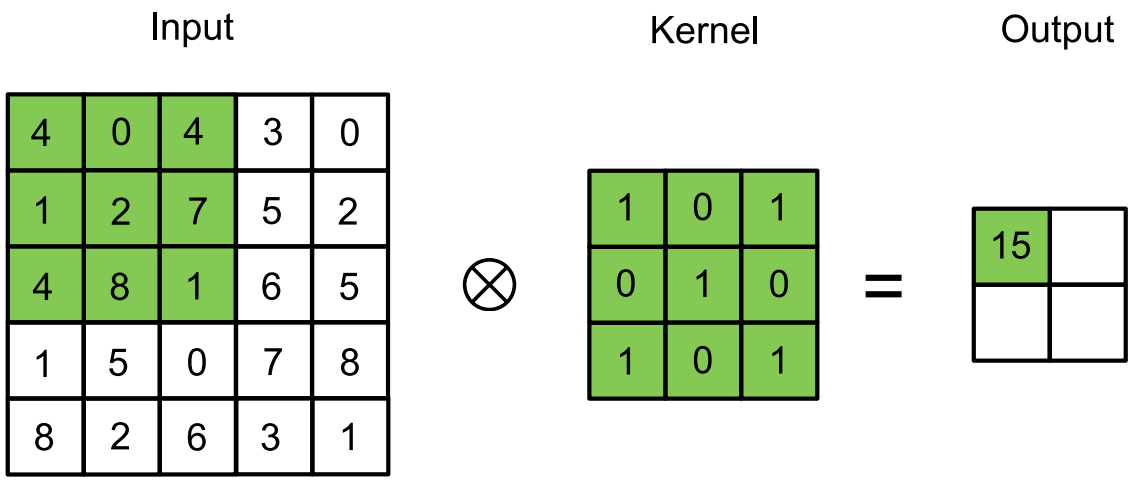


Couche convolutive avec un padding $= 1$

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

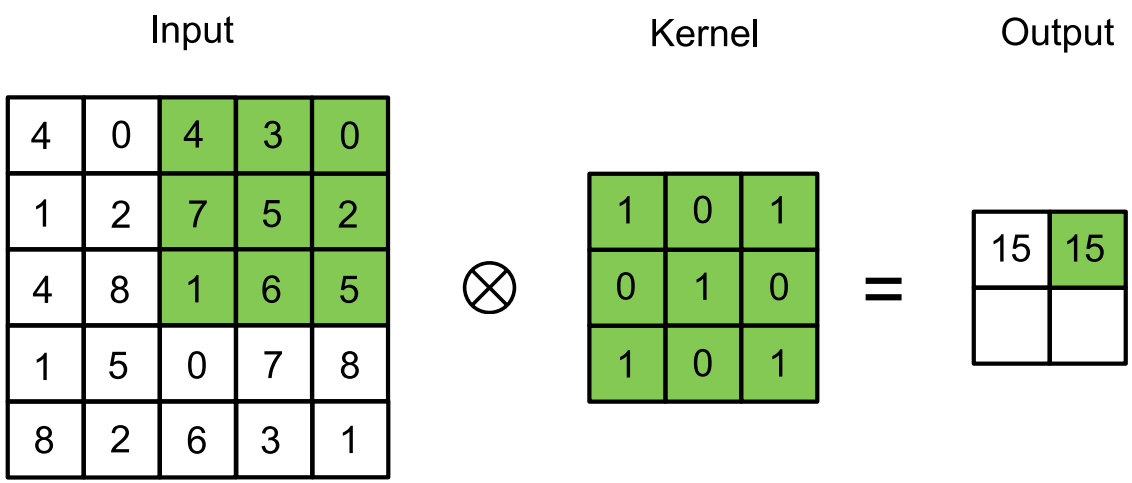


Convolution (stride $= 2$ et padding $= 0$)

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

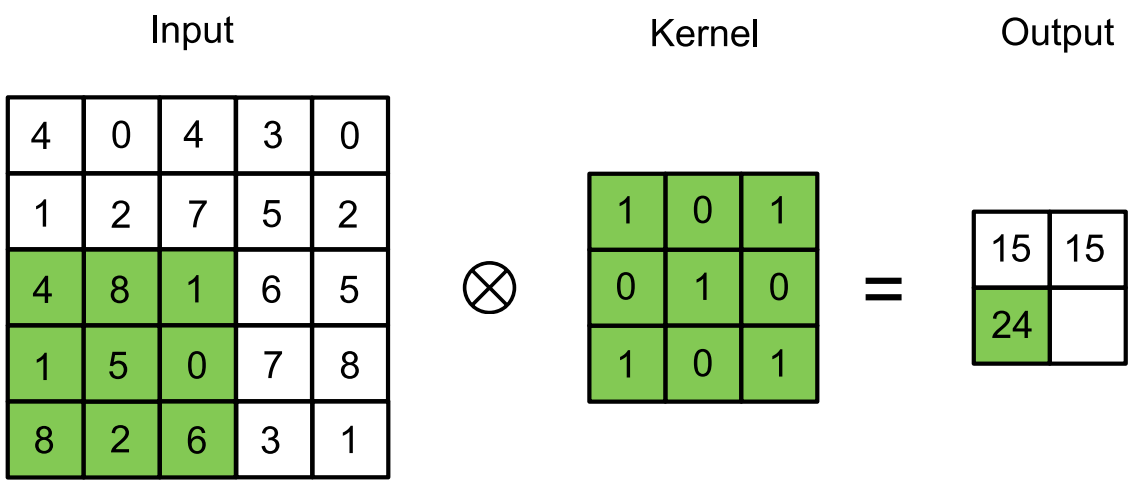


Convolution (stride $= 2$ et padding $= 0$)

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

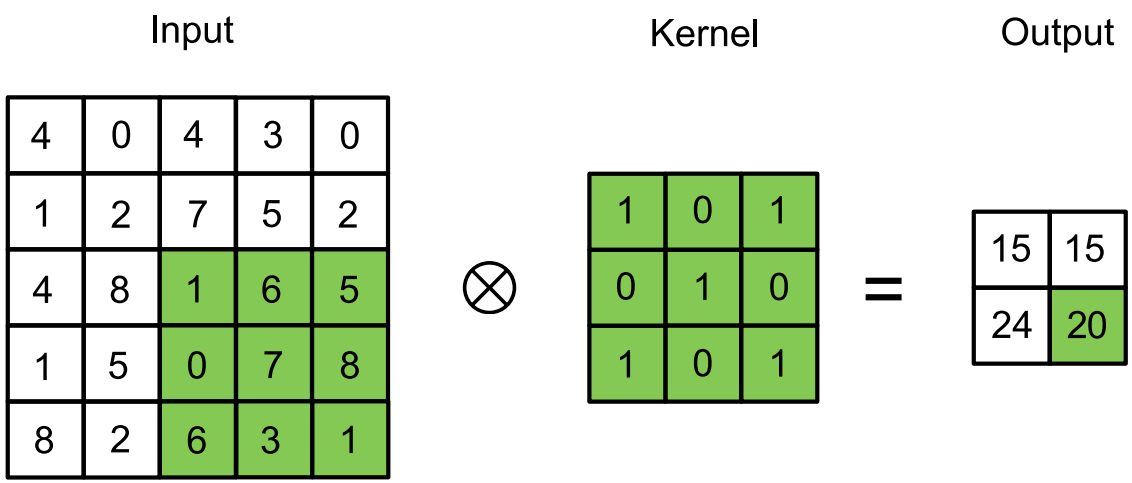


Convolution (stride $= 2$ et padding $= 0$)

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

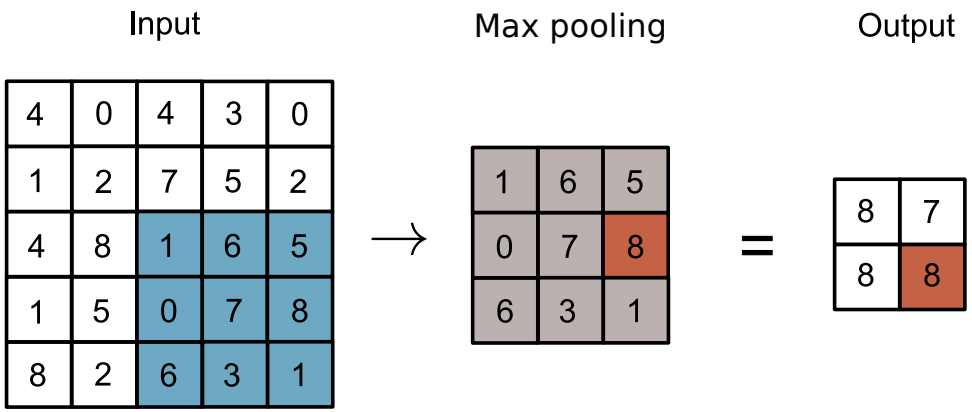


Convolution (stride $= 2$ et padding $= 0$)

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 :

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

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

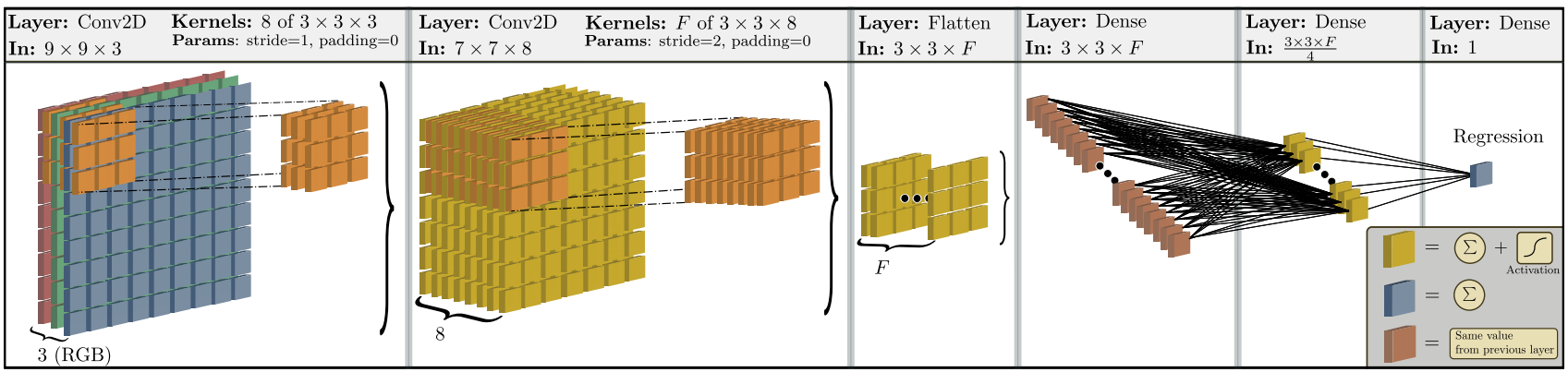
Avec :

- 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)

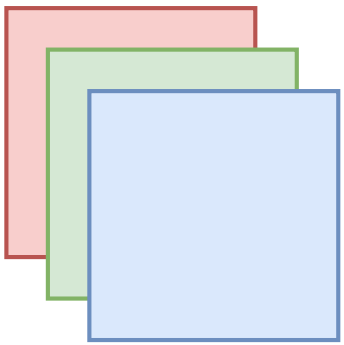


TP3 : Réseaux de neurones convolutifs

Plan

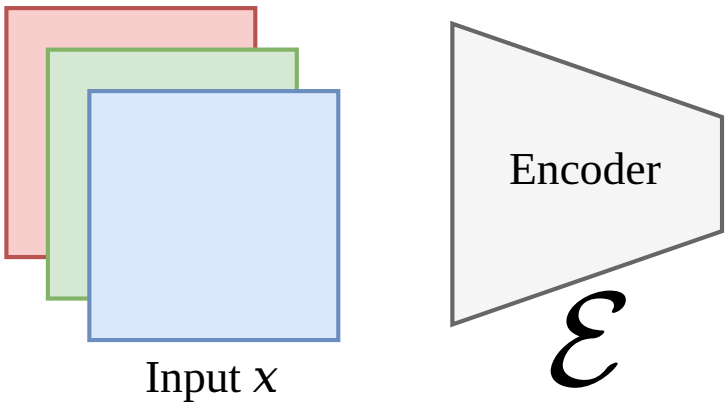
Plan

AutoEncoders : définition et applications

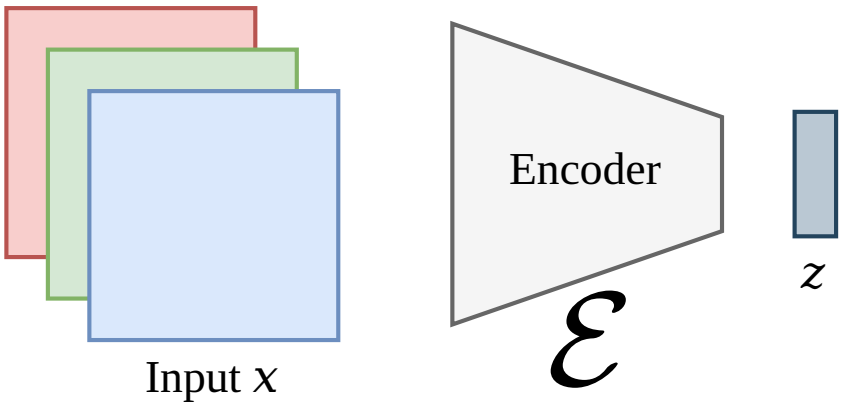


Input x

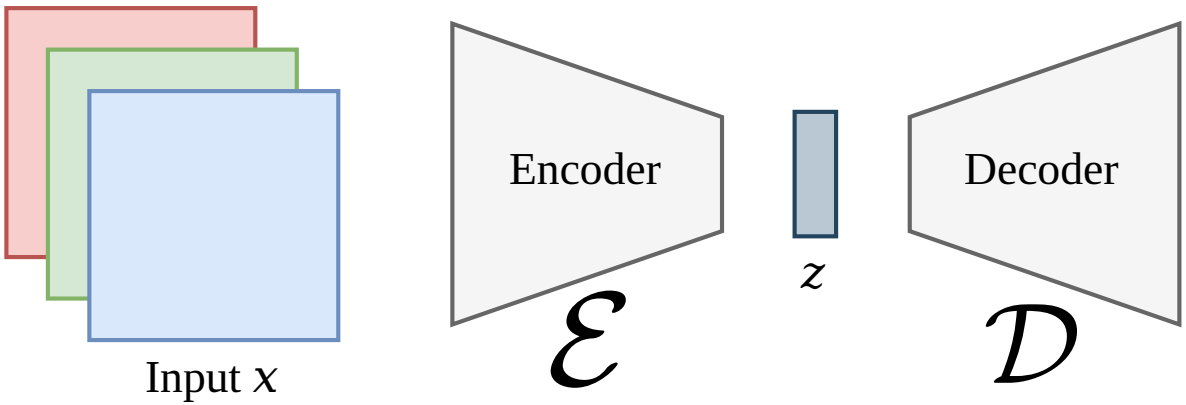
AutoEncoders : définition et applications



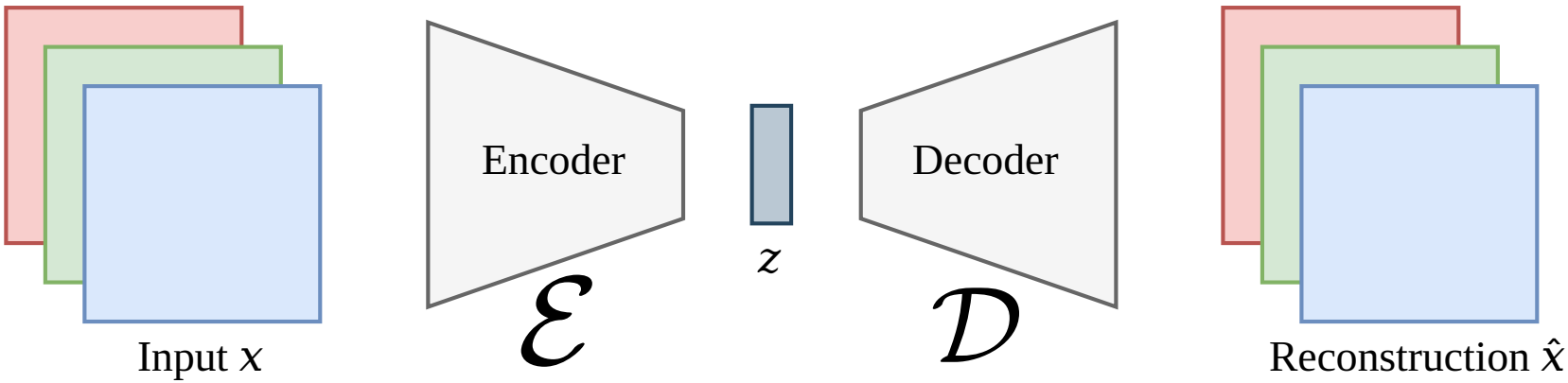
AutoEncoders : définition et applications



AutoEncoders : définition et applications



AutoEncoders : définition et applications



AutoEncoders : définition et applications

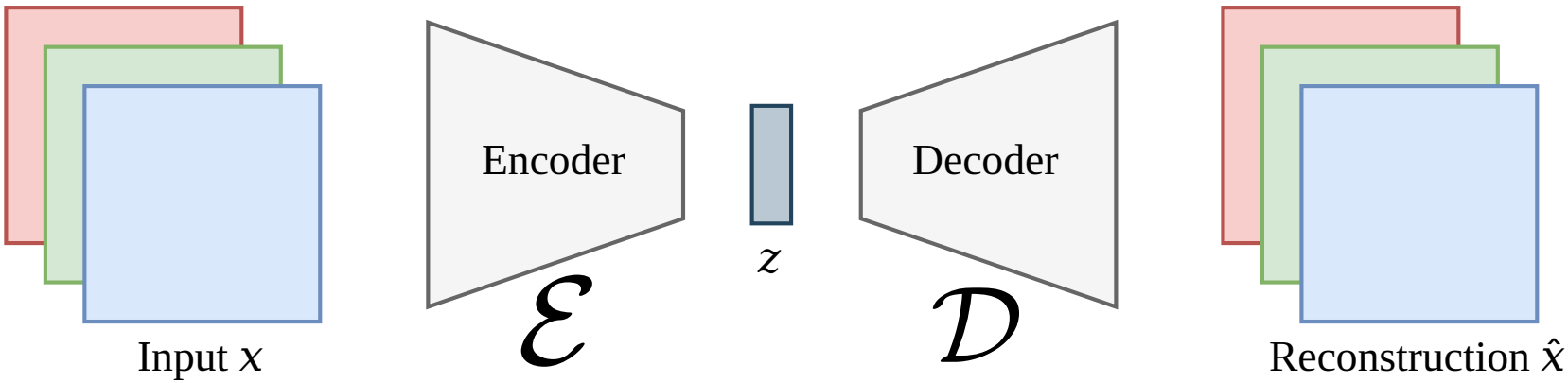


Figure – Source : https ://lightning.ai

Objectifs de l'AutoEncoders

- $\mathcal{E}$ encode les données dans un **espace latent** z
- $\mathcal{D}$ reconstruit la donnée à partir d'un **espace latent** z

AutoEncoders : définition et applications

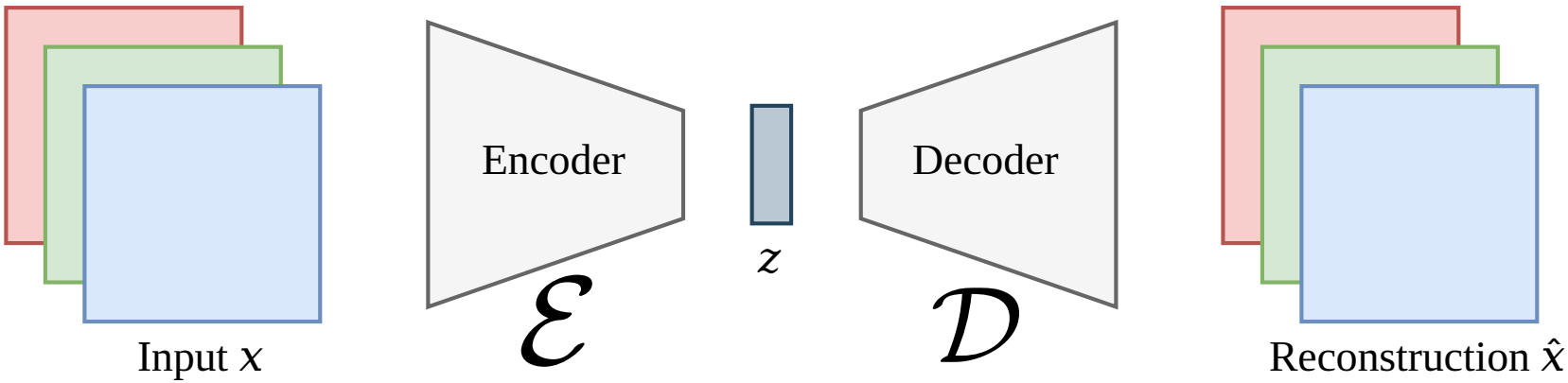


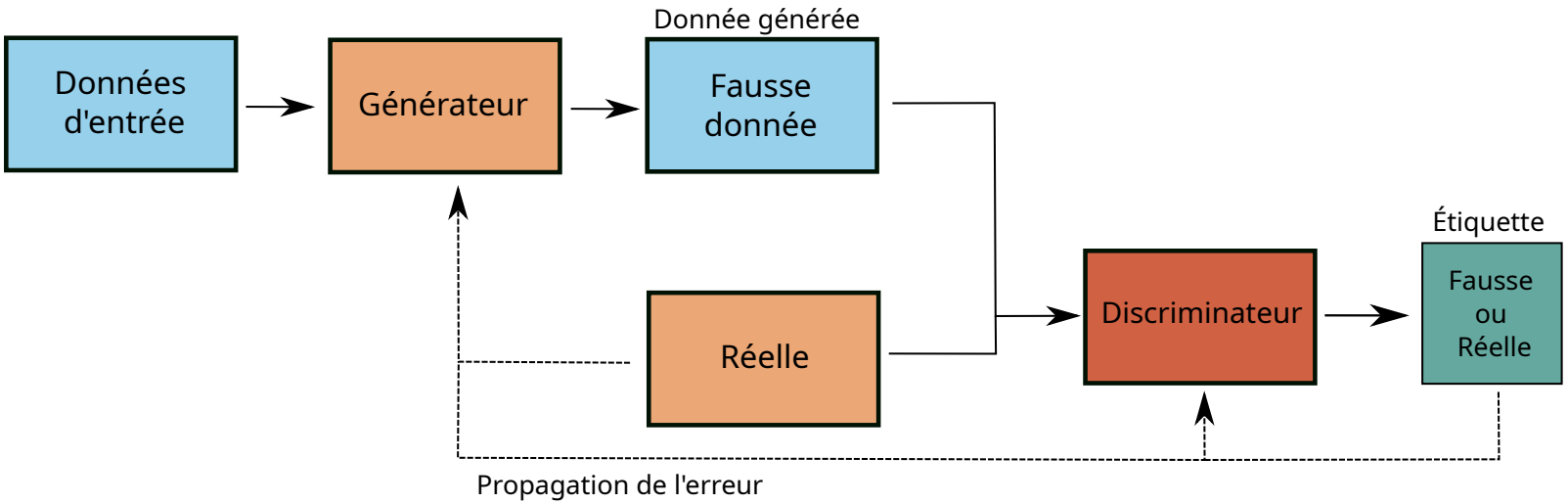
Figure – Source : https ://lightning.ai

Les applications de l'AutoEncoders

- Réduction de dimensionnalité
- Débruitage de l'information
- Détection d'anomalies (entrainés sur des données spécifique)
- Génération de données d'images (avec le Variational-AutoEncoder)

Plan

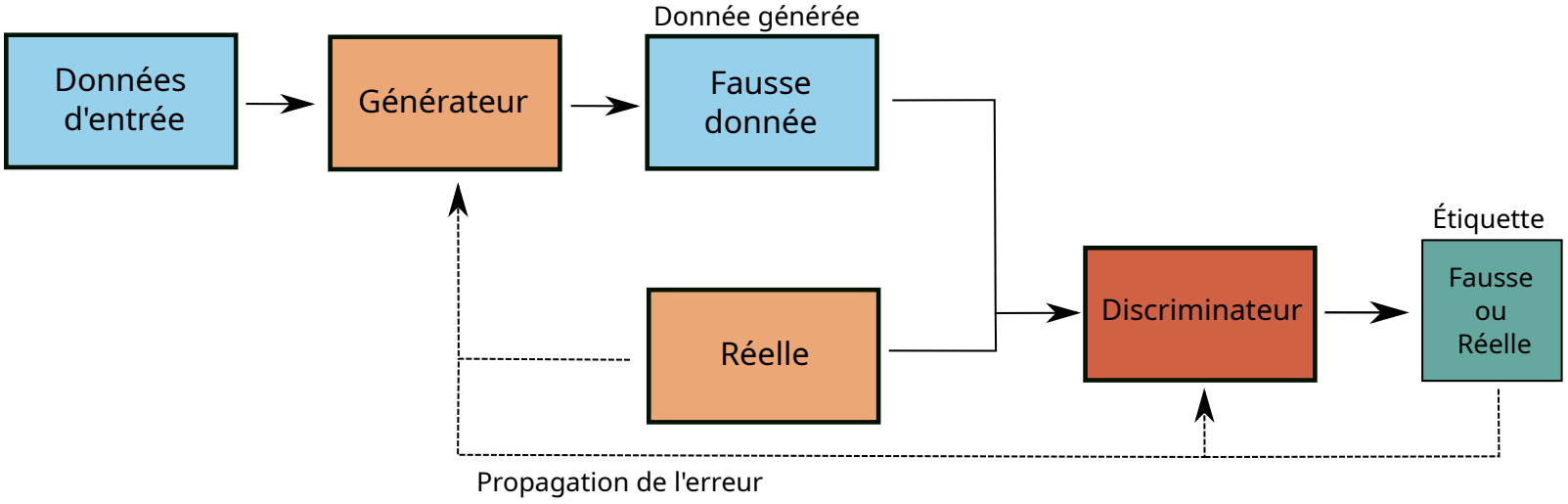
Generative Adversarial network : définition et applications



Generative Adversarial network (GAN) : objectifs

- Obtenir un générateur de données (Générateur)
- Discriminer correctement la donnée traitée (Discriminateur)

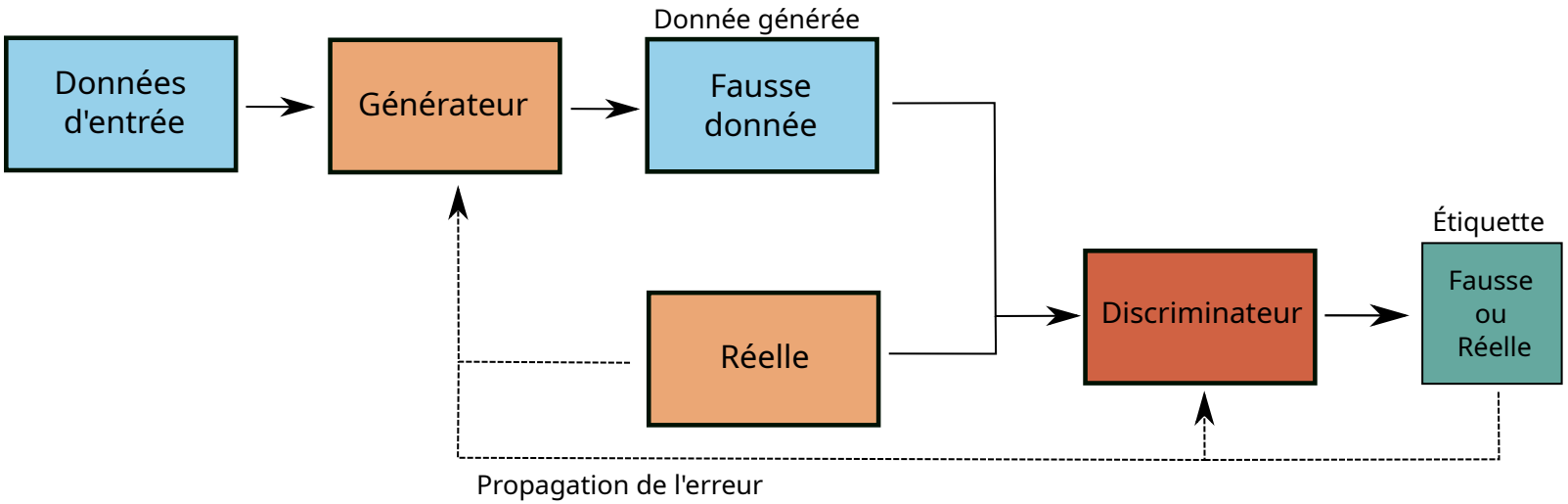
Generative Adversarial network : définition et applications



Generative Adversarial network (GAN) : fonctionnement

- Le générateur essaie d'obtenir une donnée proche de la réalité
  - le générateur part généralement d'un espace latent z
  - l'espace latent est généralement composée de donnée aléatoire
  - z peut toutefois être conditionné (donnée partiellement non aléatoire)
- Le discriminateur doit détecter si la donnée est réelle ou générée
- Le générateur apprend de son erreur pour tromper le discriminateur
  - Comment ? à partir de l'erreur du discriminateur

Generative Adversarial network : définition et applications



Generative Adversarial network (GAN) : applications

- Génération de nouvelles données
- Détection d'anomalies
- Transformer une donnée d'entrée (texte vers image par exemple)
- ...

TP4 : Les AutoEncoders