TS231 : Intelligence artificielle pour l'image

Rémi Giraud
remi.giraud@enseirb-matmeca.fr

2023-2024

2A Électronique

Slides tirés de ceux de :

- **Guillaume Bourmaud** (*ENSEIRB-MATEMCA*)
- **Michaël Clément** (*ENSEIRB-MATEMCA*)
- Aurélie Bugeau (*Univ. Bordeaux*)
- Charles Deledalle (*Brain Corp*)
- Hugo Larochelle (*Google Brain*)
- Andrew Ng (*Stanford*)
- Baptiste Pesquet (*ENSC*)

Objectifs et organisation

Organisation

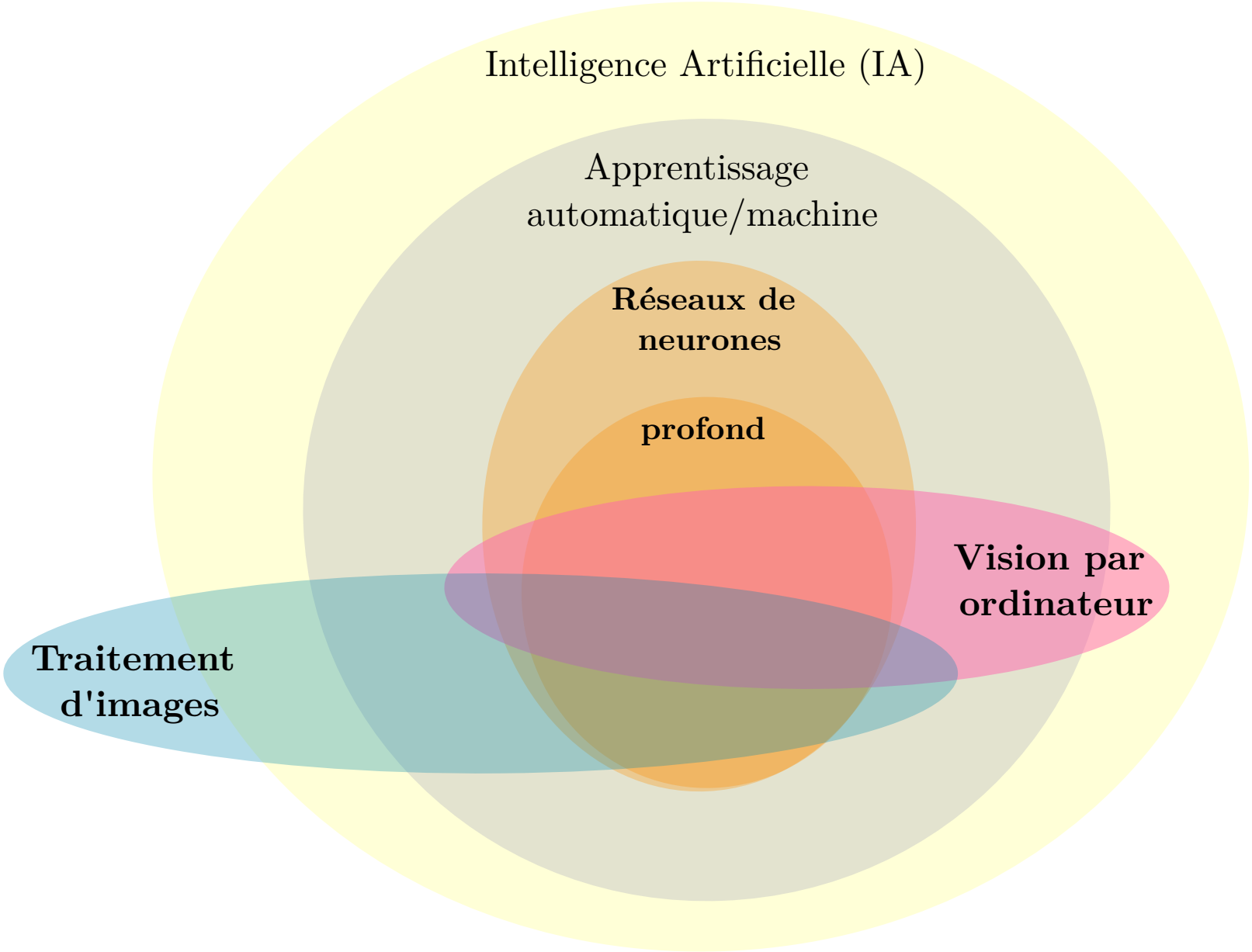
- **5h20 de cours** et **8h de TP**
- Supports de cours : https://remi-giraud.enseirb-matmeca.fr/teaching/
- **Notation** : contrôle continu (contrôle de connaissance, rendu de TP)

Objectifs

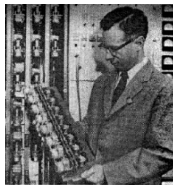
- Comprendre comment fonctionnent les **réseaux de neurones** (profonds)
- Connaître les principales **architectures** et **outils** d'apprentissage profond
- Être capable de les **implémenter**/**utiliser** pour différentes applications de **vision par ordinateur** dans un contexte d'apprentissage **supervisé**

Introduction

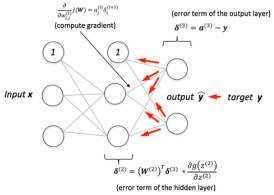
Contexte


Intelligence Artificielle (IA)
Apprentissage automatique/machine
Réseaux de neurones
profond
Vision par ordinateur
Traitement d'images

Quelques dates clés dans la chronologie de l'apprentissage profond



1958 Perceptron



1974 Backpropagation

Convolution Neural Networks for Handwritten Recognition

1998

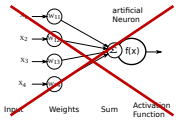
Google Brain Project on 16k Cores

2012

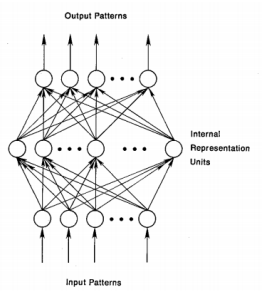
awkward silence (AI winter)

1969 Perceptrons book

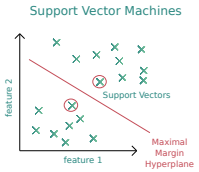
Perceptron criticized



~1980 Multilayer network



1995 SVM reigns

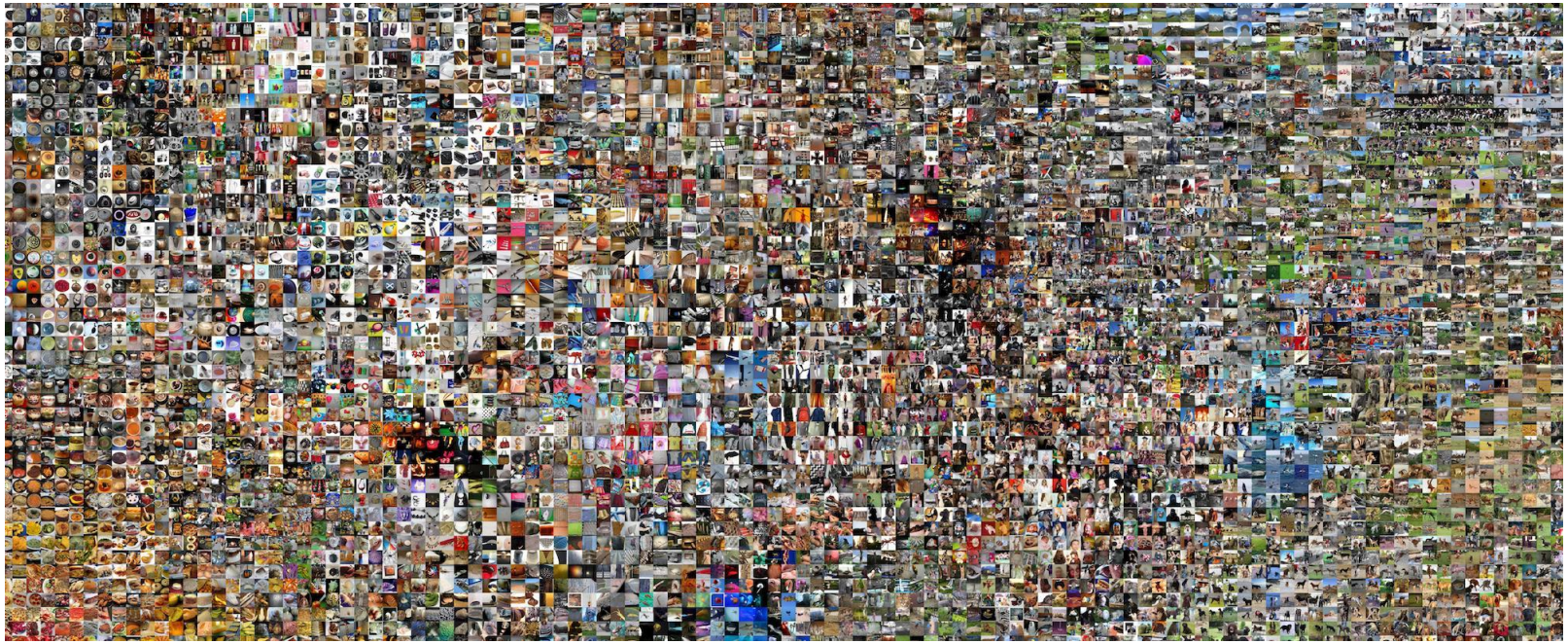


2006 Restricted Boltzmann Machine

2012 AlexNet wins ImageNet

IMAGENET

ImageNet



ImageNet:

- 12 millions d'images étiquettées,
- 22 000 classes,
- Étiquettées par *crowd-sourcing* (Amazon Mechanical Turk).

ImageNet challenge (ILSVRC)

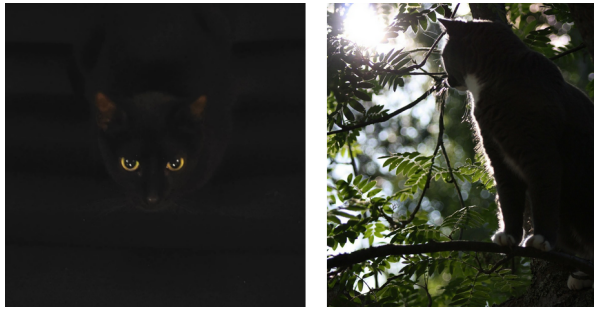
ILSVRC
- Challenge annuel depuis 2010,
- Limité à 1 000 classes,
- 1.2 million d'images de taille 256×256 pour l'entraînement,
- 50 000 images pour la validation 100 000 pour le test.

Pourquoi est-ce difficile pour l'ordinateur ?

(a) Points de vue

Pourquoi est-ce difficile pour l'ordinateur ?

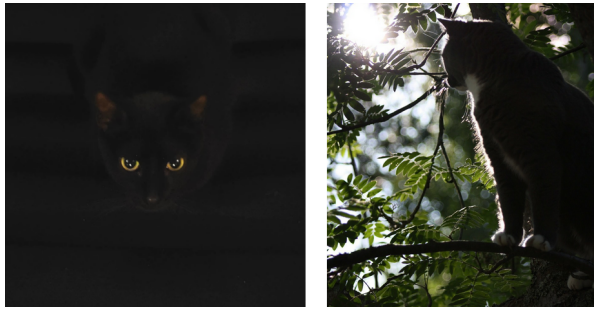
(a) Points de vue



(b) Conditions d'illumination

Pourquoi est-ce difficile pour l'ordinateur ?

(a) Points de vue

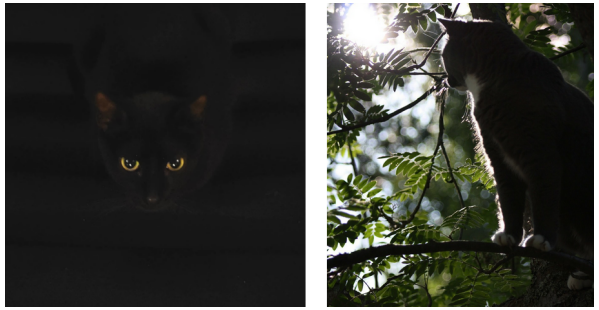


(b) Conditions d'illumination

(c) Variabilité intra-classe

Pourquoi est-ce difficile pour l'ordinateur ?

(a) Points de vue



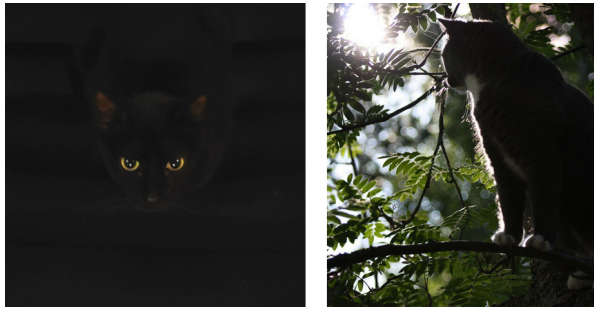
(b) Conditions d'illumination

(c) Variabilité intra-classe

(d) Occultations

Pourquoi est-ce difficile pour l'ordinateur ?

(a) Points de vue



(b) Conditions d'illumination

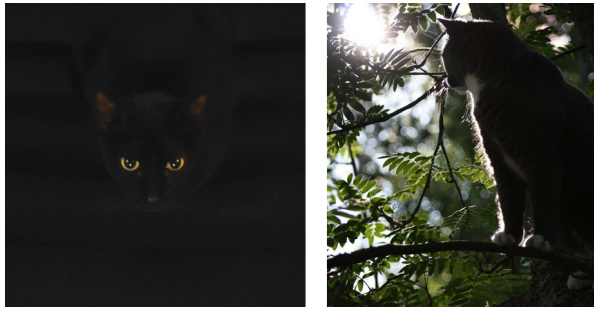
(c) Variabilité intra-classe

(d) Occultations

(e) Objets déformables

Pourquoi est-ce difficile pour l'ordinateur ?

(a) Points de vue



(b) Conditions d'illumination

(c) Variabilité intra-classe

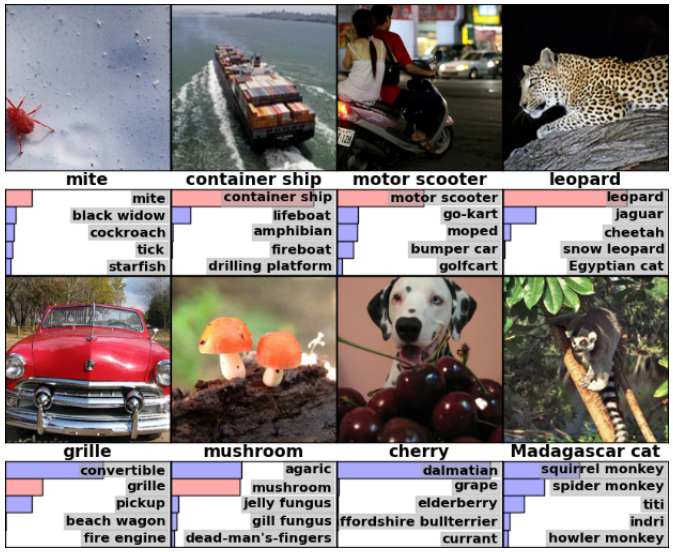
(d) Occultations

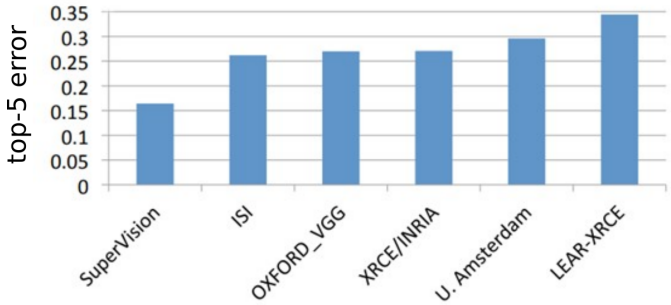
(e) Objets déformables

(f) Proximité inter-classes

AlexNet (Supervision) [1]

- Large réseaux de neurones à convolution (62.3 millions de paramètres)
- 6 jours d'apprentissage sur 2 GPUs (GTX 580 3GB)
- Utilisation des nouveaux outils d'apprentissage profond
  (calcul sur GPU, normalisation, optimisation, augmentation de données, ...)
- 1er au challenge ILSVRC2012 avec une erreur top-5 à 16.4% (2e : 26.2%).





→ Explosion de l'apprentissage profond depuis 2010 dans **tous les domaines**

1. [Krizhevsky et al., 2012] - ImageNet Classification with Deep Convolutional Neural Networks. In NeurIPS.

De nouvelles applications et de nouveaux enjeux/risques...

Automatisation de tâches/métiers

Perte d'emploi / Adaptation de la société ?

Mme Tang Yu, PDG du chinois NetDragon Websoft et de ses 6000 employés, est le premier robot à être nommé à la tête d'une société. Disponible H24, elle ne touche aucun salaire. *NetDragon Websoft*

De nouvelles applications et de nouveaux enjeux/risques...

Automatisation de tâches/métiers

Perte d'emploi / Adaptation de la société ?

Mme Tang Yu, PDG du chinois NetDragon Websoft et de ses 6000 employés, est le premier robot à être nommé à la tête d'une société. Disponible H24, elle ne touche aucun salaire. *NetDragon Websoft*

Diagnostic médical

Biais de population ?

De nouvelles applications et de nouveaux enjeux/risques...

Automatisation de tâches/métiers

Perte d'emploi / Adaptation de la société ?

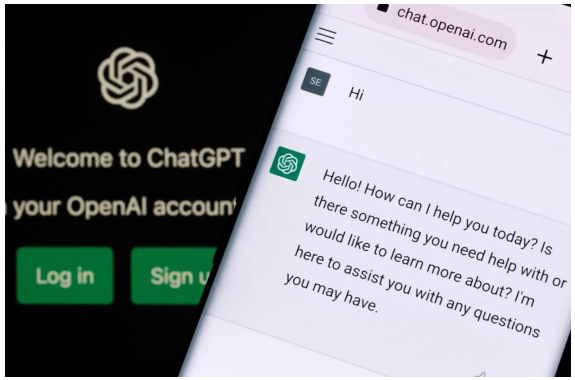
Mme Tang Yu, PDG du chinois NetDragon Websoft et de ses 6000 employés, est le premier robot à être nommé à la tête d'une société. Disponible H24, elle ne touche aucun salaire. *NetDragon Websoft*

Diagnostic médical

Biais de population ?

Bots conversationnels

Vérification des informations / Triche ?



Interface de ChatGPT (OpenAI)

De nouvelles applications et de nouveaux enjeux/risques...

Automatisation de tâches/métiers

Perte d'emploi / Adaptation de la société ?

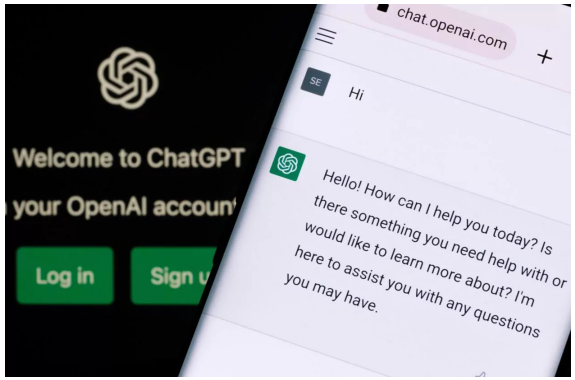
Mme Tang Yu, PDG du chinois NetDragon Websoft et de ses 6000 employés, est le premier robot à être nommé à la tête d'une société. Disponible H24, elle ne touche aucun salaire. *NetDragon Websoft*

Diagnostic médical

Biais de population ?

Bots conversationnels

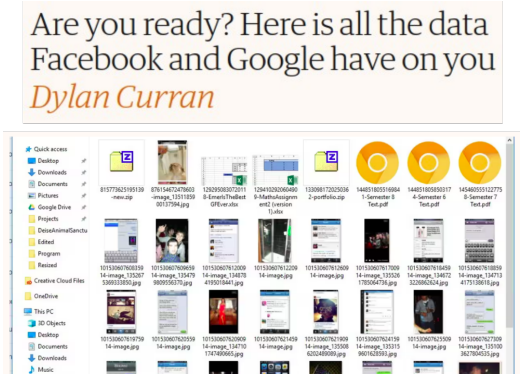
Vérification des informations / Triche ?



Interface de ChatGPT (OpenAI)

Big Data

Utilisation des données privées ?

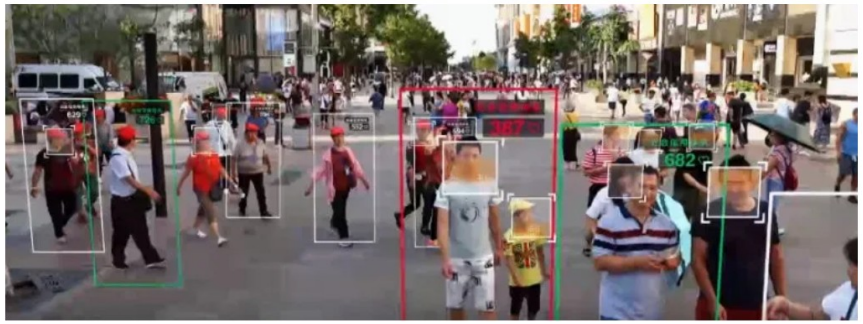


[Lien : Données privées possédées par Google]

De nouvelles applications et de nouveaux enjeux/risques...

Vidéo surveillance

Outil de contrôle / Abus ?

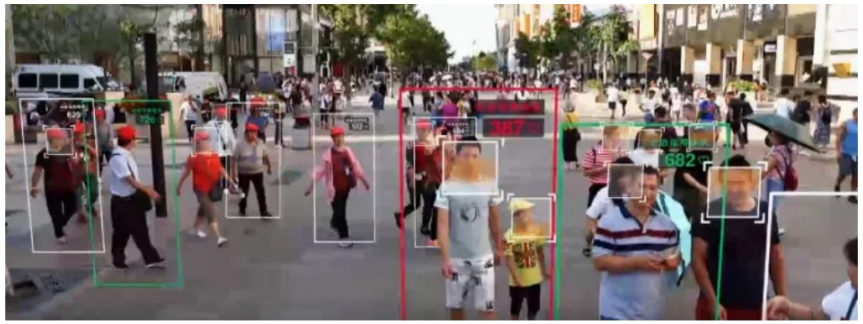


The Chinese state wants to control its citizens via a system of social scoring that punishes behavior it doesn't approve of. Image Credit: Telecoms

De nouvelles applications et de nouveaux enjeux/risques...

Vidéo surveillance

Outil de contrôle / Abus ?



The Chinese state wants to control its citizens via a system of social scoring that punishes behavior it doesn't approve of. Image Credit: Telecoms

Navigation autonome

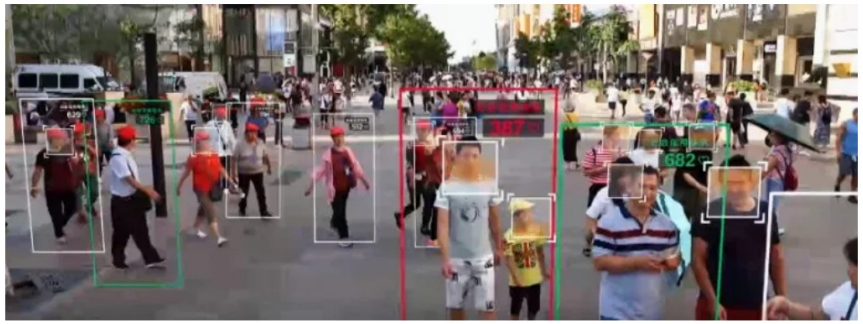
Responsabilité en cas d'accident ?

Tesla said autopilot was activated during a fatal Model X crash last week in California.

De nouvelles applications et de nouveaux enjeux/risques...

Vidéo surveillance

Outil de contrôle / Abus ?



The Chinese state wants to control its citizens via a system of social scoring that punishes behavior it doesn't approve of. Image Credit: Telecoms

Navigation autonome

Responsabilité en cas d'accident ?

Tesla said autopilot was activated during a fatal Model X crash last week in California.

Génération automatique de contenus (vidéo, image, son)

Preuve de l'authenticité ?

Vidéo de Vladimir Poutine interviewé par son double DeepFake

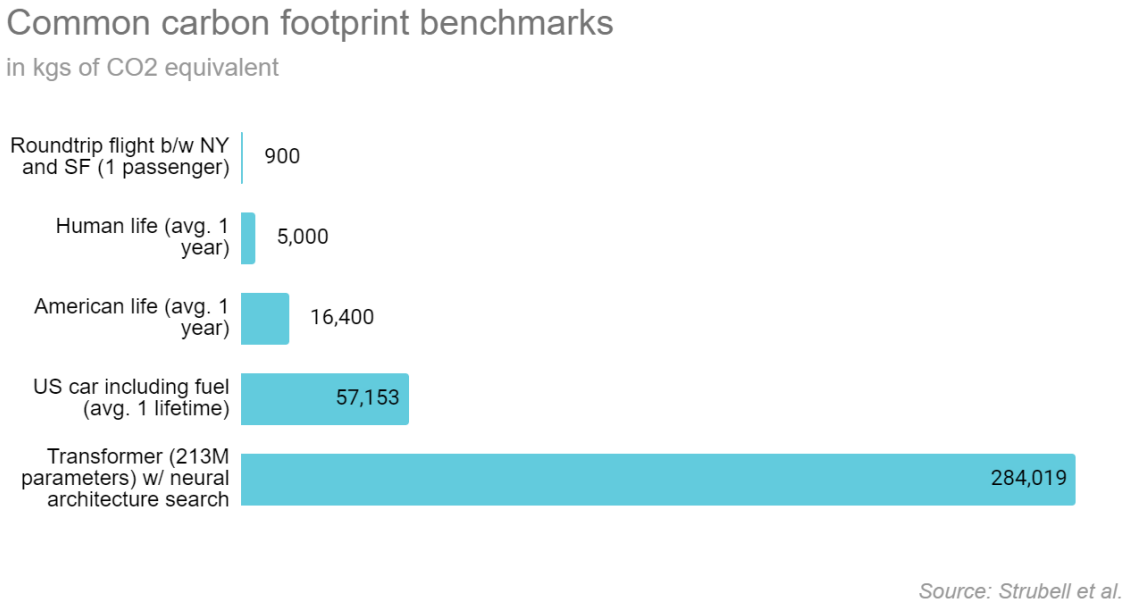
Graphistes ?

Image générée automatiquement par Midjourney

Propriété intellectuelle ?

Générique de One Piece chanté par Johnny Hallyday

Entraîner des modèles, ça prend du temps et ça consomme



Quelques outils :

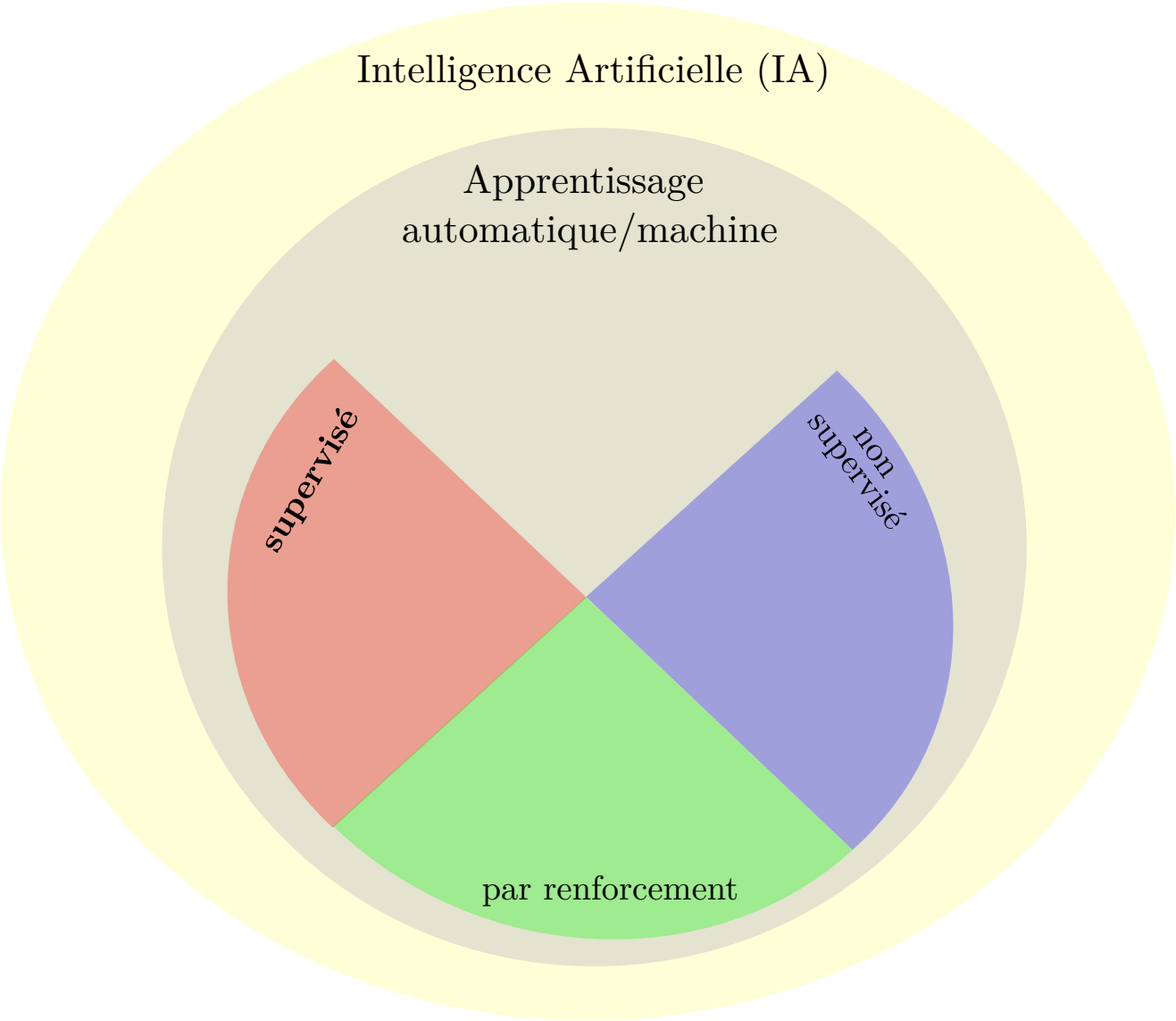
- Green Algorithms. How green are your computations?
  `https://calculator.green-algorithms.org/`
- carbontracker
  `https://pypi.org/project/carbontracker/`
- ...

Plan du cours

1. Introduction
2. Apprentissage supervisé
3. Approches paramétriques
4. Réseaux de neurones
5. Apprentissage des paramètres
6. Réseaux de neurones à convolution (CNN)
7. Réseaux de neurones profonds
8. Techniques et outils

Apprentissage supervisé

Différents types d'apprentissage



Différents types d'apprentissage

Apprentissage automatique

Apprentissage supervisé

apprentissage du modèle avec des données étiquettées

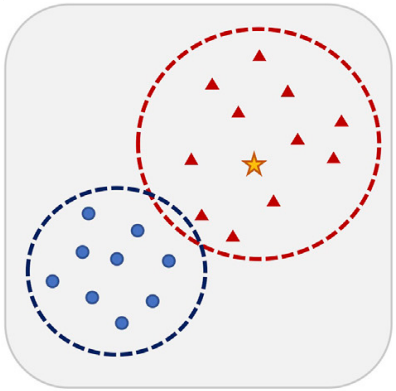
Apprentissage non supervisé

apprentissage du modèle sans données étiquettées

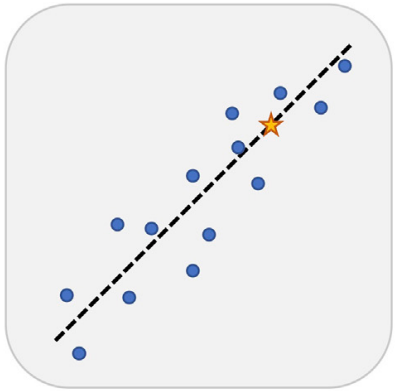
Apprentissage par renforcement

le modèle effectue des actions dans son environnement et reçoit un état mis à jour ainsi qu'un retour

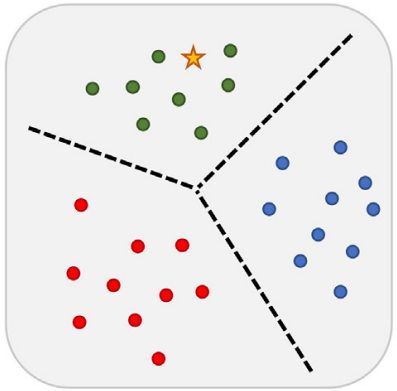
Classification

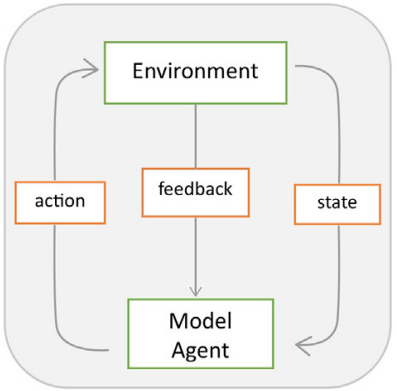


Régression



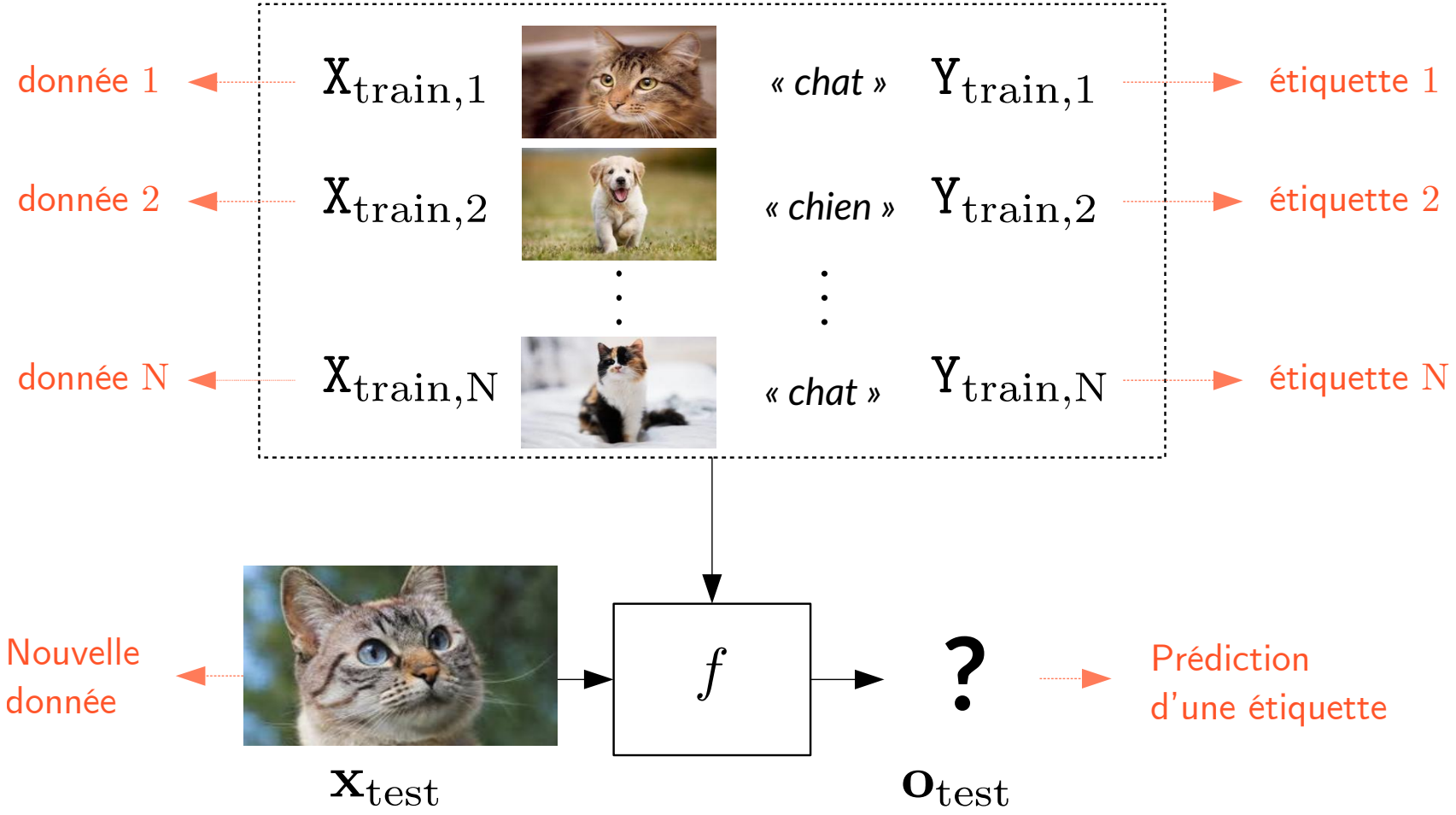
Clustering



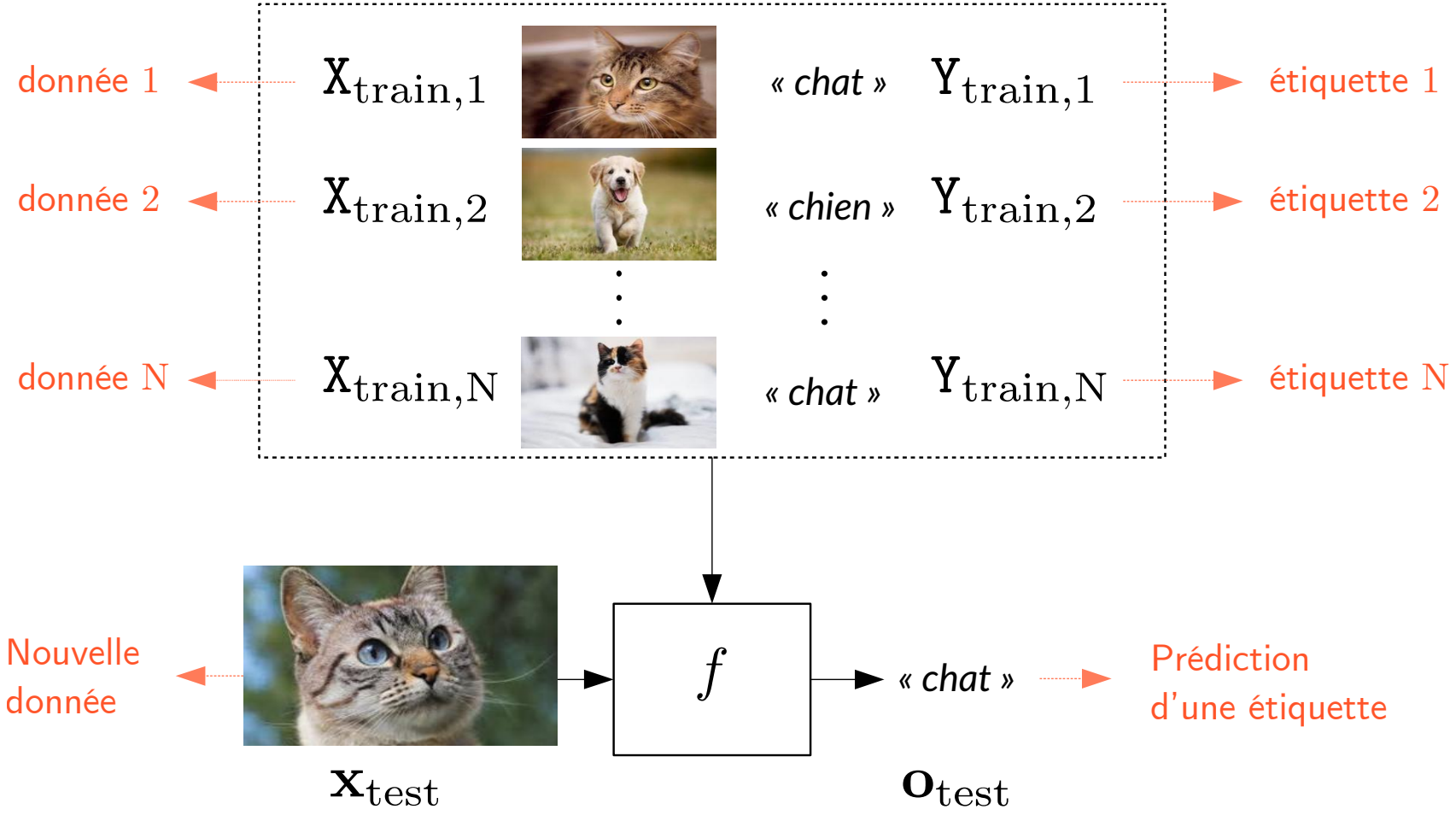


Source : [Peng et al. 2021] - Machine Learning Techniques for Personalised Medicine Approaches in Immune-Mediated Chronic Inflammatory Diseases: Applications and Challenges. In Frontiers in Pharmacology.

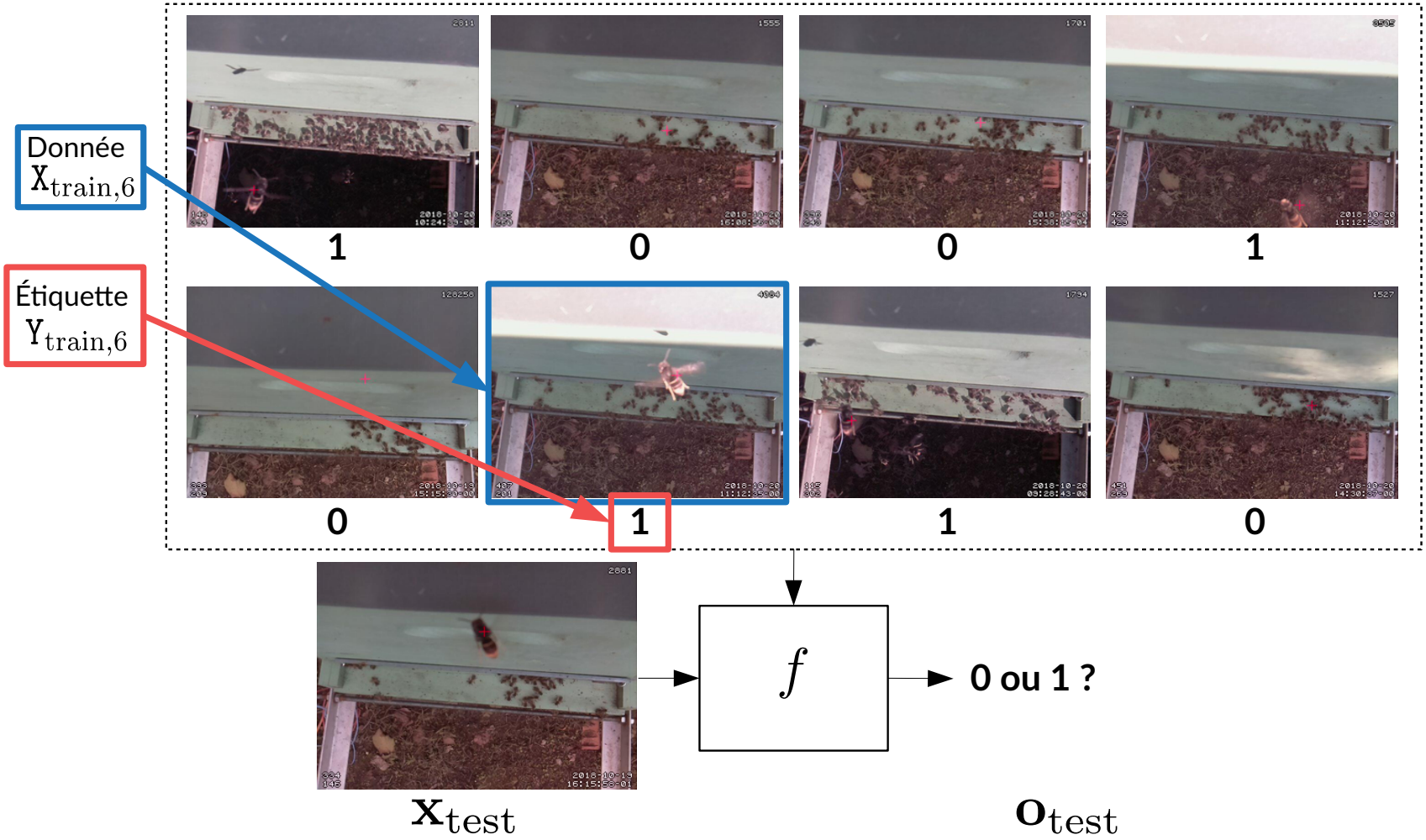
Exemple d'un classifieur chat / chien



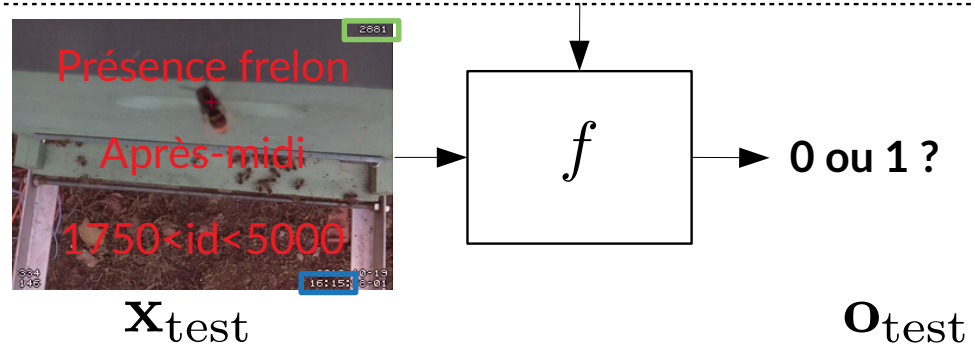
Exemple d'un classifieur chat / chien



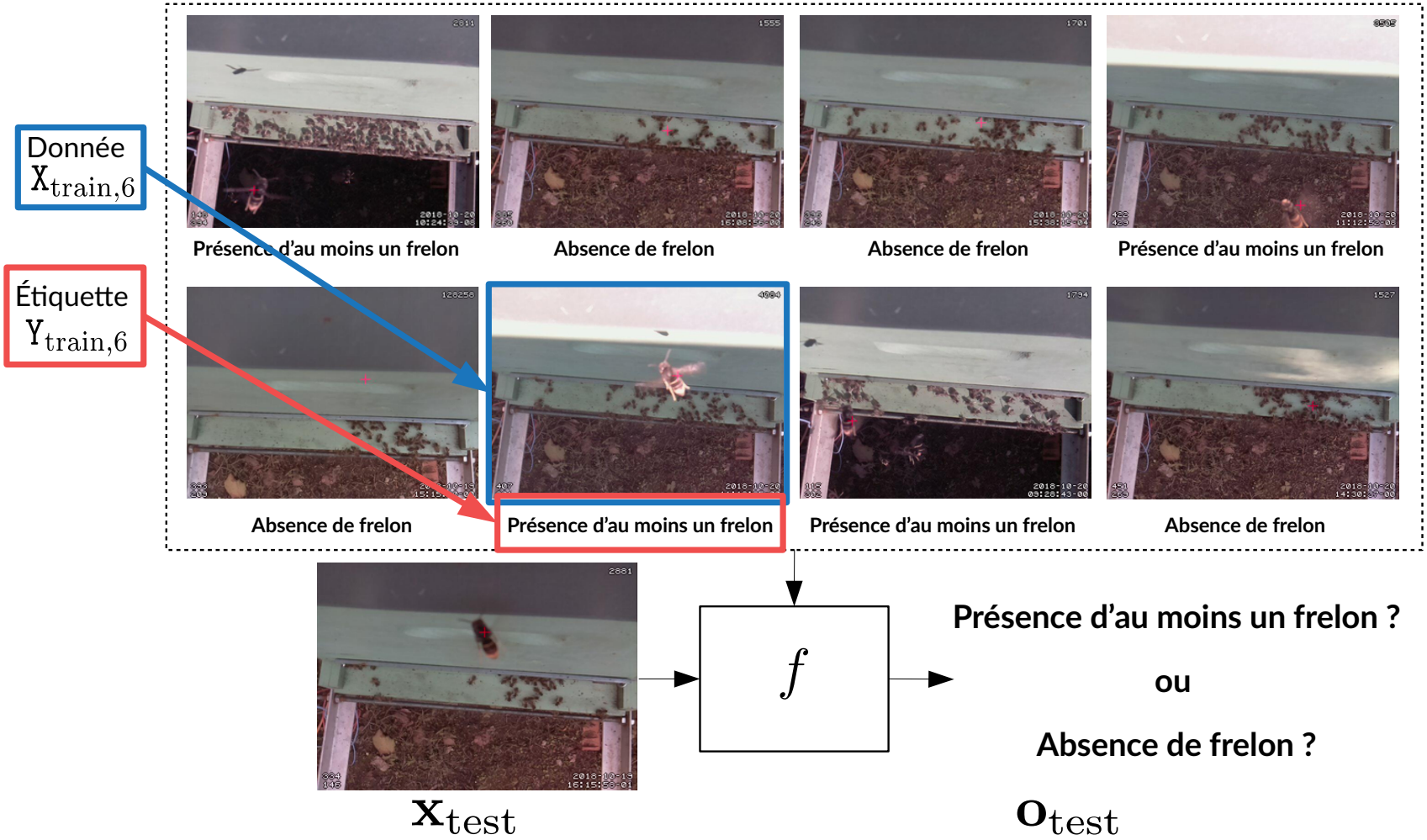
Pouvez-vous résoudre ce problème d'apprentissage supervisé ?



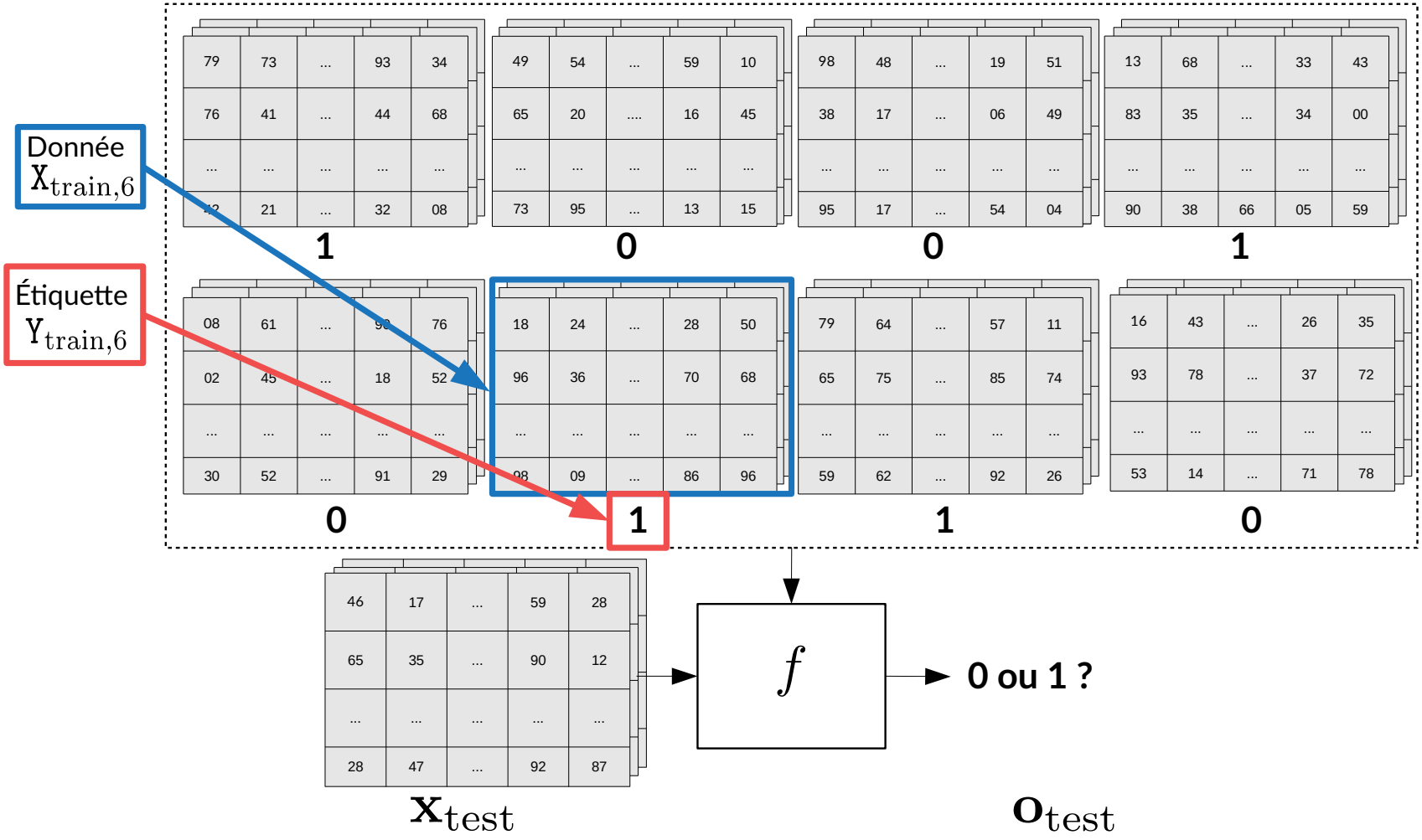
Mais de quel problème parle-t-on au juste ?



On veut que l'ordinateur apprenne à résoudre le problème suivant



Une fois mis en forme pour l'ordinateur le problème devient



Résumé

- L'apprentissage supervisé utilise des **données d'entraînement** assorties d'**étiquettes**
- L'ordinateur apprend une **fonction** qui effectue des calculs sur des tableaux de valeurs numériques et produit une valeur numérique en sortie.
- Nous **interprétons** cette valeur numérique en sortie comme une réponse à la question : « Y a-t-il (au moins) un frelon présent dans l'image ? »

- **Questions ?**
  - Quel méthode choisir pour déterminer la fonction ?
    Plus proches voisins, classification Bayésienne, SVM, réseaux de neurones, ...
  - Comment déterminer les paramètres et hyperparamètres ?
  - De quelles données avons-nous besoin ?
  - Comment évaluer la qualité de la fonction ?
  - ...

Approches paramétriques

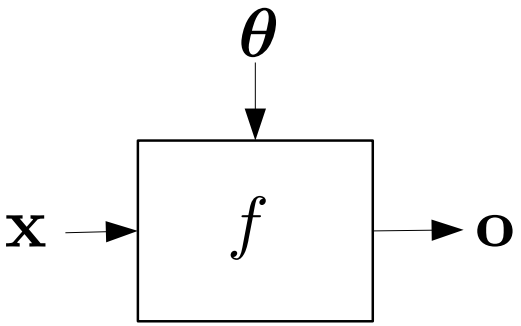
Différents formalismes

- Mathématique $\mathbf{o} = f(\mathbf{x}; \boldsymbol{\theta})$

- Informatique (Python)

```python
def f(x, theta):
    ...
    ...
    o = ...
    return o
```

- Graphique (Graphe de calcul)



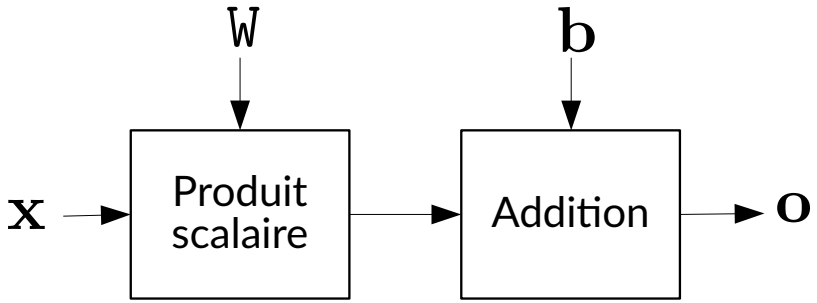
Exemple : transformation affine

- Mathématique $\mathbf{o} = f(\mathbf{x}; \boldsymbol{\theta} = \{\mathtt{W}, \mathbf{b}\}) = \mathtt{W}\mathbf{x} + \mathbf{b}$

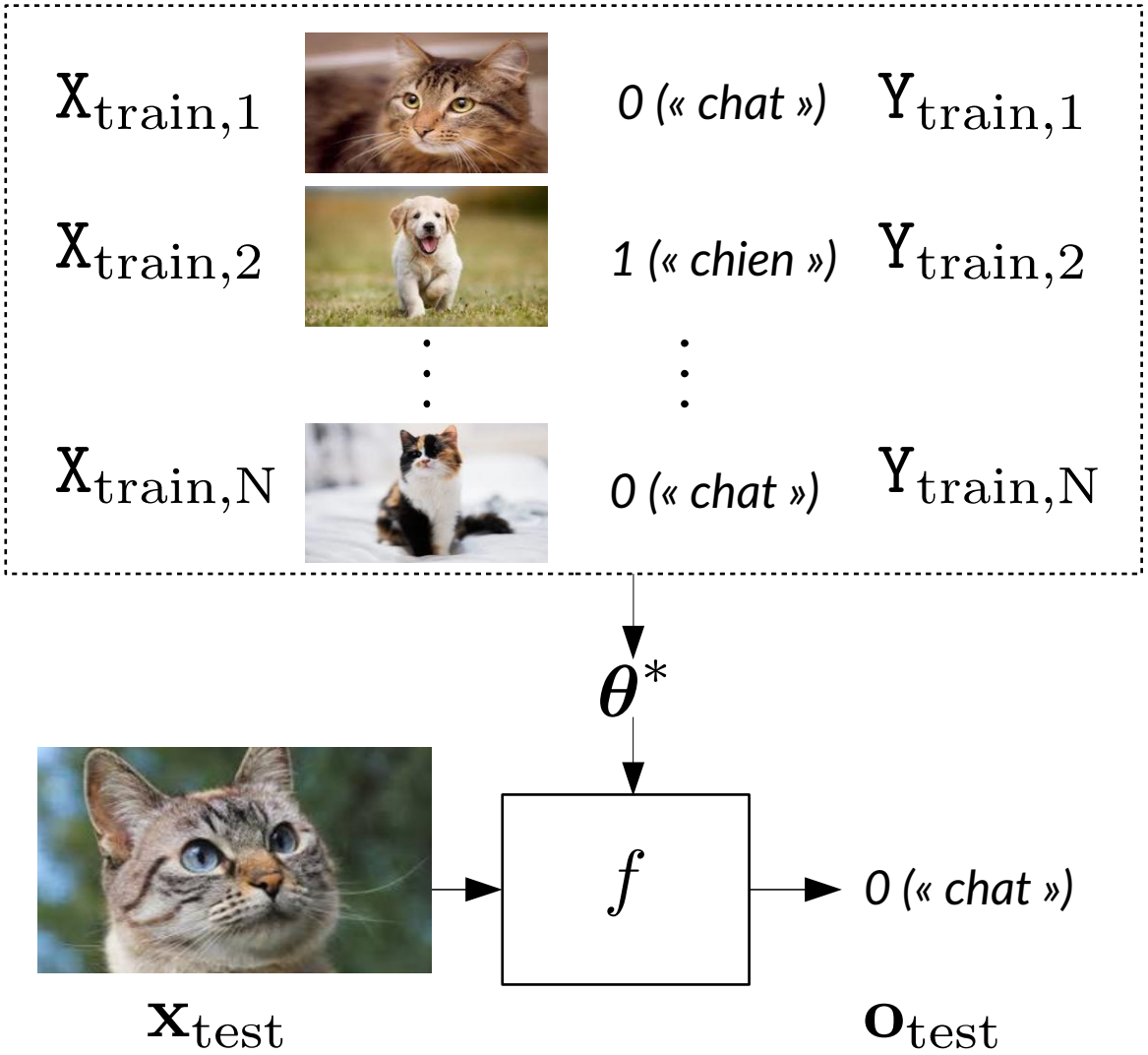
- Informatique (Python)

```python
def f(x, W, b):
    o = x.dot(W) + b
    return o
```

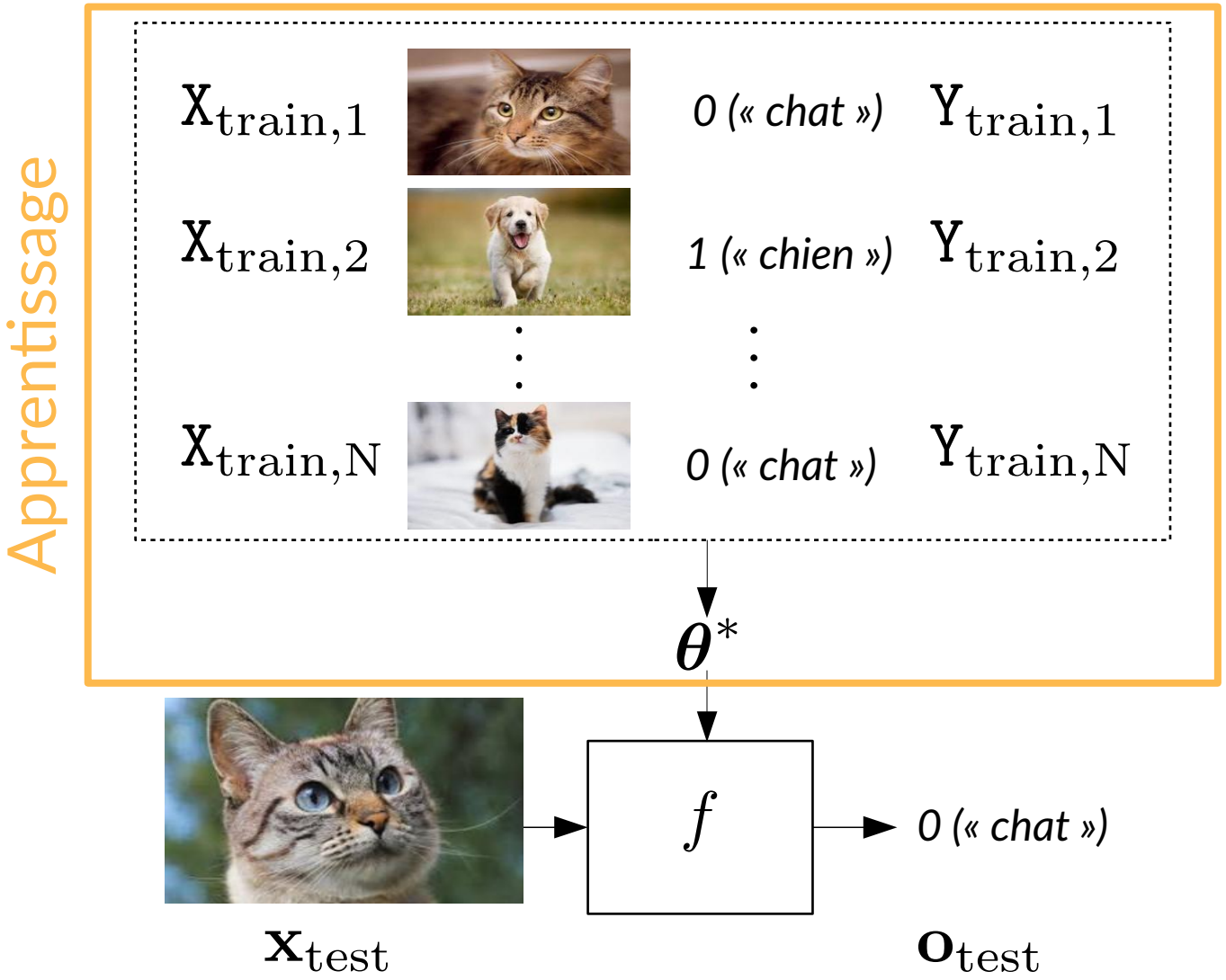
- Graphique (Graphe de calcul)



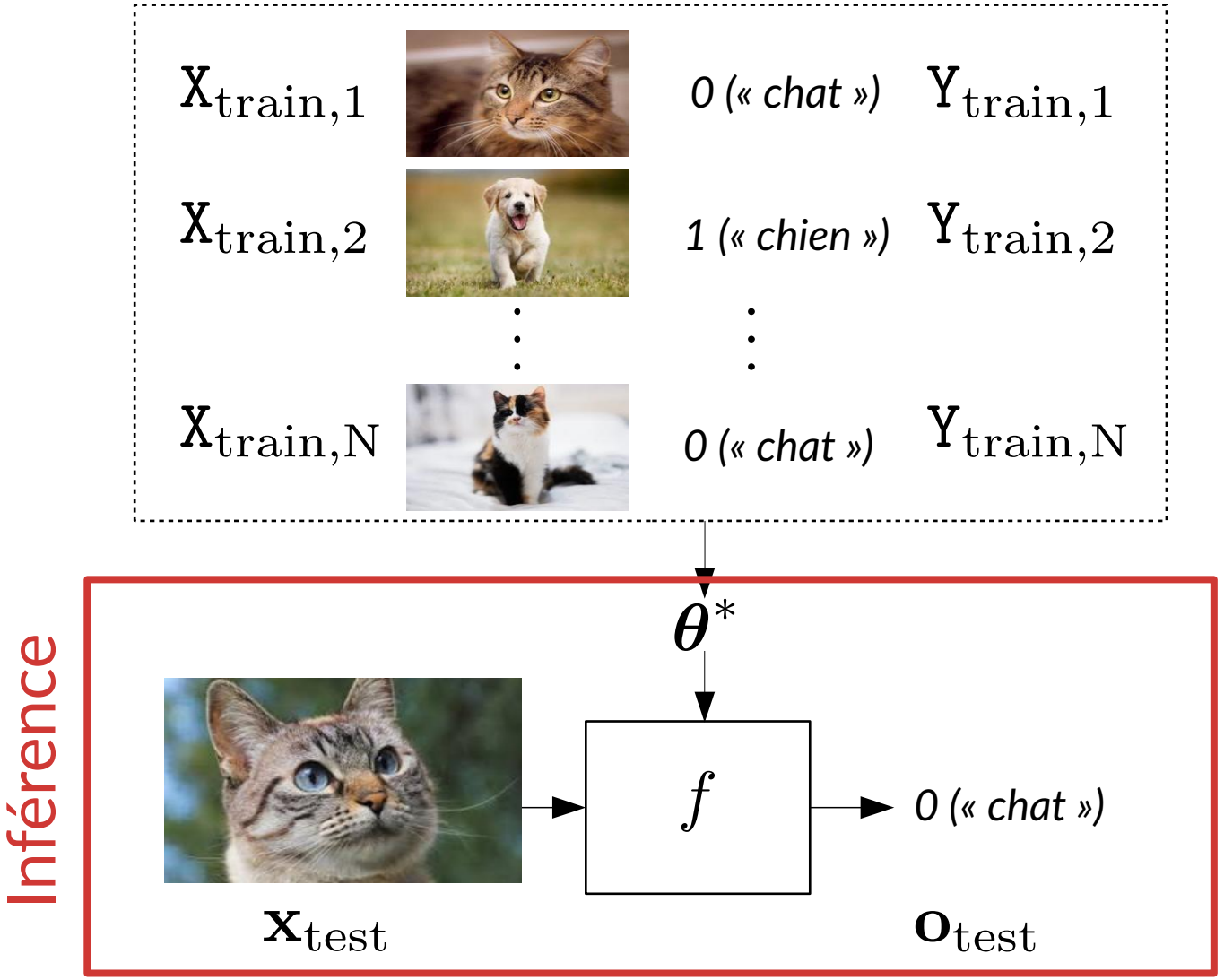
Les différentes étapes



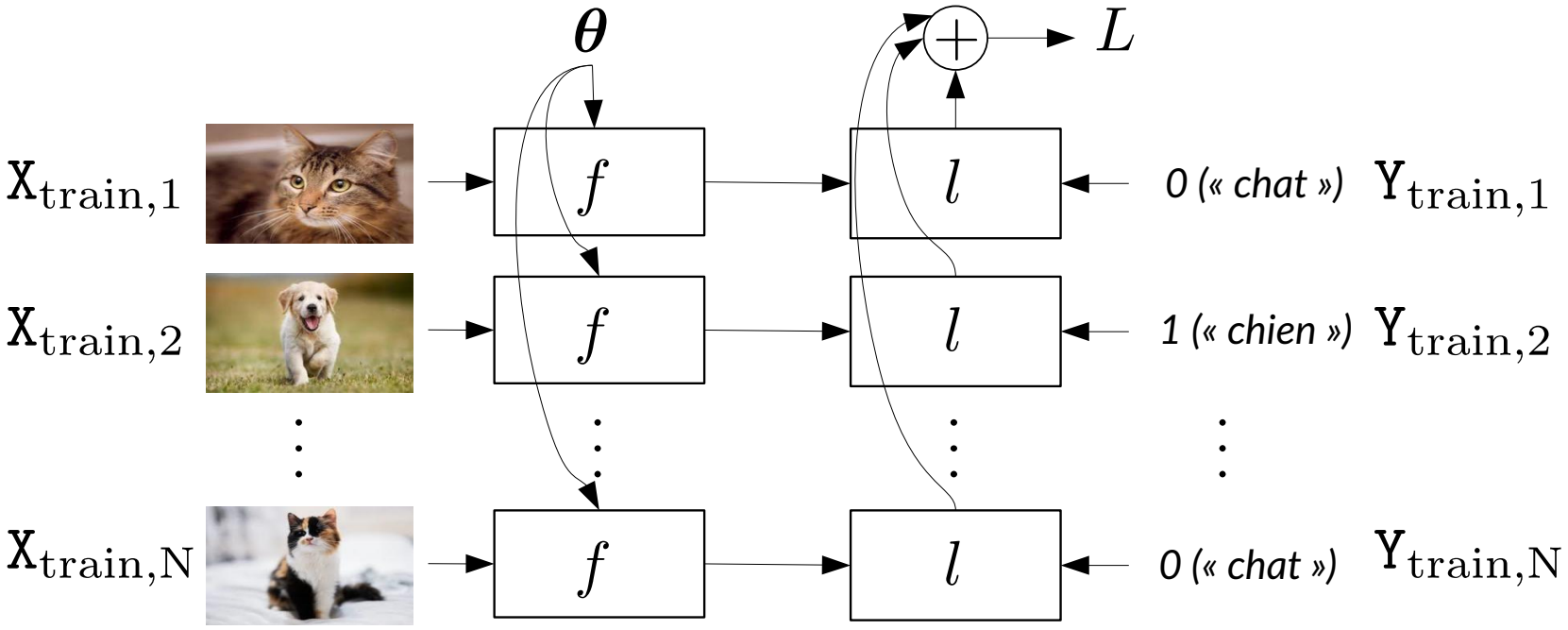
Les différentes étapes



Les différentes étapes



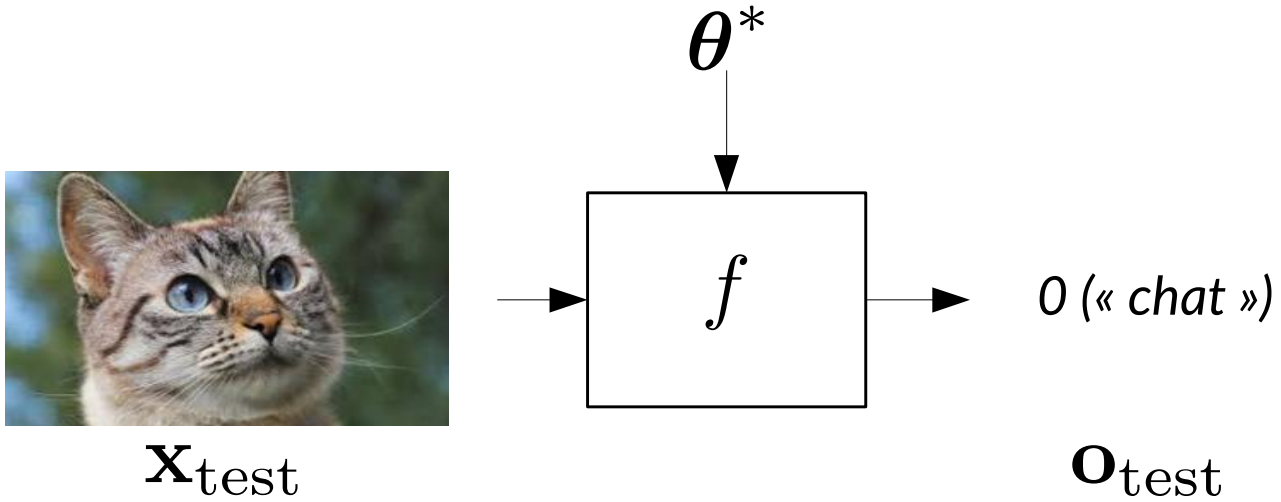
Étape d'apprentissage (*"training time"*)



$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} l\left(\mathtt{Y}_{\text{train},i}, f\left(\mathtt{X}_{\text{train},i}; \boldsymbol{\theta}\right)\right)$$

Où l est une **fonction de coût** à choisir

Étape d'inférence (*"test time"*)



$$\mathbf{o}_{\text{test}} = f\left(\mathbf{x}_{\text{test}}; \boldsymbol{\theta}^*\right)$$

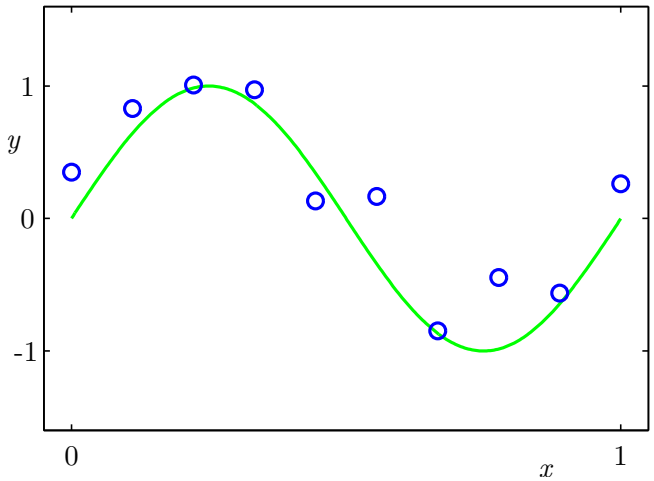
Exemple : Régression linéaire polynomiale

Régression linéaire en 1D :

- Entrée : $x \in \mathbb{R}$ Sortie : $y \in \mathbb{R}$
- Données d'entraînement :

$$\begin{aligned}\mathtt{D}_{\text{train}} &= \{(\mathtt{X}_{\text{train},1}, \mathtt{Y}_{\text{train},1}), \ldots, (\mathtt{X}_{\text{train},N}, \mathtt{Y}_{\text{train},N})\} \\ &= \{(x_1, y_1), \ldots, (x_N, y_N)\}\end{aligned}$$

- Objectif : Apprendre f capable de prédire o $(\hat{y})$ pour une entrée x : $o = f(x)$



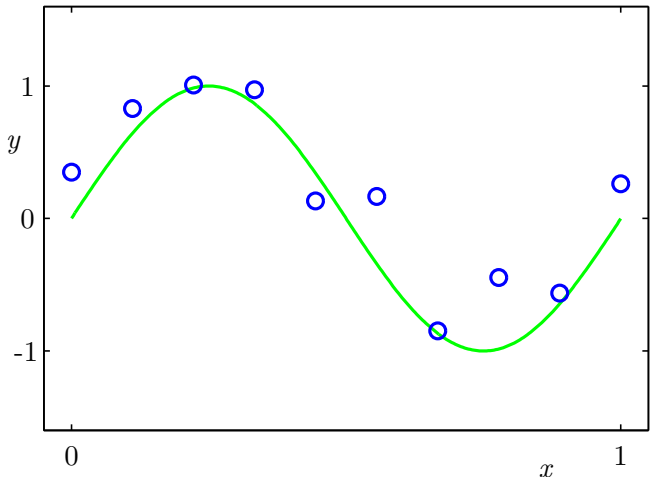
Exemple : Régression linéaire polynomiale

Régression linéaire en 1D :

- Entrée : $x \in \mathbb{R}$ Sortie : $y \in \mathbb{R}$
- Données d'entraînement :

$$\mathtt{D}_{\text{train}} = \{(\mathtt{X}_{\text{train},1}, \mathtt{Y}_{\text{train},1}), \ldots, (\mathtt{X}_{\text{train},N}, \mathtt{Y}_{\text{train},N})\}$$
$$= \{(x_1, y_1), \ldots, (x_N, y_N)\}$$

- Objectif : Apprendre f capable de prédire o $(\hat{y})$ pour une entrée x : $o = f(x)$



Régression linéaire polynomiale : On considère que des bonnes prédictions suivent une forme polynomiale. Le **modèle** f peut être défini comme :

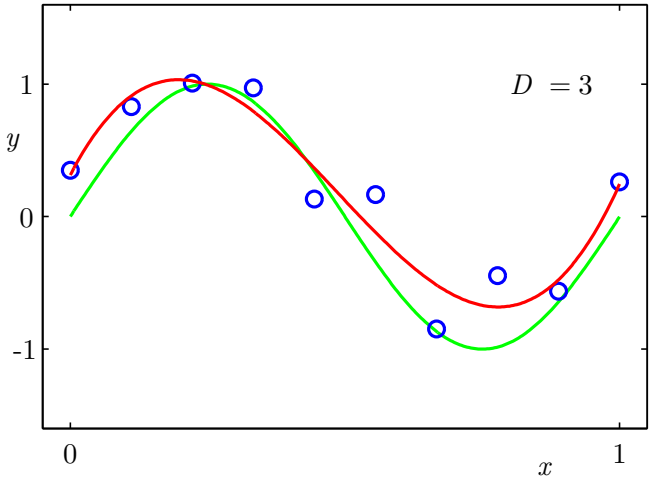
$$f(x; \boldsymbol{\theta}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_D x^D = \sum_{d=0}^{D} \theta_d x^d$$

où $\boldsymbol{\theta} = [\theta_0, \ldots, \theta_D]^\mathsf{T} \in \mathbb{R}^{D+1}$ sont les **paramètres** du modèle.

Choix du coût l

Comment trouver un "bon" $\boldsymbol{\theta}$?

$\rightarrow$ Trouver $\boldsymbol{\theta}^*$ qui minimise la différence entre les paires $o_i = f(x_i; \boldsymbol{\theta}^*)$ et y_i dans $\mathtt{D}_{\text{train}}$



Fonction de coût : erreur quadratique

$$\begin{aligned} l(y_i, o_i) &= (y_i - o_i)^2 \\ &= (y_i - f(x_i; \boldsymbol{\theta}))^2 \end{aligned}$$

Optimisation : Fonction **convexe**, donc un seul minimum global et calculable [1]

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{N} l(y_i, o_i) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{N} (y_i - f(x_i; \boldsymbol{\theta}))^2$$

Inférence :

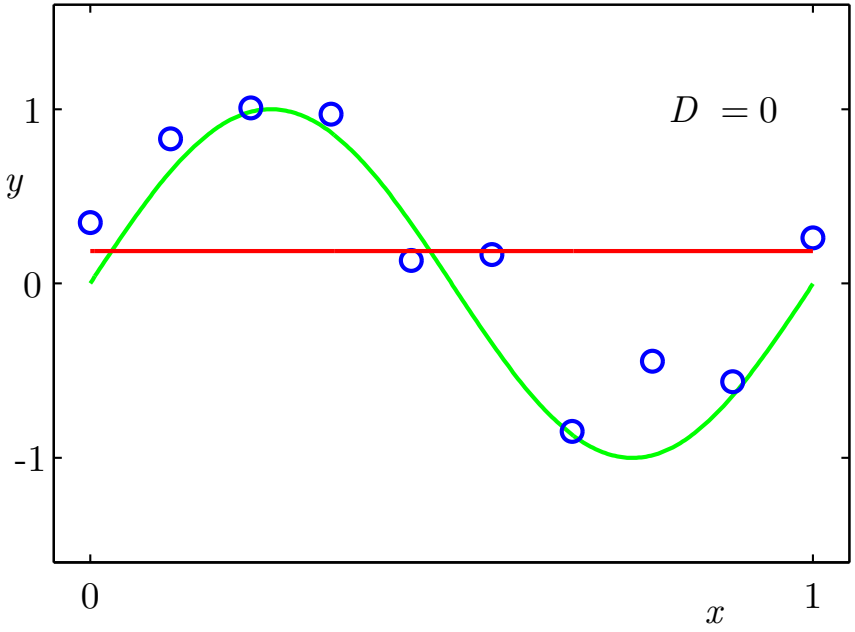
$$o_{\text{test}} = f(x_{\text{test}}; \boldsymbol{\theta}^*) = \theta_0^* + \theta_1^* x_{\text{test}} + \ldots + \theta_D^* x_{\text{test}}^D = \sum_{d=0}^{D} \theta_d^* x_{\text{test}}^d$$

1. Pour les détails de la résolution :
https://eli.thegreenplace.net/2014/derivation-of-the-normal-equation-for-linear-regression

Choix de la dimension D

Comment trouver un "bon" D ?

$$f(x;\boldsymbol{\theta}) = \sum_{d=0}^{D} \theta_d x^d$$



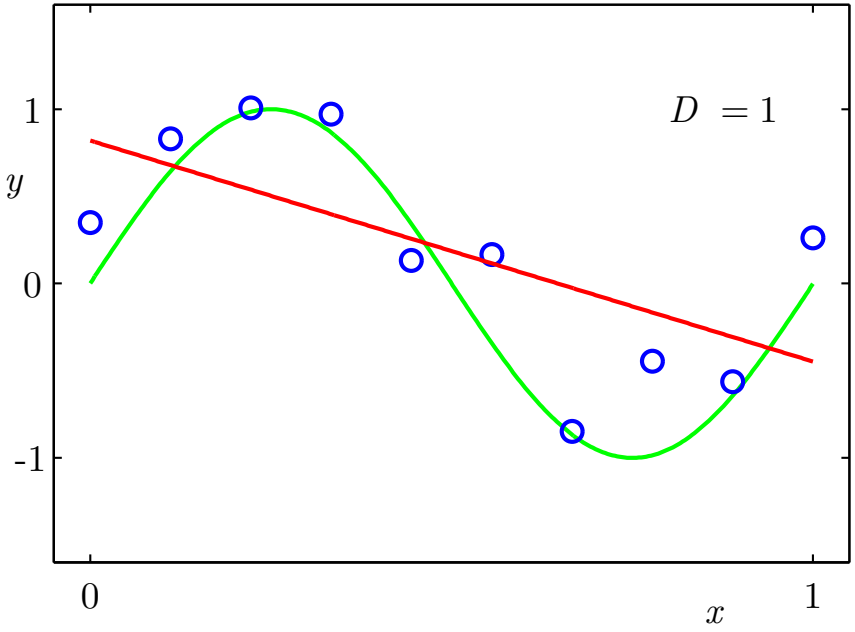
Trop faible : grande erreur sur les données d'entraînement, représentation trop simple
$\rightarrow$ **Sous-apprentissage (*underfitting*)**

D est appelé **hyperparamètre** du modèle

Choix de la dimension D

Comment trouver un "bon" D ?

$$f(x;\boldsymbol{\theta}) = \sum_{d=0}^{D} \theta_d x^d$$



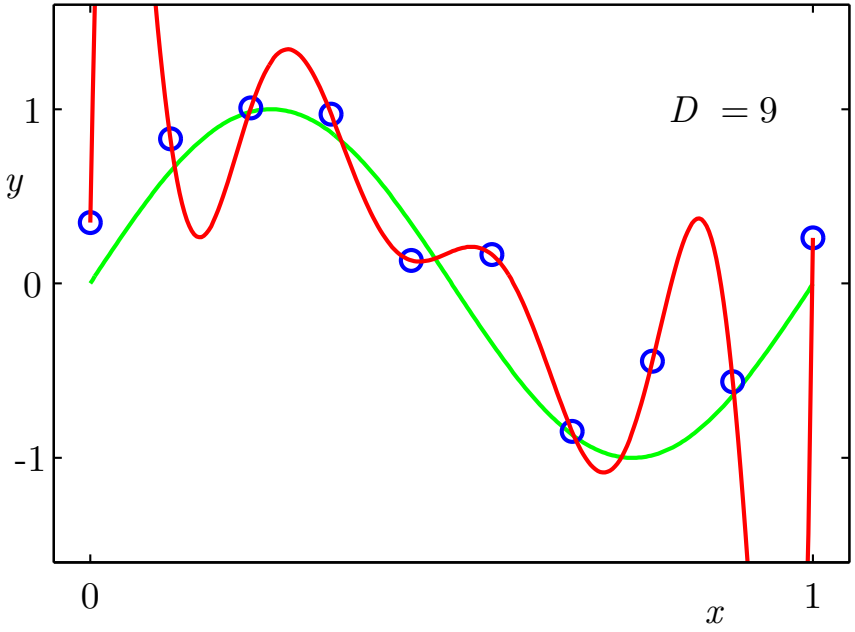
Trop faible : grande erreur sur les données d'entraînement, représentation trop simple
$\rightarrow$ **Sous-apprentissage (*underfitting*)**

D est appelé **hyperparamètre** du modèle

Choix de la dimension D

Comment trouver un "bon" D ?

$$f(x;\boldsymbol{\theta}) = \sum_{d=0}^{D} \theta_d x^d$$



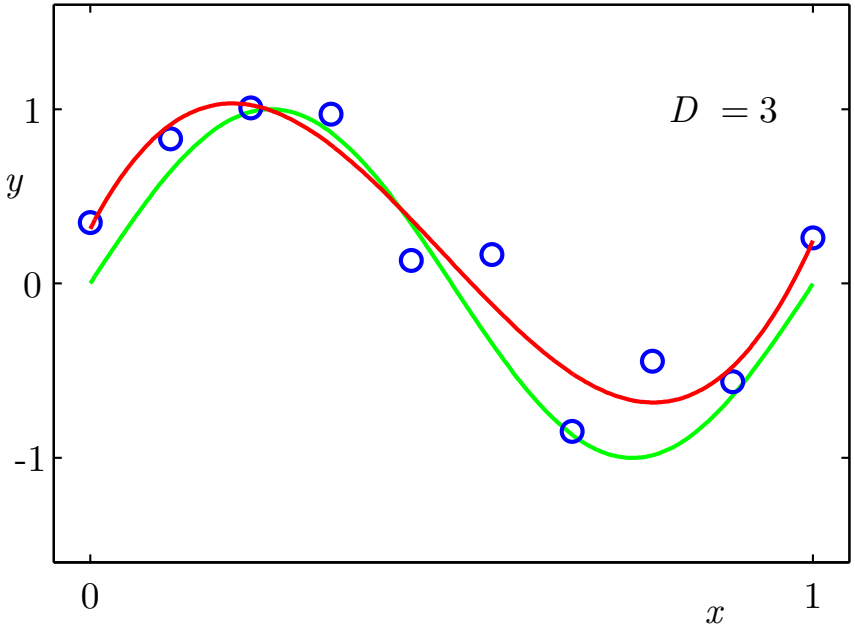
Trop forte : le modèle apprend "par cœur" les données d'entraînement, représentation trop complexe
$\rightarrow$ **Sur-apprentissage (*overfitting*)**

D est appelé **hyperparamètre** du modèle

Choix de la dimension D

Comment trouver un "bon" D ?

$$f(x;\boldsymbol{\theta}) = \sum_{d=0}^{D} \theta_d x^d$$

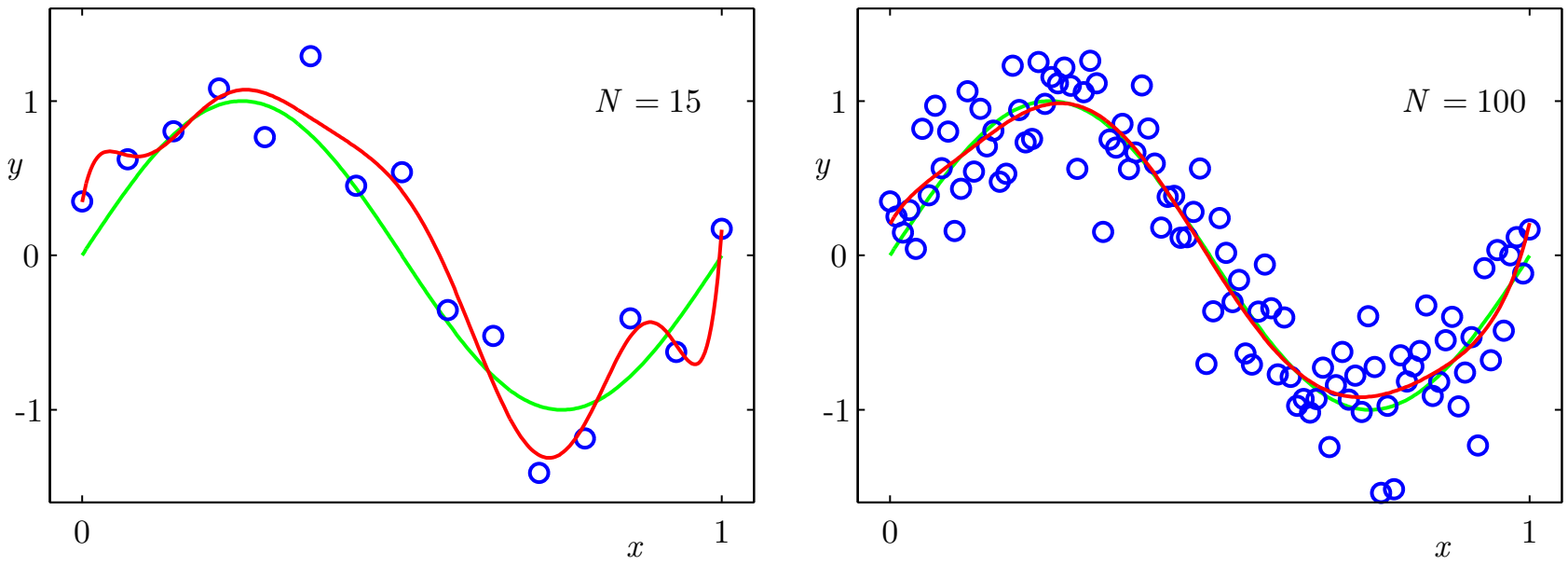


On cherche un bon compromis qui trouve la tendance générale, mais sans le bruit, pour **généraliser** aux nouvelles données de test

D est appelé **hyperparamètre** du modèle

Choix de la dimension D

Plus le **nombre de données d'entraînement augmente**, plus le modèle est susceptible de **généraliser**.



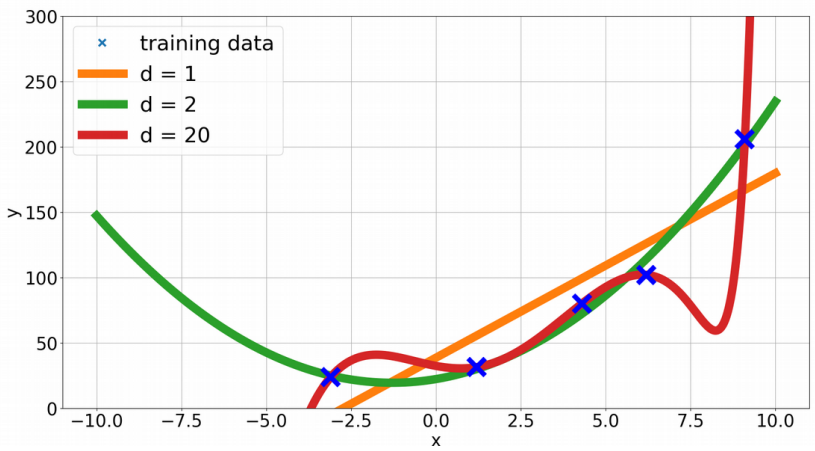
Exemple de régression : $N = 5$, $\mathbf{X} \in \mathbb{R}$ et $\mathbf{Y} \in \mathbb{R}$

$$\mathtt{X}_{\text{train}} = [\ \text{-3.1} \quad 1.2 \quad 4.3 \quad 6.2 \quad 9.1\ ]$$
$$\mathtt{Y}_{\text{train}} = [\ 23.7 \quad 31.3 \quad 79.9 \quad 101.9 \quad 205.5\ ]$$

Fonction : $f(x; \boldsymbol{\theta}) = \boldsymbol{\theta}^T [1 \ x \ \ldots \ x^D]$ hyperparamètre : D

Fonction de coût :

$$l(y, o) = (y - o)^2$$



Optimisation :

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{N} l(\mathtt{Y}_{\text{train},i}, f(\mathtt{X}_{\text{train},i}; \boldsymbol{\theta}))$$
$$= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{N} (\mathtt{Y}_{\text{train},i} - f(\mathtt{X}_{\text{train},i}; \boldsymbol{\theta}))^2$$

Inférence :

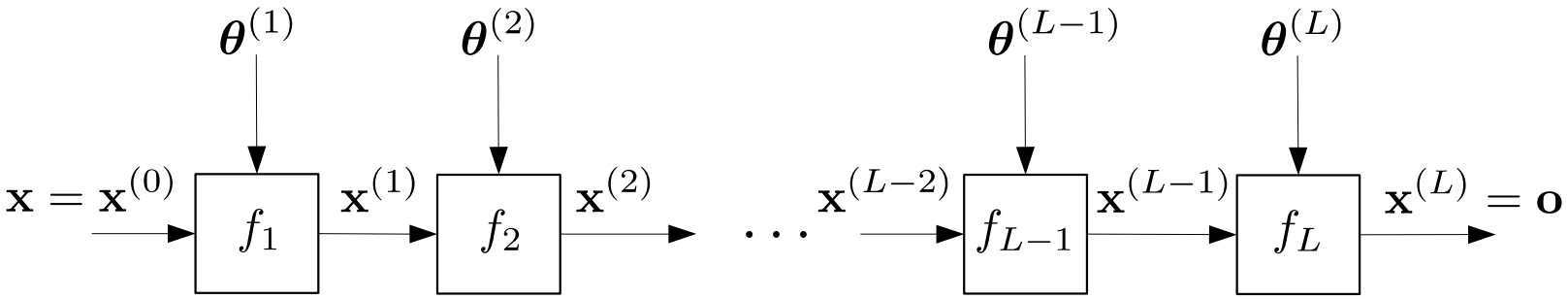
$$o_{\text{test}} = f(x_{\text{test}}; \boldsymbol{\theta}^*) = \theta_0^* + \theta_1^* x_{\text{test}} + \ldots + \theta_D^* x_{\text{test}}^D = \sum_{d=0}^{D} \theta_d^* x_{\text{test}}^d$$

Réseaux de neurones

Contexte

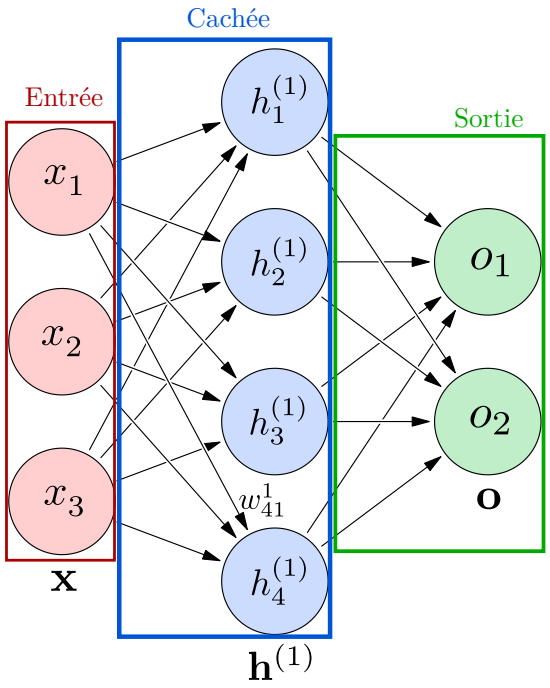
- Méthode d'apprentissage supervisée "inspirée" du cerveau humain.
- Consiste en l'inter-connexion de plusieurs petites unités appelées neurones.
- Introduit dans les années 50 (perceptron), très populaire dans les années 90, et réapparu en 2010 avec l'apprentissage profond.
- Aussi appelé Perceptron multicouche (Multi-Layer Perceptron (MLP)).
- Plus simplement, on verra que :

Réseau de neurones = Composition de fonctions paramétriques



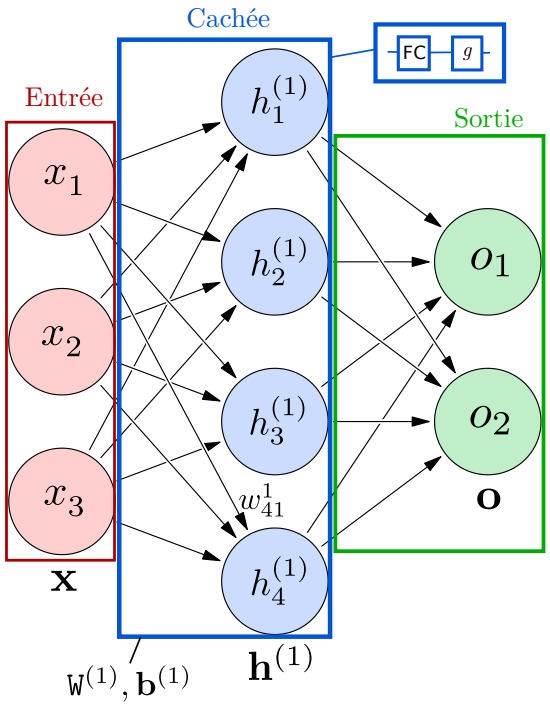
$$f\left(\mathbf{x};\boldsymbol{\theta}\right) = f_L\left(f_{L-1}\left(...f_2\left(f_1\left(\mathbf{x};\boldsymbol{\theta}^{(1)}\right);\boldsymbol{\theta}^{(2)}\right)...;\boldsymbol{\theta}^{(L-1)}\right);\boldsymbol{\theta}^{(L)}\right)$$

Perceptron multicouche (MLP)



- Inter-connexion de "neurones artificiels" issus de :
  - Transformation affine (connexions pondérées)
  - Fonction d'activation non linéaire
- Chaque niveau dans le graphe est appelé couche :
  - D'entrée $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$
  - Cachée(s) $\mathbf{h}^{(i)} = \left\{h_1^{(i)}, h_2^{(i)}, \ldots, h_{N_i}^{(i)}\right\}$
  - De sortie $\mathbf{o} = \{o_1, o_2, \ldots, o_{N_o}\}$
- Chaque neurone dans les couches cachées agit comme un classifieur ou un détecteur de motifs
- Réseau de neurones *feed-forward* (pas de cycle)

Perceptron multicouche (MLP)



– Transformation affine = **FC** ("Fully Connected")

$$\mathrm{FC}(\mathbf{x}; \boldsymbol{\theta} = \{\mathtt{W}, b\}) = \mathtt{W}\mathbf{x} + \mathbf{b}$$

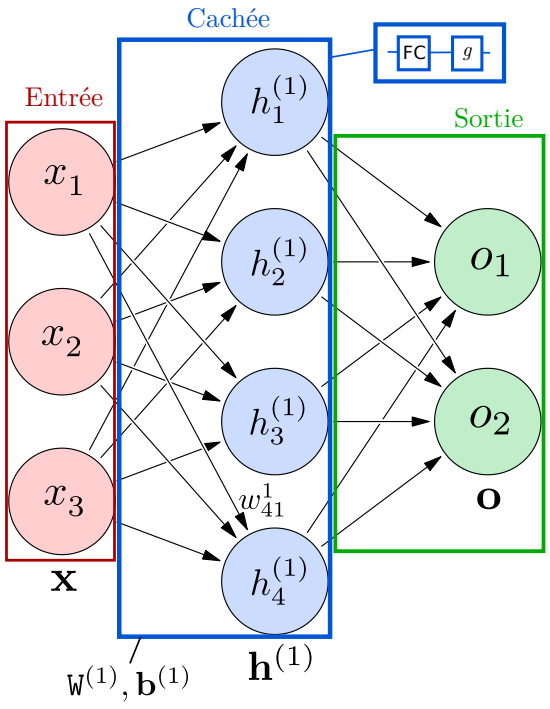
$$\mathbf{h}^{(1)} = \left(\mathtt{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right)$$

$$\begin{bmatrix} h_1^{(1)} \\ h_2^{(1)} \\ h_3^{(1)} \\ h_4^{(1)} \end{bmatrix} = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 \\ w_{41}^1 & w_{42}^1 & w_{43}^1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \end{bmatrix}$$

$\mathtt{W}^{(k)} = \{w_{ij}^k\}$ les **poids** entre le neurone précédent j et le suivant i à la couche k

$\mathbf{b}^{(k)} = \{b_i^k\}$ les **biais** du neurone suivant i à la couche k

Perceptron multicouche (MLP)



– Transformation affine = **FC** ("Fully Connected")

$$\mathrm{FC}(\mathbf{x}; \boldsymbol{\theta} = \{\mathtt{W}, b\}) = \mathtt{W}\mathbf{x} + \mathbf{b}$$

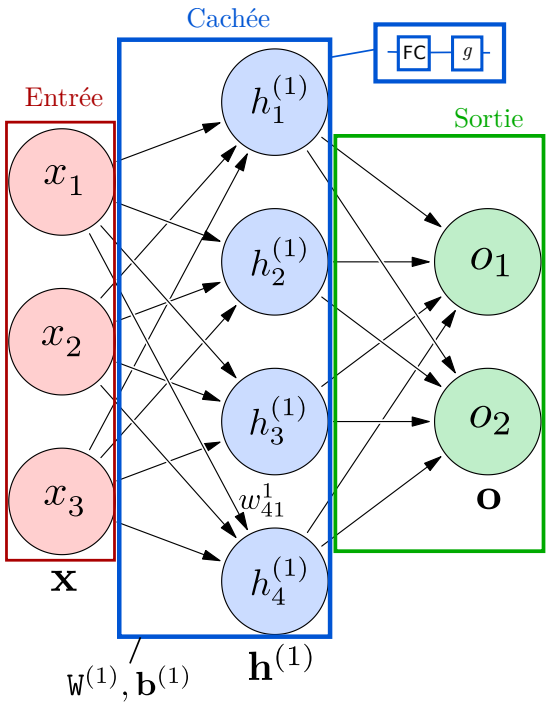
$$\mathbf{h}^{(1)} = \left(\mathtt{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right)$$

$$\begin{aligned}
h_1^{(1)} &= \left(w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 + b_1^1\right) \\
h_2^{(1)} &= \left(w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 + b_2^1\right) \\
h_3^{(1)} &= \left(w_{31}^1 x_1 + w_{32}^1 x_2 + w_{33}^1 x_3 + b_3^1\right) \\
h_4^{(1)} &= \left(w_{41}^1 x_1 + w_{42}^1 x_2 + w_{43}^1 x_3 + b_4^1\right)
\end{aligned}$$

$\mathtt{W}^{(k)} = \{w_{ij}^k\}$ les **poids** entre le neurone précédent j et le suivant i à la couche k

$\mathbf{b}^{(k)} = \{b_i^k\}$ les **biais** du neurone suivant i à la couche k

Perceptron multicouche (MLP)



– Transformation affine = **FC** ("Fully Connected")

$$\mathrm{FC}(\mathbf{x}; \boldsymbol{\theta} = \{\mathtt{W}, b\}) = \mathtt{W}\mathbf{x} + \mathbf{b}$$

– Fonction d'activation : $g(\mathbf{x})$

$$\mathbf{h}^{(1)} = g_1\left(\mathtt{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right)$$

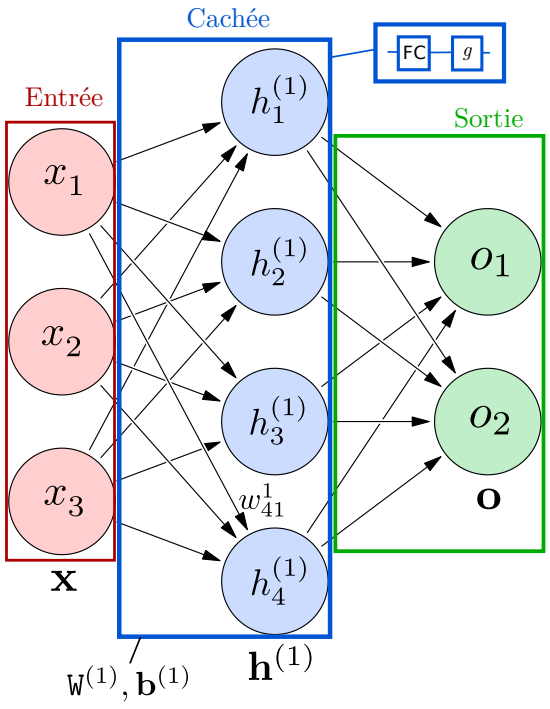
$$h_1^{(1)} = g_1\left(w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 + b_1^1\right)$$
$$h_2^{(1)} = g_1\left(w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 + b_2^1\right)$$
$$h_3^{(1)} = g_1\left(w_{31}^1 x_1 + w_{32}^1 x_2 + w_{33}^1 x_3 + b_3^1\right)$$
$$h_4^{(1)} = g_1\left(w_{41}^1 x_1 + w_{42}^1 x_2 + w_{43}^1 x_3 + b_4^1\right)$$

$\mathtt{W}^{(k)} = \{w_{ij}^k\}$ les **poids** entre le neurone précédent j et le suivant i à la couche k

$\mathbf{b}^{(k)} = \{b_i^k\}$ les **biais** du neurone suivant i à la couche k

g_k la **fonction d'activation** appliquée à chaque élément de l'entrée à la couche k

Perceptron multicouche (MLP)



– Transformation affine = **FC** ("Fully Connected")

$$\mathrm{FC}(\mathbf{x}; \boldsymbol{\theta} = \{\mathtt{W}, b\}) = \mathtt{W}\mathbf{x} + \mathbf{b}$$

– Fonction d'activation : $g(\mathbf{x})$

$$\mathbf{h}^{(1)} = g_1\left(\mathtt{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right)$$

$$h_1^{(1)} = g_1\left(w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 + b_1^1\right)$$
$$h_2^{(1)} = g_1\left(w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 + b_2^1\right)$$
$$h_3^{(1)} = g_1\left(w_{31}^1 x_1 + w_{32}^1 x_2 + w_{33}^1 x_3 + b_3^1\right)$$
$$h_4^{(1)} = g_1\left(w_{41}^1 x_1 + w_{42}^1 x_2 + w_{43}^1 x_3 + b_4^1\right)$$

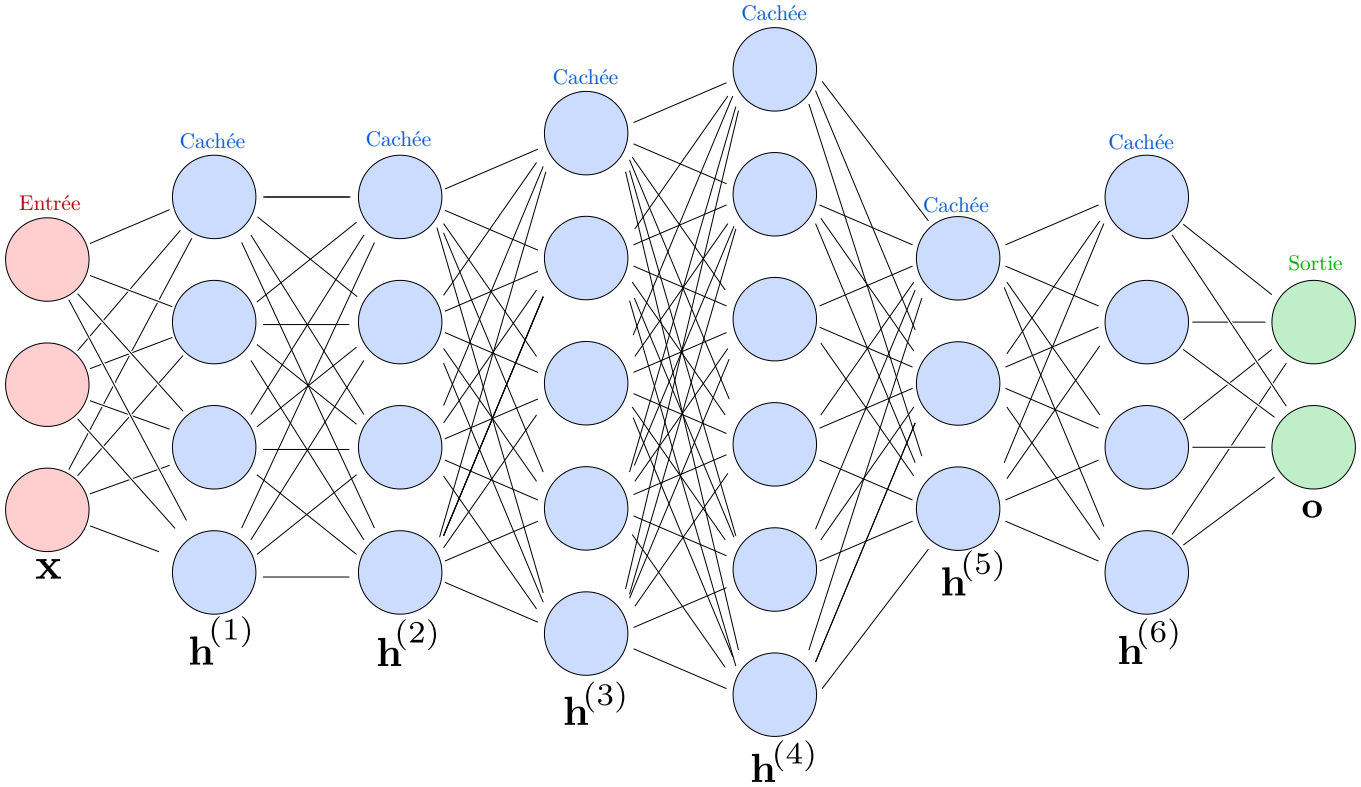
$\mathtt{W}^{(k)} = \{w_{ij}^k\}$ les **poids** entre le neurone précédent j et le suivant i à la couche k

$\mathbf{b}^{(k)} = \{b_i^k\}$ les **biais** du neurone suivant i à la couche k

g_k la **fonction d'activation** appliquée à chaque élément de l'entrée à la couche k

Couche de neurones = composition de 2 fonctions paramétriques (ici)

Architecture d'un réseau de neurones

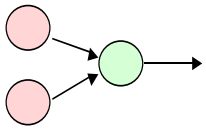


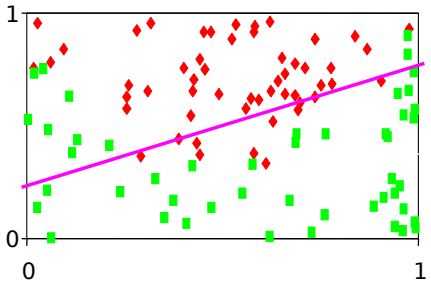
- Il peut avoir seulement une seule couche cachée (shallow network)
- Il peut avoir plusieurs couches cachées (deep network)
- Chaque couche (cachée ou de sortie) peut avoir une taille et une fonction d'activation différentes

Frontière de décision

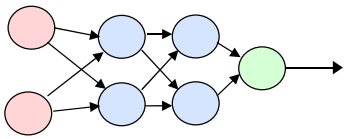
L'architecture du réseau définit la forme de la limite de décision :

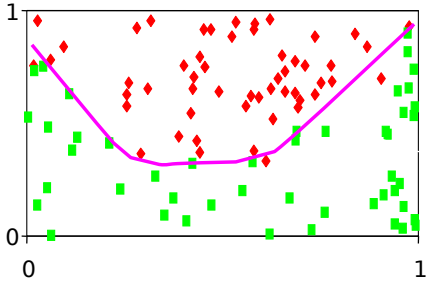
1 neurone



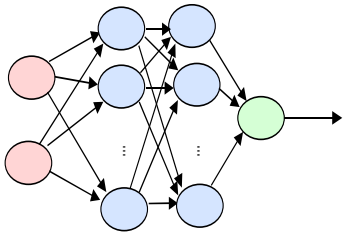


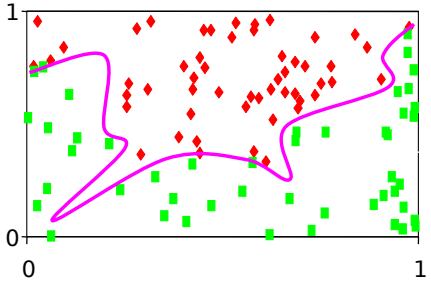
2+2+1 neurones





10+10+1 neurones





Complexité/capacité du réseau

$\Rightarrow$

Compromis entre généralisation et sur-apprentissage

Fonctions d'activation

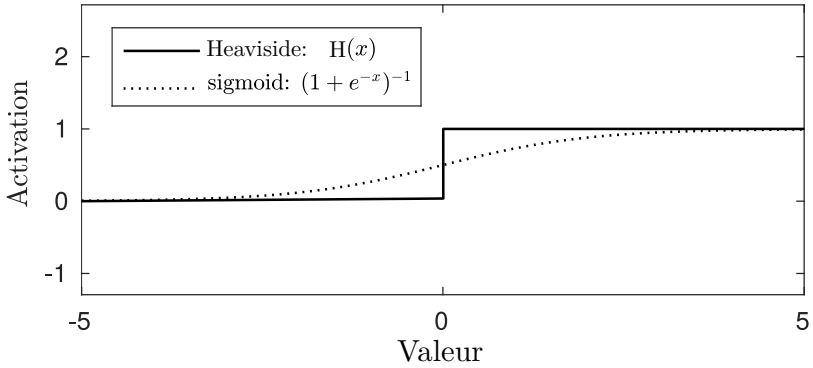
Fonction de seuil : par exemple la fonction de Heaviside (échelon)

$$g(a) = \mathrm{H}(a) = \begin{cases} 0 & \text{if} \quad a < 0 \\ 1 & \text{otherwise.} \end{cases}$$

- Des discontinuités dans les couches cachées rendent l'optimisation difficile
- Il est préférable d'utiliser des fonctions d'activation **différentiable**

Sigmoïde :

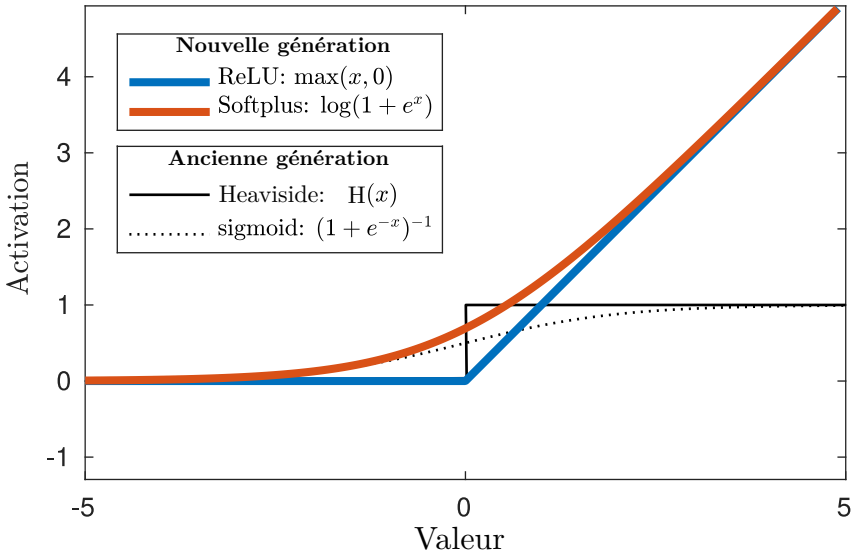
$$g(a) = \frac{1}{1 + e^{-a}} \in [0, 1]$$



- Approximation différentiable de la fonction échelon
- Seuil/comportement linéaire pour les fortes/faibles valeurs

Fonctions d'activation "modernes" :

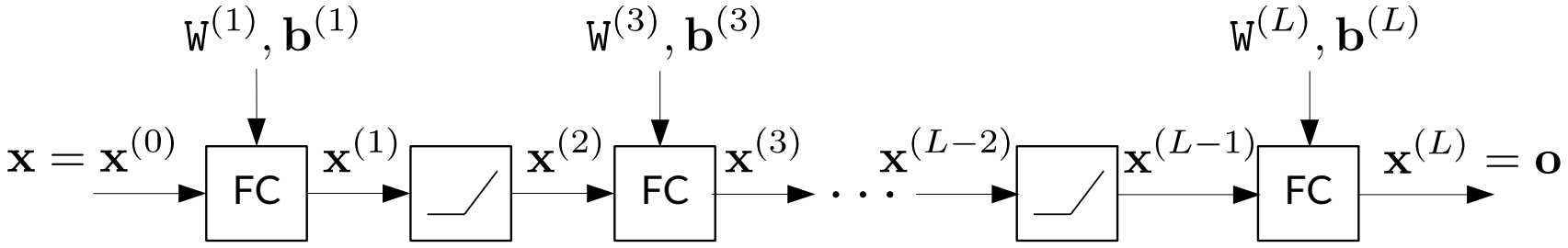
$$\underbrace{g(a) = \max(a, 0)}_{\text{ReLU}} \quad ou \quad \underbrace{g(a) = \log(1 + e^a)}_{\text{Softplus}}$$



- De nombreuses architectures de réseaux de neurones récentes utilisent la fonction d'activation *Rectified Linear Unit* (ReLU) pour les couches cachées.
- Elle entraîne beaucoup plus rapidement, est plus expressive que la fonction sigmoïde et évite le problème de disparition du gradient (*vanishing gradient*).

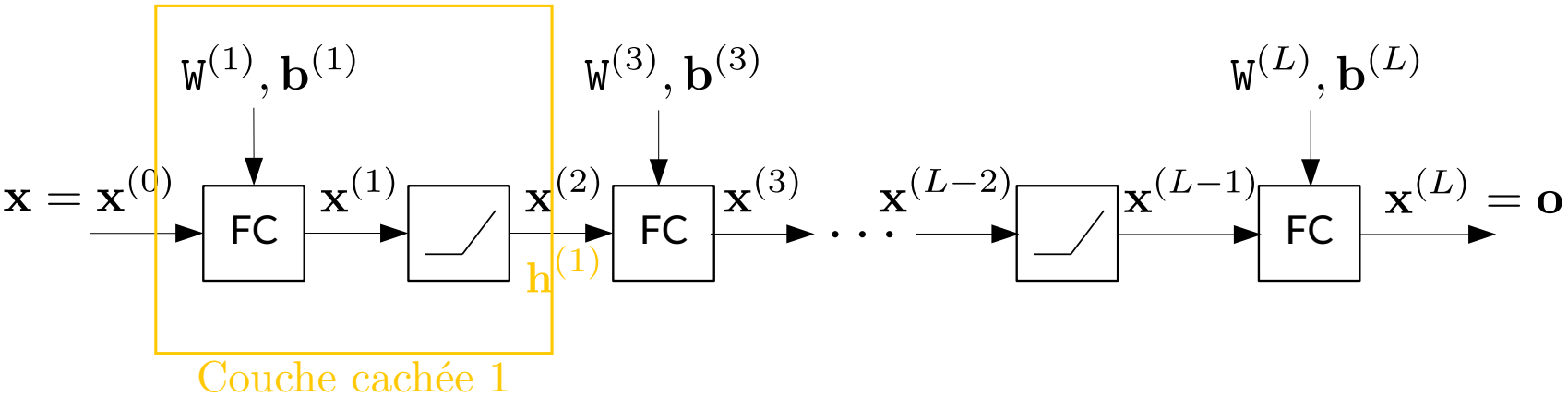
Composition de fonctions paramétriques : FC + ReLU

$$f(\mathbf{x};\boldsymbol{\theta}) = \mathrm{MLP}\left(\mathbf{x};\boldsymbol{\theta} = \left\{\mathtt{W}^{(l)}, \mathbf{b}^{(l)}\right\}_l\right)$$



Composition de fonctions paramétriques : FC + ReLU

$$f(\mathbf{x};\boldsymbol{\theta}) = \mathrm{MLP}\left(\mathbf{x};\boldsymbol{\theta} = \left\{\mathtt{W}^{(l)}, \mathbf{b}^{(l)}\right\}_l\right)$$



Attention !

Dans cette représentation décomposant toutes les fonctions paramétriques :

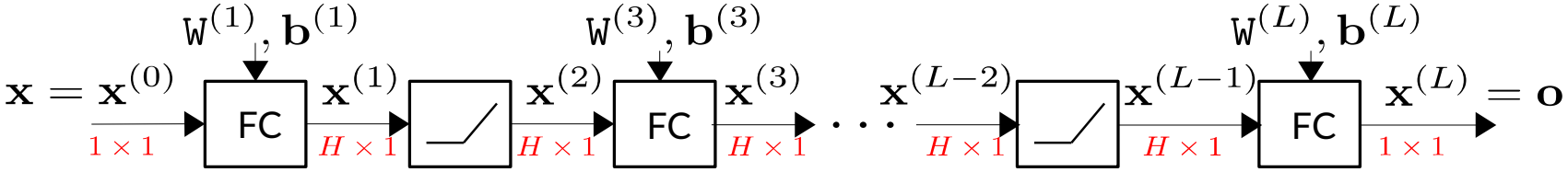
$\mathbf{h}^{(i)} = \mathbf{x}^{(2i)}$ et $\{\mathtt{W}^{(2i-1)}, \mathbf{b}^{(2i-1)}\}$ paramètres de la couche cachée i

Perceptron multicouche (MLP)

Exemple de régression : $N = 5$, $\mathbf{X} \in \mathbb{R}$ et $\mathbf{Y} \in \mathbb{R}$

$$\mathtt{X}_{\text{train}} = [\ \text{-3.1} \quad 1.2 \quad 4.3 \quad 6.2 \quad 9.1\ ]$$
$$\mathtt{Y}_{\text{train}} = [\ 23.7 \quad 31.3 \quad 79.9 \quad 101.9 \quad 205.5\ ]$$

Fonction : $f(x; \boldsymbol{\theta}) = \mathrm{MLP}\left(x; \boldsymbol{\theta} = \left\{\mathtt{W}^{(l)}, \mathbf{b}^{(l)}\right\}_l\right)$ hyperparamètres : H, L



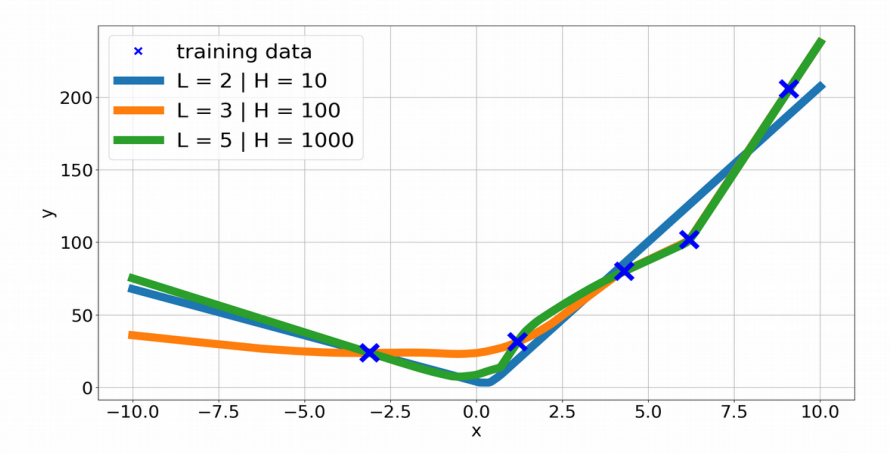
Fonction de coût :

$$l(y, o) = (y - o)^2$$

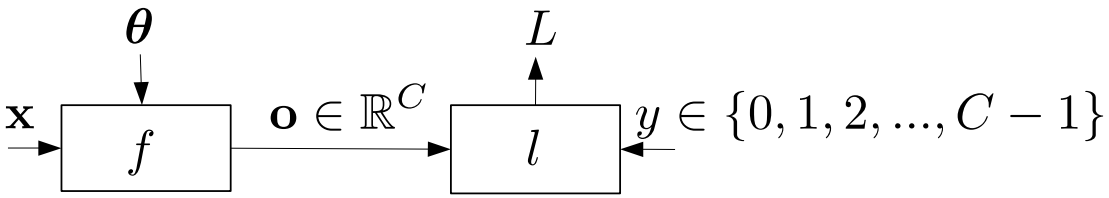
Optimisation :

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{N} \left(\mathtt{Y}_{\text{train},i} - f(\mathtt{X}_{\text{train},i}; \boldsymbol{\theta})\right)^2$$

Inférence : $o_{\text{test}} = f(x_{\text{test}}; \boldsymbol{\theta}^*)$

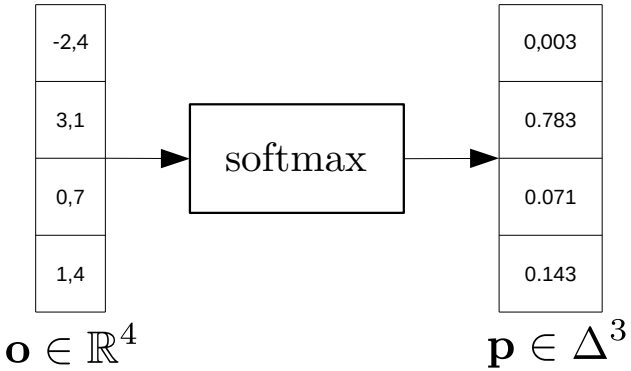


Réseau de classification



- **Softmax** : Approximation dérivable de la fonction argmax

$$\text{softmax}(\mathbf{o})[y] = \frac{\exp(\mathbf{o}[y])}{\sum_{c=1}^{C} \exp(\mathbf{o}[c])}$$



avec $\displaystyle\sum_{i=0}^{3} \mathbf{p}_i = 1$

$\mathbf{p}_i \geq 0$ pour $i = 0, ..., 3$

- **Cross-entropy** : $l(y, \mathbf{o}) = -\ln(\text{softmax}(\mathbf{o})[y])$

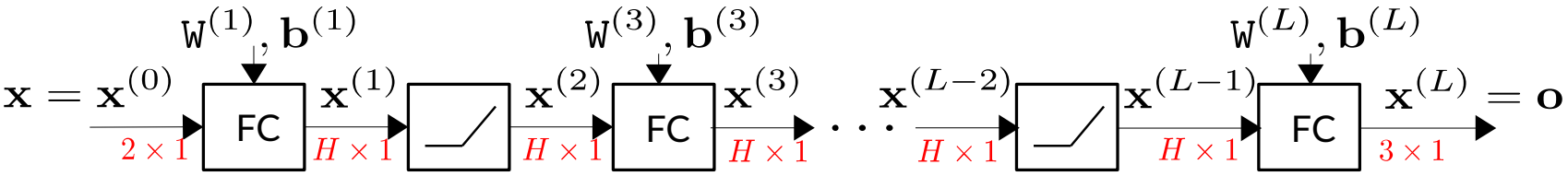
Ex. : Si $y = 2$, $L = -\ln(0.071) = 2.647$

Perceptron multicouche (MLP)

Exemple de classification : $N = 5$, $\mathbf{X} \in \mathbb{R}^2$ et $\mathbf{Y} \in \{0, 1, 2\}$

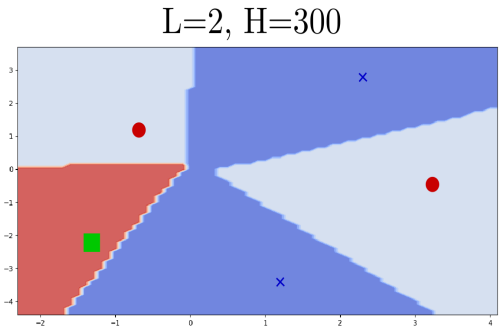
$\mathtt{X}_{\text{train}} = [\ (1.2,\ -3.4)\quad (2.3,\ 2.8)\quad (-0.7,\ 1.2)\quad (3.2,\ -0.4)\quad (-1.3,\ 2.3)\ ]$
$\mathtt{Y}_{\text{train}} = [\ 0\quad 0\quad 1\quad 1\quad 2\ ]$

Fonction : $f(x; \boldsymbol{\theta}) = \mathrm{MLP}\left(x; \boldsymbol{\theta} = \left\{\mathtt{W}^{(l)}, \mathbf{b}^{(l)}\right\}_l\right)$ hyperparamètres : H, L



Fonction de coût :

$$l(y, \mathbf{o}) = -\ln(\mathrm{softmax}(\mathbf{o})[y])$$

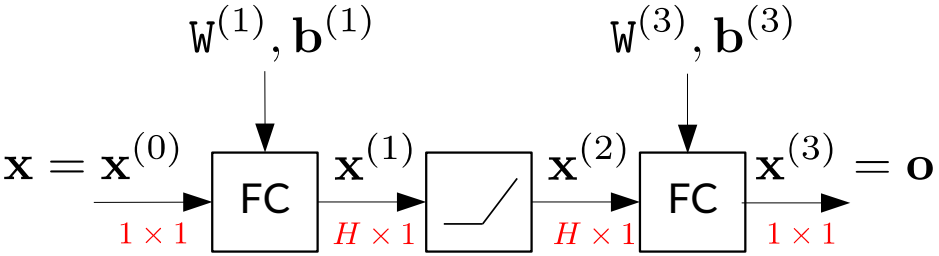


Optimisation :

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\mathrm{argmin}} \sum_{i=1}^{N} -\ln\left(\mathrm{softmax}\left(\mathrm{MLP}\left(\mathtt{X}_{\text{train},i}; \boldsymbol{\theta}\right)\right)\left[\mathtt{Y}_{\text{train},i}\right]\right)$$

Inférence : $\mathbf{o}_{\text{test}} = f(x_{\text{test}}; \boldsymbol{\theta}^*)$ classe prédite : $\mathrm{argmax}\ \mathbf{o}_{\text{test}}$

MLP à une couche cachée en 1D

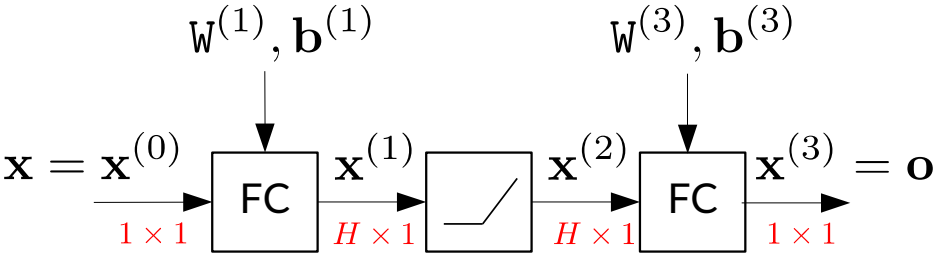


$$f(x) = \mathbf{w}_3^T \left(\mathrm{ReLU}\left(\mathbf{w}_1 x + \mathbf{b}_1\right)\right) + b_3$$

avec	$\mathtt{W}^{(1)} = \mathbf{w}_1$	Vecteur colonne $\mathrm{H} \times 1$
	$\mathbf{b}^{(1)} = \mathbf{b}_1$	Vecteur colonne $\mathrm{H} \times 1$
	$\mathtt{W}^{(3)} = \mathbf{w}_3^T$	Vecteur ligne $1 \times \mathrm{H}$
	$\mathbf{b}^{(3)} = b_3$	Scalaire 1×1

Nombre total de paramètres : $3 \times \mathrm{H} + 1$

MLP à une couche cachée en 1D



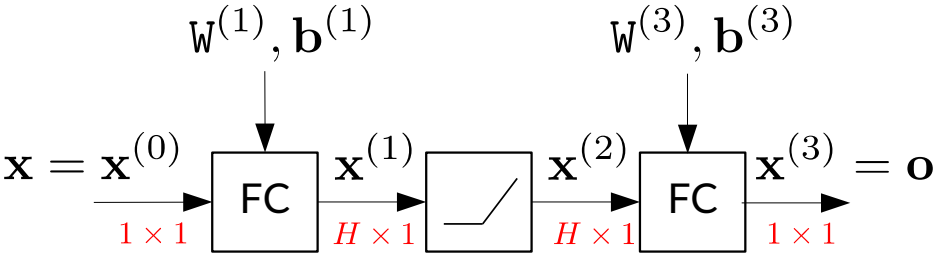
$$f(x) = \mathbf{w}_3^T \left(\mathrm{ReLU}\left(\mathbf{w}_1 x + \mathbf{b}_1\right)\right) + b_3 = \sum_{j=1}^{H} \mathbf{w}_{3,j} \mathrm{ReLU}\left(\mathbf{w}_{1,j} x + \mathbf{b}_{1,j}\right) + b_3$$

$\rightarrow$ somme pondérée de fonctions ReLU

avec	$\mathtt{W}^{(1)} = \mathbf{w}_1$	Vecteur colonne $\mathrm{H} \times 1$
	$\mathbf{b}^{(1)} = \mathbf{b}_1$	Vecteur colonne $\mathrm{H} \times 1$
	$\mathtt{W}^{(3)} = \mathbf{w}_3^T$	Vecteur ligne $1 \times \mathrm{H}$
	$\mathbf{b}^{(3)} = b_3$	Scalaire 1×1

Nombre total de paramètres : $3 \times \mathrm{H} + 1$

MLP à une couche cachée en 1D

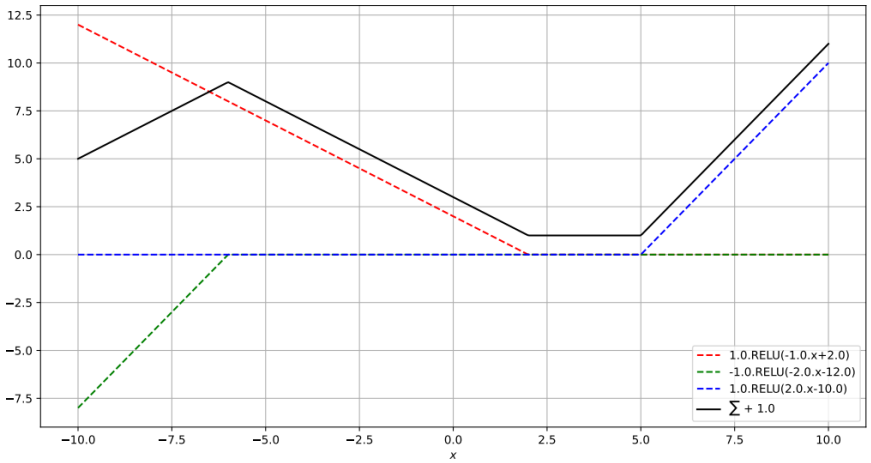


$$f(x) = \mathbf{w}_3^T \left(\mathrm{ReLU}\left(\mathbf{w}_1 x + \mathbf{b}_1\right)\right) + b_3 = \sum_{j=1}^{H} \mathbf{w}_{3,j} \mathrm{ReLU}\left(\mathbf{w}_{1,j} x + \mathbf{b}_{1,j}\right) + b_3$$

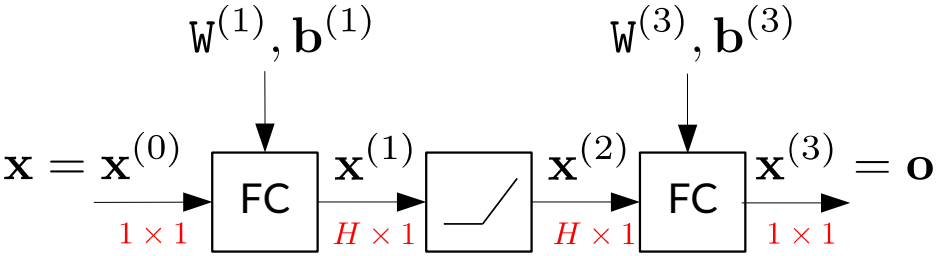
$\rightarrow$ somme pondérée de fonctions ReLU

Exemple numérique avec $\mathrm{H} = 3$:

$\rightarrow$ Fonction continue et affine par morceaux



MLP à une couche cachée en 1D

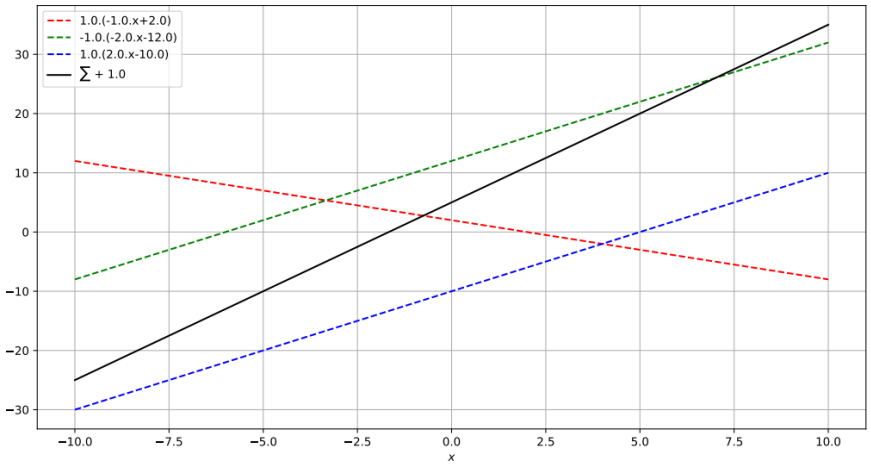


$$f(x) = \mathbf{w}_3^T \left(\mathrm{ReLU}\left(\mathbf{w}_1 x + \mathbf{b}_1\right)\right) + b_3 = \sum_{j=1}^{H} \mathbf{w}_{3,j} \mathrm{ReLU}\left(\mathbf{w}_{1,j} x + \mathbf{b}_{1,j}\right) + b_3$$

$\rightarrow$ somme pondérée de fonctions ReLU

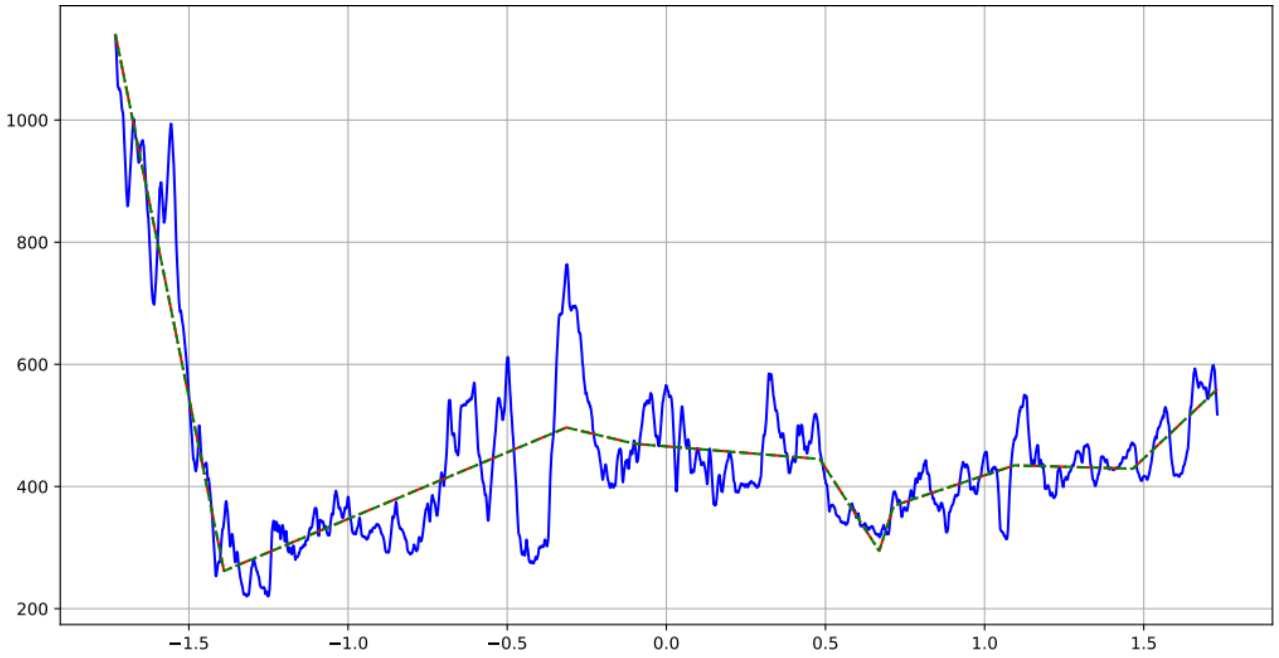
Et sans ReLU ?

$\rightarrow$ Une somme de fonctions linéaires est une fonction linéaire...



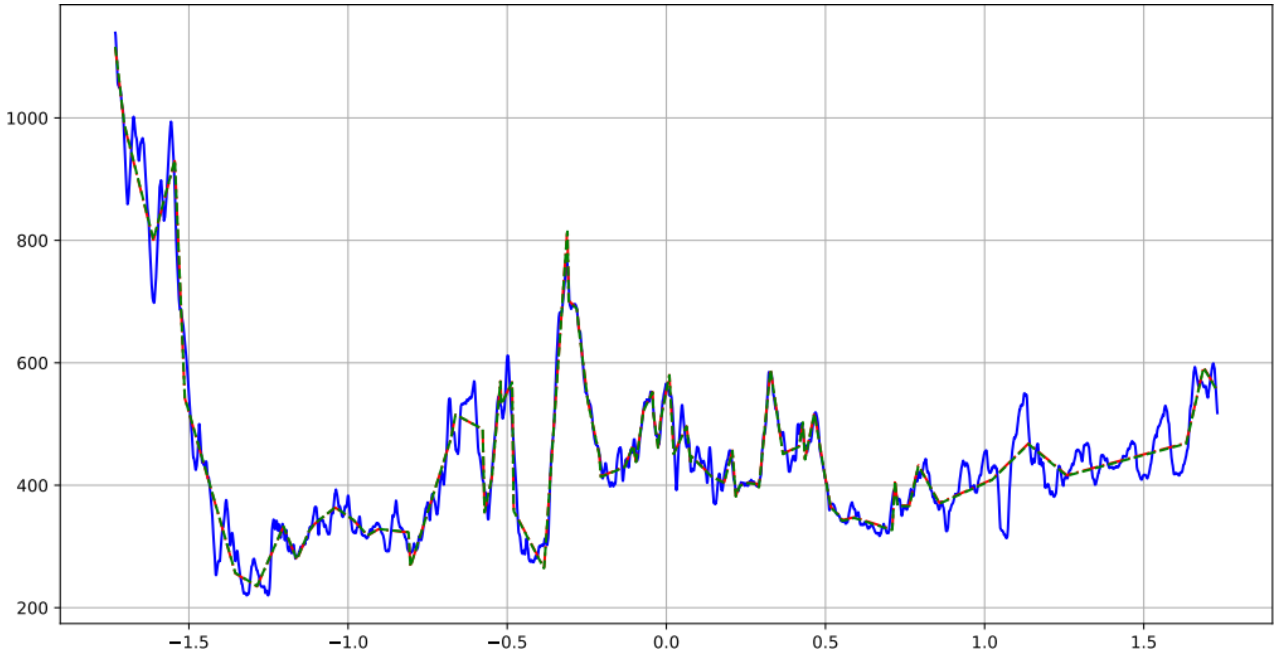
MLP à une couche cachée en 1D

Optimisation d'un MLP sur un profil de terrain : $\mathrm{H} = 10$



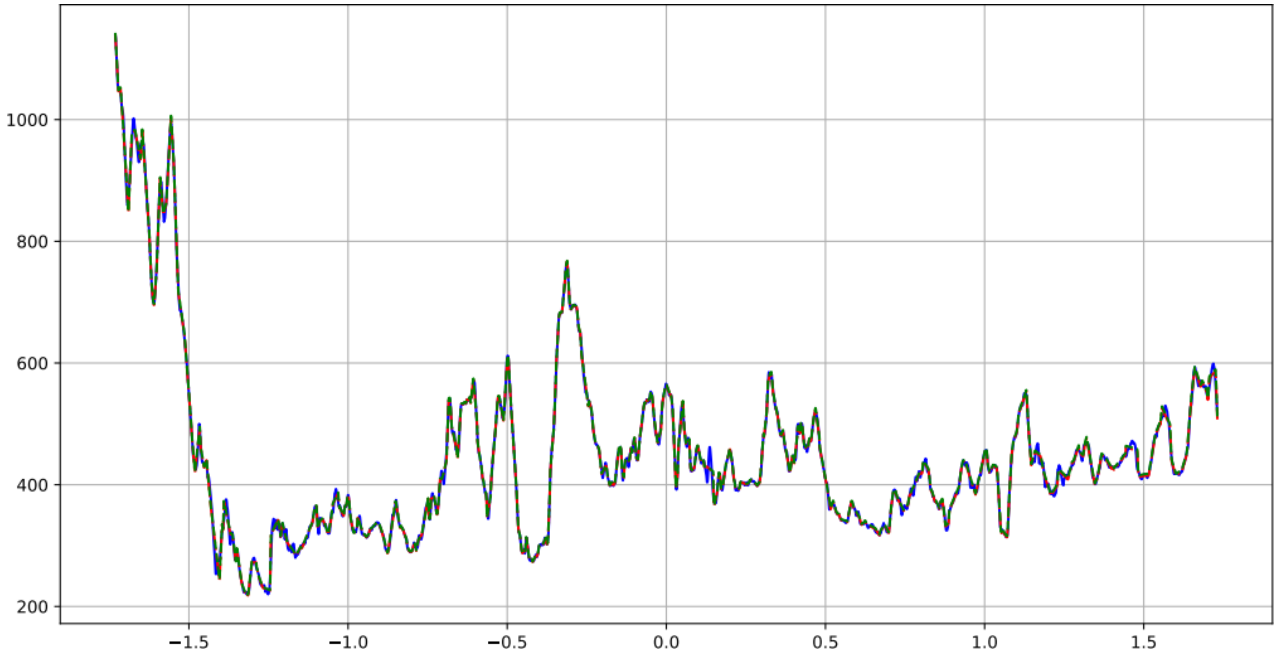
MLP à une couche cachée en 1D

Optimisation d'un MLP sur un profil de terrain : $\mathrm{H} = 100$



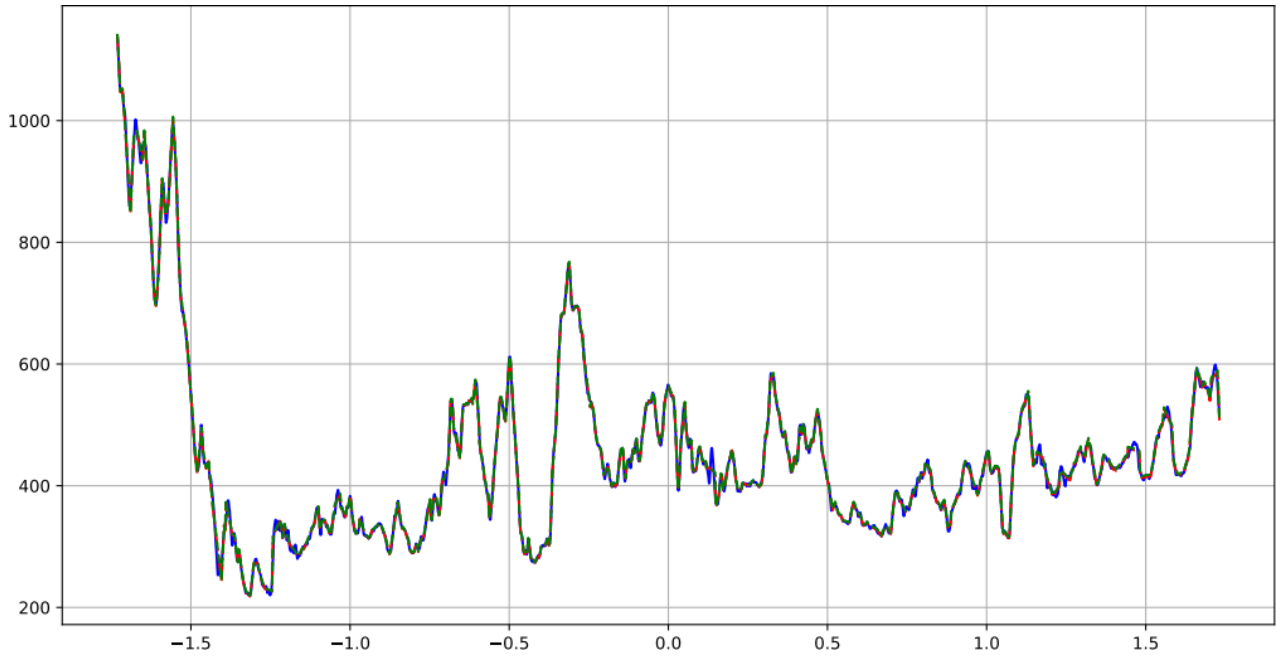
MLP à une couche cachée en 1D

Optimisation d'un MLP sur un profil de terrain : $\mathrm{H} = 1000$



MLP à une couche cachée en 1D

Optimisation d'un MLP sur un profil de terrain : $\mathrm{H} = 1000$



$\rightarrow$ Illustration du théorème d'approximation universelle

(Hornik et al, 1989; Cybenko, 1989)

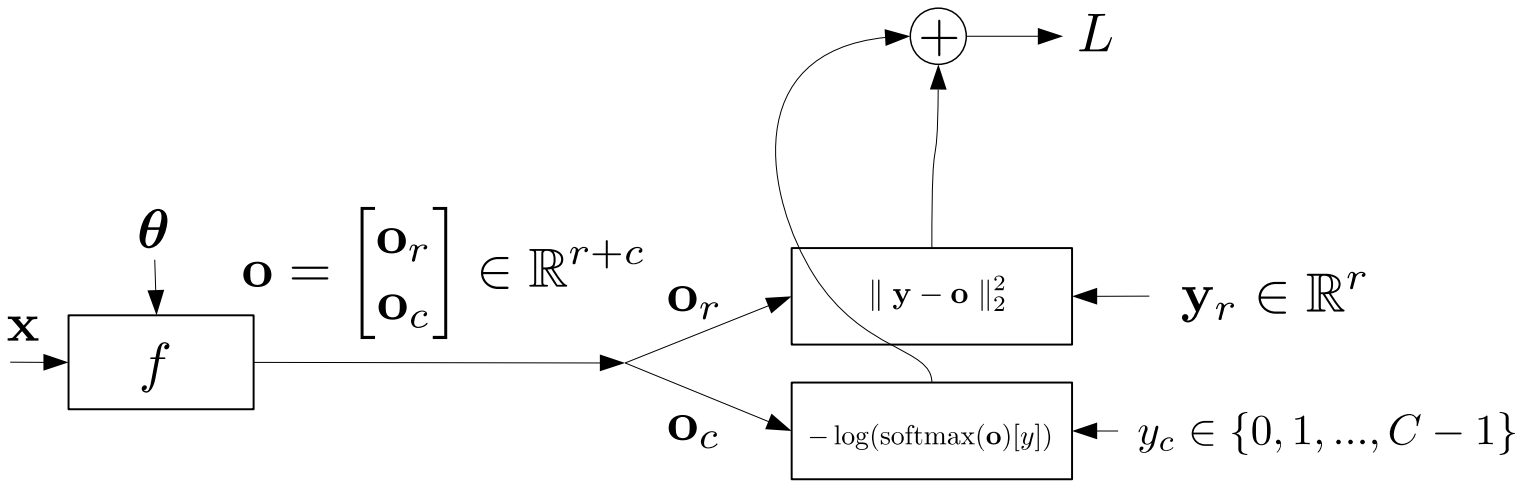
"*Toute fonction continue peut être approximée par un réseau à une couche cachée (shallow), avec un nombre suffisant de neurones.*"

Apprentissage des paramètres

Choix du coût l

- **Régression** : $\mathbf{y} \in \mathbb{R}^n \quad \mathbf{o} \in \mathbb{R}^n$
  - Erreur quadratique : $\|\mathbf{y} - \mathbf{o}\|_2^2 = \sum_{i=0}^{n-1} (\mathbf{y}[i] - \mathbf{o}[i])^2$
  - Somme des valeurs absolues : $\|\mathbf{y} - \mathbf{o}\|_1 = \sum_{i=0}^{n-1} |\mathbf{y}[i] - \mathbf{o}[i]|$
- **Classification** : $y \in \{0, 1, ..., C-1\}, \quad \mathbf{o} \in \mathbb{R}^C$
  - Cross-entropy : $-\ln(\mathrm{softmax}(\mathbf{o})[y])$
- **Combinaison de coûts** : $L = -\ln(\mathrm{softmax}(\mathbf{o})[y]) + \lambda\|\mathbf{y} - \mathbf{o}\|_2^2$

  avec λ hyperparamètre à déterminer



Objectif : Déterminer θ^*

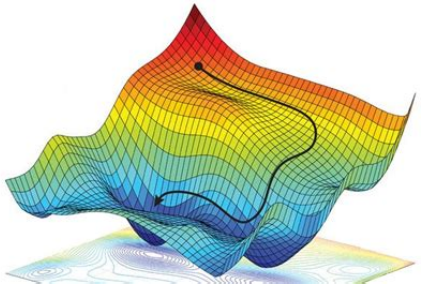
- On a $\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} L(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{N} l(\mathtt{Y}_{\text{train},i}, f(\mathtt{X}_{\text{train},i}; \boldsymbol{\theta}))$

  avec $f(\mathbf{x}; \boldsymbol{\theta}) = f_L\left(f_{L-1}\left(\ldots f_2\left(f_1\left(\mathbf{x}; \boldsymbol{\theta}^{(1)}\right); \boldsymbol{\theta}^{(2)}\right)\ldots; \boldsymbol{\theta}^{(L-1)}\right); \boldsymbol{\theta}^{(L)}\right)$

  Déterminer $\boldsymbol{\theta}^*$ implique que $\dfrac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \left[\dfrac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^{(1)}}, \ldots, \dfrac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^{(L)}}\right]^\top = 0$

- Pour cela on effectue une **Descente de gradient** :
  Mise à jour des paramètres par pas dans la direction opposée du gradient de la fonction de coût :

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}$$



avec $\alpha > 0$ hyperparamètre appelé **pas d'apprentissage** (*learning rate*)

Calcul du gradient

Rappel : Théorème de dérivation des fonctions composées

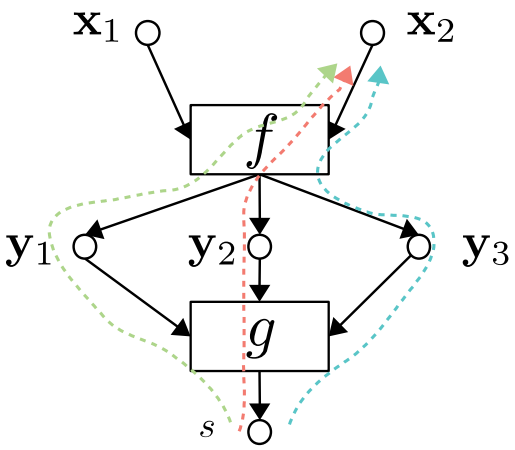
f dérivable sur $\mathbf{I}$ et g dérivable sur $f(\mathbf{I})$: $\quad (g \circ f)' = (g' \circ f) \times f'$

Calcul du gradient

Rappel : Théorème de dérivation des fonctions composées

f dérivable sur $\mathbf{I}$ et g dérivable sur $f(\mathbf{I})$: $\quad (g \circ f)' = (g' \circ f) \times f'$

Exemple :



$$\frac{\partial s}{\partial \mathbf{x}_2} = \boxed{\frac{\partial s}{\partial \mathbf{y}_1}\bigg|_{\mathbf{y}=f(\mathbf{x})} \frac{\partial \mathbf{y}_1}{\partial \mathbf{x}_2}} + \boxed{\frac{\partial s}{\partial \mathbf{y}_2}\bigg|_{\mathbf{y}=f(\mathbf{x})} \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}_2}} + \boxed{\frac{\partial s}{\partial \mathbf{y}_3}\bigg|_{\mathbf{y}=f(\mathbf{x})} \frac{\partial \mathbf{y}_3}{\partial \mathbf{x}_2}}$$

Formule : Dérivation élément par élément

$$\frac{\partial s}{\partial \mathbf{x}_i} = \sum_{j=1}^{M} \frac{\partial s}{\partial \mathbf{y}_j}\bigg|_{\mathbf{y}=f(\mathbf{x})} \frac{\partial \mathbf{y}_j}{\partial \mathbf{x}_i}$$

Calcul du gradient

Rappel : Théorème de dérivation des fonctions composées

f dérivable sur $\mathbf{I}$ et g dérivable sur $f(\mathbf{I})$: $\quad (g \circ f)' = (g' \circ f) \times f'$

$$\frac{\partial s}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial s}{\partial \mathbf{x}_1} \\ \frac{\partial s}{\partial \mathbf{x}_2} \\ \frac{\partial s}{\partial \mathbf{x}_3} \\ \vdots \end{bmatrix}^\top = \begin{bmatrix} \sum_{j=1}^{M} \frac{\partial s}{\partial \mathbf{y}_j}\Big|_{\mathbf{y}=f(\mathbf{x})} \frac{\partial \mathbf{y}_j}{\partial \mathbf{x}_1} \\ \sum_{j=1}^{M} \frac{\partial s}{\partial \mathbf{y}_j}\Big|_{\mathbf{y}=f(\mathbf{x})} \frac{\partial \mathbf{y}_j}{\partial \mathbf{x}_2} \\ \sum_{j=1}^{M} \frac{\partial s}{\partial \mathbf{y}_j}\Big|_{\mathbf{y}=f(\mathbf{x})} \frac{\partial \mathbf{y}_j}{\partial \mathbf{x}_3} \\ \vdots \end{bmatrix}^\top = \begin{bmatrix} \frac{\partial s}{\partial \mathbf{y}_1}\Big|_{\mathbf{y}=f(\mathbf{x})} \\ \frac{\partial s}{\partial \mathbf{y}_2}\Big|_{\mathbf{y}=f(\mathbf{x})} \\ \vdots \end{bmatrix}^\top \begin{bmatrix} \frac{\partial \mathbf{y}_1}{\partial \mathbf{x}_1} & \frac{\partial \mathbf{y}_1}{\partial \mathbf{x}_2} & \frac{\partial \mathbf{y}_1}{\partial \mathbf{x}_3} & \cdots \\ \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}_1} & \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}_2} & \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}_3} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} = \underbrace{\frac{\partial s}{\partial \mathbf{y}}\Big|_{\mathbf{y}=f(\mathbf{x})} \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}}_{\doteq \tilde{f}(\mathbf{x}, \frac{\partial s}{\partial \mathbf{y}}|_{\mathbf{y}=f(\mathbf{x})})}$$

$$\frac{\partial s}{\partial \mathbf{y}} = \begin{bmatrix} \frac{\partial s}{\partial \mathbf{y}_1} \\ \frac{\partial s}{\partial \mathbf{y}_2} \\ \vdots \end{bmatrix}^\top = \underbrace{1 \cdot \frac{\partial g(\mathbf{y})}{\partial \mathbf{y}}}_{\doteq \tilde{g}(\mathbf{y}, 1)}$$

$$\frac{\partial s}{\partial \mathbf{x}} = \tilde{f}\left(\mathbf{x}, \tilde{g}\left(\mathbf{y}, 1\right)\right)$$

Composition de fonctions

=

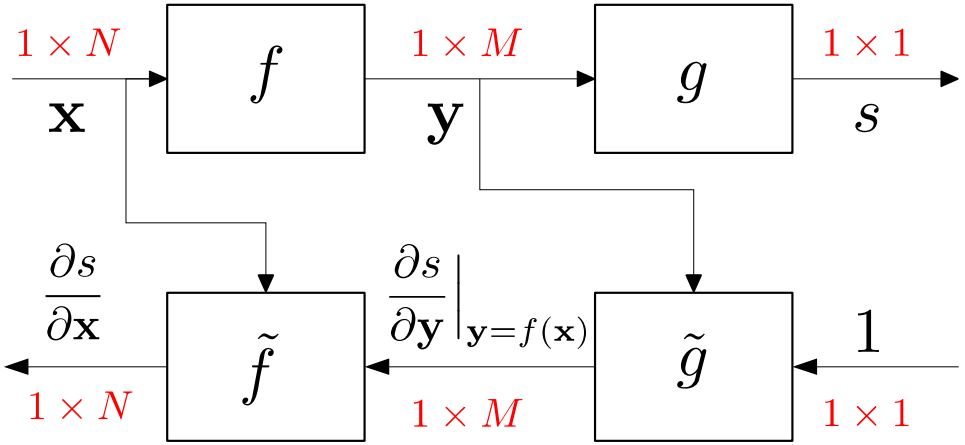
Réseaux de neurones

Calcul du gradient

Rappel : Théorème de dérivation des fonctions composées

f dérivable sur $\mathbf{I}$ et g dérivable sur $f(\mathbf{I})$: $\quad (g \circ f)' = (g' \circ f) \times f'$

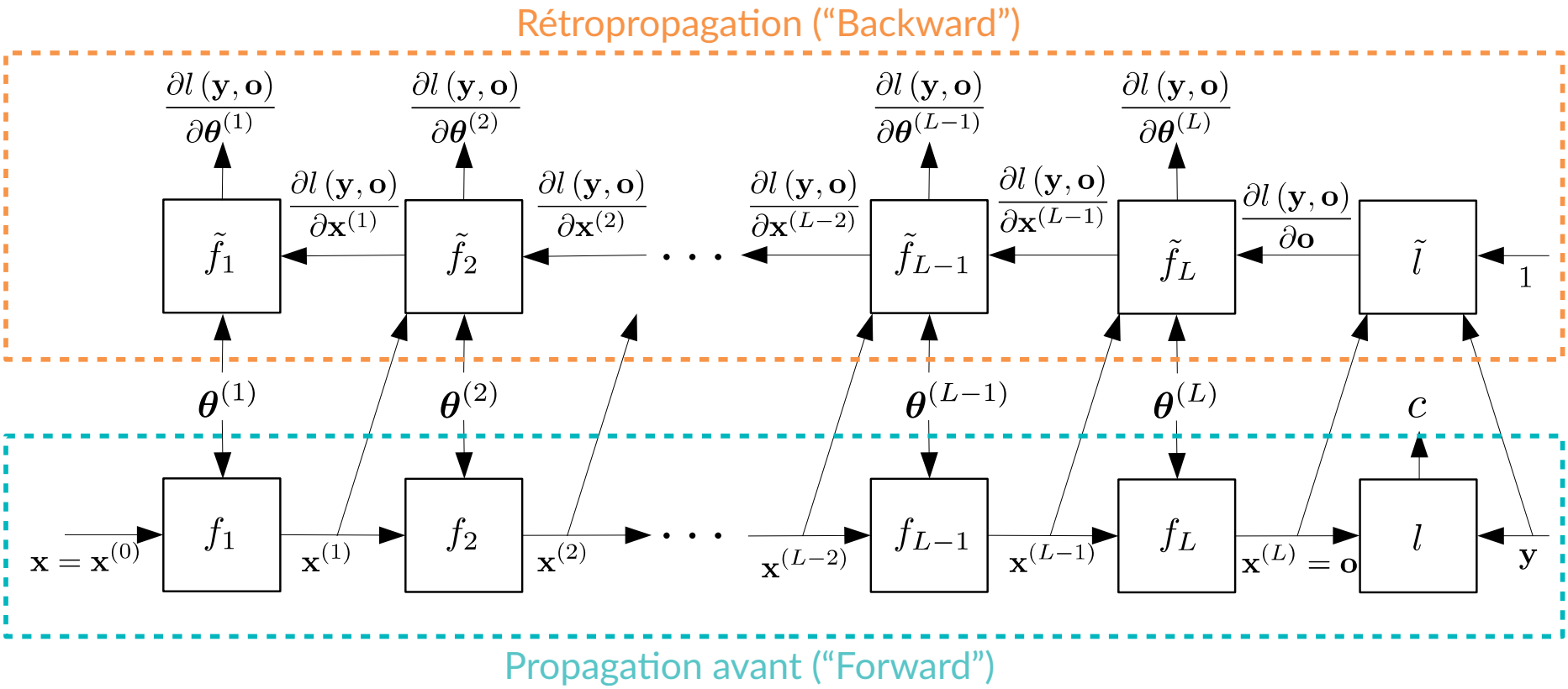
$$\frac{\partial s}{\partial \mathbf{x}} = \tilde{f}\left(\mathbf{x}, \tilde{g}\left(\mathbf{y}, 1\right)\right)$$



Calcul du gradient

Programmation différentielle

Rétropropagation des gradients à travers la chaîne de fonctions paramétriques



Calcul de gradient - Exemple

Considérons l'équation suivante pour $\mathbf{x} \in \mathbb{R}^N$ et $y \in \mathbb{R}$:

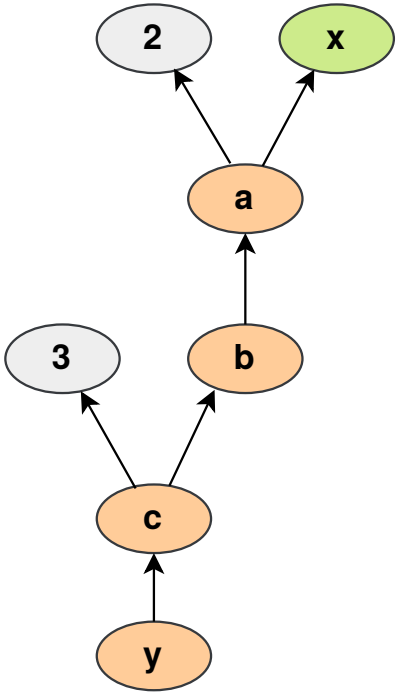
$$y = \frac{1}{N} \sum_{i=0}^{N-1} [(x_i + 2)^2 + 3]$$

Calcul de gradient - Exemple

Considérons l'équation suivante pour $\mathbf{x} \in \mathbb{R}^N$ et $y \in \mathbb{R}$:

$$y = \frac{1}{N} \sum_{i=0}^{N-1} [(x_i + 2)^2 + 3]$$

Graphe de calcul :
(sens inverse)



Calcul de gradient - Exemple

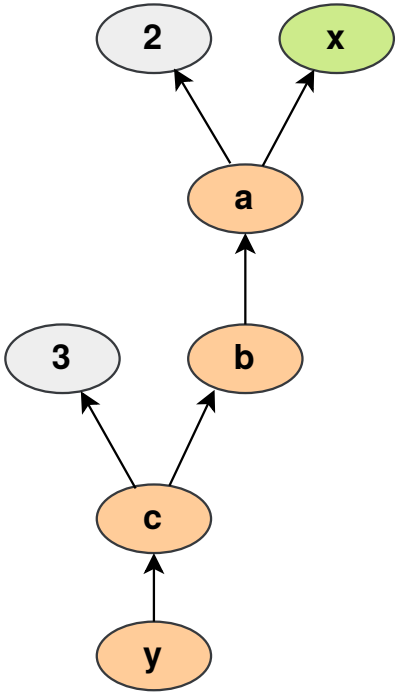
Considérons l'équation suivante pour $\mathbf{x} \in \mathbb{R}^N$ et $y \in \mathbb{R}$:

$$y = \frac{1}{N} \sum_{i=0}^{N-1} [(x_i + 2)^2 + 3]$$

D'après la règle de compositions on a :

$$\frac{\partial y}{\partial x_i} = \frac{\partial y}{\partial c_i} \frac{\partial c_i}{\partial b_i} \frac{\partial b_i}{\partial a_i} \frac{\partial a_i}{\partial x_i}$$

Graphe de calcul :
(sens inverse)



Calcul de gradient - Exemple

Considérons l'équation suivante pour $\mathbf{x} \in \mathbb{R}^N$ et $y \in \mathbb{R}$:

$$y = \frac{1}{N} \sum_{i=0}^{N-1} [(x_i + 2)^2 + 3]$$

D'après la règle de compositions on a :

$$\frac{\partial y}{\partial x_i} = \frac{\partial y}{\partial c_i} \frac{\partial c_i}{\partial b_i} \frac{\partial b_i}{\partial a_i} \frac{\partial a_i}{\partial x_i}$$

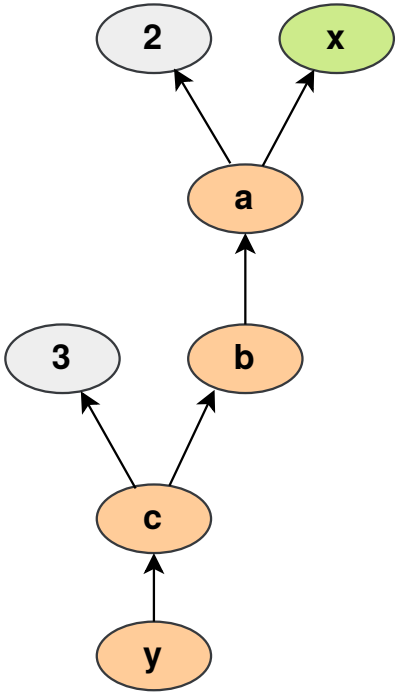
Pour $\mathbf{x} = [0, 1, 2]$,

$$\frac{\partial a_i}{\partial x_i} = 1 \quad \frac{\partial b_i}{\partial a_i} = 2.a_i \quad \frac{\partial c_i}{\partial b_i} = 1 \quad \frac{\partial y}{\partial c_i} = \frac{1}{3}$$

Et,

$$\frac{\partial y}{\partial \mathbf{x}} = [\frac{4}{3}, 2, \frac{8}{3}]$$

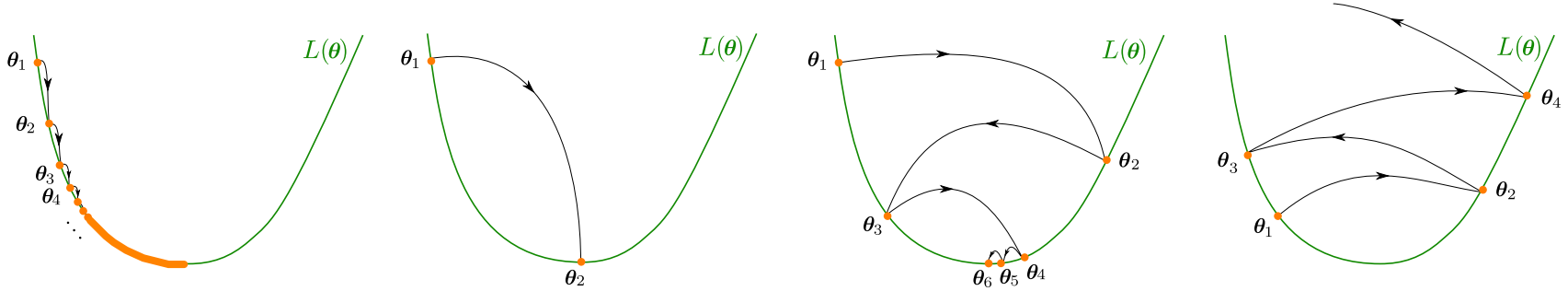
Graphe de calcul :
(sens inverse)



Descente de gradient

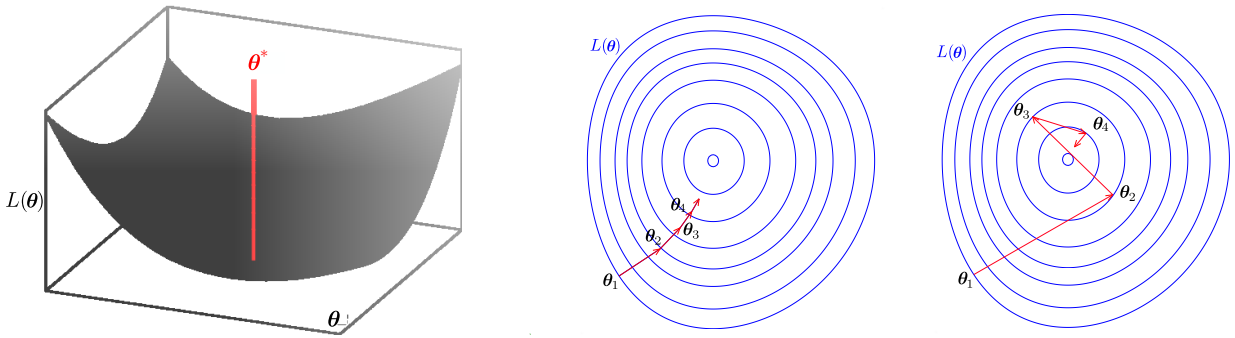
Optimisation : $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}$ avec α pas d'apprentissage

Exemples (Dimension 1)

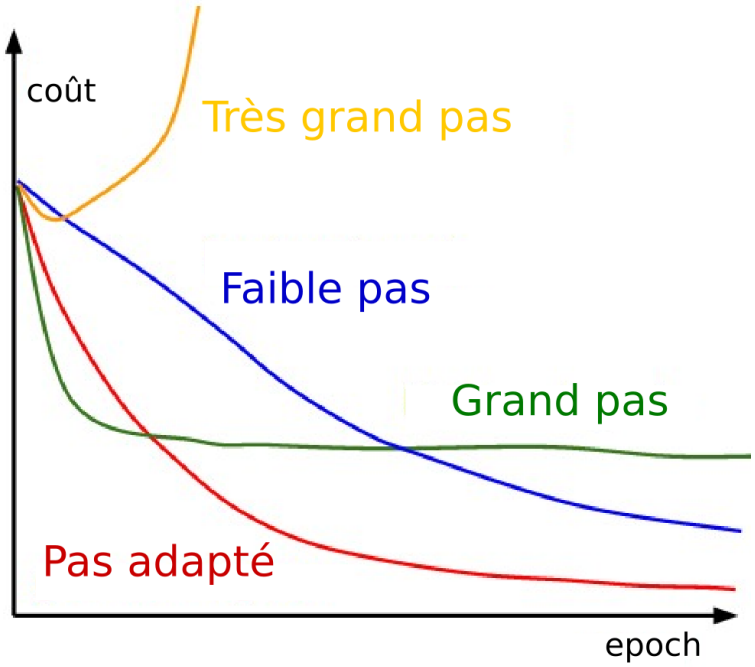


Petit pas	Pas adapté	Grand pas	Trop grand pas
Convergence lente	Convergence rapide	Convergence lente	Divergence

Exemples (Dimension 2)



Infuence du pas d'apprentissage



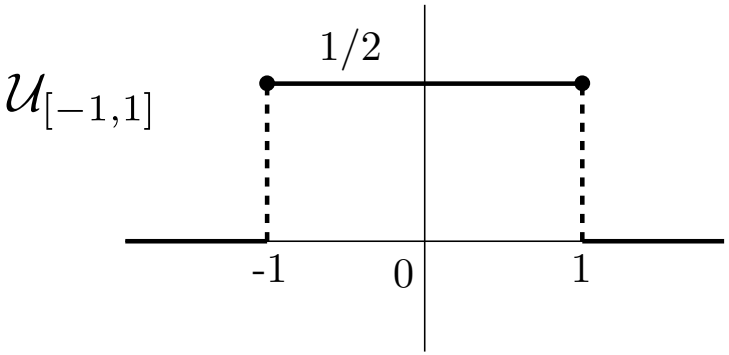
- **En pratique**
  Tester différentes valeurs du pas d'apprentissage en visualisant à chaque fois l'évolution du coût d'apprentissage, et du coût de validation (sur des données ne servant pas au calcul de l'optimisation)

Initialisation des paramètres

Loi Gaussienne ou uniforme

- Initialisation de *Xavier*[1] (loi uniforme $\mathcal{U}$)

$$\mathtt{W}^{(i)} = \frac{1}{\sqrt{N_{i-1}}}\mathcal{U}_{[-1,1]}(N_i, N_{i-1}) \qquad \mathbf{b}^{(i)} = \frac{1}{\sqrt{N_{i-1}}}\mathcal{U}_{[-1,1]}(N_i)$$



- *Kaiming* initialization[2]
- Autres...[3]

1. [Glorot et al., 2010] - Understanding the difficulty of training deep feedforward neural networks. PMLR.
2. [He et al., 2015] - Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. ICCV.
3. [Mishkin, et al., 2016] - All you need is a good init. ICLR.

Apprentissage sur une grande base de données

Descente de gradient (GD) : $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \sum_{i=1}^{N_{\text{train}}} \frac{\partial l(\mathtt{Y}_{\text{train},i}, f(\mathtt{X}_{\text{train},i}; \boldsymbol{\theta}))}{\partial \boldsymbol{\theta}} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}$

Problème

La *forward pass* doit être calculée pour toutes les données d'entraînement
$\rightarrow$ Trop lent et trop coûteux en mémoire pour de grands datasets

Solution

Descente de gradient stochastique (SGD) :

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \sum_{i \in \Omega_k} \frac{\partial l(\mathtt{Y}_{\text{train},i}, f(\mathtt{X}_{\text{train},i}; \boldsymbol{\theta}))}{\partial \boldsymbol{\theta}} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}$$

- Décomposition <u>aléatoire</u> de l'ensemble d'entraînement en sous-ensembles Ω_k de même taille (**batch**)
- Chaque sous-ensemble aléatoire est appelé ***minibatch***
- Une ***epoch*** est quand chaque exemple de l'ensemble d'entraînement a été parcouru au moins une fois
- Apprentissage des paramètres par itération des epochs

Avantages et inconvénients de la SGD

Le gradient de la SGD est "bruité" car calculé sur un batch

- **Inconvénient :**
  L'apprentissage peut être compliqué/lent si le "bruit" est trop important (exemple : taille du minibatch trop faible)

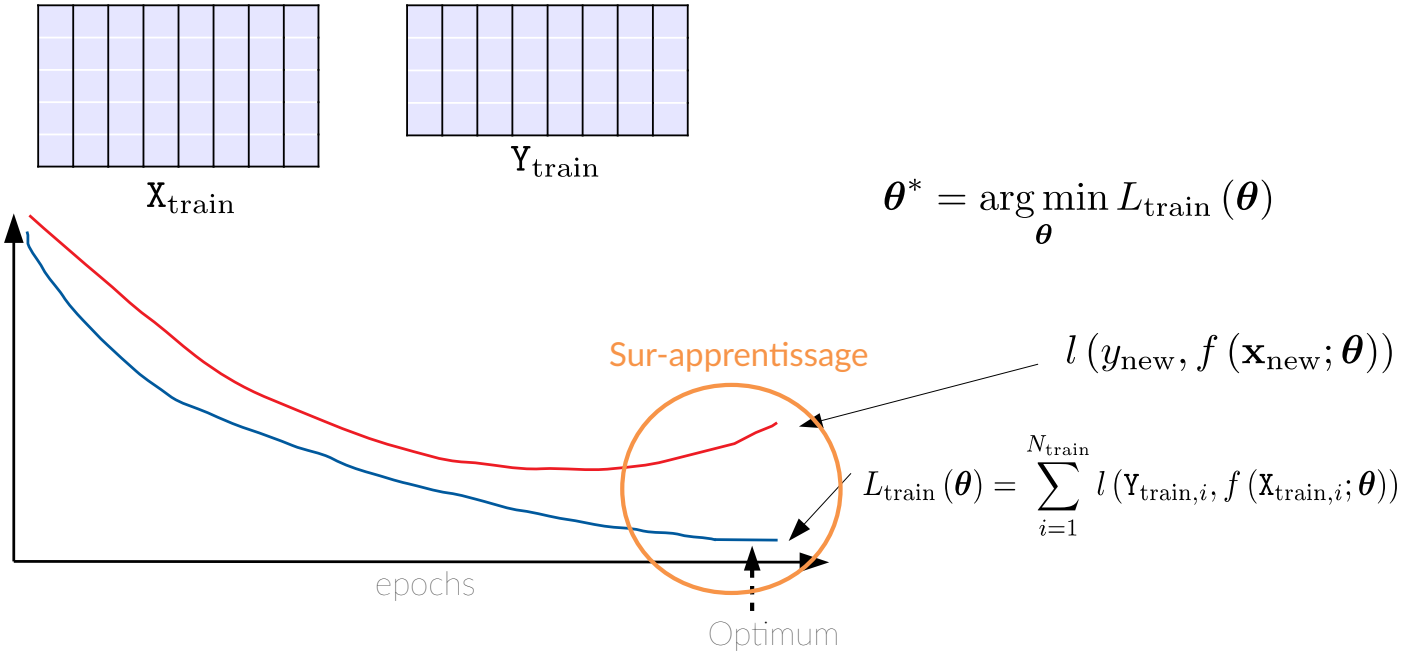
- **Avantage 1** :
  Ce "bruit" peut permettre de sortir ou d'éviter de mauvais minima locaux

- **Avantage 2** :
  Le gradient est très rapide à calculer et la demande en mémoire peut être satisfaite par un GPU

- **Avantage 3 (empirique)** :
  L'utilisation de petits minibatchs (32-512) conduit à une bien meilleure généralisation que l'utilisation de grands minibatchs

Utilisation d’un ensemble de validation

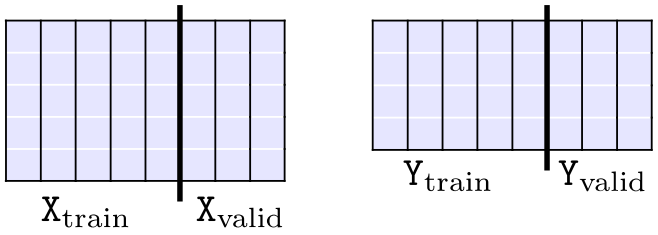
- Risque de sur-apprentissage toujours présent



Apprendre à “bien” prédire sur de nouvelles données

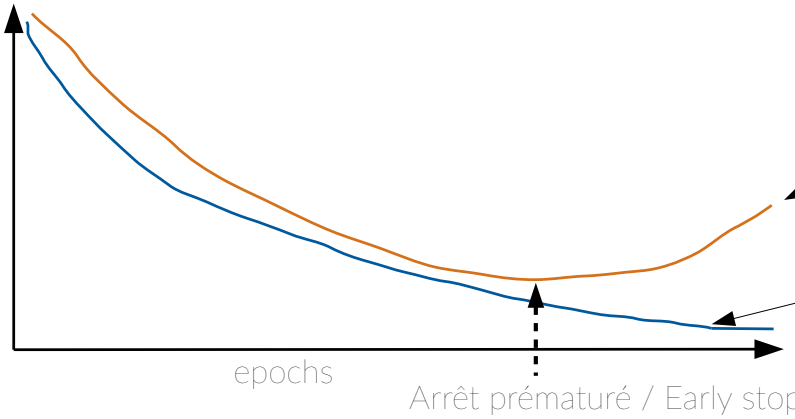
Utilisation d’un ensemble de validation

- Risque de sur-apprentissage toujours présent
- On extrait des données de l’ensemble d’entraînement pour créer un ensemble de **validation** (sur lequel le réseau n’apprendra jamais)
- On suit les deux performances pour déterminer α et calculer $\boldsymbol{\theta}^*$
- Possibilité d’arrêter l’entraînement après un nombre d’epochs successives telles que $L(\boldsymbol{\theta}_{k+1}) > L(\boldsymbol{\theta}^*)$ (***early stopping***)



$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L_{\text{valid}}(\boldsymbol{\theta})$$

$$\cancel{\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L_{\text{train}}(\boldsymbol{\theta})}$$



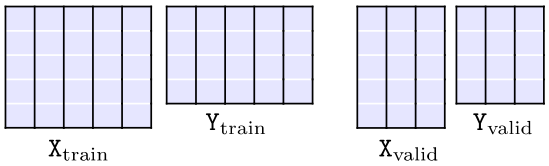
$$L_{\text{valid}}(\boldsymbol{\theta}) = \sum_{i=1}^{N_{\text{valid}}} l\left(\mathtt{Y}_{\text{valid},i}, f\left(\mathtt{X}_{\text{valid},i}; \boldsymbol{\theta}\right)\right)$$

$$L_{\text{train}}(\boldsymbol{\theta}) = \sum_{i=1}^{N_{\text{train}}} l\left(\mathtt{Y}_{\text{train},i}, f\left(\mathtt{X}_{\text{train},i}; \boldsymbol{\theta}\right)\right)$$

Résumé de l'étape d'apprentissage

1) Découper une fois pour toutes la base de données en :

- une base d'apprentissage ("training set")
- une base de validation ("validation set")



2) Lancer une descente de gradient stochastique (SGD) avec arrêt prématuré

- Au début d'une epoch, découper la base d'apprentissage aléatoirement en minibatchs
- SGD sur chaque minibatch : $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \sum_{i \in \Omega_k} \frac{\partial l(\mathtt{Y}_{\text{train},i}, f(\mathtt{X}_{\text{train},i}; \boldsymbol{\theta}))}{\partial \boldsymbol{\theta}} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}$
- À la fin d'une epoch, calculer : $L_{\text{valid}}(\boldsymbol{\theta}) = \sum_{i=1}^{N_{\text{valid}}} l(\mathtt{Y}_{\text{valid},i}, f(\mathtt{X}_{\text{valid},i}; \boldsymbol{\theta}))$
- Stocker la valeur actuelle de $\boldsymbol{\theta}$ si le coût de validation est plus faible que le précédent meilleur coût
- Stopper l'entraînement lorsqu'on est en régime de sur-apprentissage

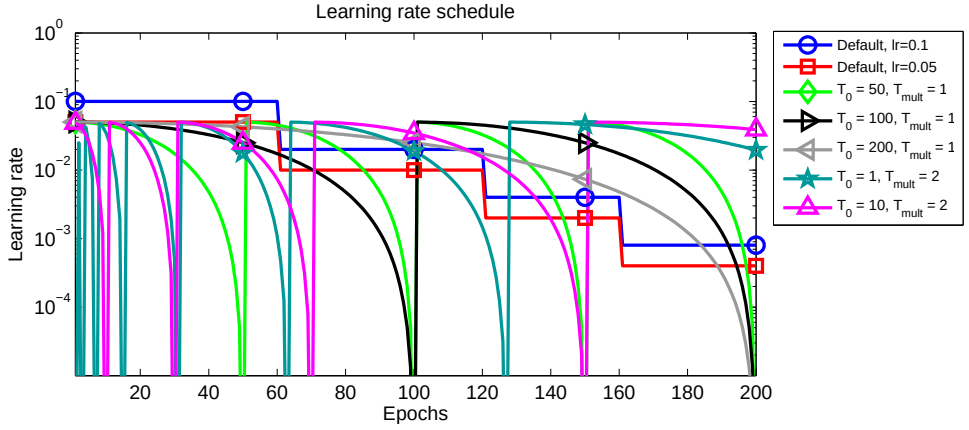
Bonnes pratiques

- Lancer un apprentissage sur un seul minibatch jusqu'à obtention d'un coût d'apprentissage de zéro
  - Est-ce que le réseau est fonctionnel et capable d'apprendre par cœur ?
- Visualiser tout ce qu'il est possible de visualiser
  - Entrées $\rightarrow$ plage de valeurs (erreur classique : données non normalisées)
  - Sorties
  - Valeurs des paramètres
  - Valeur du pas d'apprentissage
  - Coûts (d'apprentissage, de validation, ...)
  - Gradients
  - ...

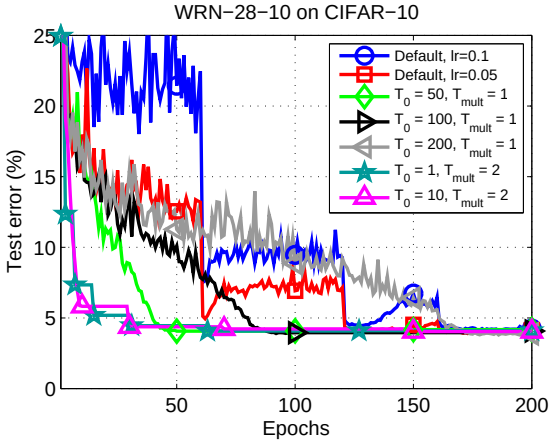
Évolution du pas d'apprentissage

Différentes stratégies d'évolution du pas d'apprentisage au cours de l'optimisation sont possibles[1] :

- Constant
- Décroissant
- Cyclique
- Réduction sur plateau



→ Vitesses de convergence différentes



1. [Loshchilov et al., 2016] - SGDR: Stochastic gradient descent with warm restarts. arXiv preprint.

SGD avec moment (*with momentum*)

SGD

$$\mathbf{g}_{k+1} = \sum_{i \in \Omega_k} \frac{\partial l(\mathtt{Y}_{\text{train},i}, f(\mathtt{X}_{\text{train},i}; \boldsymbol{\theta}))}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}$$

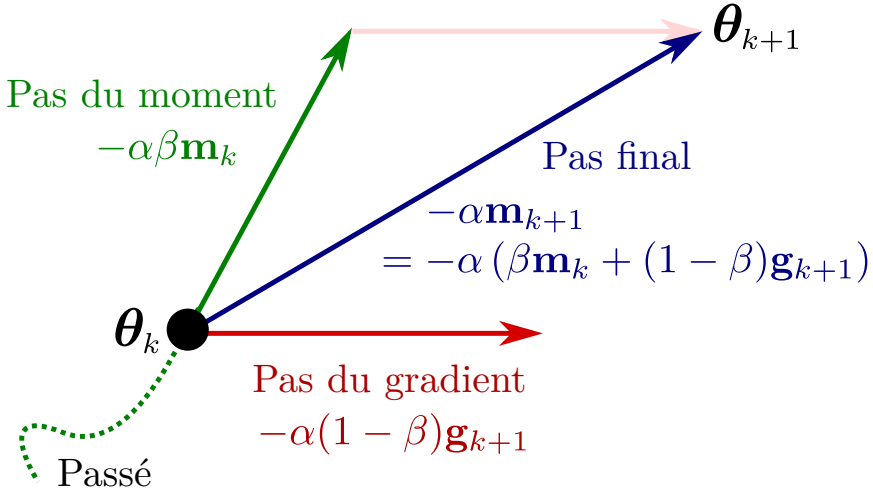
$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \mathbf{g}_{k+1}$$

SGD avec moment

$$\mathbf{g}_{k+1} = \sum_{i \in \Omega_k} \frac{\partial l(\mathtt{Y}_{\text{train},i}, f(\mathtt{X}_{\text{train},i}; \boldsymbol{\theta}))}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}$$

$$\mathbf{m}_{k+1} = \beta \mathbf{m}_k + (1-\beta)\mathbf{g}_{k+1}$$

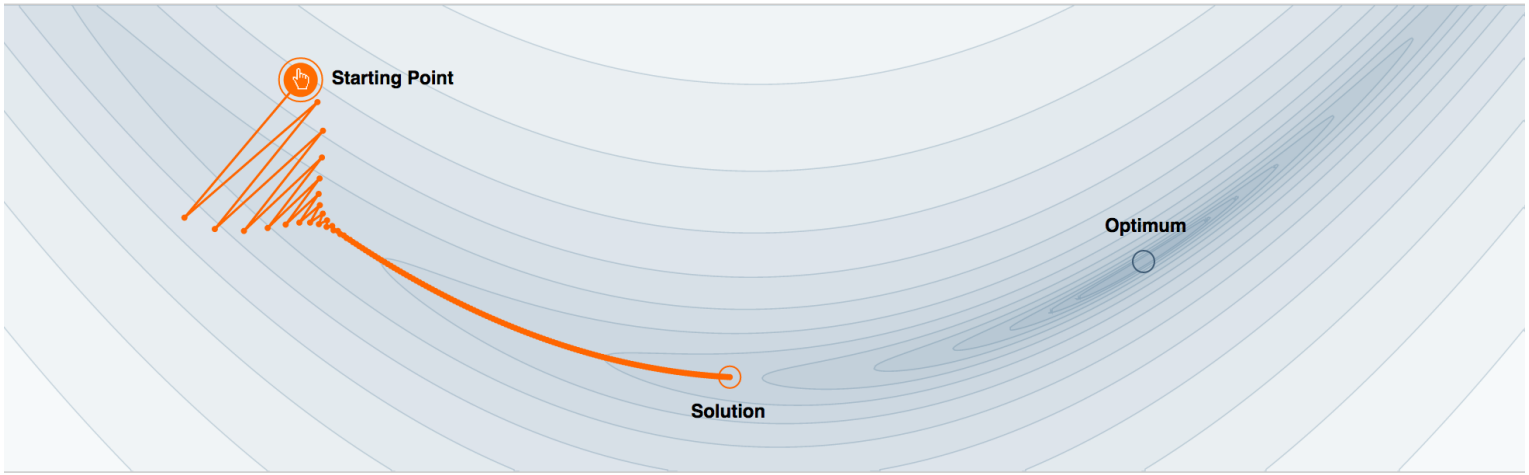
$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \mathbf{m}_{k+1}$$



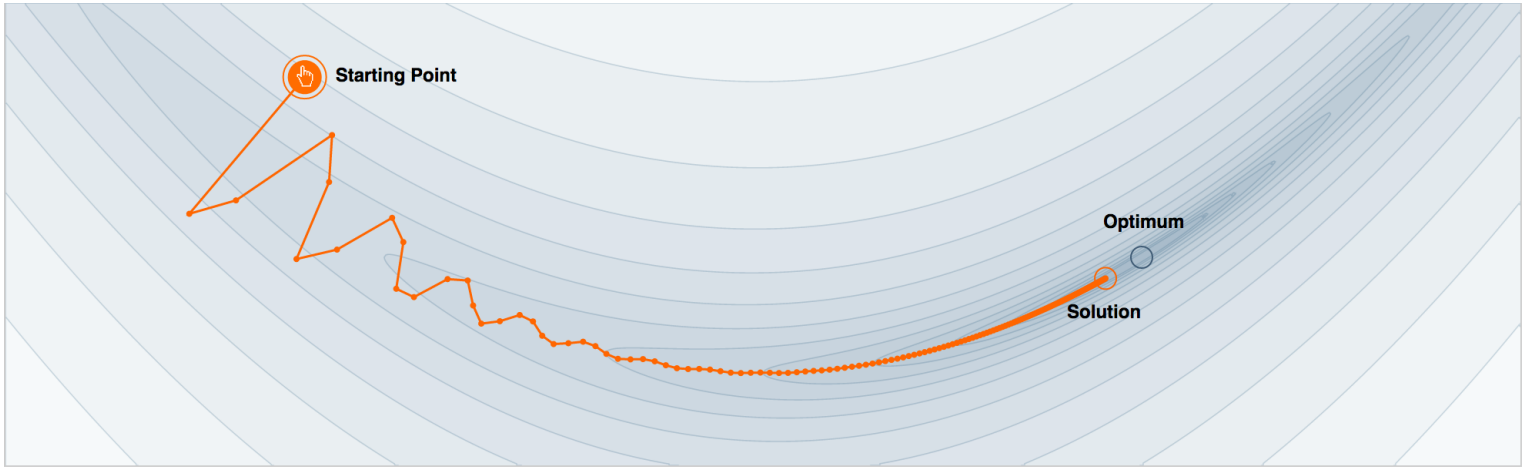
$\rightarrow$ Débruitage des gradients de la SGD en utilisant l'inertie ($\beta \approx 0.9$)

Optimisation des paramètres

SGD



SGD avec moment



"Adam: A Method for Stochastic Gradient Descent" [1]

Idée : Avoir un pas d'apprentissage par paramètre

$$\mathbf{g}_{k+1} = \sum_{i \in \Omega_k} \frac{\partial l(\mathtt{Y}_{\text{train},i}, f(\mathtt{X}_{\text{train},i}; \boldsymbol{\theta}))}{\partial \boldsymbol{\theta}} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}$$

$$\mathbf{m}_{k+1} = \frac{1}{1-\beta_1^k} \left(\beta_1 \mathbf{m}_k + (1-\beta_1)\mathbf{g}_{k+1}\right)$$

$$\mathbf{v}_{k+1} = \frac{1}{1-\beta_2^k} \left(\beta_2 \mathbf{v}_k + (1-\beta_2)\mathbf{g}_{k+1}^2\right)$$ avec $\mathbf{g}_{k+1}^2$ carré de chaque élément de $\mathbf{g}_k$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \frac{\mathbf{m}_{k+1}}{\sqrt{\mathbf{v}_{k+1}} + \epsilon}$$ avec $\sqrt{\mathbf{v}_{k+1}}$ racine carré de chaque élément de $\mathbf{v}_{k+1}$

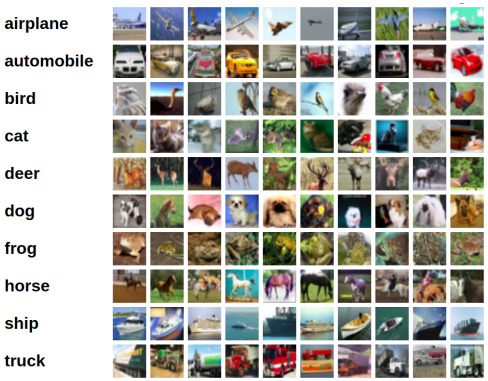
→ Peut permettre d'atteindre plus rapidement un minimum local

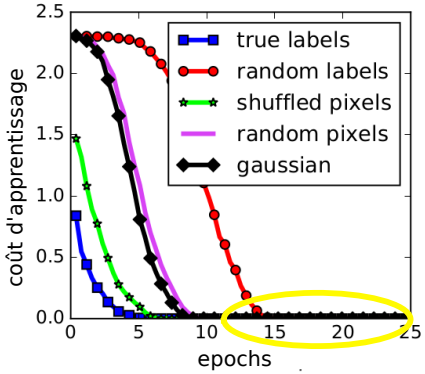
→ Augmentation du coût en mémoire (calcul du carré du gradient)

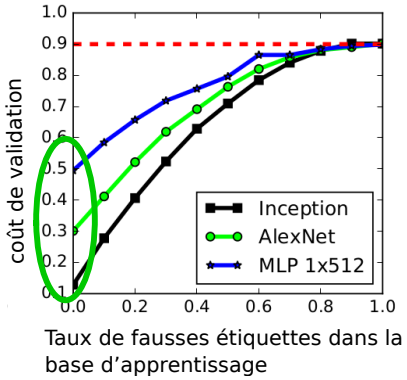
1. [Kingma et al., 2015] - Adam: A method for stochastic gradient descent. In ICLR.

Pourquoi les réseaux de neurones fonctionnent ?

Constat sur CIFAR-10 : Apprentissage "par cœur" de l'association image/étiquettes (O)... mais bonnes performances sur des images de validation (O) [1]







Hypothèses [2] :

1) Biais introduit par l'architecture (*inductive bias*)
2) Biais introduit par la descente de gradient stochastique

1. [Zhang et al., 2017] - Understanding Deep Learning requires rethinking generalization. In ICLR.
2. https://guillefix.me/nnbias/
   https://hackmd.io/75gt3X6WQbu1_A3pF8svWg
   [Valle-Pérez et al., 2019] - Deep learning generalizes because the parameter-function map is biased towards simple functions. In ICLR.
   [Smith et al., 2021] - On the origin of implicit regularization on stochastic gradient descent. In ICLR.

Réseaux de neurones à convolution

Limites d'une transformation affine générale (FC)

Exemple d'une simple transformation affine où la résolution de l'image d'entrée est préservée :

H=480

W=640

$$\mathbf{x} : 640 \times 480 \times 3 \approx 10^6$$

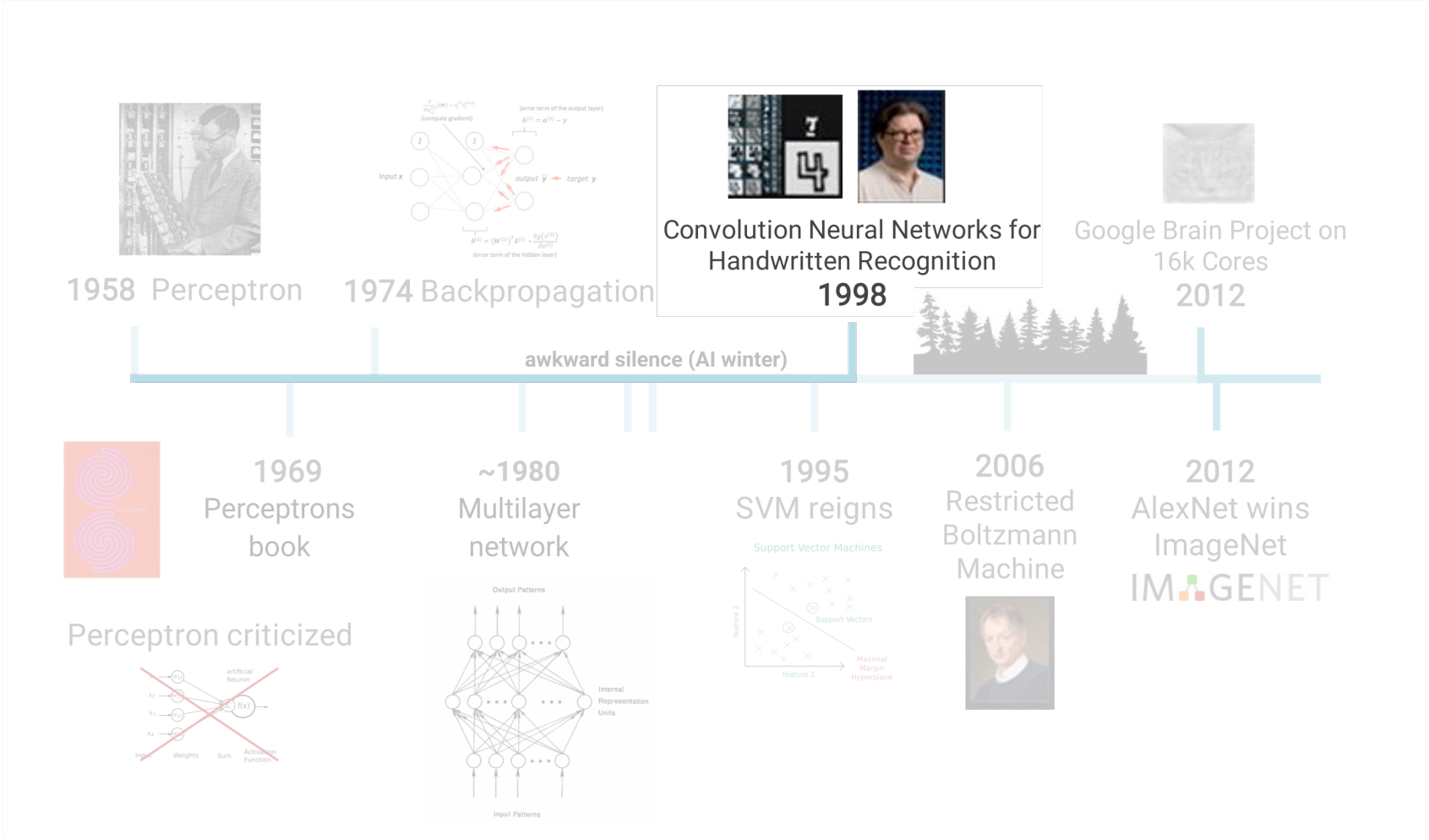
$$\mathbf{y} = \mathtt{W}\mathbf{x} + \mathbf{b}$$

10^6x1 10^6x10^6 10^6x1 10^6x1

→ Occupation mémoire de `W` : 4 octets x 10^6 x 10^6 = 4To

→ Nombre de "multiplications+additions" également très élevé

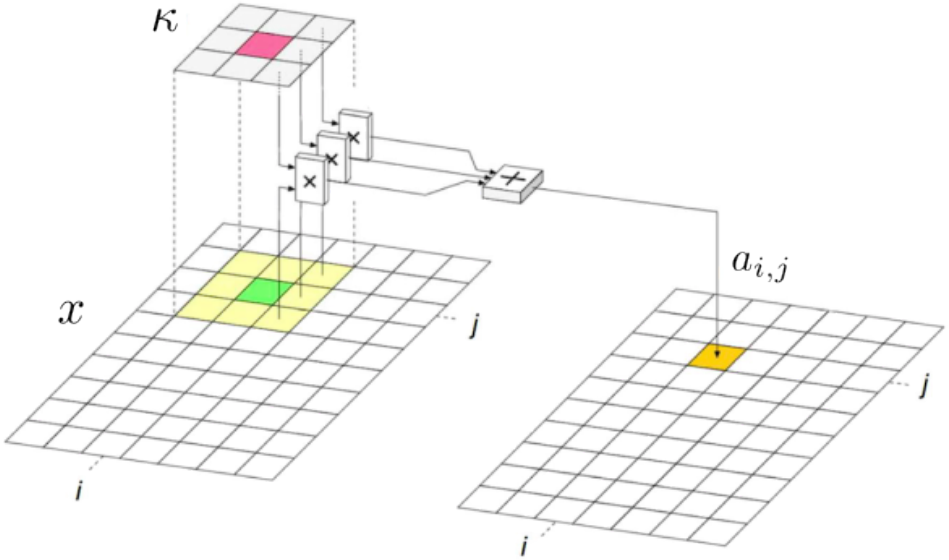
Chronologie



Filtres linéaires : convolutions 2D

- Produit de convolution $I * \kappa$ d'une image I par un filtre κ (aussi appelé noyau (*kernel*):

$$(I * \kappa)(i,j) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} I(i+u, j+v)\, \kappa(u,v)$$



Ici κ de taille $S = 3 \times 3$

$$\Rightarrow \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} \equiv \sum_{u=-1}^{+1} \sum_{v=-1}^{+1}$$

κ est centré sur (i,j)

Exemple de filtre de convolution 2D

Exemple : filtre moyenneur

Moyenne des pixels dans un voisinage local, par exemple avec $S = 3 \times 3$:

$$\kappa = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & \underline{\mathbf{1/9}} & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & \underline{\mathbf{1}} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Exemple : filtre moyenneur

Moyenne des pixels dans un voisinage local, par exemple avec $S = 3 \times 3$:

$$\kappa = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & \underline{\mathbf{1/9}} & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & \underline{\mathbf{1}} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Image originale

Filtre moyenneur 3×3

Exemple : filtre moyenneur

Moyenne des pixels dans un voisinage local, par exemple avec $S = 3 \times 3$:

$$\kappa = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & \underline{\mathbf{1/9}} & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & \underline{\mathbf{1}} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Image originale

Filtre moyenneur 5×5

Exemple : filtre moyenneur

Moyenne des pixels dans un voisinage local, par exemple avec $S = 3 \times 3$:

$$\kappa = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & \underline{\mathbf{1/9}} & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & \underline{\mathbf{1}} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Image originale

Filtre moyenneur 7×7

Convolution étape par étape

$$\text{Filtre } \kappa = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & \underline{\mathbf{1}} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0								

Convolution étape par étape

$$\text{Filtre } \kappa = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & \underline{\mathbf{1}} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10							

Convolution étape par étape

$$\text{Filtre } \kappa = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & \underline{\mathbf{1}} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10	20						

Exemple de filtre de convolution 2D

Convolution étape par étape

$$\text{Filtre } \kappa = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & \underline{\mathbf{1}} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10	20	30					

Exemple de filtre de convolution 2D

Convolution étape par étape

$$\text{Filtre } \kappa = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & \underline{\mathbf{1}} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10	20	30	30				

Exemple de filtre de convolution 2D

Convolution étape par étape

$$\text{Filtre } \kappa = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & \underline{\mathbf{1}} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Convolution étape par étape

$$\text{Filtre } \kappa = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & \underline{\mathbf{1}} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

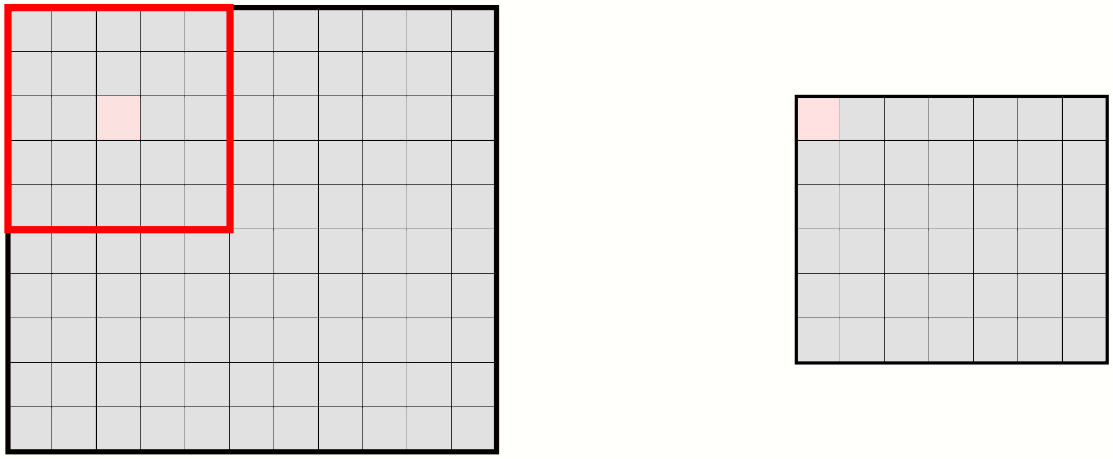
Et les effets de bord ?

Exemple de filtre de convolution 2D

Zero-padding

Ajout de zéros autour de l'image (demi taille du filtre)

$\rightarrow$ même dimension en sortie

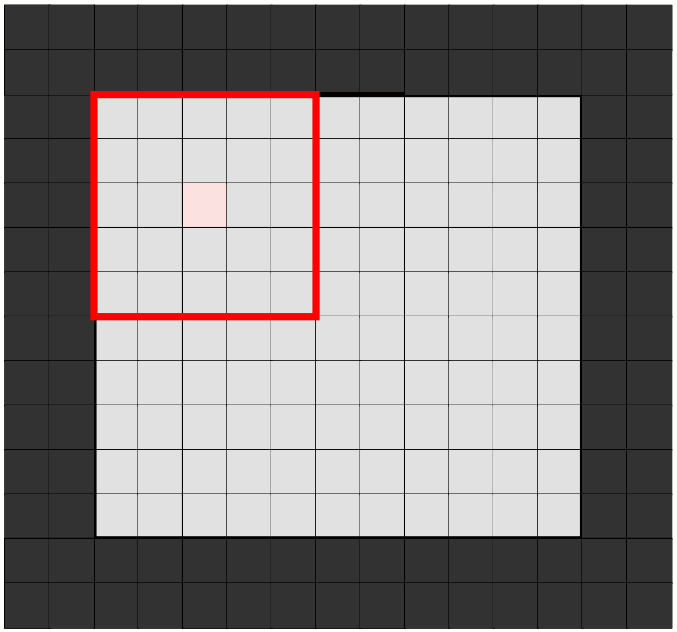


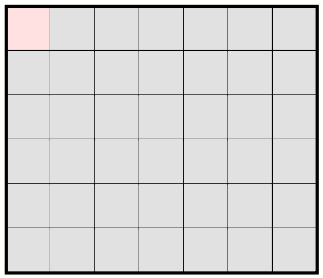
Exemple de filtre de convolution 2D

Zero-padding

Ajout de zéros autour de l'image (demi taille du filtre)

$\rightarrow$ même dimension en sortie



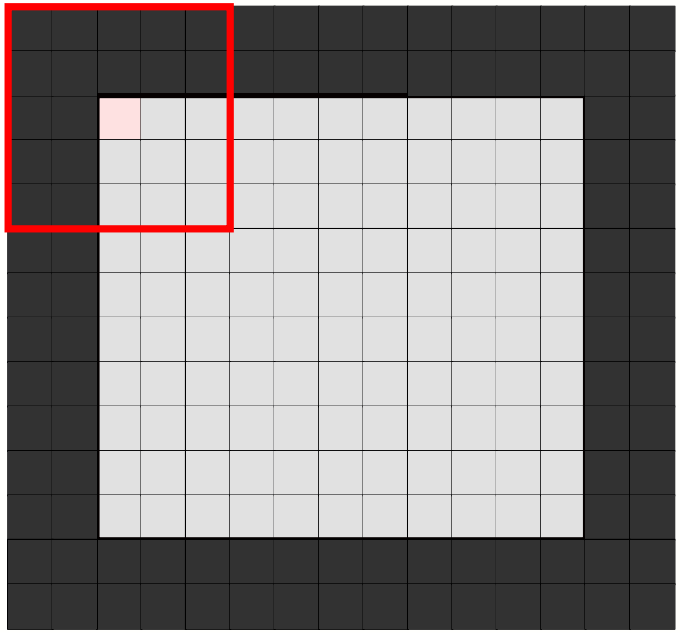


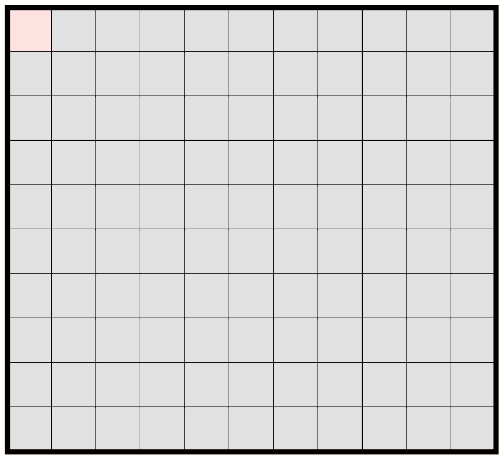
Exemple de filtre de convolution 2D

Zero-padding

Ajout de zéros autour de l'image (demi taille du filtre)

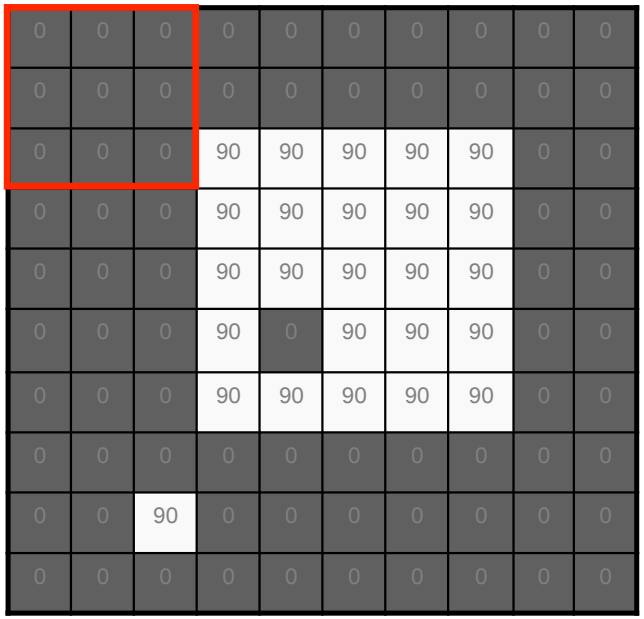
$\rightarrow$ même dimension en sortie

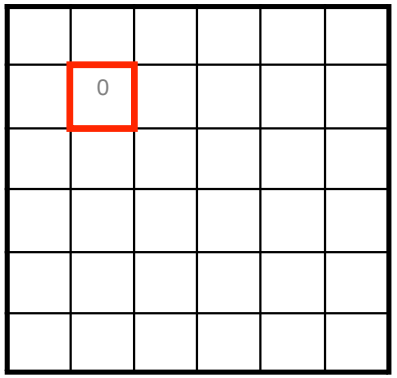




Stride

Saut de position → Réduction de dimension

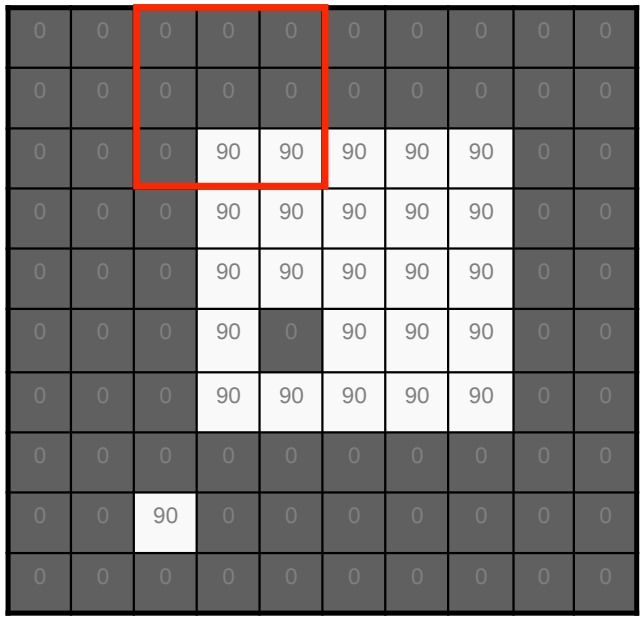


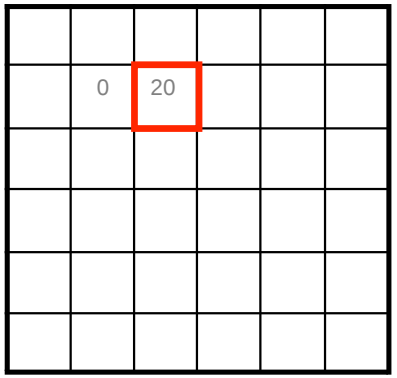


Exemple : stride de 2

Stride

Saut de position → Réduction de dimension

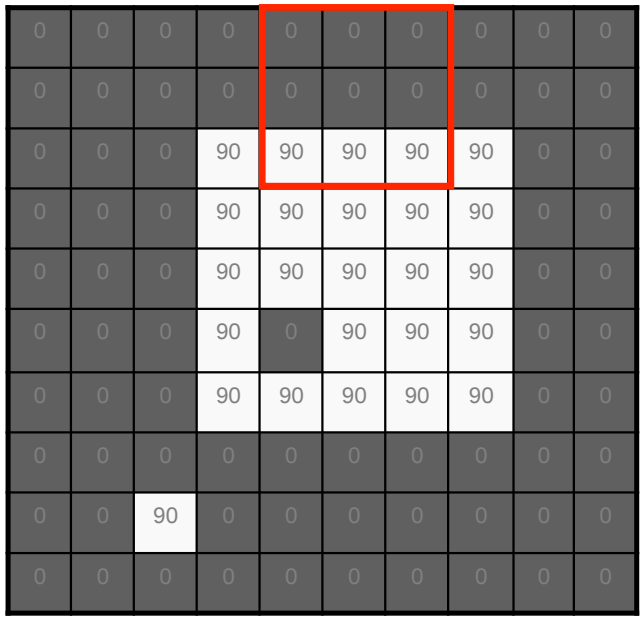


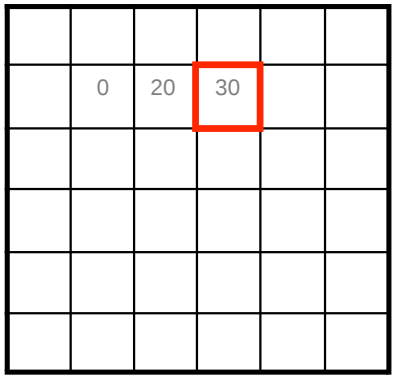


Exemple : stride de 2

Stride

Saut de position → Réduction de dimension

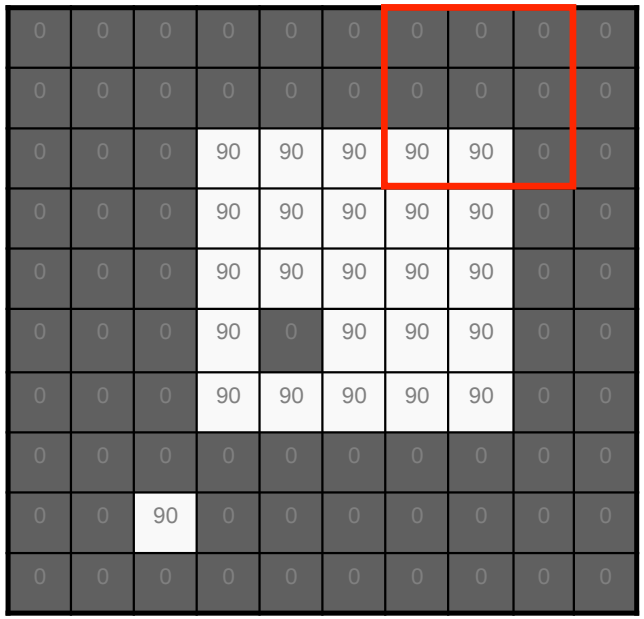


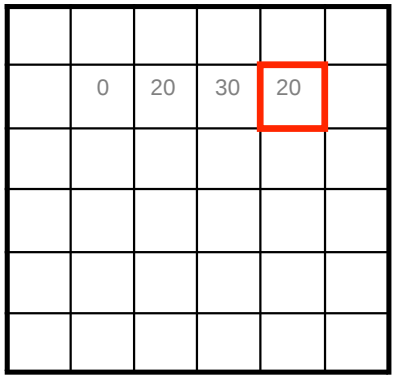


Exemple : stride de 2

Stride

Saut de position → Réduction de dimension





Exemple : stride de 2

Stride

Saut de position → Réduction de dimension

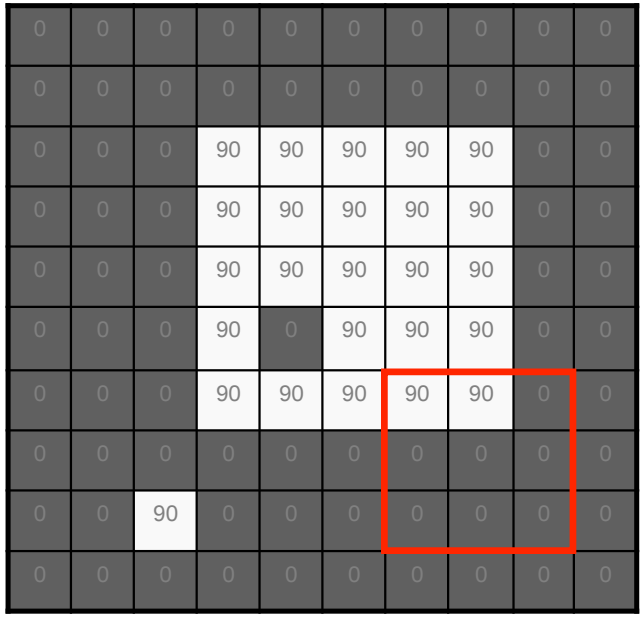
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

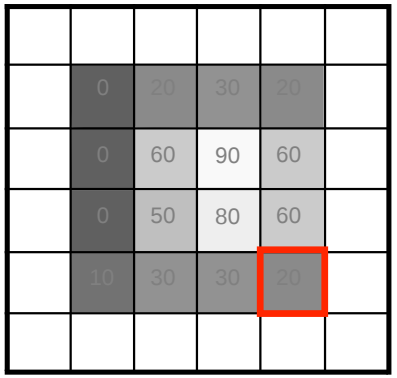
	0	20	30	20	
	0				

Exemple : stride de 2

Stride

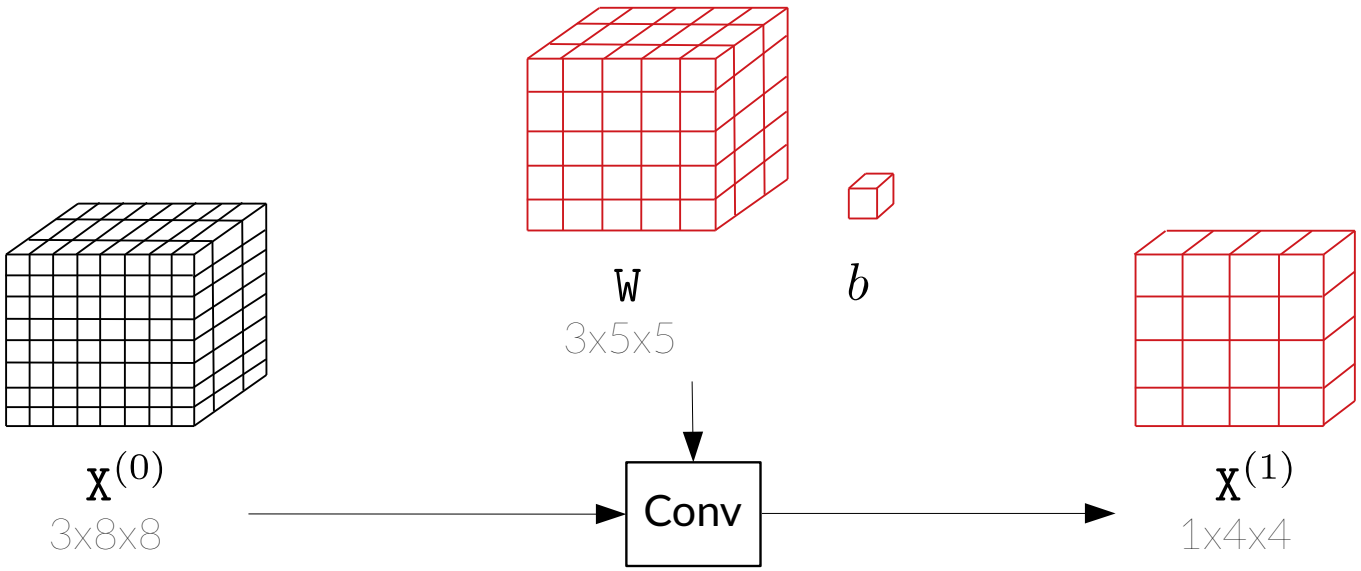
Saut de position → Réduction de dimension





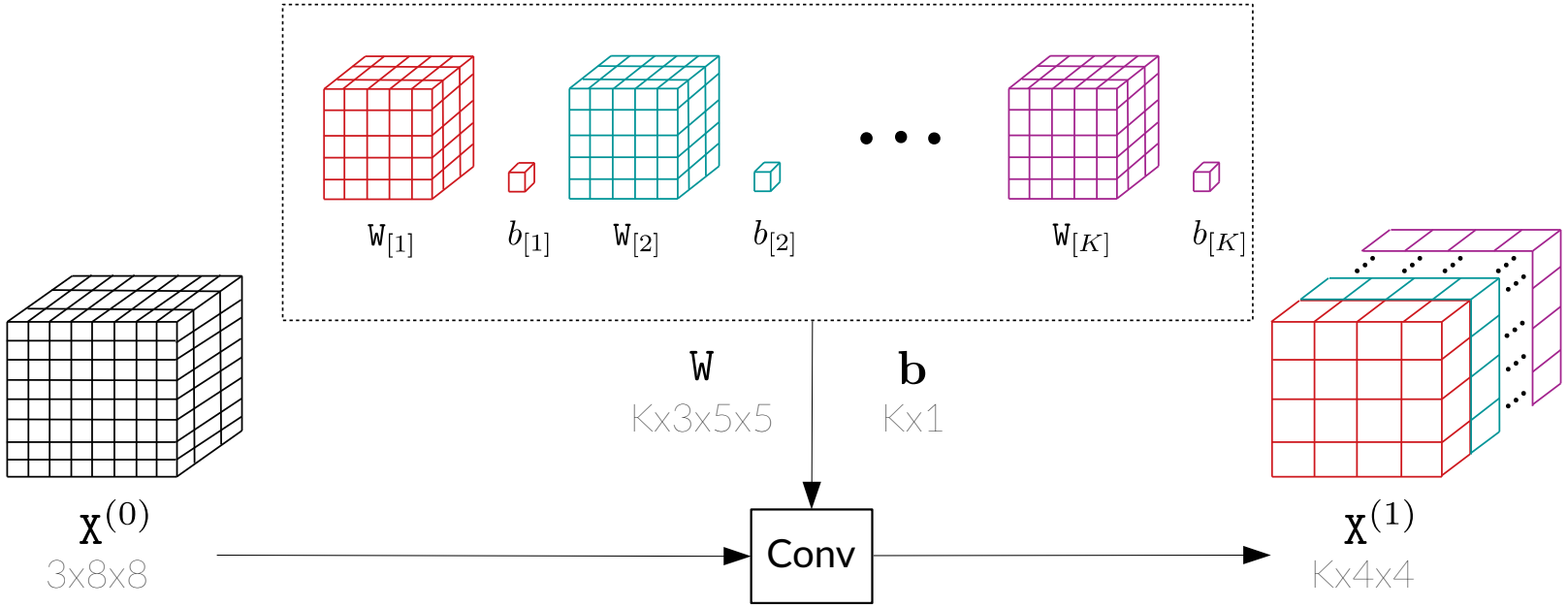
Exemple : stride de 2

Couche de convolution : un seul filtre



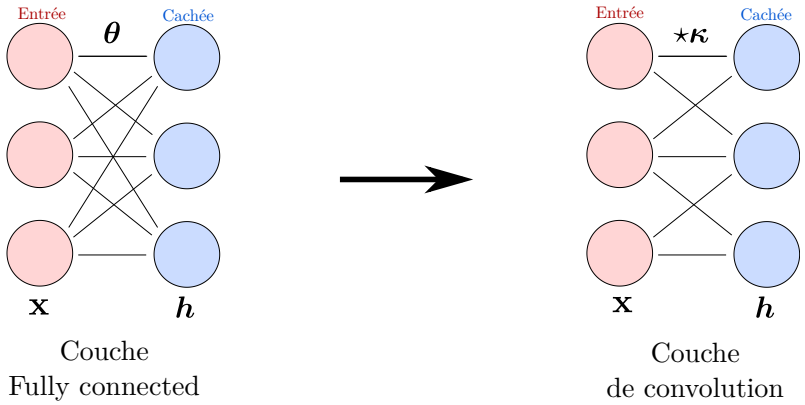
$$\mathtt{X}^{(1)}_{i,j} = \sum_{k=0}^{2} \sum_{m=0}^{4} \sum_{n=0}^{4} \mathtt{W}_{k,m,n} \mathtt{X}^{(0)}_{k,i+m,j+n} + b$$

Couche de convolution : K filtres

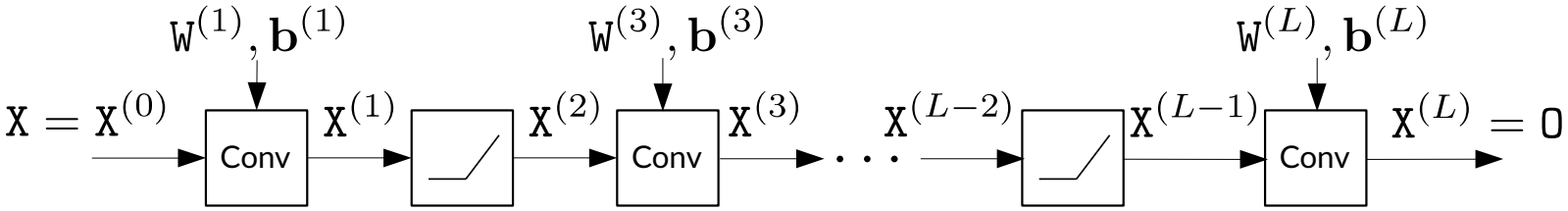


Que sont les Réseaux de neurones à convolution (CNNs) ?

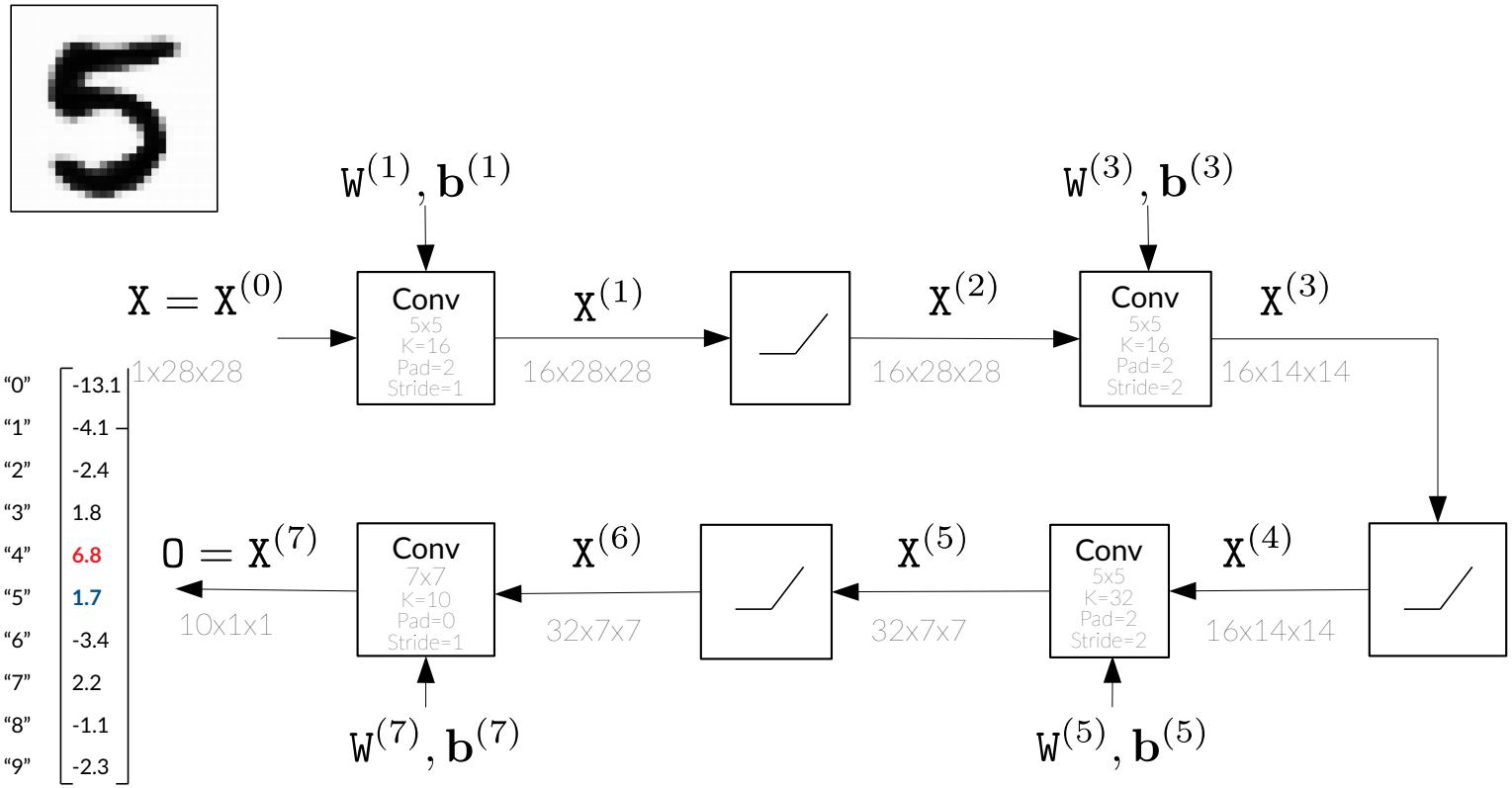
- CNN = MLP avec transformations affines ~~générales~~ **spécifiques**



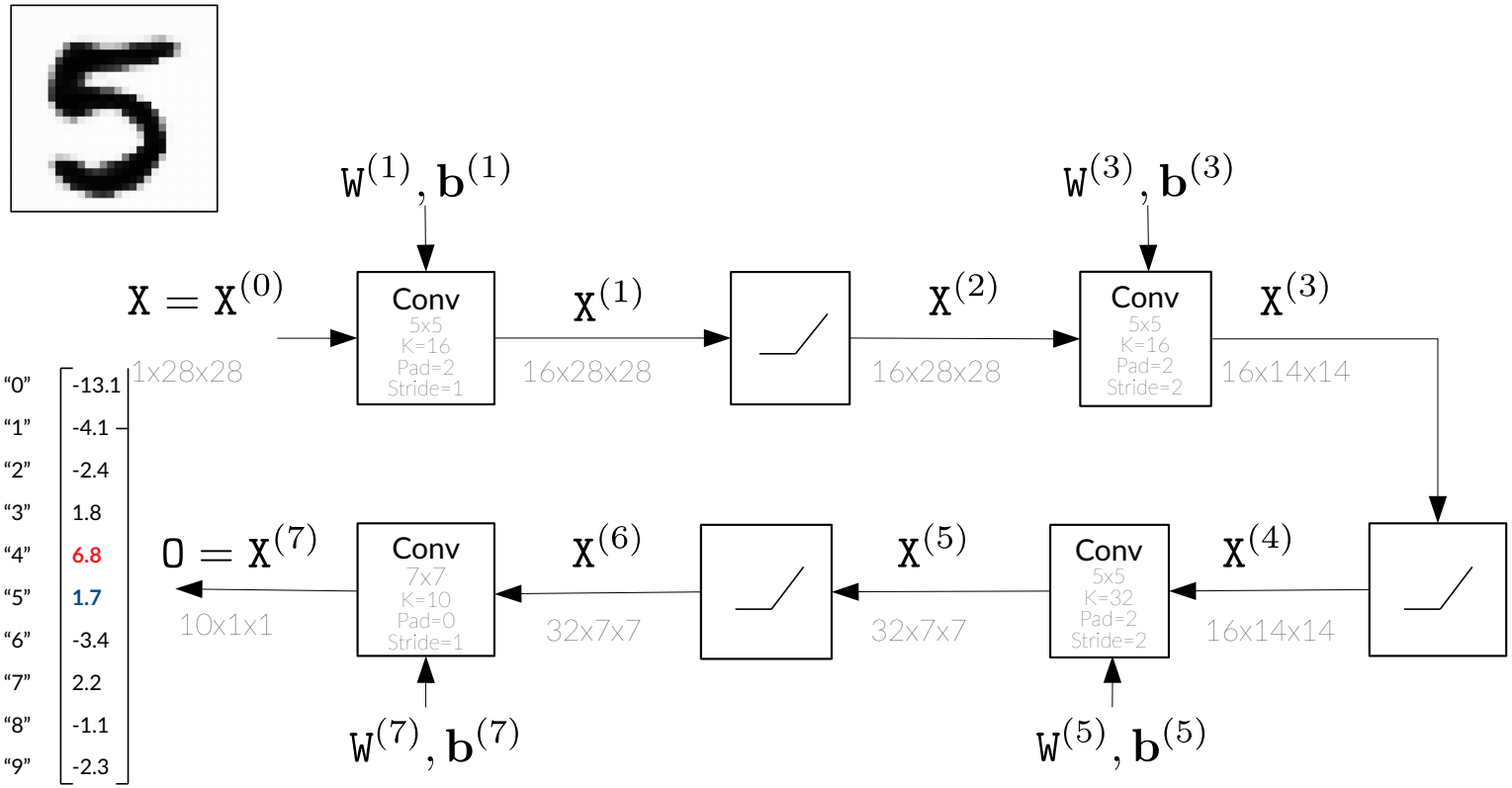
- Même représentation comme une composition de fonctions paramétriques



Exemple d'apprentissage d'un CNN sur MNIST



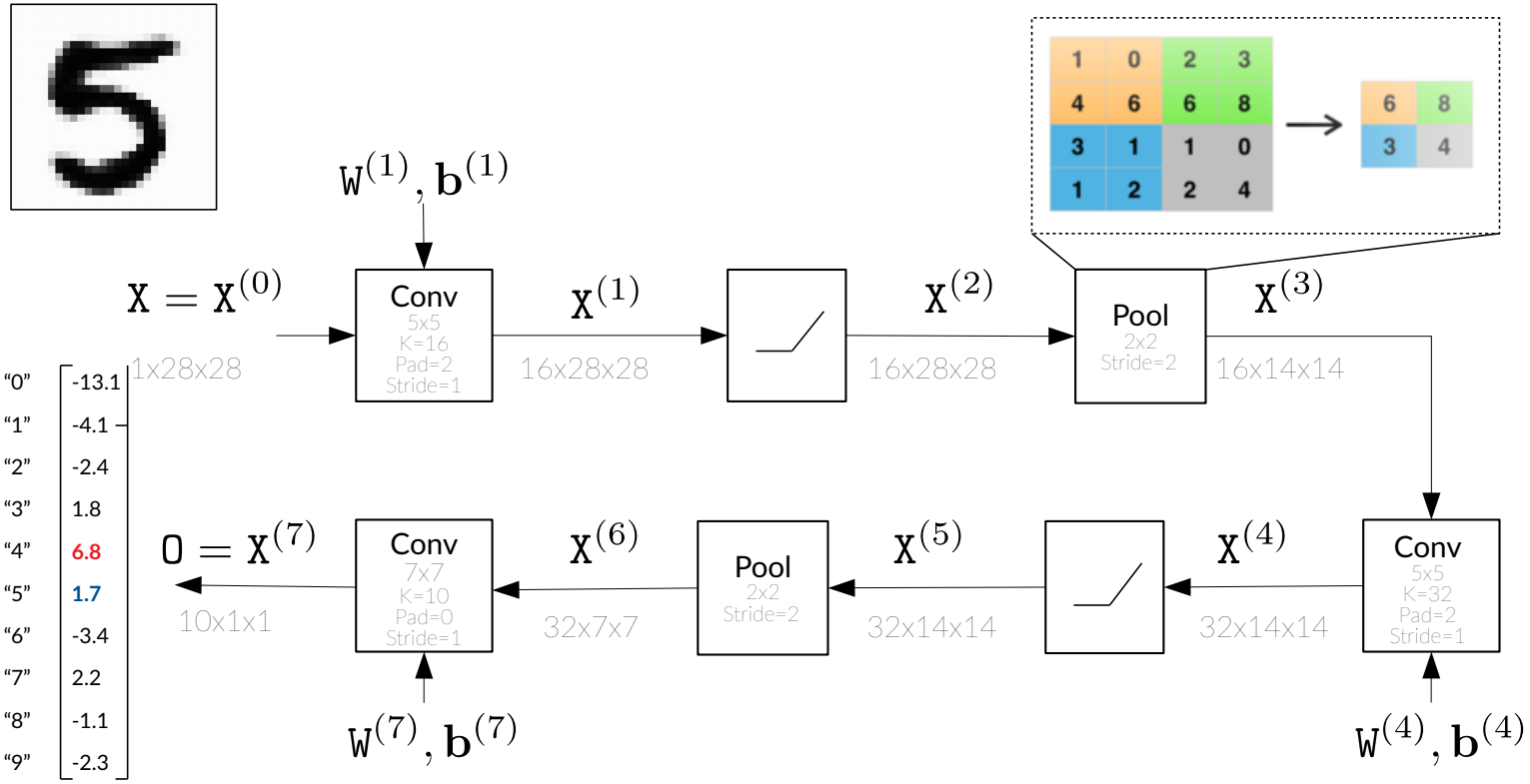
Exemple d'apprentissage d'un CNN sur MNIST



Nombre de paramètres :

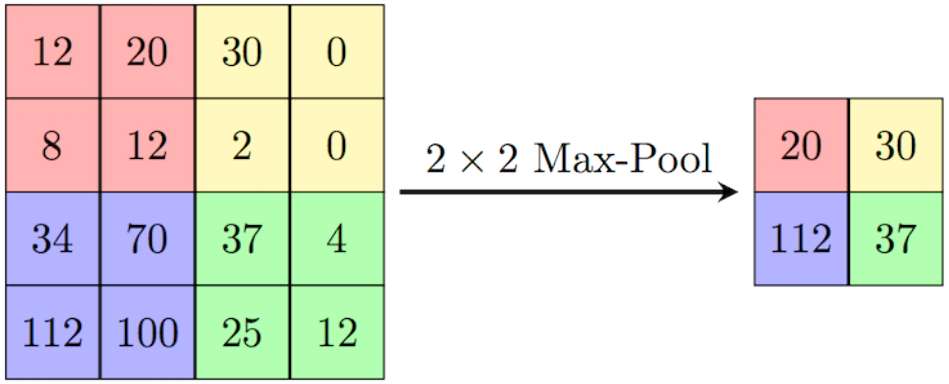
$$\underbrace{(1\text{x}5\text{x}5+1)\text{x}16}_{\mathtt{W}^{(1)},\ \mathbf{b}^{(1)}} + \underbrace{(16\text{x}5\text{x}5+1)\text{x}16}_{\mathtt{W}^{(3)},\ \mathbf{b}^{(3)}} + \underbrace{(16\text{x}5\text{x}5+1)\text{x}32}_{\mathtt{W}^{(5)},\ \mathbf{b}^{(5)}} + \underbrace{(32\text{x}7\text{x}7+1)\text{x}10}_{\mathtt{W}^{(7)},\ \mathbf{b}^{(7)}} = \underbrace{35354}_{\text{Total}}$$

Autre exemple d'apprentissage d'un CNN sur MNIST



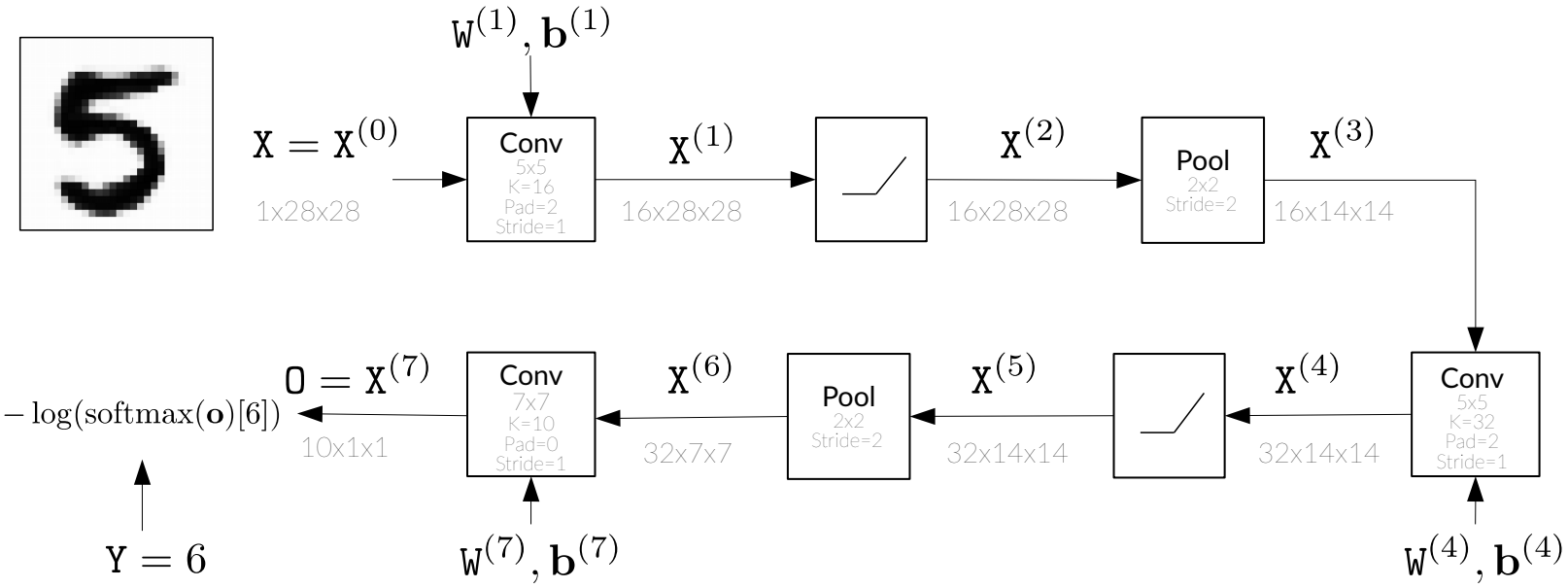
Couche de *pooling*

- Utilisée (il y a quelques années) après une couche de convolution pour agréger l'information
  - Partitionne l'image en une grille de fenêtres de taille $z \times z$ (généralement $z = 2$),
  - max-pooling: prend le $\max$ dans la fenêtre,

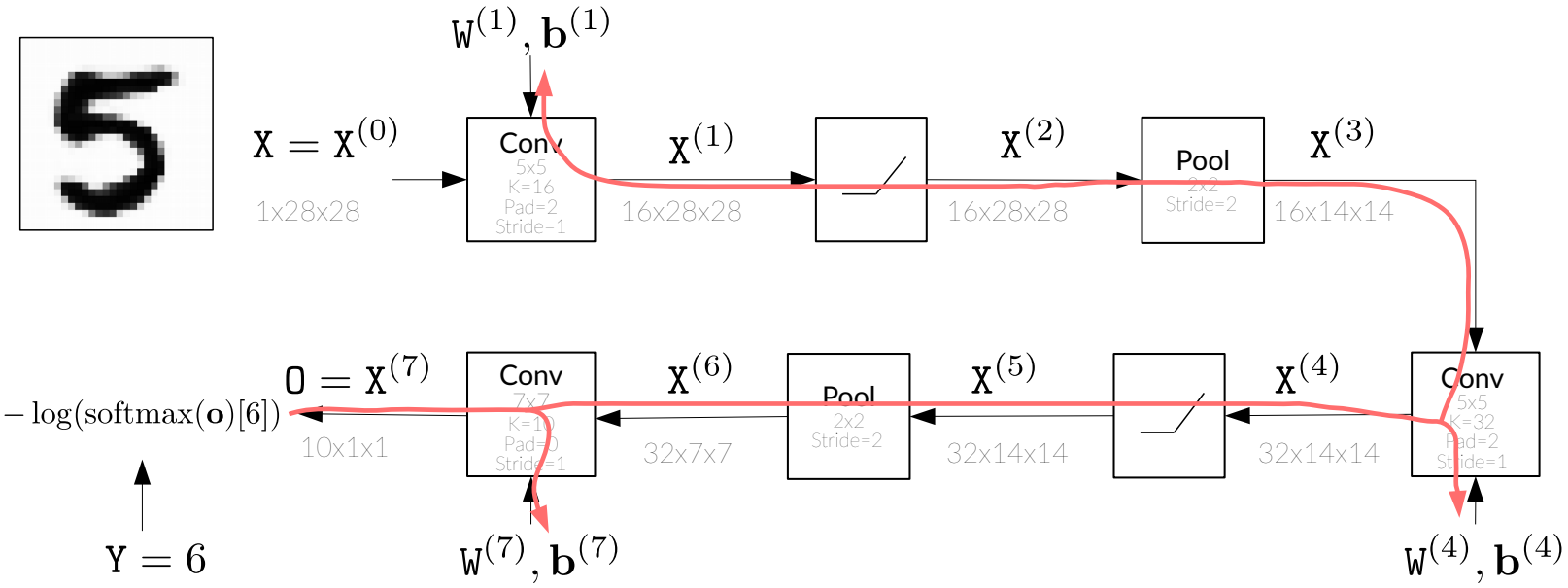


  - average-pooling: prend la moyenne.

Exemple d'apprentissage d'un CNN sur MNIST

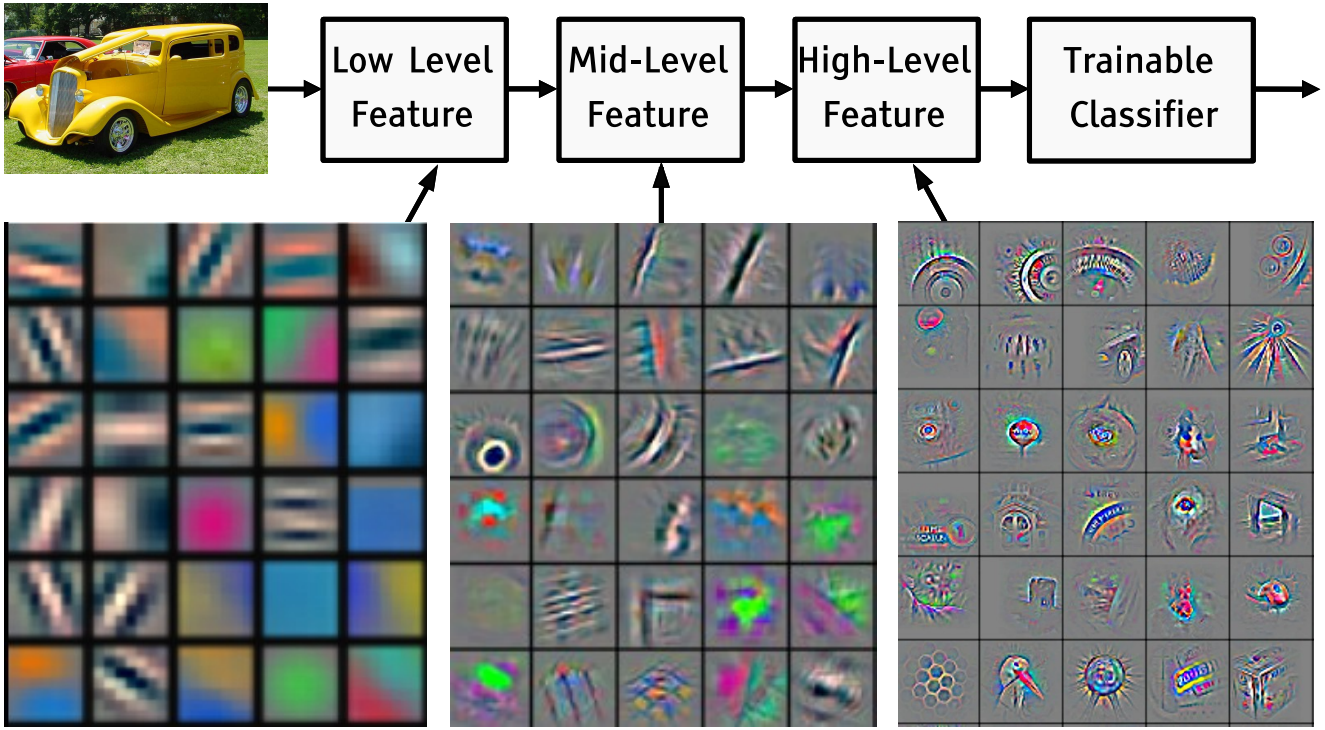


Exemple d'apprentissage d'un CNN sur MNIST



Idée derrière les CNNs

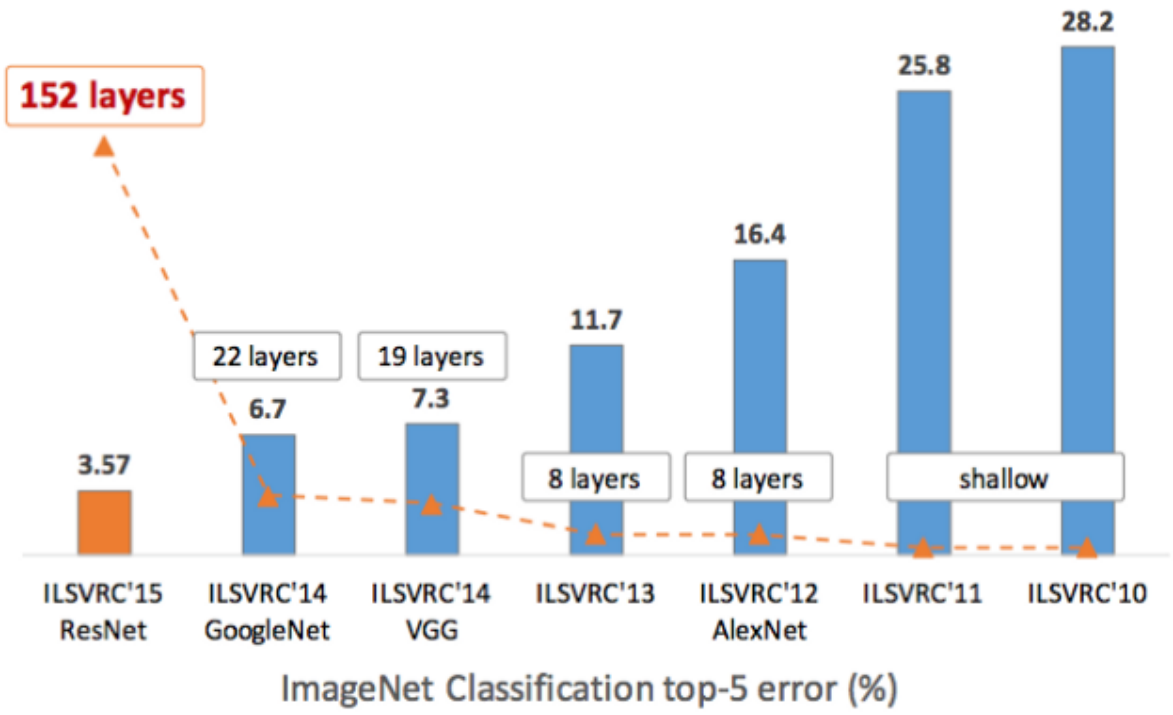
- Hiérarchie des représentations avec des niveaux d'abstraction croissants
- Chaque étape est une sorte de transformation de caractéristiques pouvant être entraînée [1] :



1. [Zeiler et al., 2014] - Visualizing and Understanding Convolutional Networks. In ECCV.

Réseaux de neurones profonds

De plus en plus profond (nombre de couches cachées)

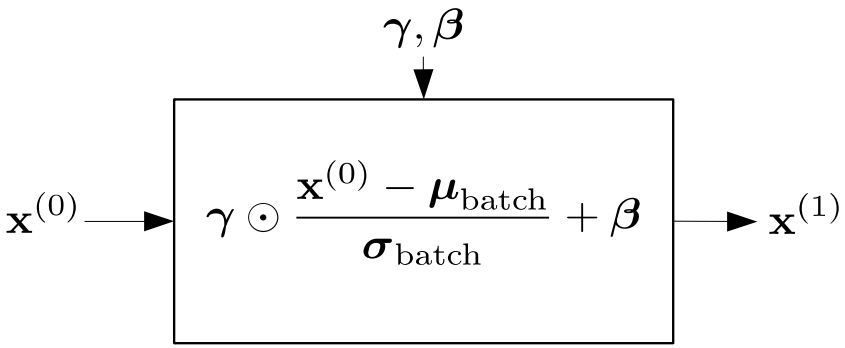


→ **Comment entraîner de tels réseaux ?**

Minimums locaux ? Stabilité ? Temps d'entraînement ?

Couche de *Batch Normalization*[1]

Normalisation des valeurs des minibatchs en sortie des convolutions



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

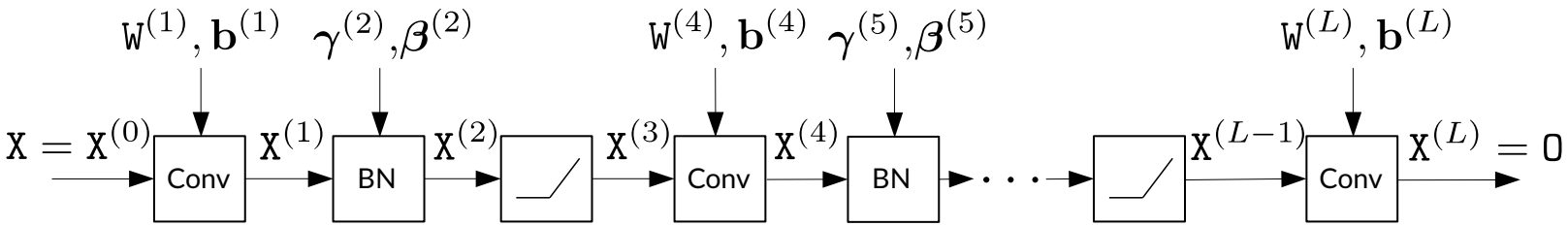
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

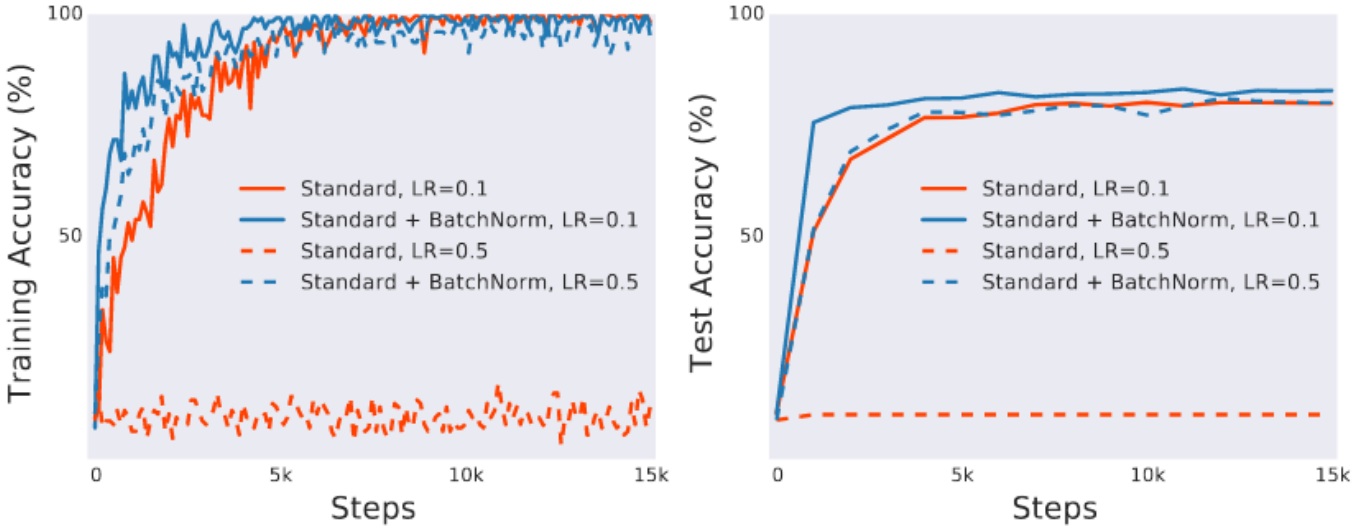


1. [Ioffe et al., 2015] - Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint.

Couche de *Batch Normalization*[1]

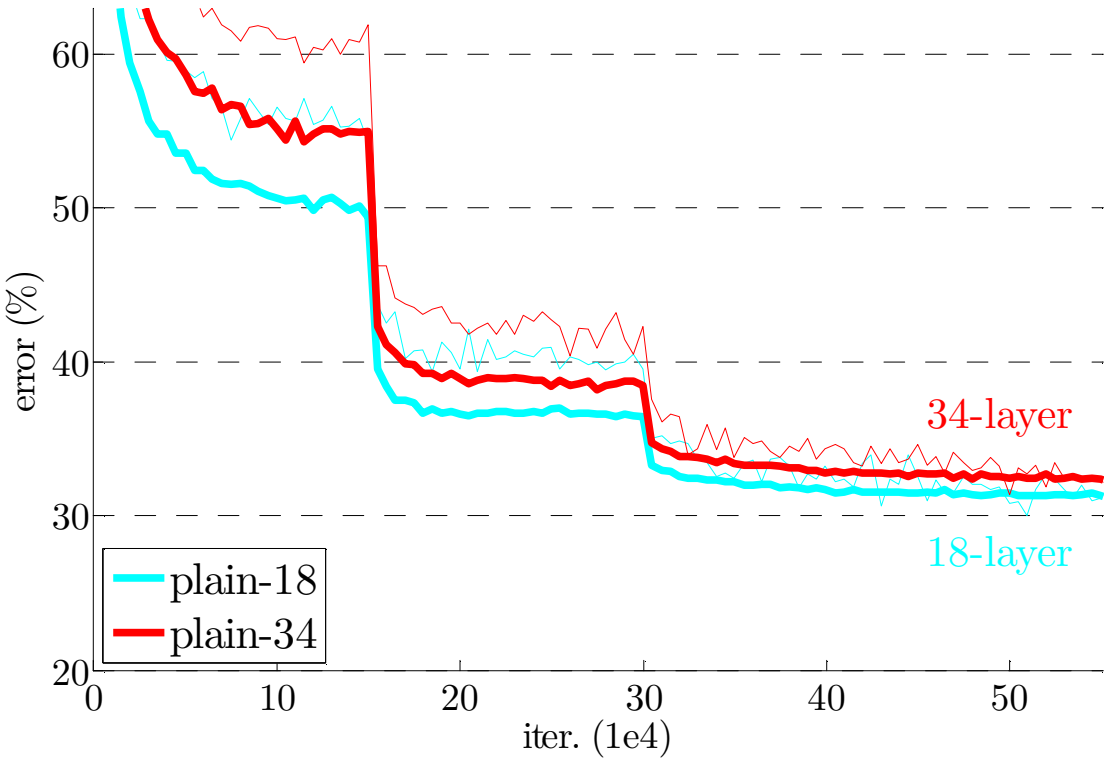
L'utilisation de couches de *batch normalization* rend le problème d'optimisation plus "lisse", ce qui implique :

- Initialisation des paramètres moins critique
- Possibilité d'utilisation d'un plus grand pas d'apprentissage



1. [Santurkar et al., 2018] - How does batch normalization help optimization? In NeurIPS.

Suffisant ?

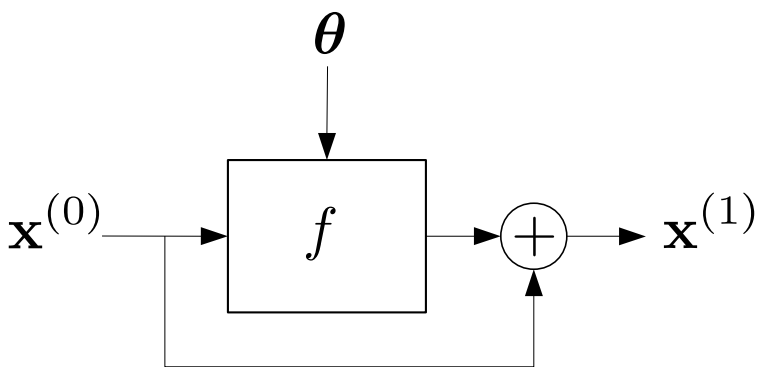

60
50
40
30
20
error (%)
0
10
20
30
40
50
iter. (1e4)
34-layer
18-layer
plain-18
plain-34

Connexion résiduelle

Ajout de l'entrée à la sortie

Rend une couche plus "linéaire" donc :

- Réduit sa "capacité" $\rightarrow$ augmentation du nombre de couches et donc de la consommation et de la mémoire pour un même résultat



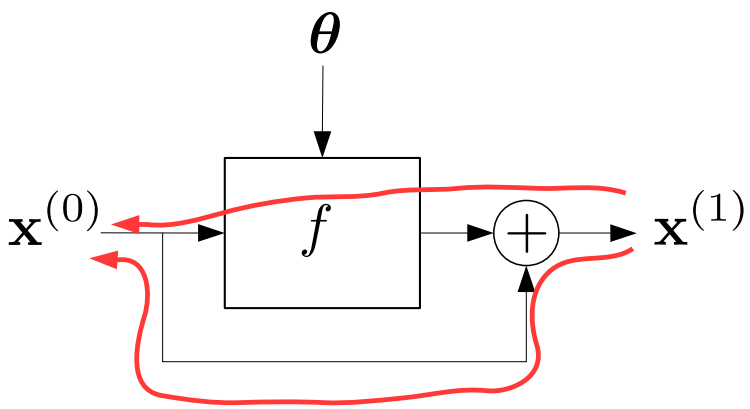
$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + f(\mathbf{x}^{(0)}; \boldsymbol{\theta})$$

Connexion résiduelle

Ajout de l'entrée à la sortie

Rend une couche plus "linéaire" donc :

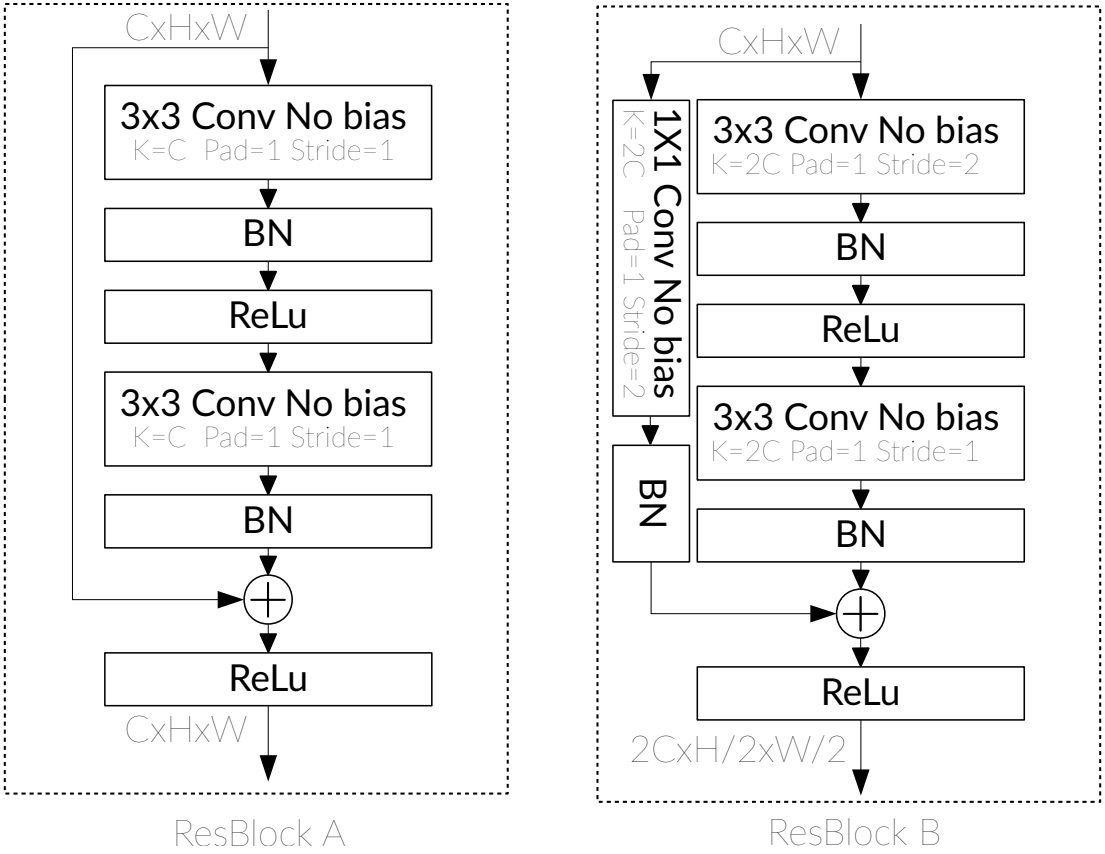
- Réduit sa "capacité" $\rightarrow$ augmentation du nombre de couches et donc de la consommation et de la mémoire pour un même résultat
- Mais facilite la propagation du gradient $\rightarrow$ plus de couches conduit à de meilleurs résultats (en théorie)



$$\frac{\partial \mathbf{x}^{(1)}}{\partial \mathbf{x}^{(0)}} = \mathtt{I} + \frac{\partial f(\mathbf{x}^{(0)}; \boldsymbol{\theta})}{\partial \mathbf{x}^{(0)}}$$

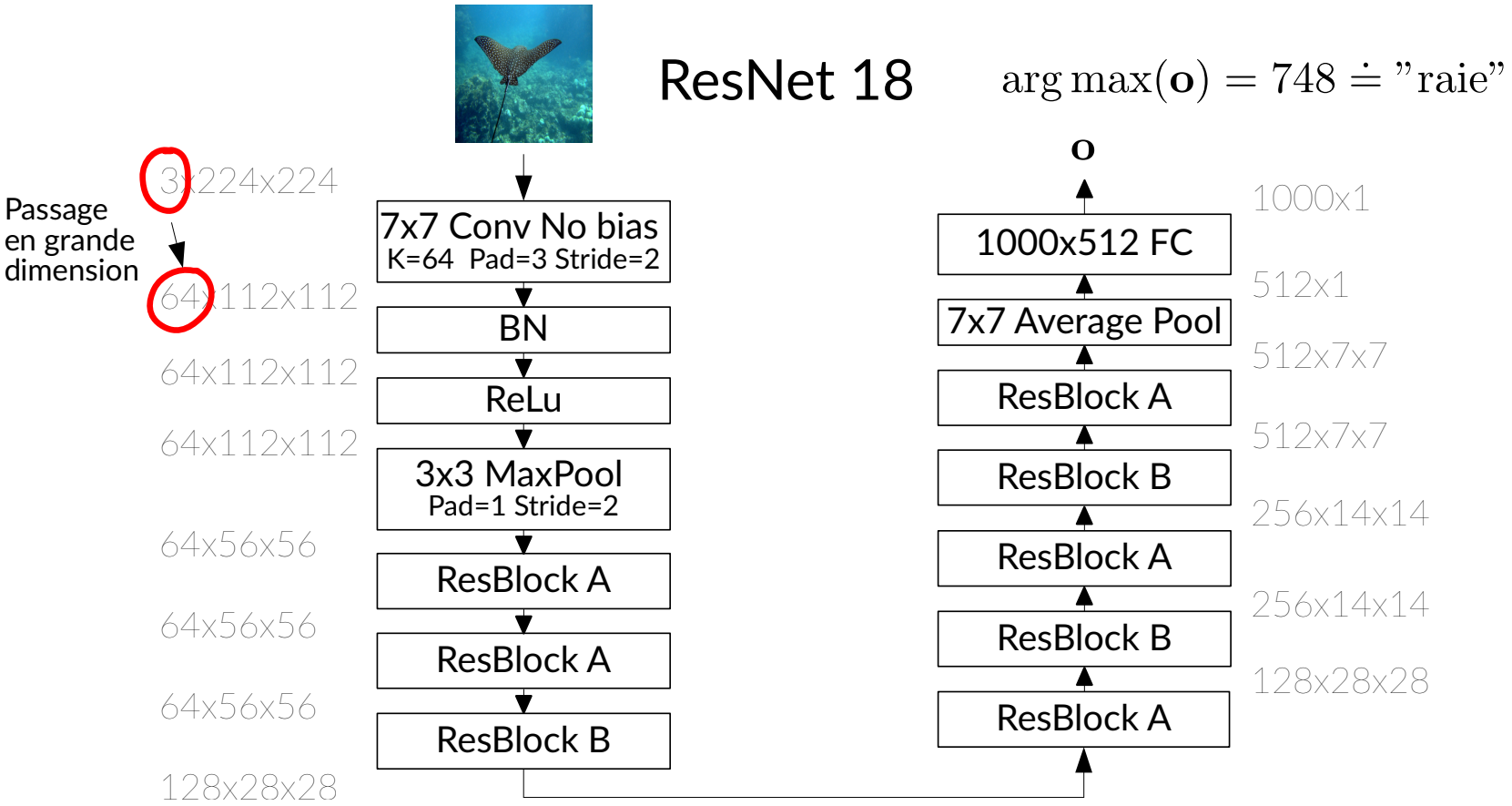
Architecture utilisant la connexion résiduelle [1]

ResNet



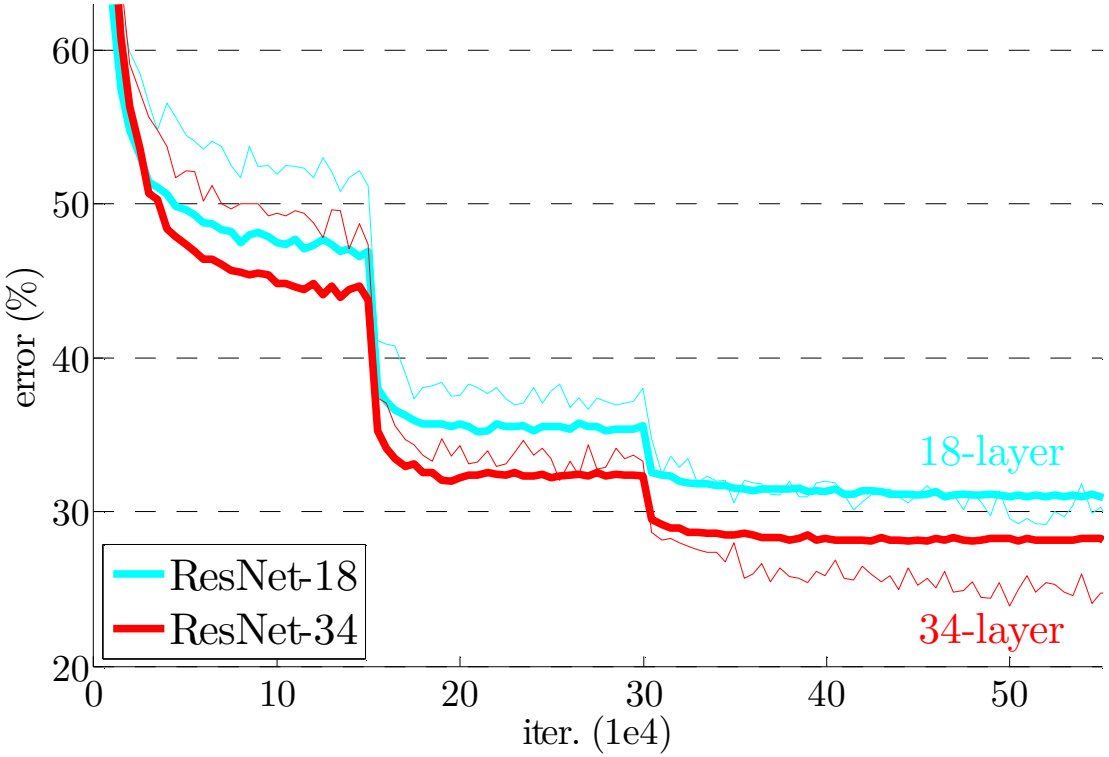
1. [Kaiming et al., 2016] - Deep residual learning for image recognition. In CVPR.

Architecture utilisant la connexion résiduelle [1]

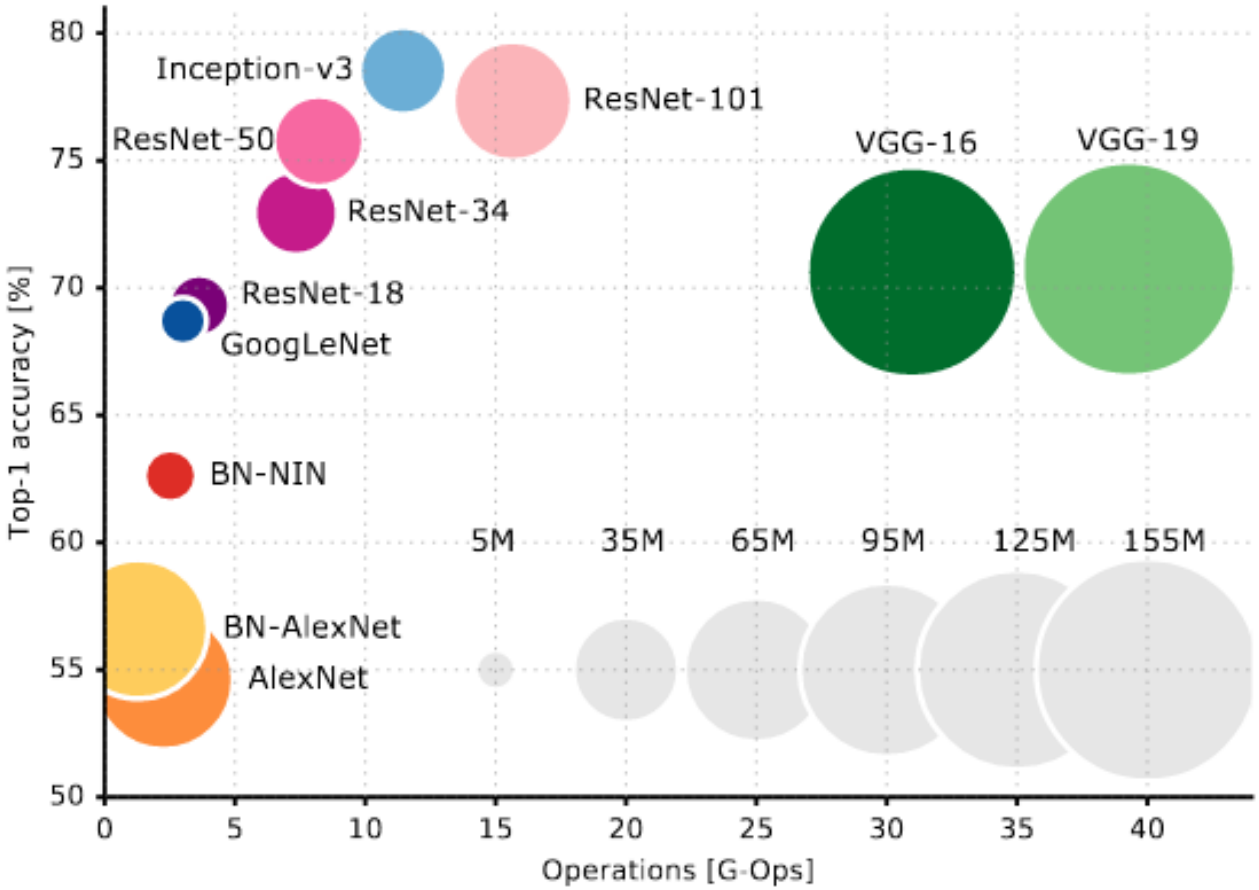


1. [Kaiming et al., 2016] - Deep residual learning for image recognition. In CVPR.

Architecture utilisant la connexion résiduelle



Précision vs Nombre de paramètres vs Nombre d'opérations [1]



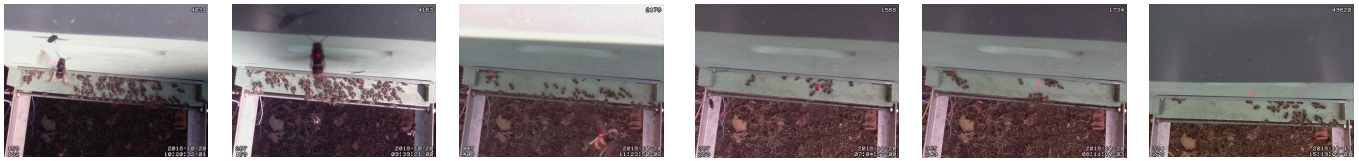
1. [Canziani et al., 2016] - An analysis of deep neural network models for practical applications. arXiv preprint.

Techniques et outils

Spécialisation d'un réseau (*fine-tuning*)

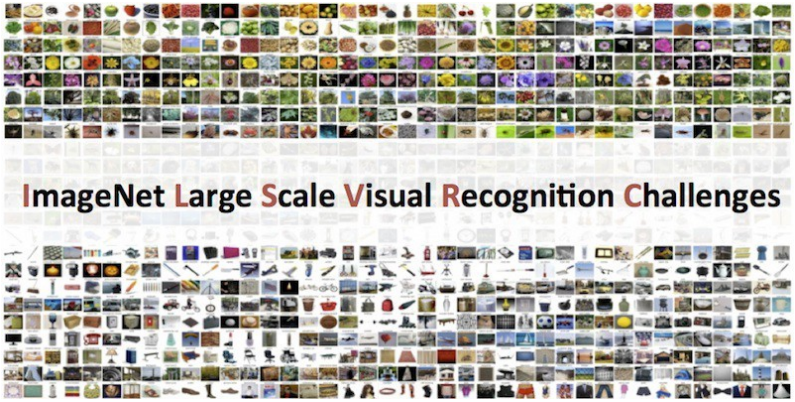
Exemple :

- Nouvelle application de détection de frelon asiatique



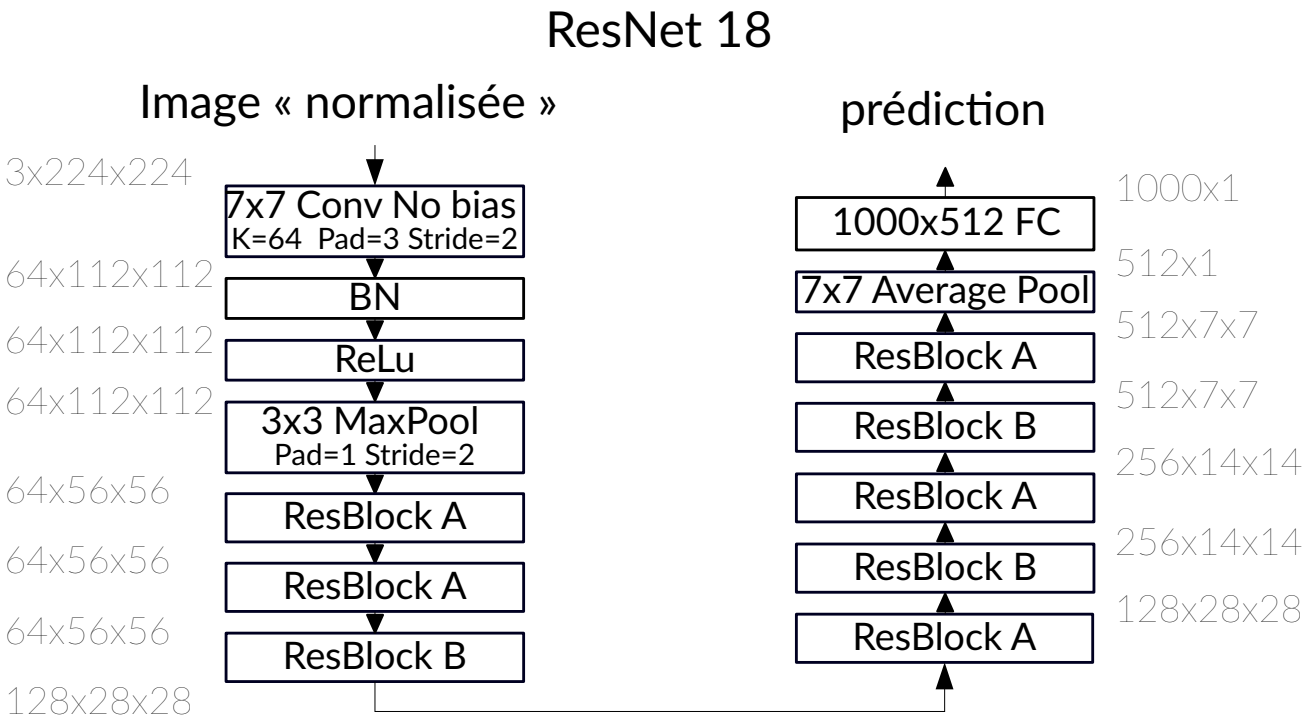
Présence (1042 images) Absence (1844 images)

- À disposition : Réseau (par ex. ResNet) pré-entraîné pour la classification sur le dataset ImageNet (par ex. 1000 classes)



Spécialisation d'un réseau (*fine-tuning*)

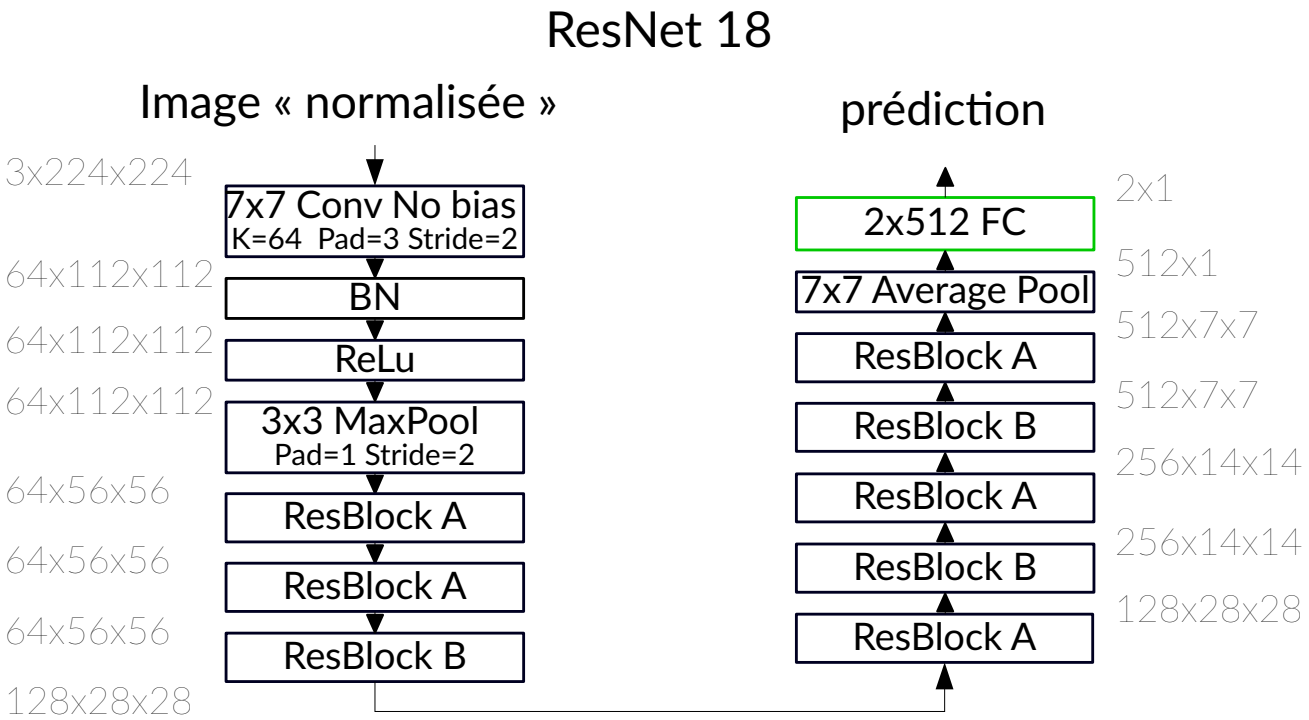
Adaptation du réseau à la nouvelle application en conservant les paramètres des couches cachées



Paramètres initialisés par un ResNet18 entraîné sur ImageNet

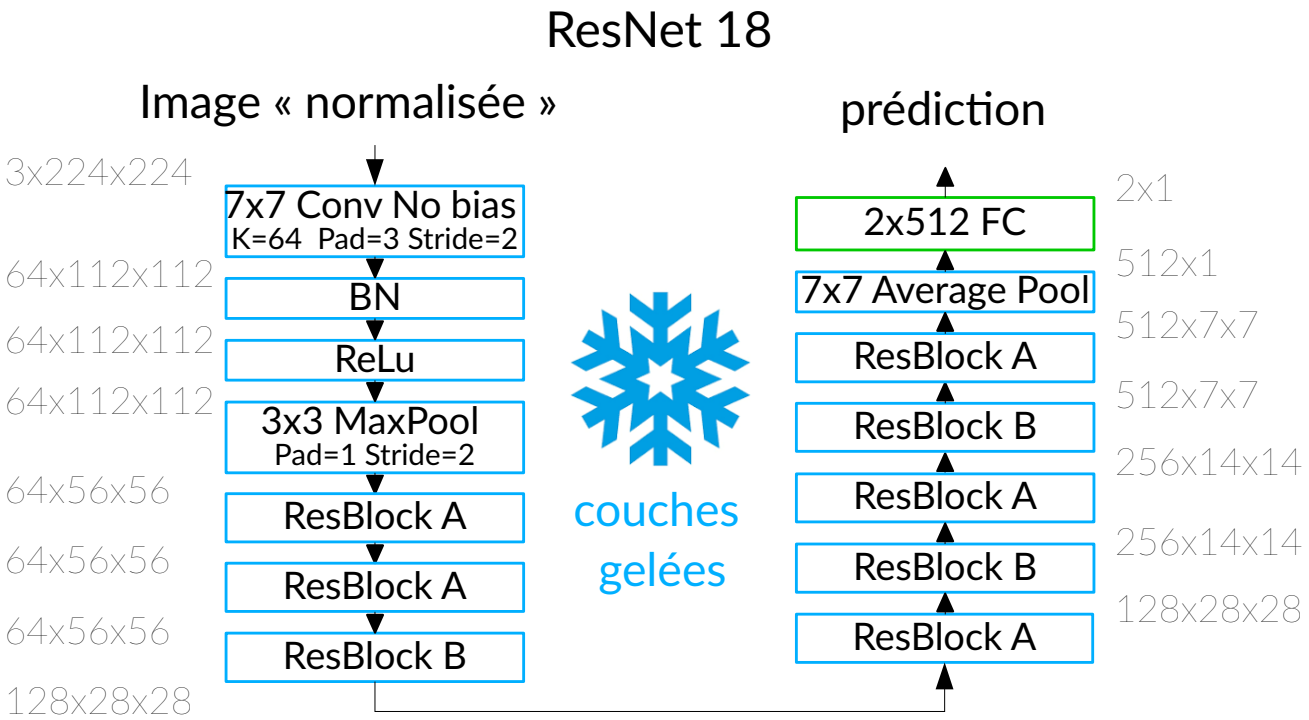
Spécialisation d'un réseau (*fine-tuning*)

Adaptation du réseau à la nouvelle application en conservant les paramètres des couches cachées



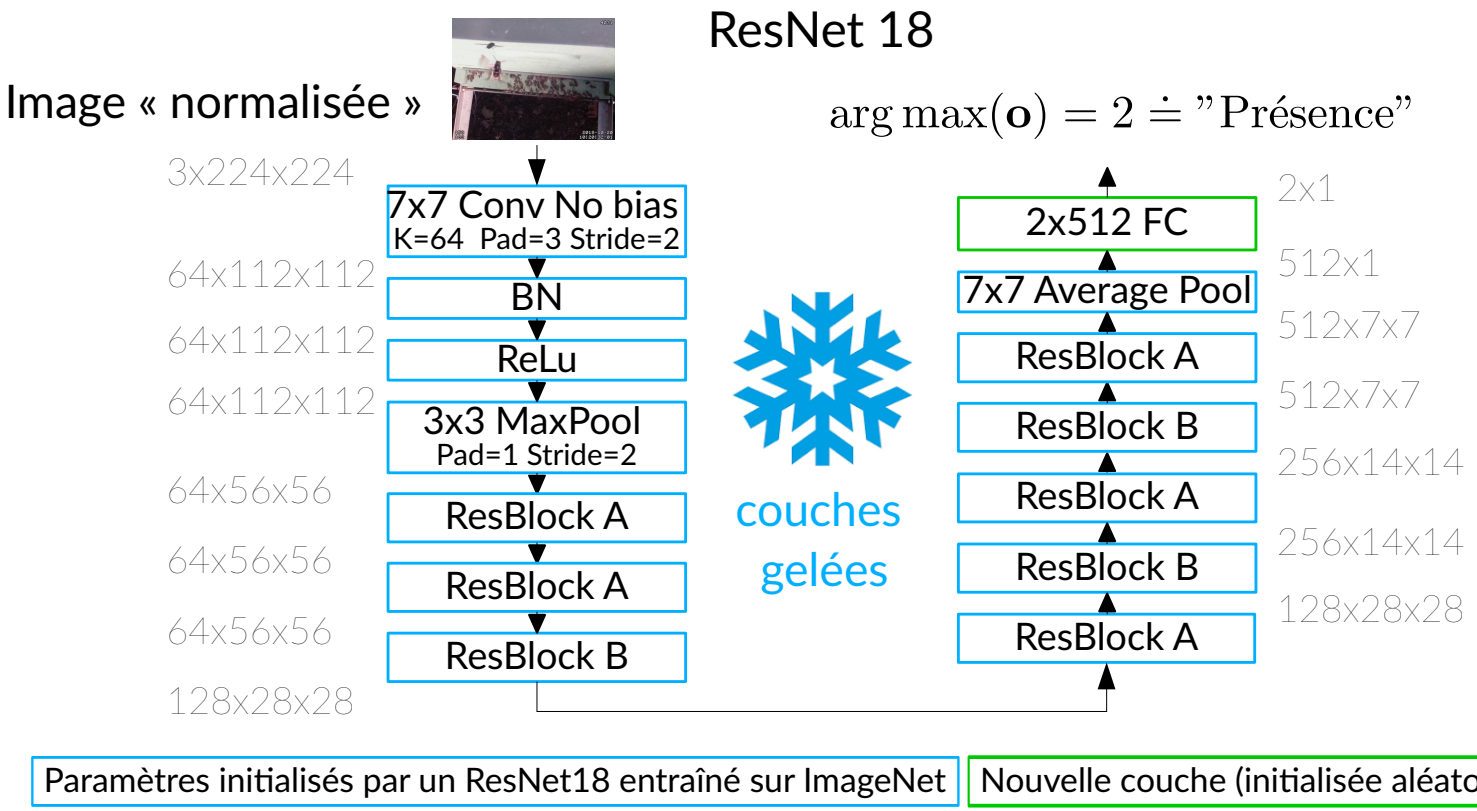
Spécialisation d'un réseau (*fine-tuning*)

Adaptation du réseau à la nouvelle application en conservant les paramètres des couches cachées



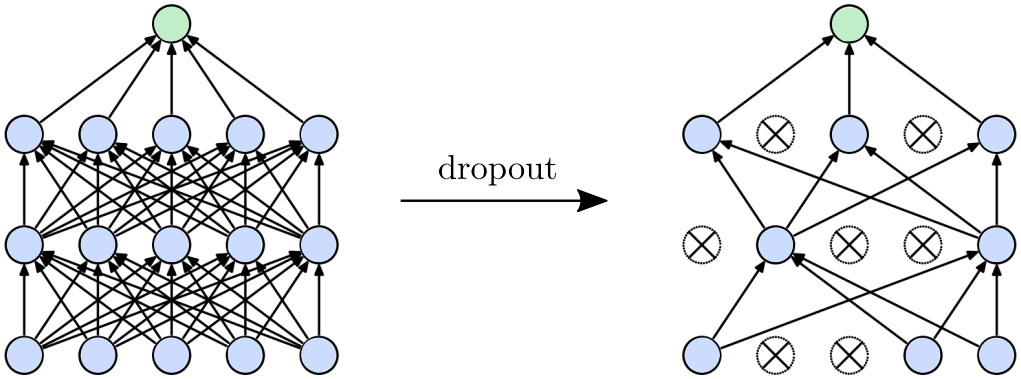
Spécialisation d'un réseau (*fine-tuning*)

Adaptation du réseau à la nouvelle application en conservant les paramètres des couches cachées



Dropout

- Éteint des neurones selon une probabilité p

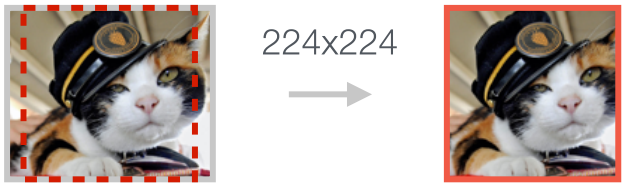


- Empêche les unités de trop s'adapter (dépendent les unes des autres)
- Permet au réseau d'apprendre des fonctionnalités plus robustes et spécifiques
- Converge plus rapidement et chaque époque coûte moins cher

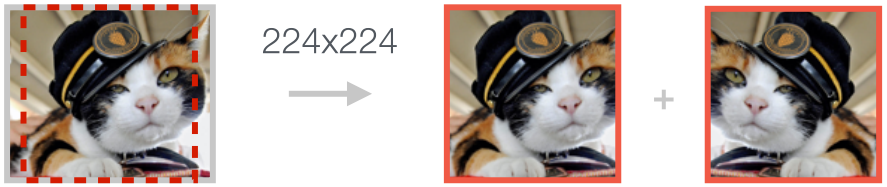
Augmentation de données

Idée : générer artificiellement des données directement depuis celles d'entraînement

Pas d'augmentation (=1 image)



Flips (=2 images)

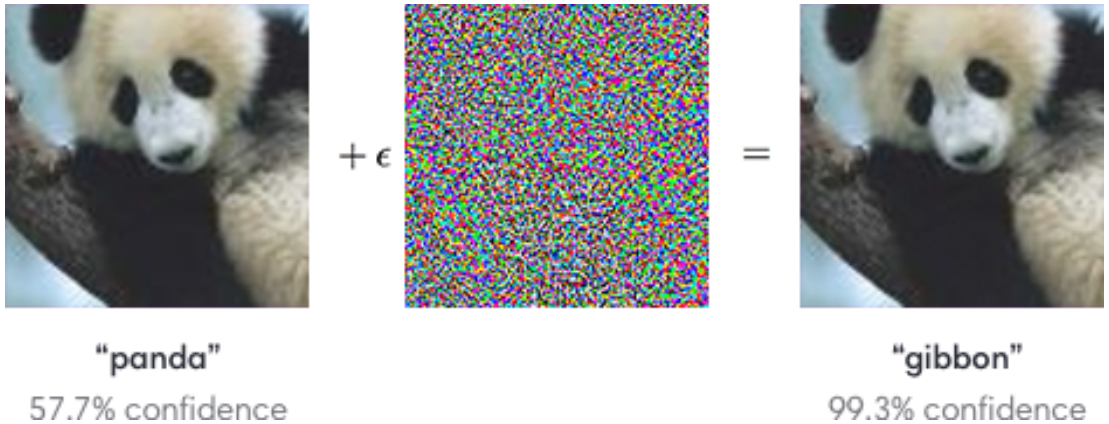


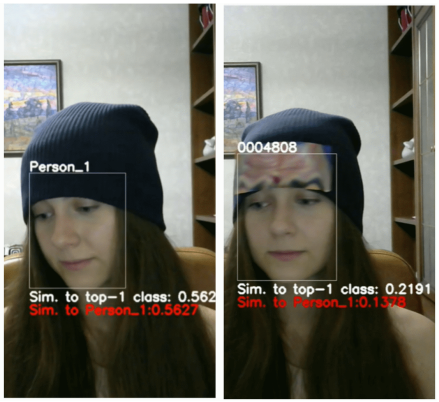
Crop + Flips (=10 images)

Autres transformations : rotations, mix, luminosité, bruit, . . .

Adversarial networks

Idée : Réussir à tromper un réseau avec un léger bruit ou motif spécifique [1]

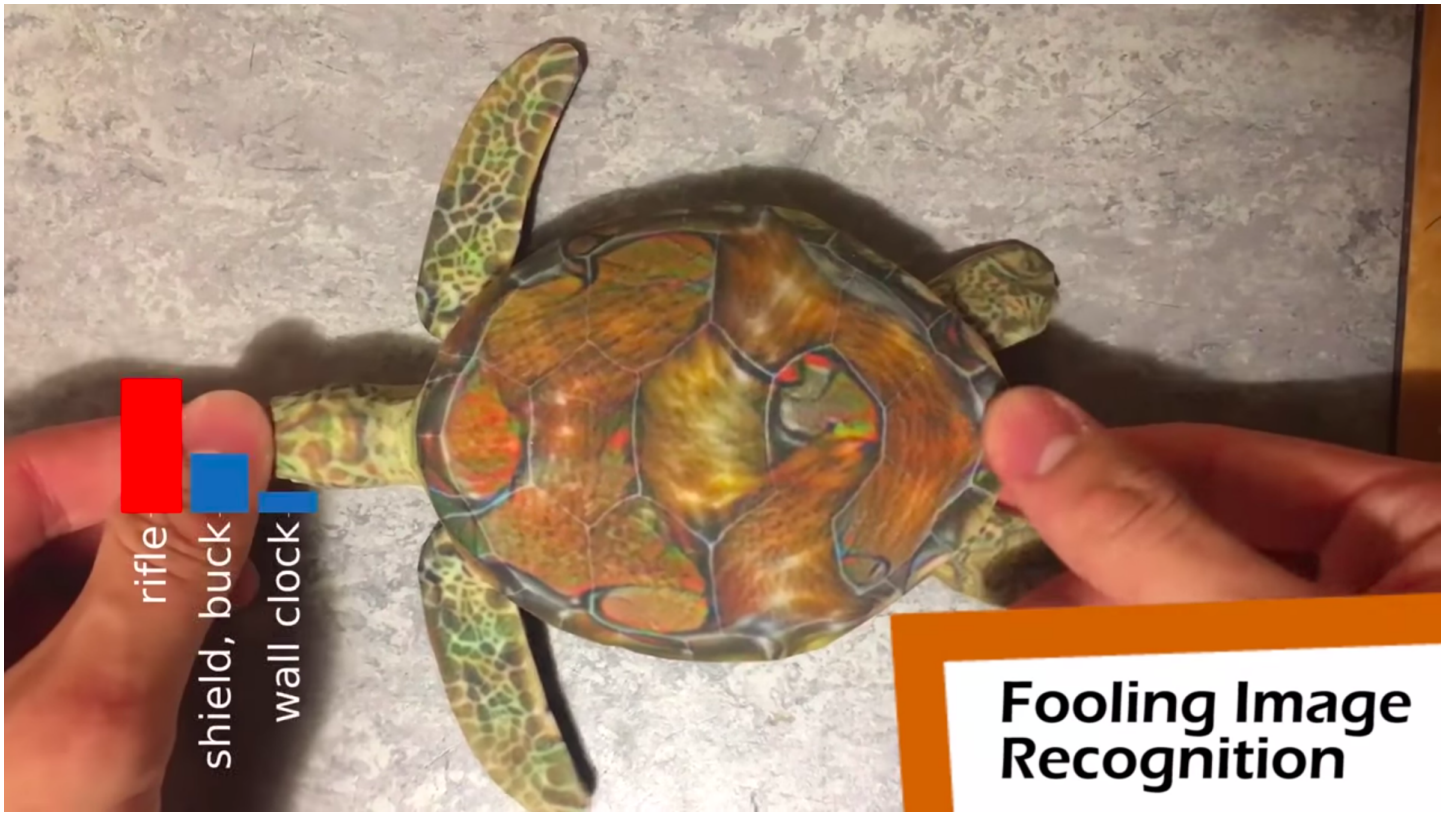




1. https://openai.com/blog/adversarial-example-research/
[Komkov et al., 2020] - ADVHAT: Real-world adversarial attack on ArcFace face ID system. In ICPR.
[Eykholt et al., 2018] - Robust Physical-World Attacks on Deep Learning Visual Classification. In CVPR.

Adversarial networks

Idée : Réussir à tromper un réseau avec un léger bruit ou motif spécifique



Vidéo : `https://www.youtube.com/watch?v=piYnd_wYlT8`

Bibliothèques de *Deep Learning*

PyTorch

Numpy sur GPU

```
# Create a numpy array.
x = np.array([[1, 2], [3, 4]])

# Convert the numpy array to a torch tensor.
y = torch.from_numpy(x)

# Convert the torch tensor to a numpy array.
z = y.numpy()
```

Autograd

```
# Create tensors.
x = torch.tensor(1., requires_grad=True)
w = torch.tensor(2., requires_grad=True)
b = torch.tensor(3., requires_grad=True)

# Build a computational graph.
y = w * x + b    # y = 2 * x + 3

# Compute gradients.
y.backward()

# Print out the gradients.
print(x.grad)    # x.grad = 2
print(w.grad)    # w.grad = 1
print(b.grad)    # b.grad = 1
```

PyTorch : Chargement de données (*Dataloader*)

Définition du Dataloader

```
train_loader = t.utils.data.DataLoader(dataset=train_set,
                                       batch_size=batch_size,
                                       shuffle=True,
                                       num_workers=2)
```

Nombre de processus qui vont préparer des minibatches en parallèle sur CPU(s).

Définition du GPU

```
device = t.device('cuda:0' if t.cuda.is_available() else 'cpu')
```

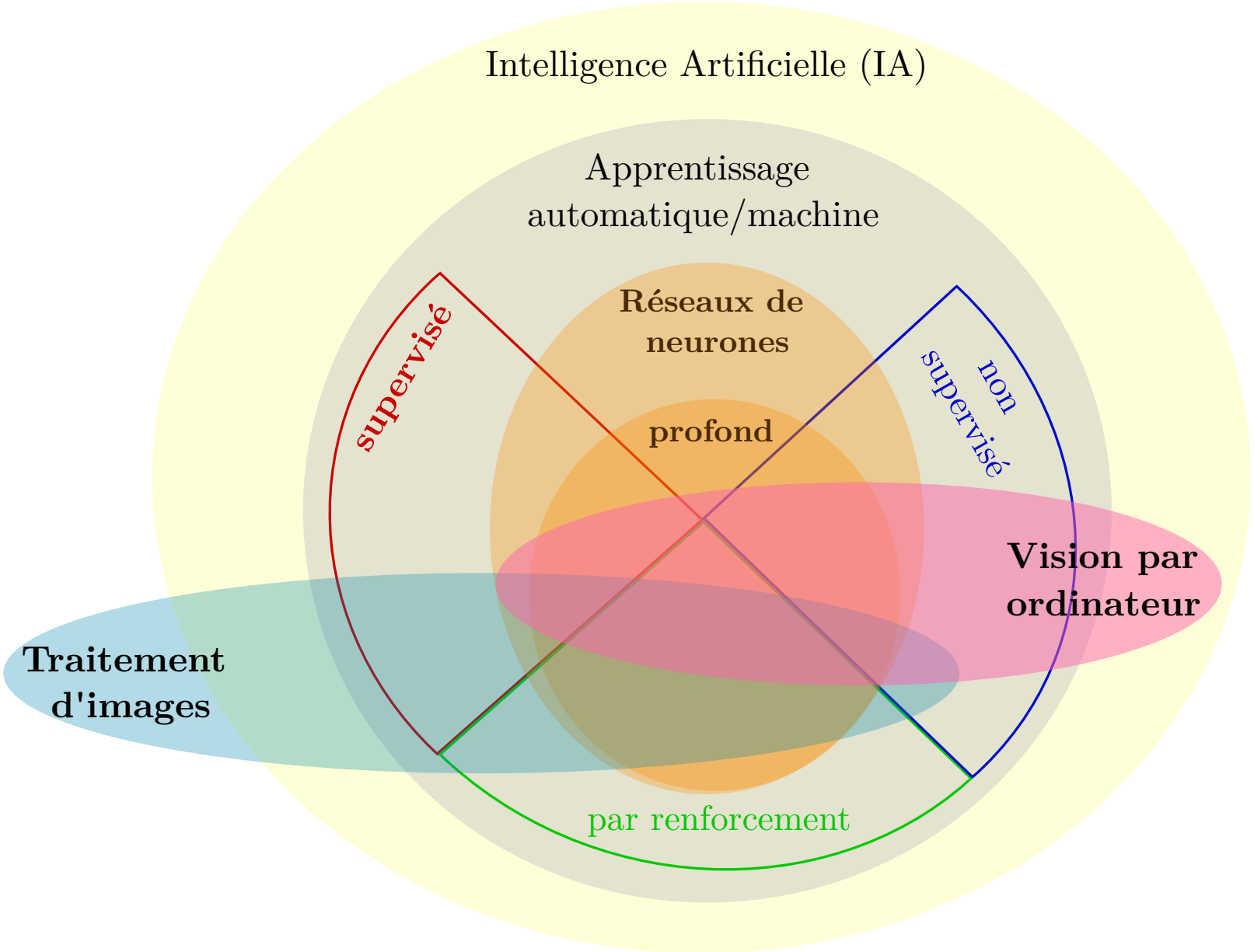
Boucle principale d'apprentissage

```
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)
        ...
```

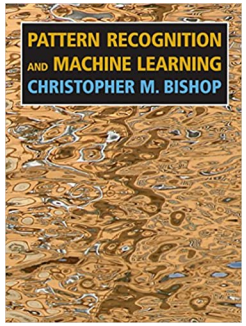
Mise à disposition d'un minibatch

Transfert du minibatch au GPU

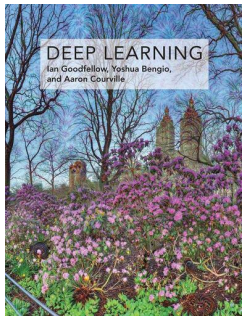
Intelligence artificielle pour l'image



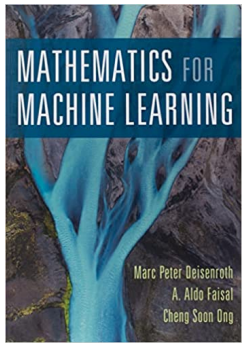
Quelques conseils de lecture



Pattern Recognition and Machine Learning
Christopher M. Bishop
`https://www.microsoft.com/en-us/research/people/cmbishop/prml-book/`



Deep Learning
Ian Goodfellow, Yoshua Bengio, Aaron Courville
`https://www.deeplearningbook.org`



Mathematics for Machine Learning
Marc Peter Deisenroth, A. Aldo Faisal, Cheng Soon Ong
`https://mml-book.com`