TSIA1 : Intelligence artificielle pour l'image

Rémi Giraud
remi.giraud@enseirb-matmeca.fr

2025-2026

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

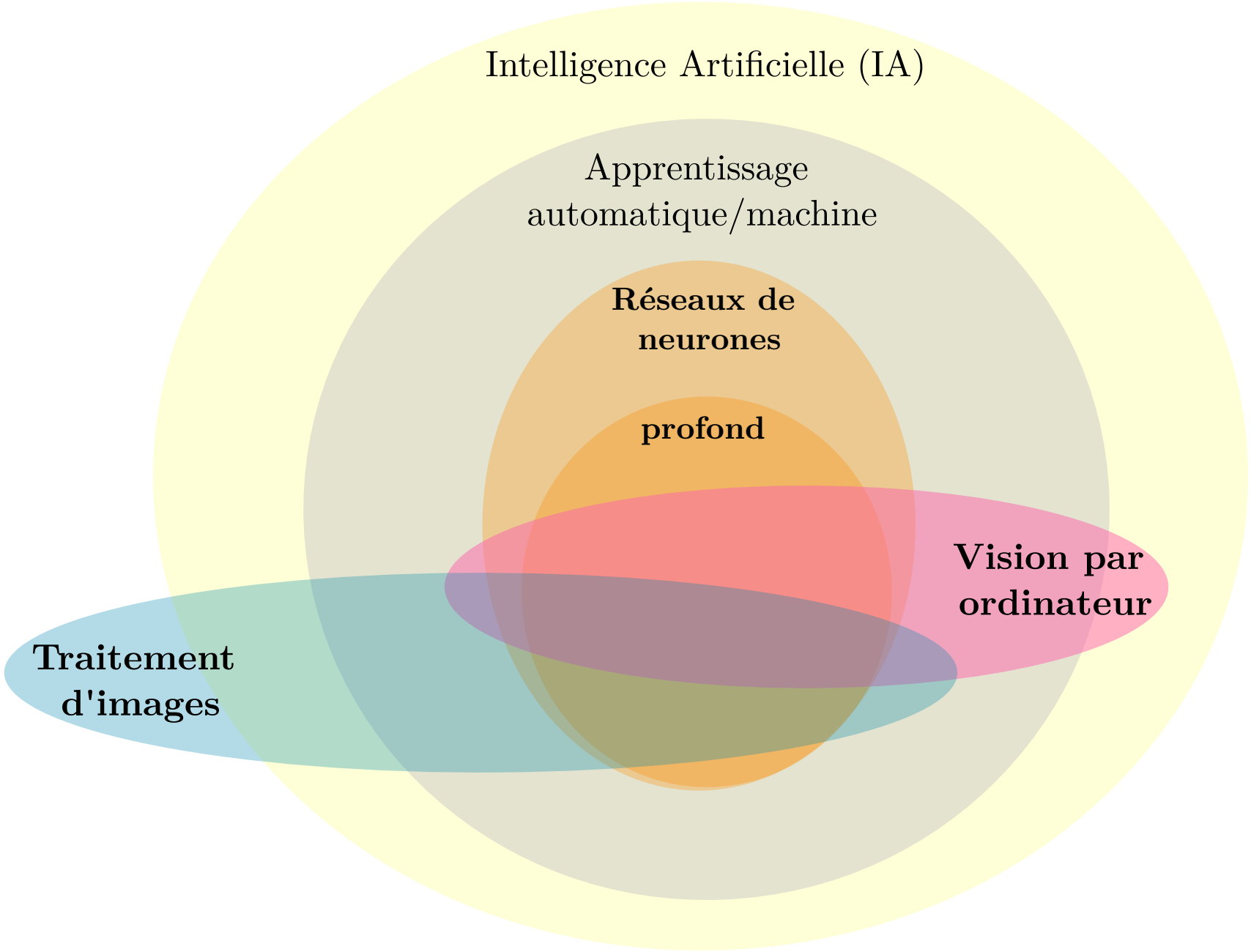
- **5h20 de cours** et **8h de TP**
- Supports de cours : https://remi-giraud.enseirb-matmeca.fr/teaching/
- **Notation** : contrôle continu (contrôle de connaissance, rendu de TP)

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**

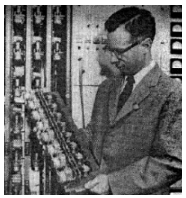
Introduction

Contexte

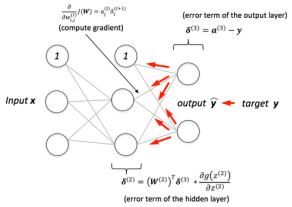

Intelligence Artificielle (IA)
Apprentissage automatique/machine
Réseaux de neurones
profond
Vision par ordinateur
Traitement d'images

Chronologie

Quelques dates clés dans la chronologie de l'apprentissage profond



1958 Perceptron



1974 Backpropagation

Convolution Neural Networks for Handwritten Recognition

1998

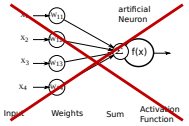
Google Brain Project on 16k Cores

2012

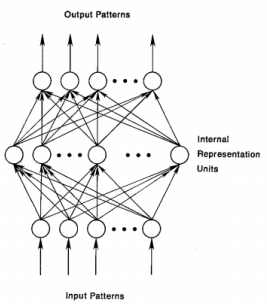
awkward silence (AI winter)

1969 Perceptrons book

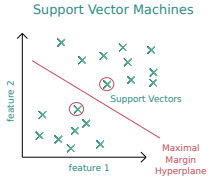
Perceptron criticized



~1980 Multilayer network



1995 SVM reigns

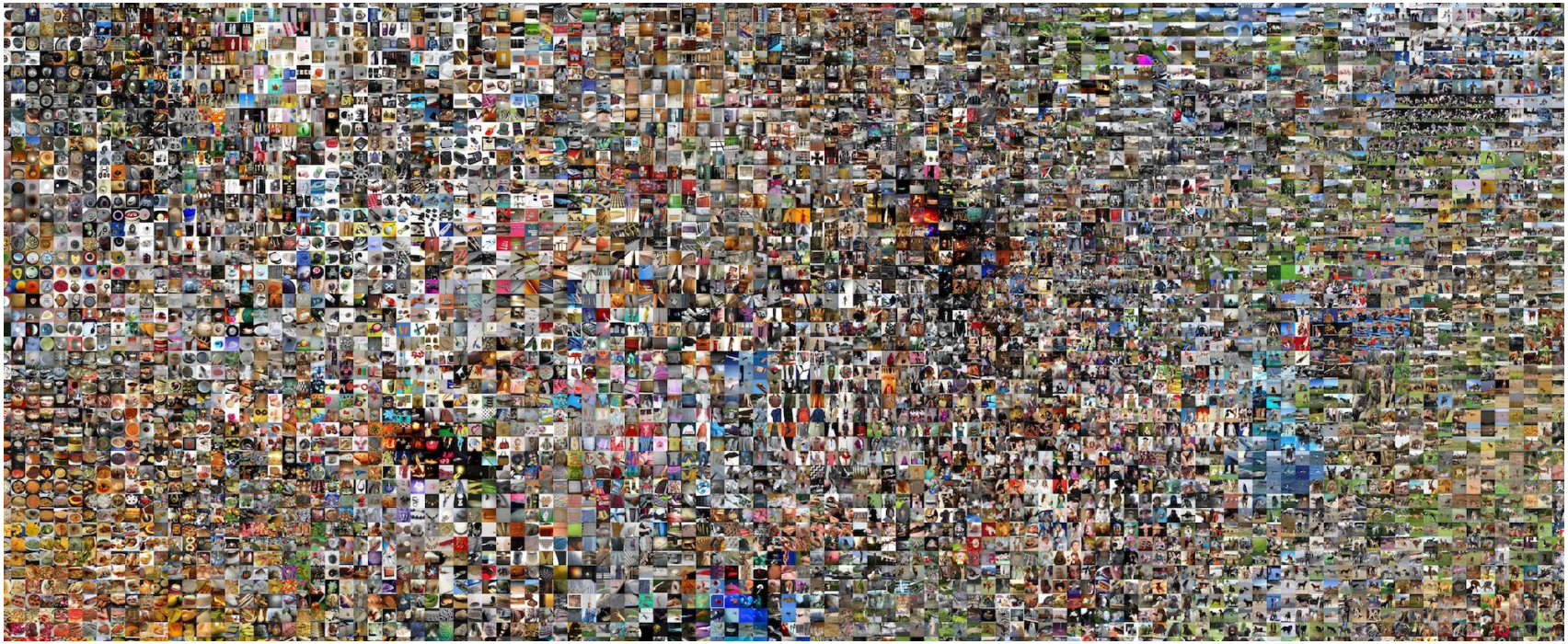


2006 Restricted Boltzmann Machine

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.

Vision par ordinateur – Difficultés

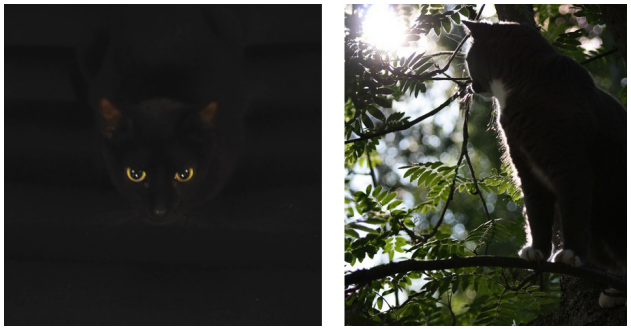
Pourquoi est-ce difficile pour l'ordinateur ?

(a) Points de vue

Vision par ordinateur – Difficultés

Pourquoi est-ce difficile pour l'ordinateur ?

(a) Points de vue

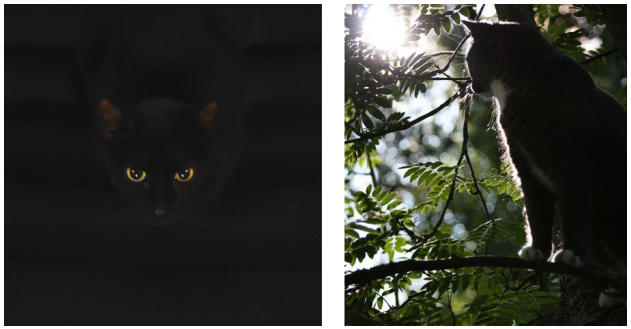


(b) Conditions d'illumination

Vision par ordinateur – Difficultés

Pourquoi est-ce difficile pour l'ordinateur ?

(a) Points de vue



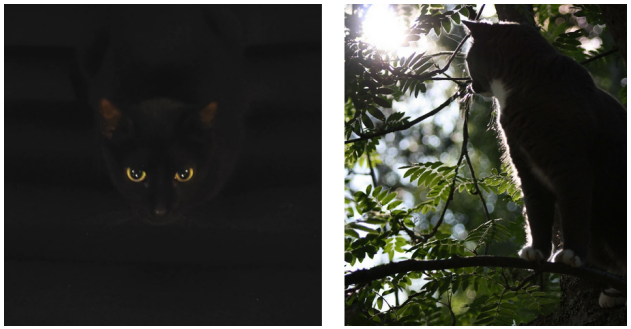
(b) Conditions d'illumination

(c) Variabilité intra-classe

Vision par ordinateur – Difficultés

Pourquoi est-ce difficile pour l'ordinateur ?

(a) Points de vue



(b) Conditions d'illumination

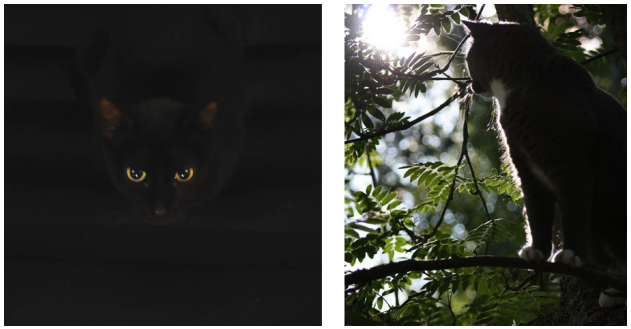
(c) Variabilité intra-classe

(d) Occultations

Vision par ordinateur – Difficultés

Pourquoi est-ce difficile pour l'ordinateur ?

(a) Points de vue



(b) Conditions d'illumination

(c) Variabilité intra-classe

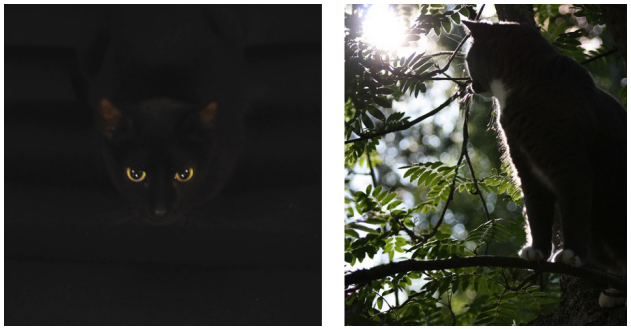
(d) Occultations

(e) Objets déformables

Vision par ordinateur – Difficultés

Pourquoi est-ce difficile pour l'ordinateur ?

(a) Points de vue



(b) Conditions d'illumination

(c) Variabilité intra-classe

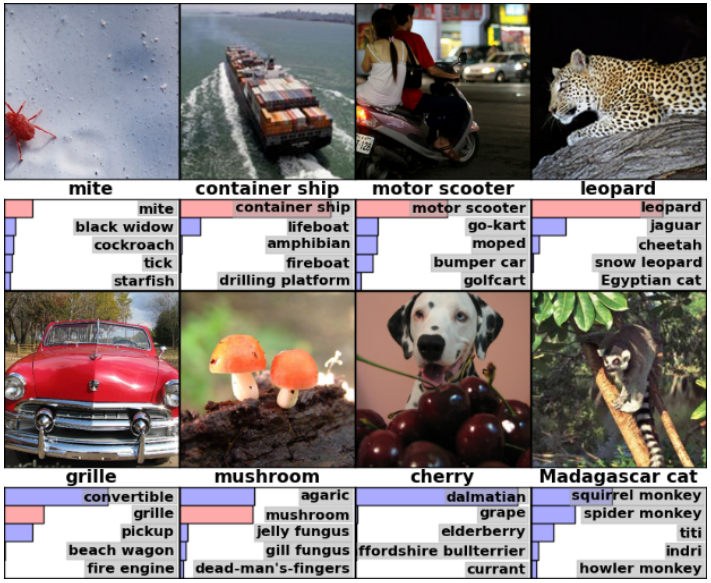
(d) Occultations

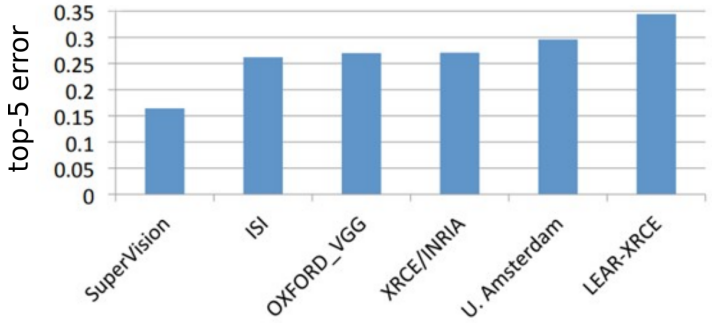
(e) Objets déformables

(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 ?

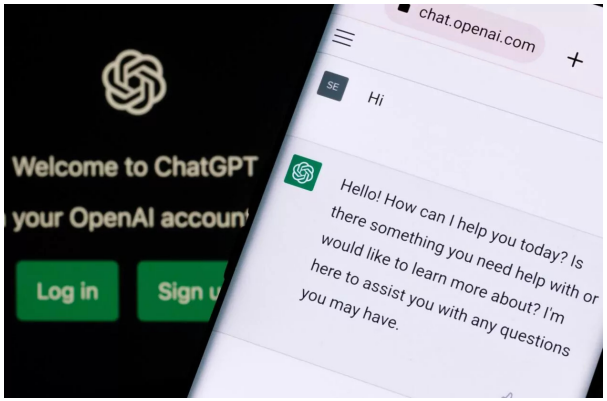
Mme Tang Yu, PDG du chinois NetDragon Websoft et de ses 6000 employés, est le premier robot à être nommé à la tête d'une société. Disponible H24, elle ne touche aucun salaire. *NetDragon Websoft*

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 ?

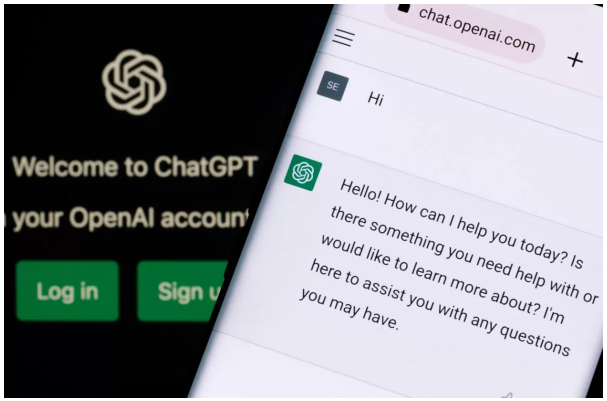
Mme Tang Yu, PDG du chinois NetDragon Websoft et de ses 6000 employés, est le premier robot à être nommé à la tête d'une société. Disponible H24, elle ne touche aucun salaire. *NetDragon Websoft*

Diagnostic médical

Biais de population ?

Bots conversationnels

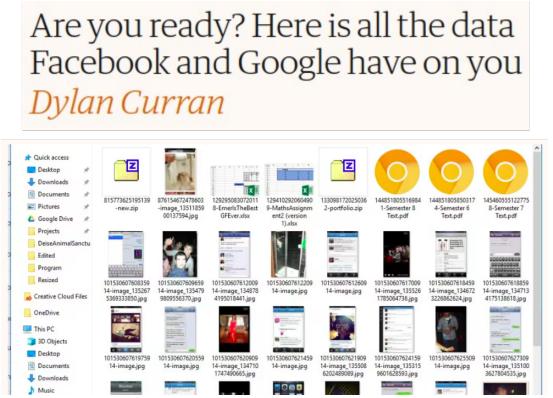
Vérification des informations / Triche ?



Interface de ChatGPT (OpenAI)

Big Data

Utilisation des données privées ?

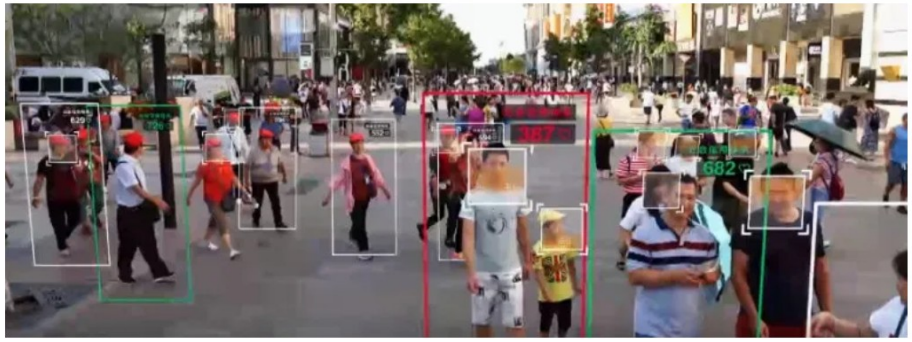


[Lien : Données privées possédées par Google]

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

Vidéo surveillance

Outil de contrôle / Abus ?

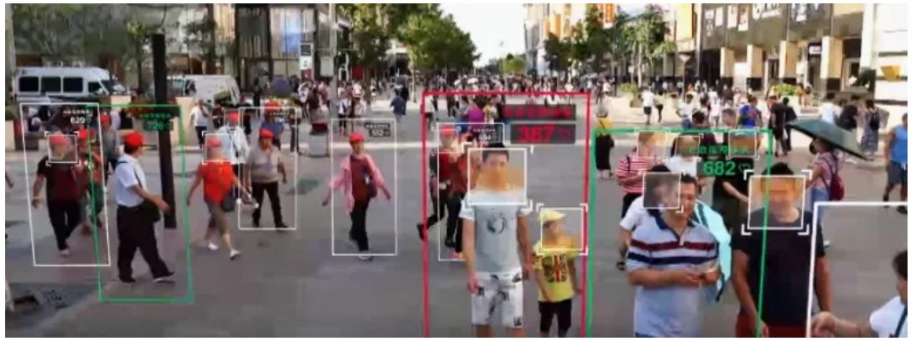


The Chinese state wants to control its citizens via a system of social scoring that punishes behavior it doesn't approve of. Image Credit: Telecoms

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

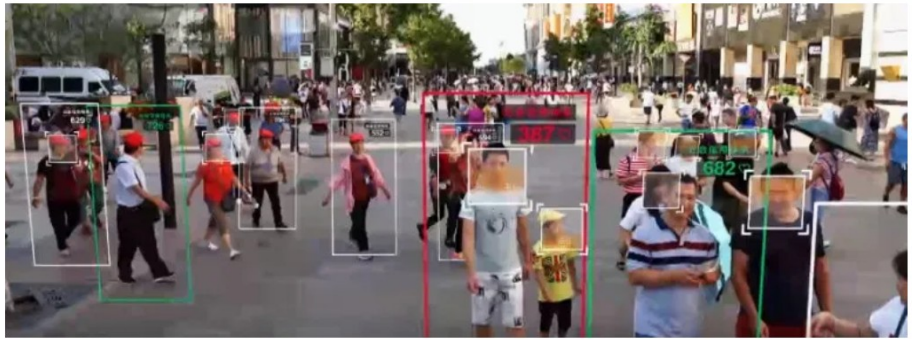
Responsabilité en cas d'accident ?

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 ?

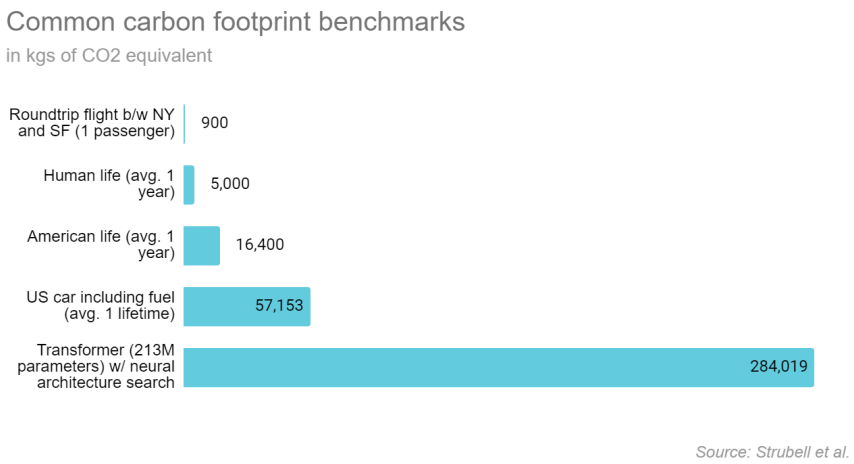
Image générée automatiquement par Midjourney

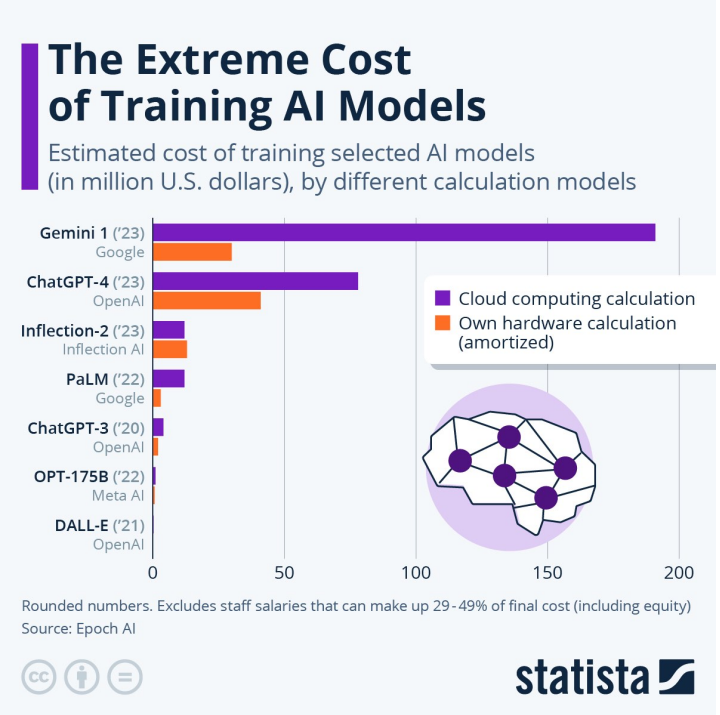
Propriété intellectuelle ?

Générique de One Piece chanté par Johnny Hallyday

La dimension écologique

Entraîner des modèles, ça prend du temps et ça consomme





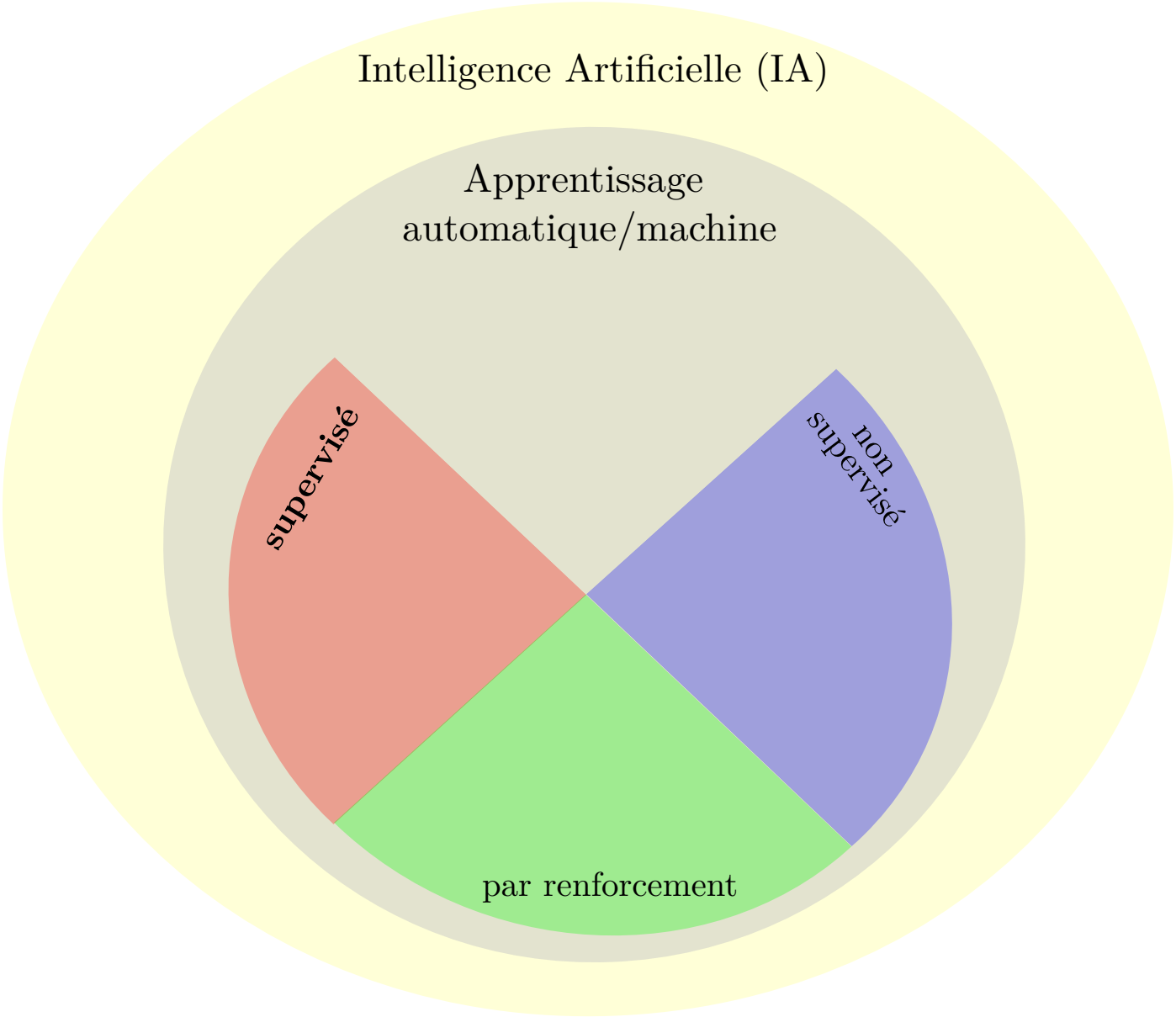
Quelques outils :

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

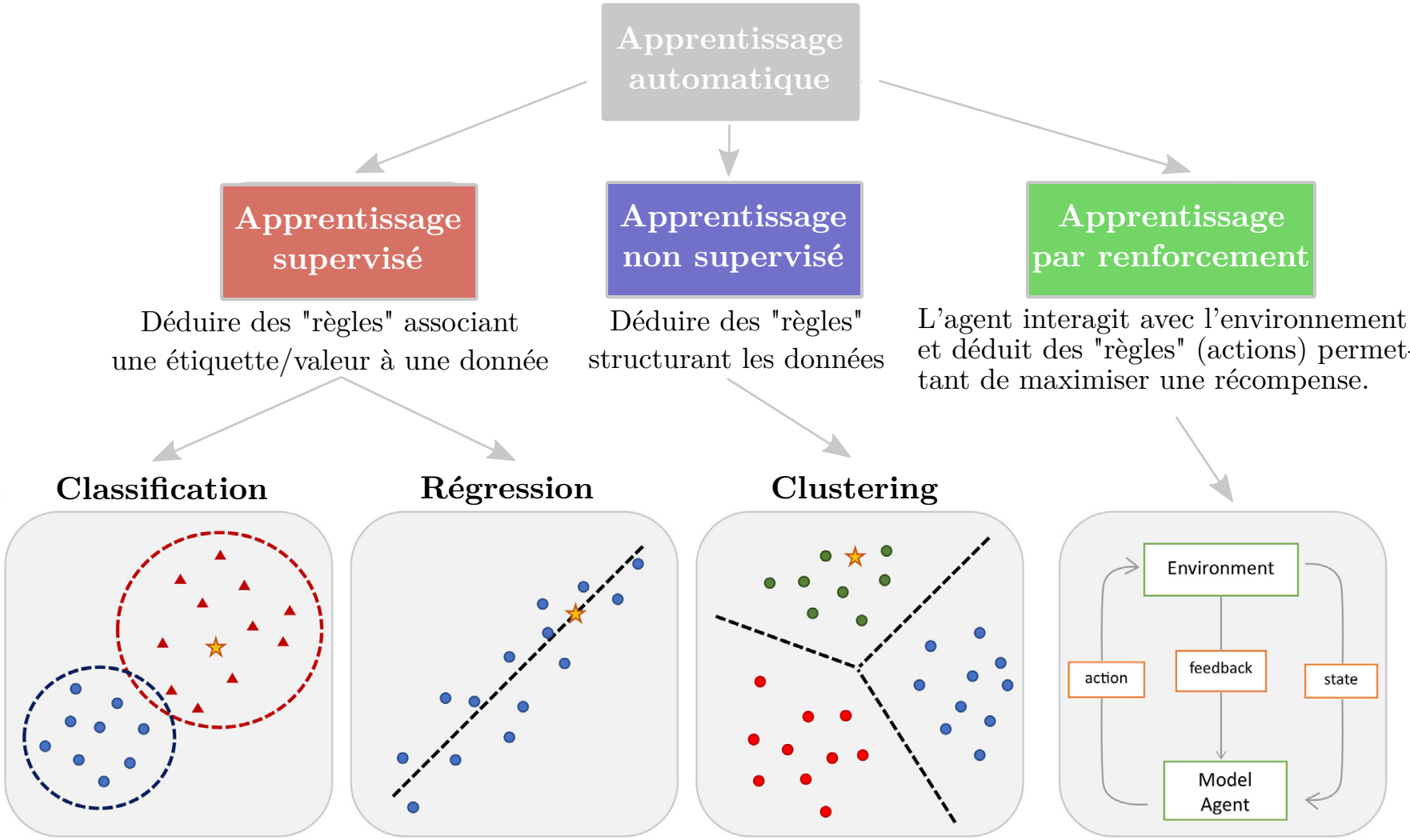
Plan du cours

Apprentissage supervisé

Différents types d'apprentissage

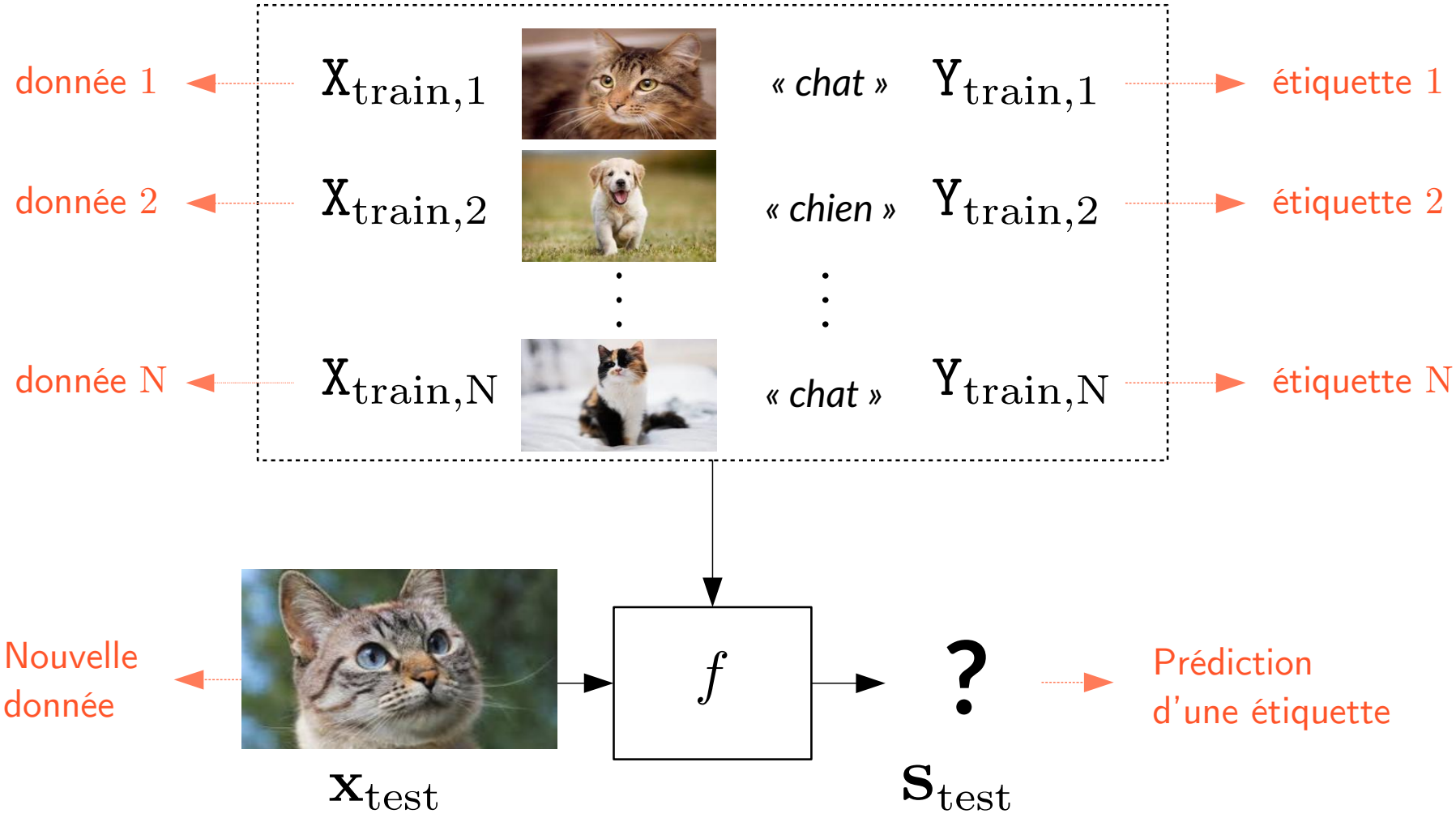


Différents types d'apprentissage

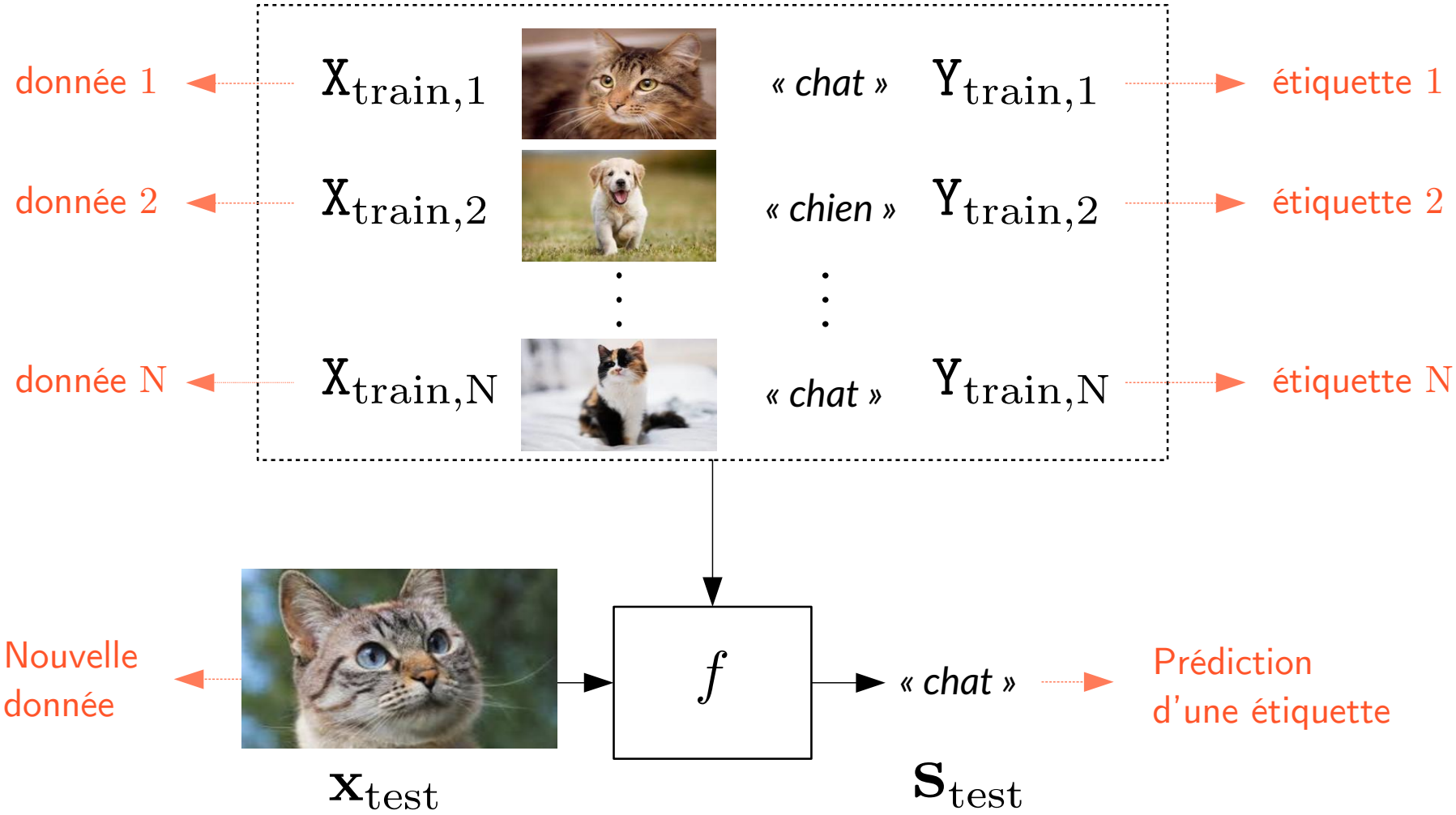


Source : [Peng et al. 2021] - Machine Learning Techniques for Personalised Medicine Approaches in Immune-Mediated Chronic Inflammatory Diseases: Applications and Challenges. In Frontiers in Pharmacology.

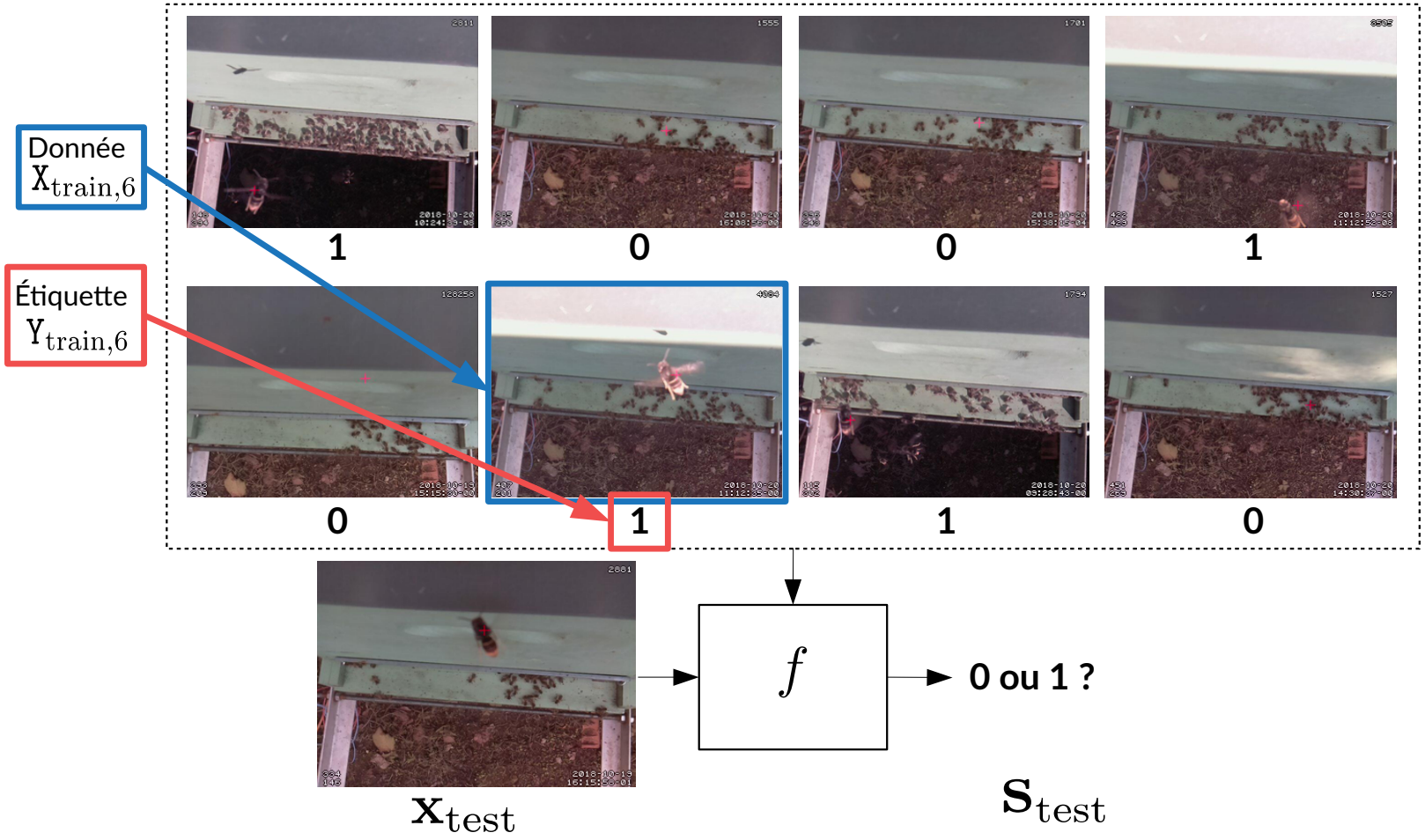
Exemple d'un classifieur chat / chien



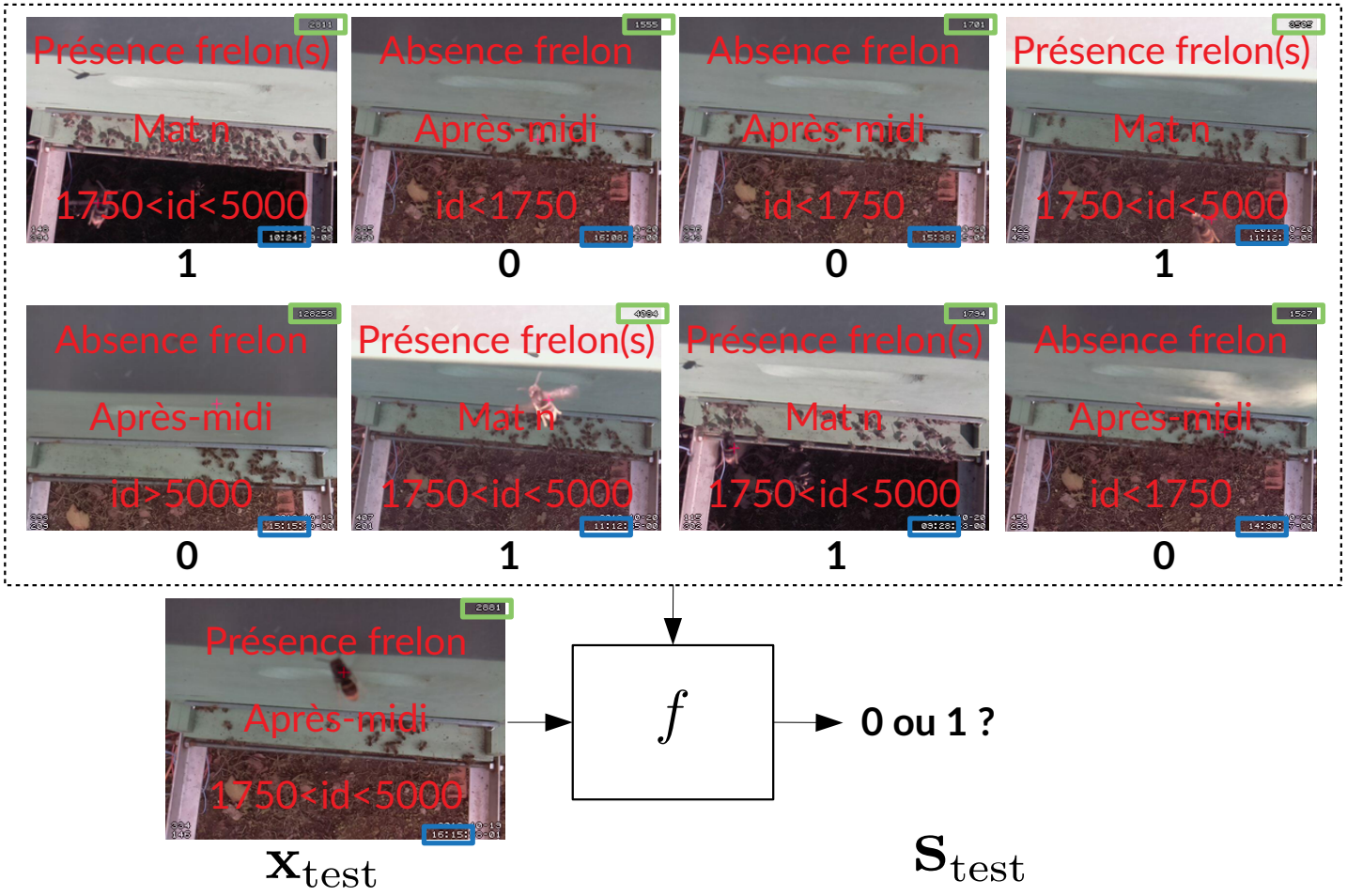
Exemple d'un classifieur chat / chien



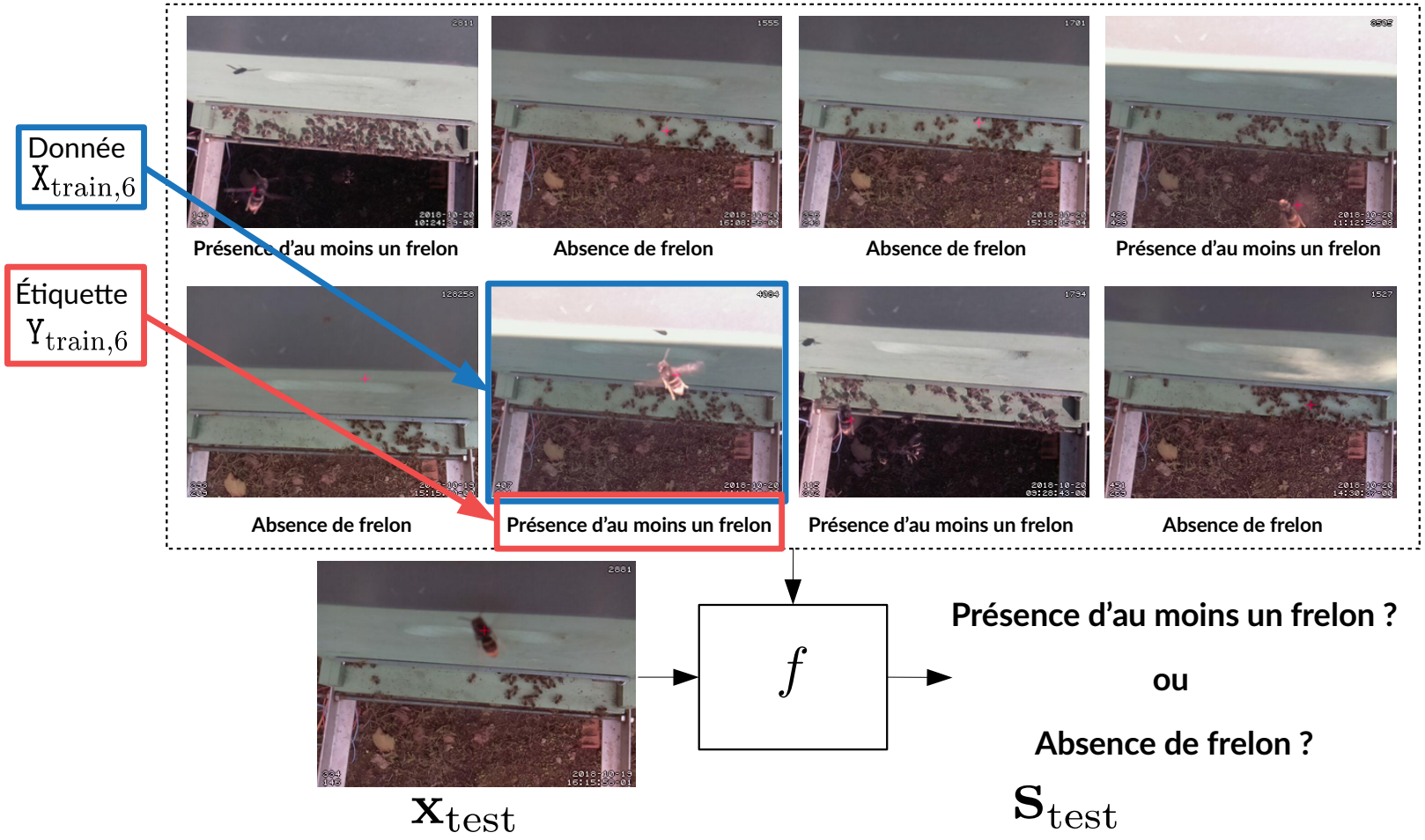
Pouvez-vous résoudre ce problème d'apprentissage supervisé ?



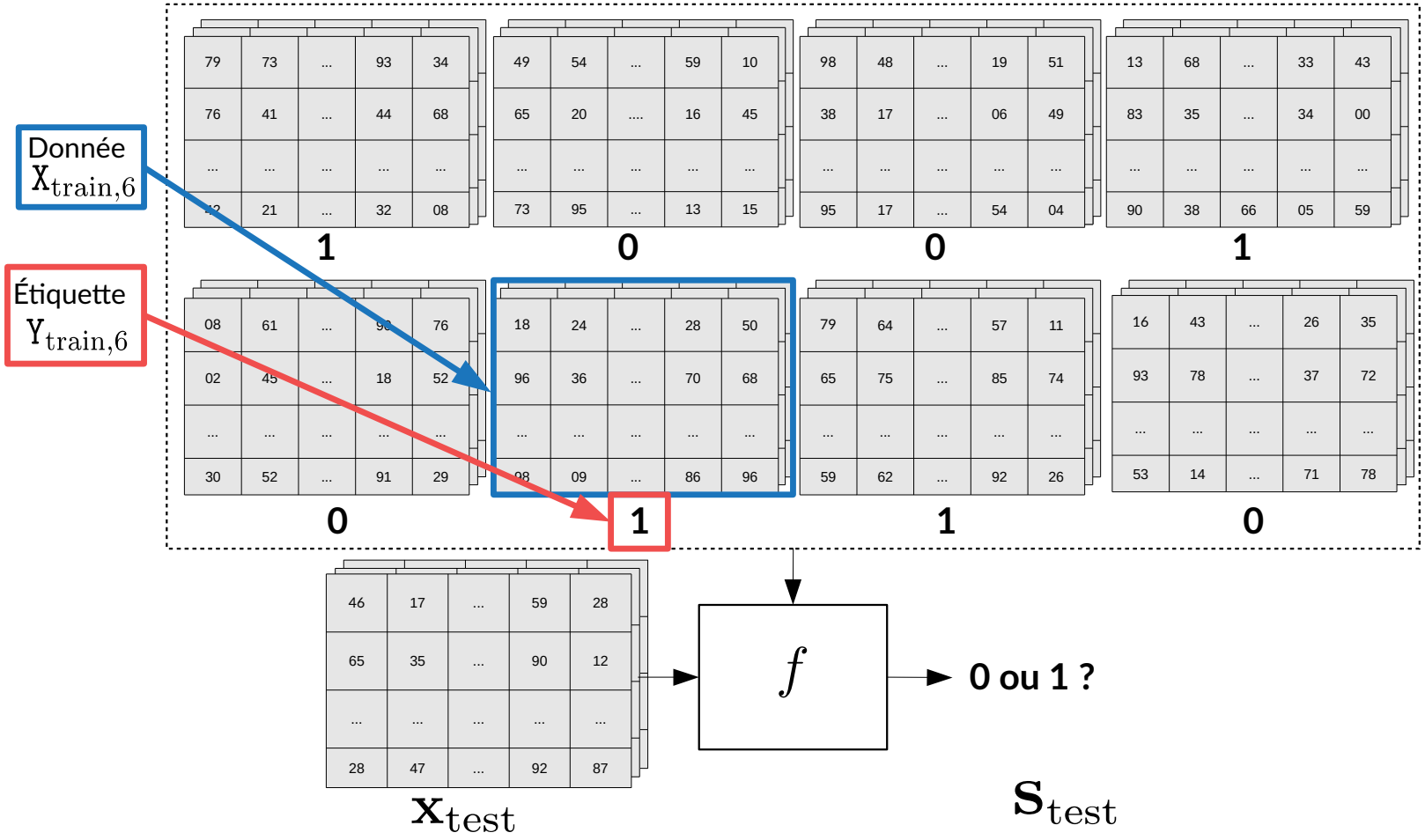
Mais de quel problème parle-t-on au juste ?



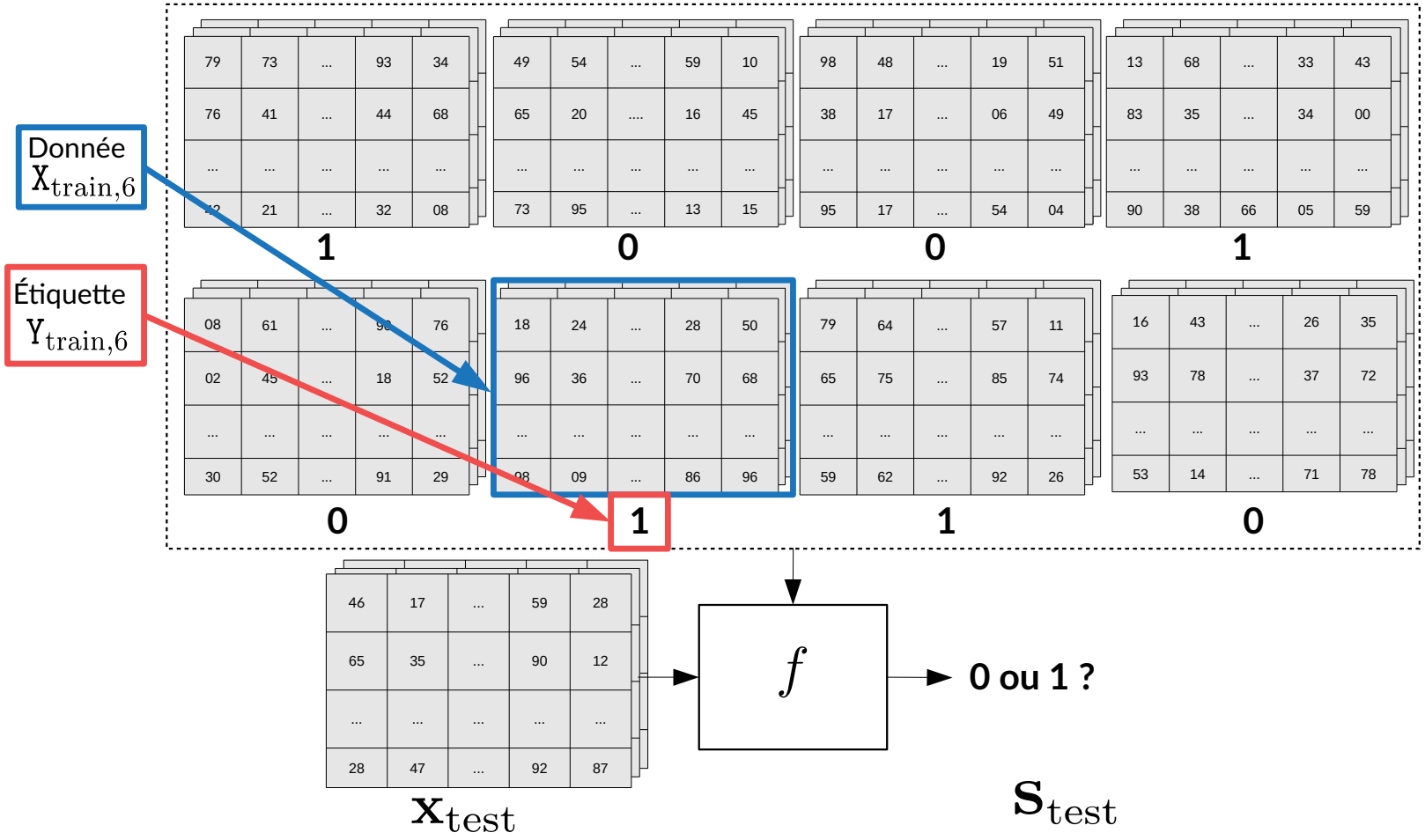
On veut que l'ordinateur apprenne à résoudre le problème suivant



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



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



Les règles déduites ne seront (probablement) pas interprétables → "Boîte noire"

C'est nous qui interprétons : "0" = présence frelon ; "1" = absence frelon

Résumé

- L'apprentissage supervisé utilise des **données d'entraînement** assorties d'**étiquettes**
- L'ordinateur apprend "des règles"/une **fonction** qui effectue des calculs sur des tableaux de valeurs numériques et produit une valeur numérique en sortie.
- Dans le cas d'une classification, nous **interprétons** cette valeur numérique en sortie comme une étiquette "sémantique".

- **Questions ?**
  - Quelle 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

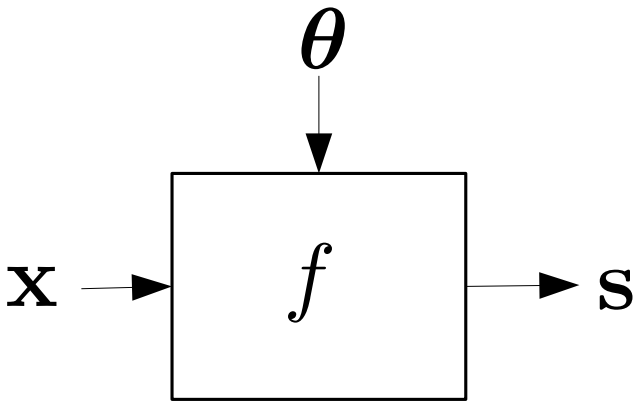
- Mathématique $\mathbf{s} = f(\mathbf{x}; \boldsymbol{\theta})$ $\boldsymbol{\theta}$ = paramètres

---

- Informatique (Python)

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

- Graphique (Graphe de calcul)



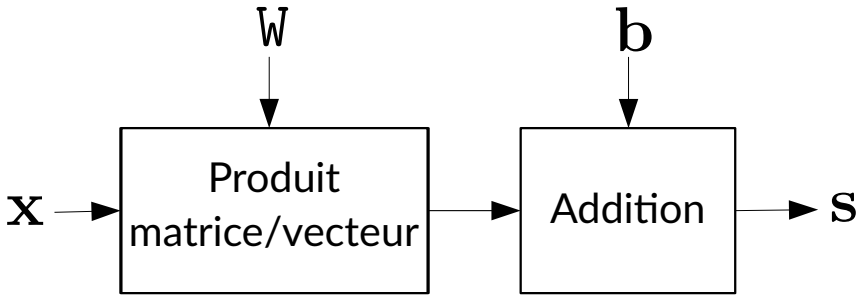
Exemple : transformation affine

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

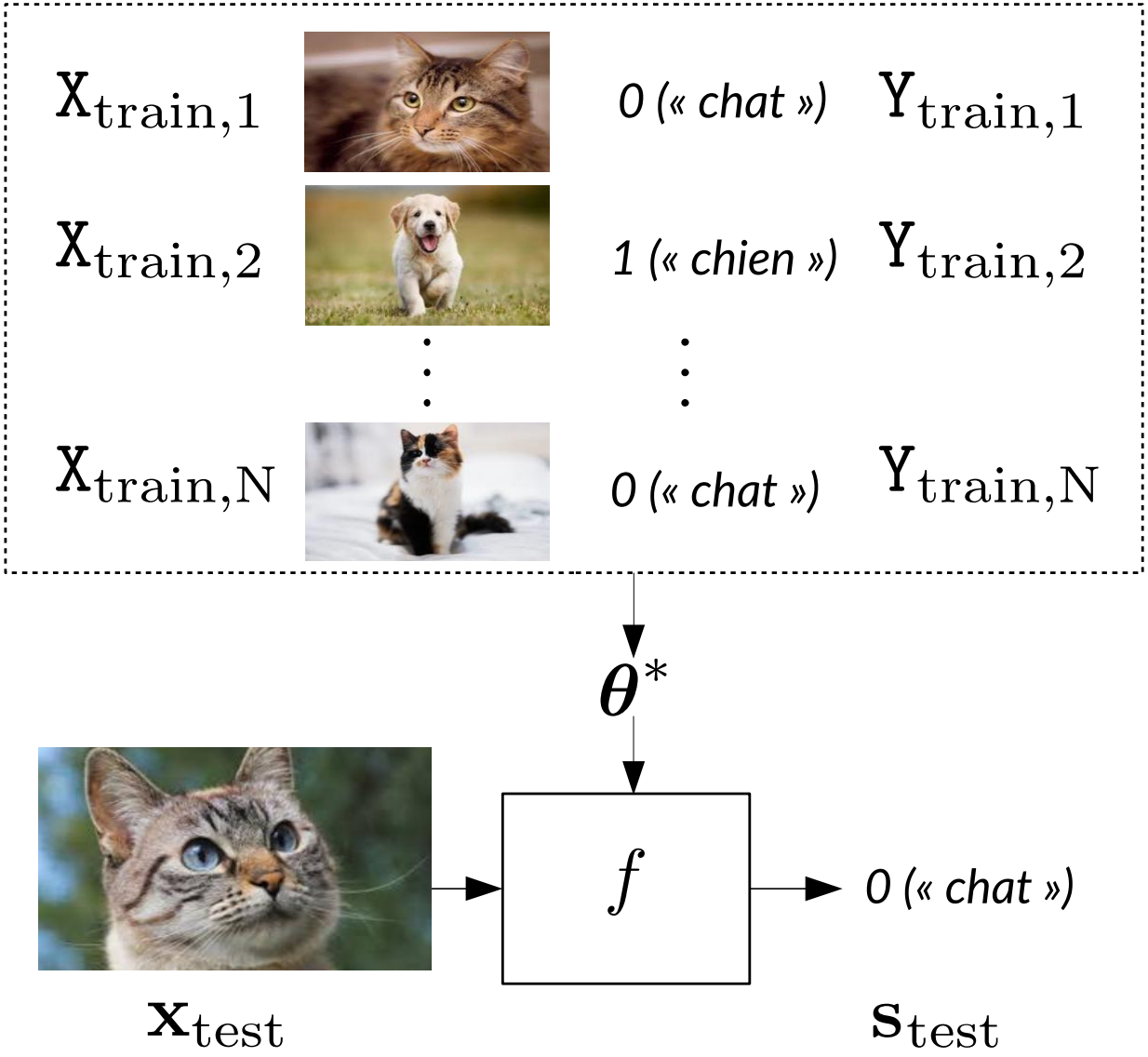
- Informatique (Python)

```
def f(x, W, b):
    s = x @ W +b
    return s
```

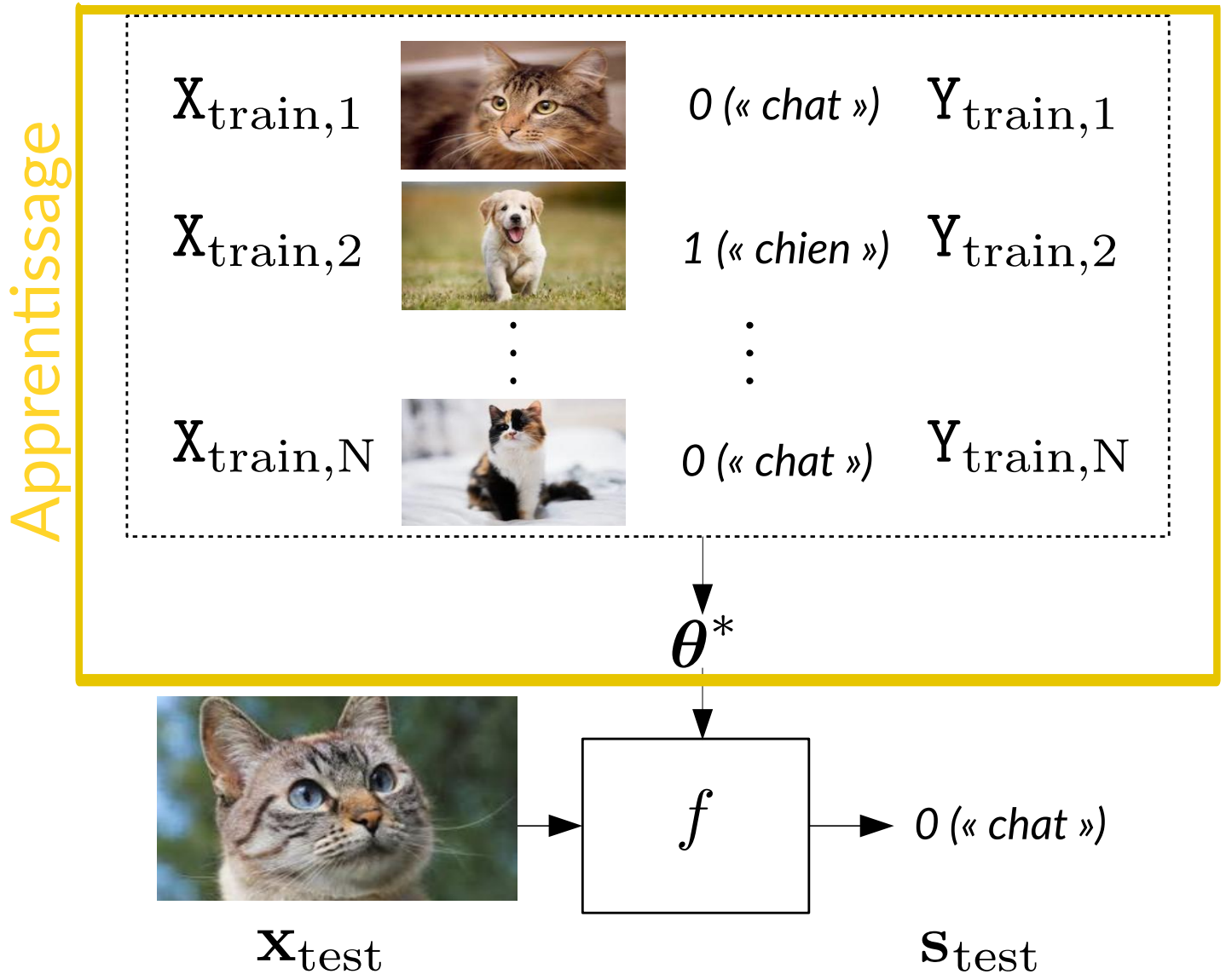
- Graphique (Graphe de calcul)



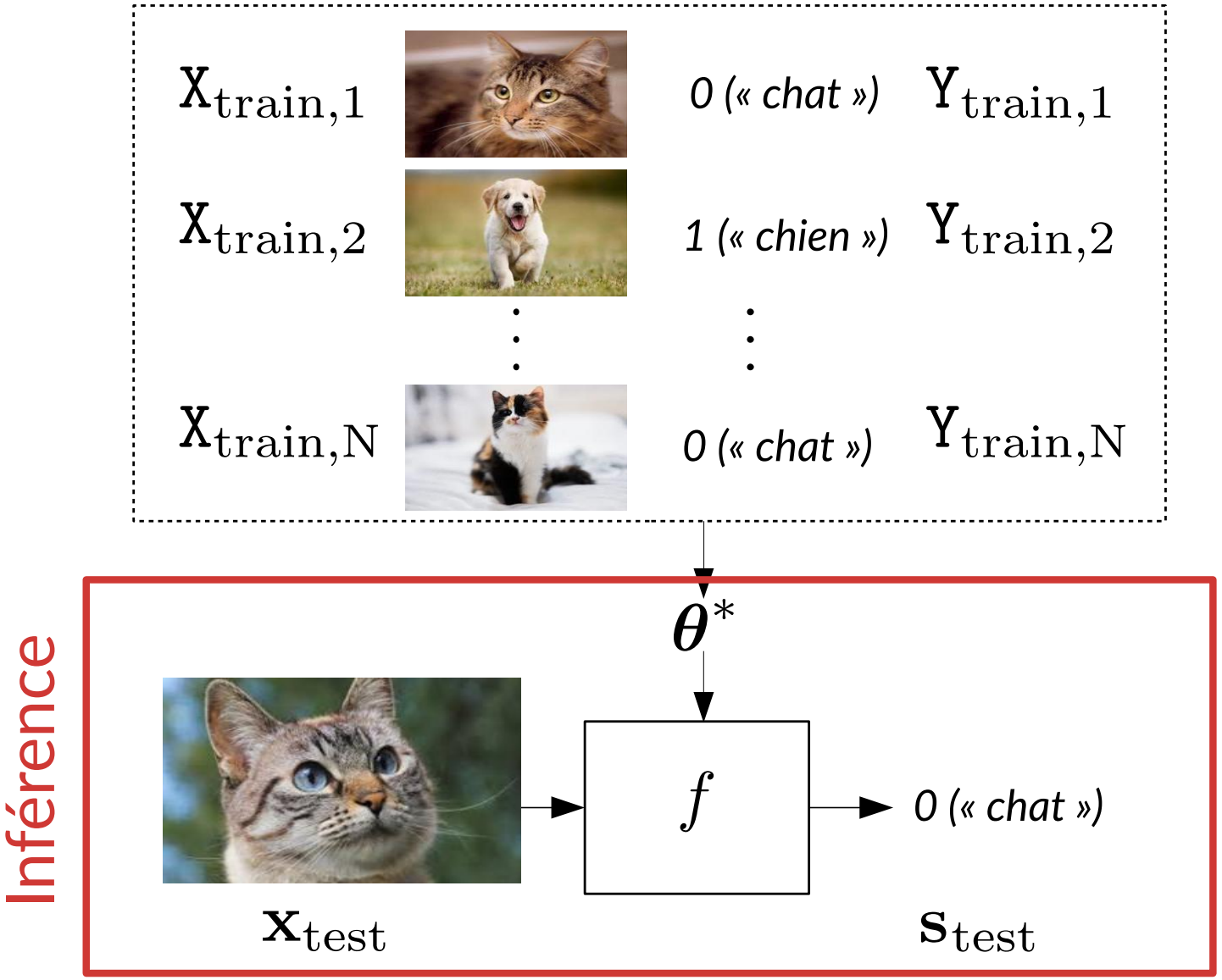
Les différentes étapes



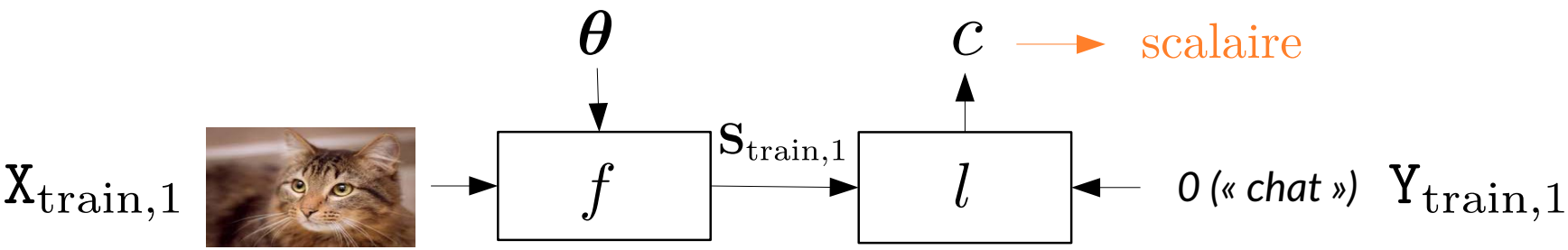
Les différentes étapes



Les différentes étapes

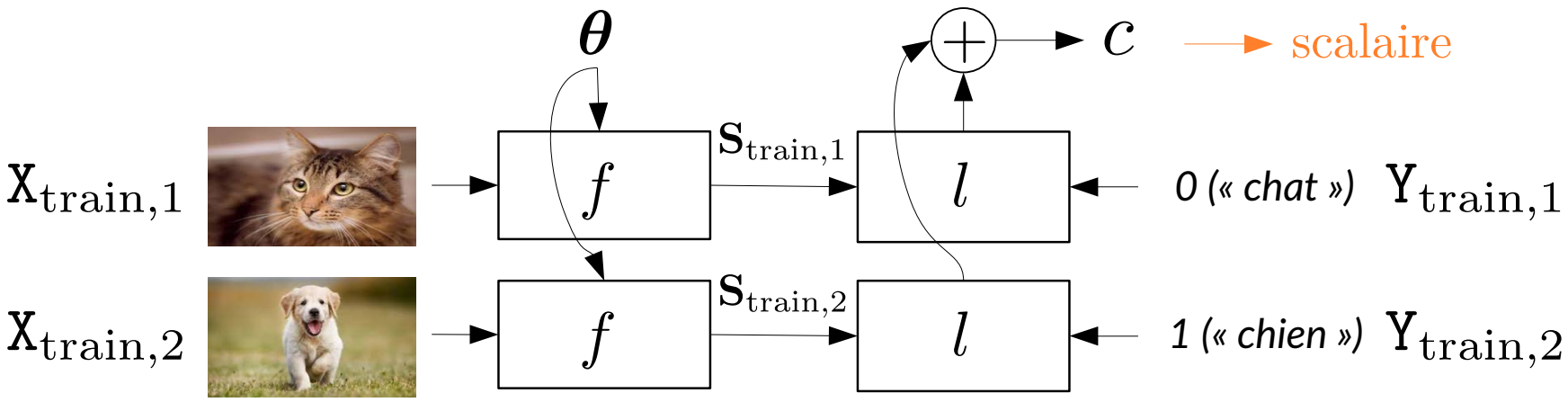


Étape d'apprentissage (*"training time"*)



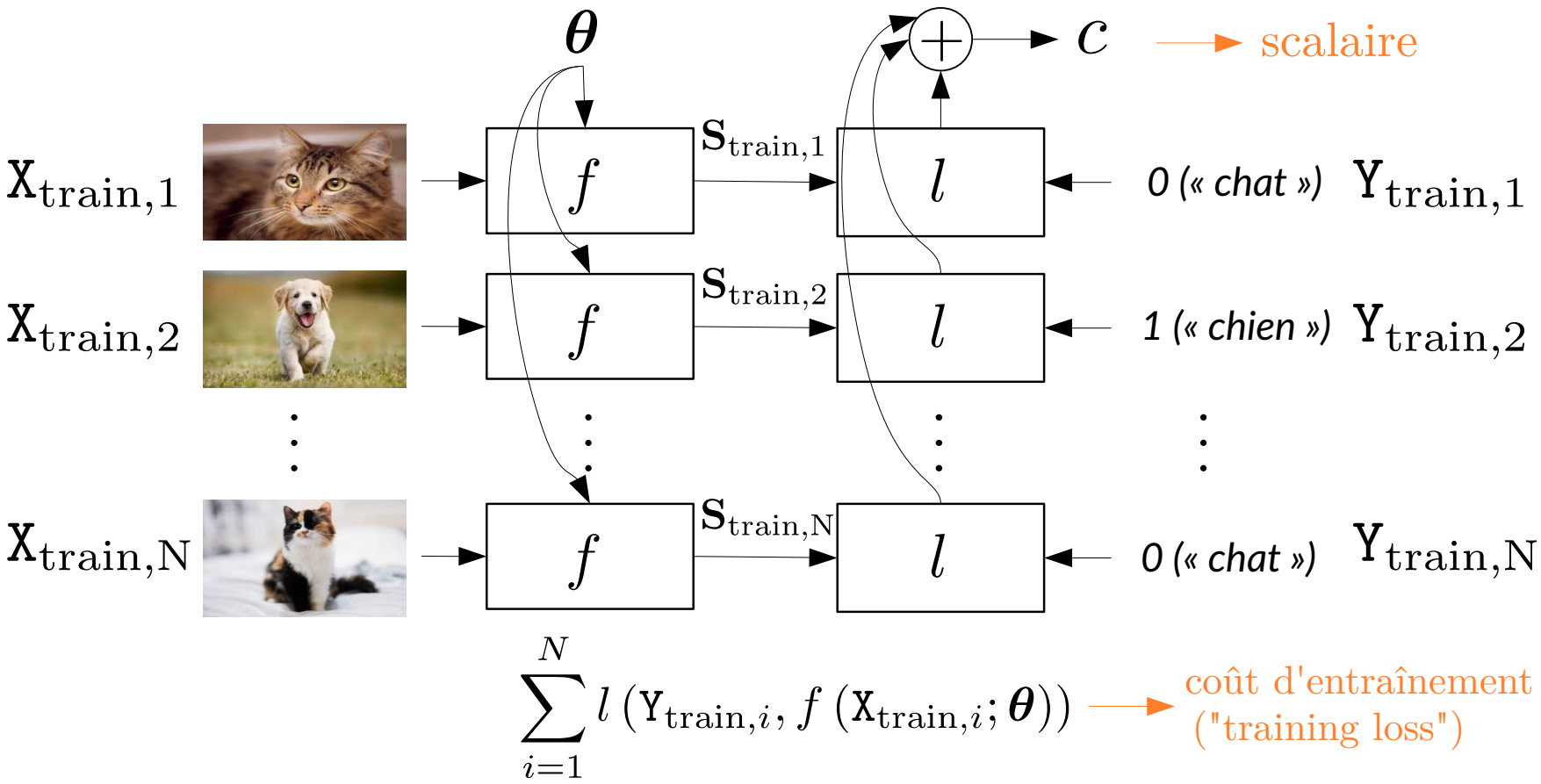
Où l est une **fonction de coût** ("loss function") à choisir, permettant de comparer la prédiction du réseau à l'étiquette

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



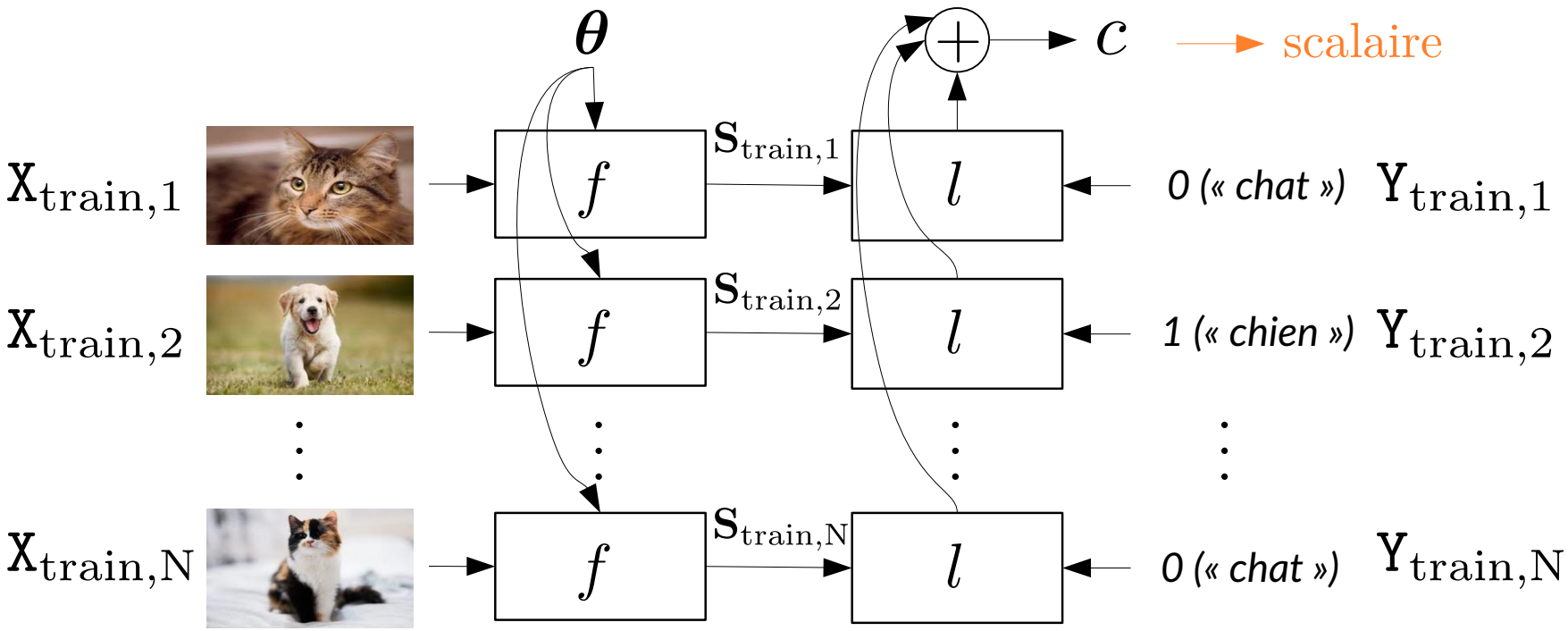
Où l est une **fonction de coût** ("loss function") à choisir, permettant de comparer la prédiction du réseau à l'étiquette

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



Où l est une **fonction de coût** ("loss function") à choisir, permettant de comparer la prédiction du réseau à l'étiquette

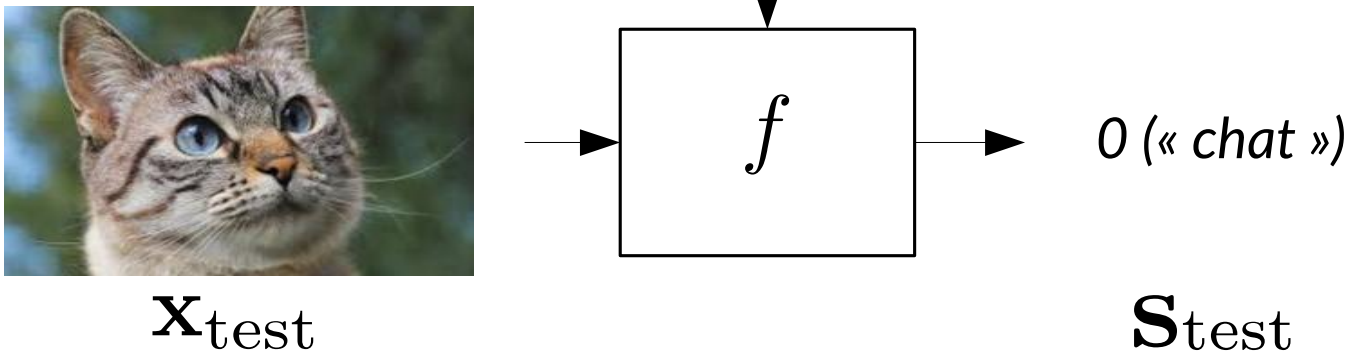
Étape d'apprentissage (*"training time"*)



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

Où l est une **fonction de coût** ("loss function") à choisir, permettant de comparer la prédiction du réseau à l'étiquette

Étape d'inférence (*"test time"*)



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

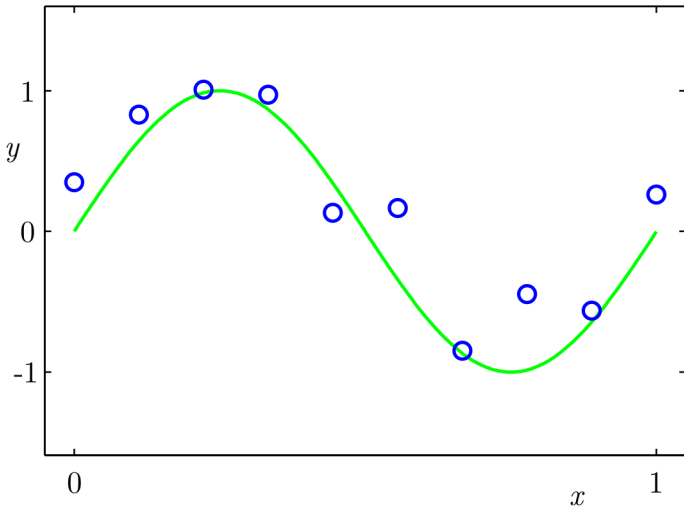
Exemple : Régression linéaire polynomiale

Régression linéaire en 1D :

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

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

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



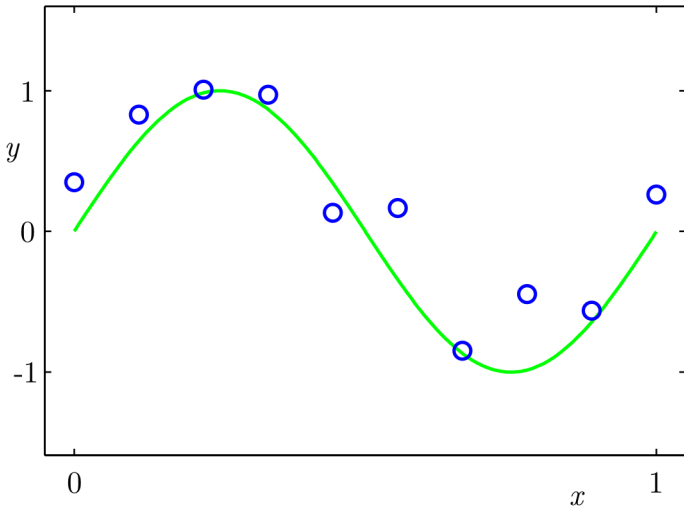
Exemple : Régression linéaire polynomiale

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 s ($\hat{y}$) pour une entrée x : $s = f(x)$



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

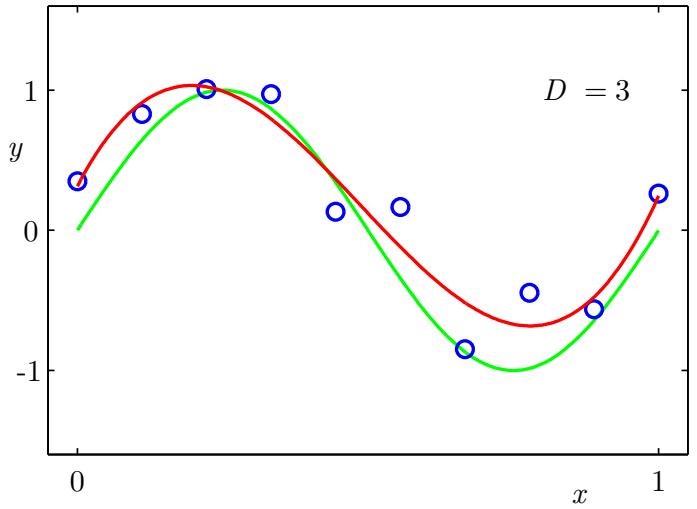
$$f(x; \boldsymbol{\theta}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_D x^D = \sum_{d=0}^{D} \theta_d x^d$$

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 $s_i = f(x_i; \boldsymbol{\theta}^*)$ et y_i dans $\mathsf{D}_{\text{train}}$



Fonction de coût : erreur quadratique

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

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

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

Inférence :

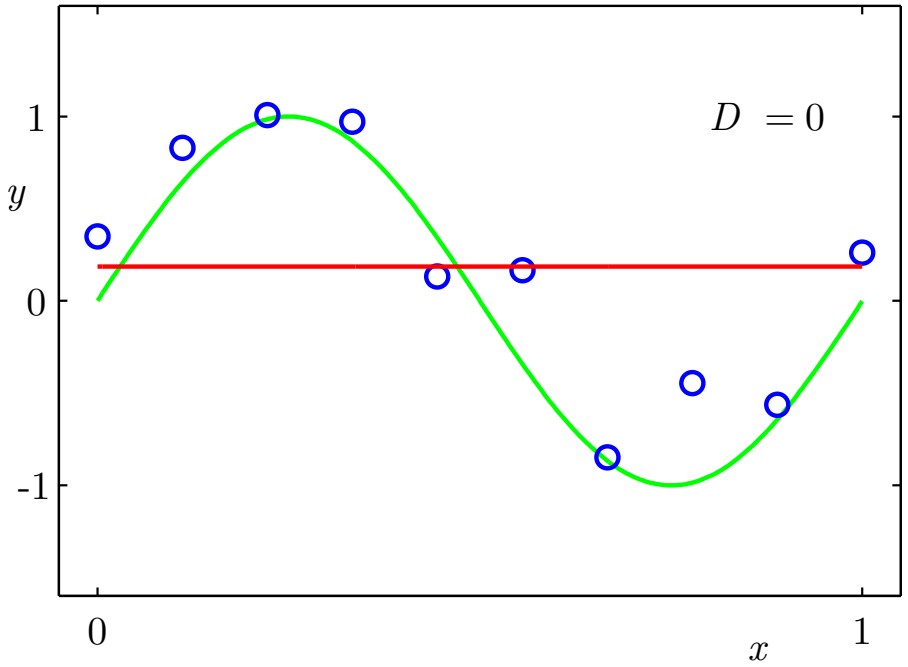
$$s_{\text{test}} = f(x_{\text{test}}; \boldsymbol{\theta}^*) = \theta_0^* + \theta_1^* x_{\text{test}} + \ldots + \theta_D^* x_{\text{test}}^D = \sum_{d=0}^{D} \theta_d^* x_{\text{test}}^d$$

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

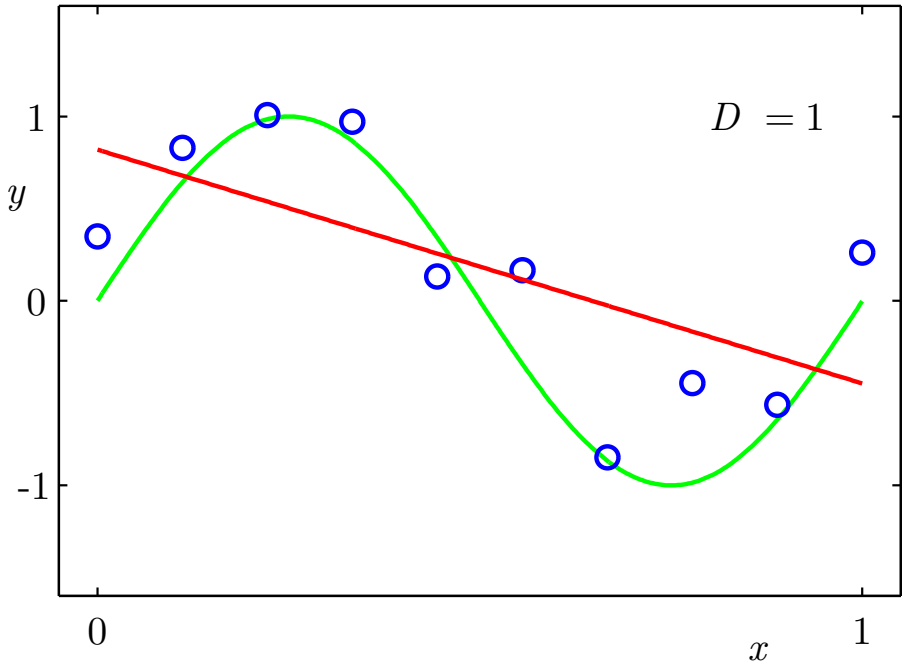
$\rightarrow$ **Sous-apprentissage (*underfitting*)**

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

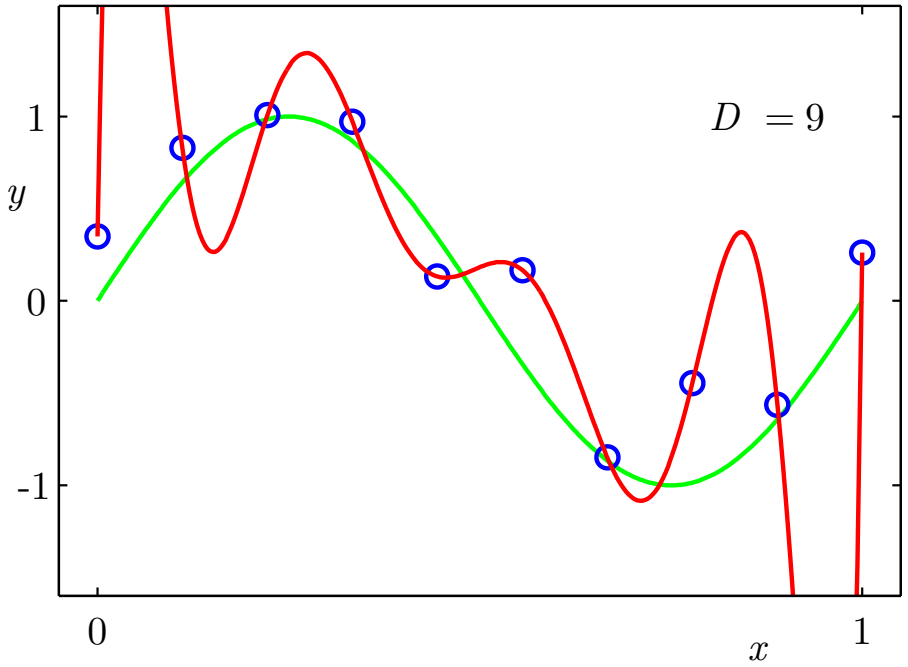
$\rightarrow$ **Sous-apprentissage (*underfitting*)**

D est appelé **hyperparamètre** du modèle

Choix de la dimension D

Comment trouver un "bon" D ?

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



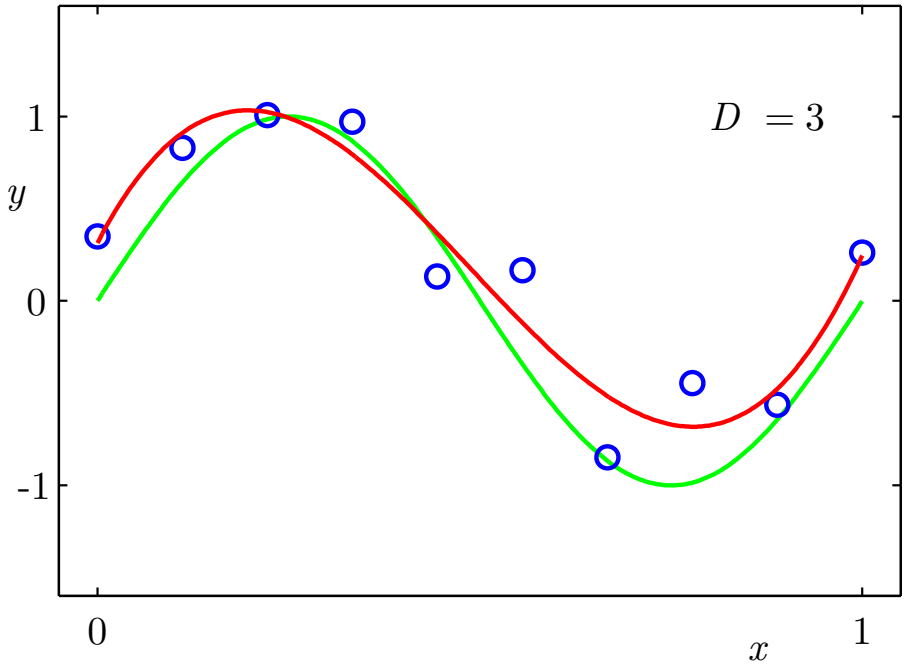
Trop forte : le modèle apprend "par cœur" les données d'entraînement, représentation trop complexe
$\rightarrow$ **Sur-apprentissage (*overfitting*)**

D est appelé **hyperparamètre** du modèle

Choix de la dimension D

Comment trouver un "bon" D ?

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

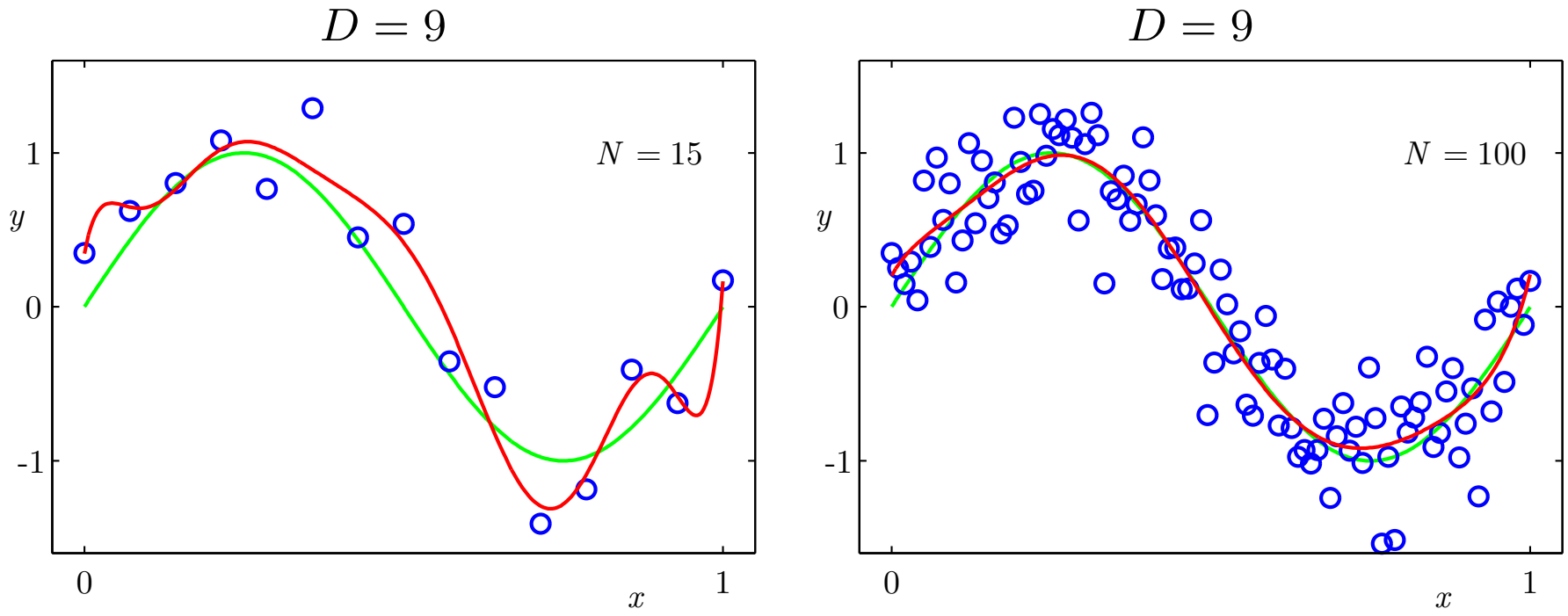


On cherche un bon compromis qui trouve la tendance générale, mais sans le bruit, pour **généraliser** aux nouvelles données de test

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**.



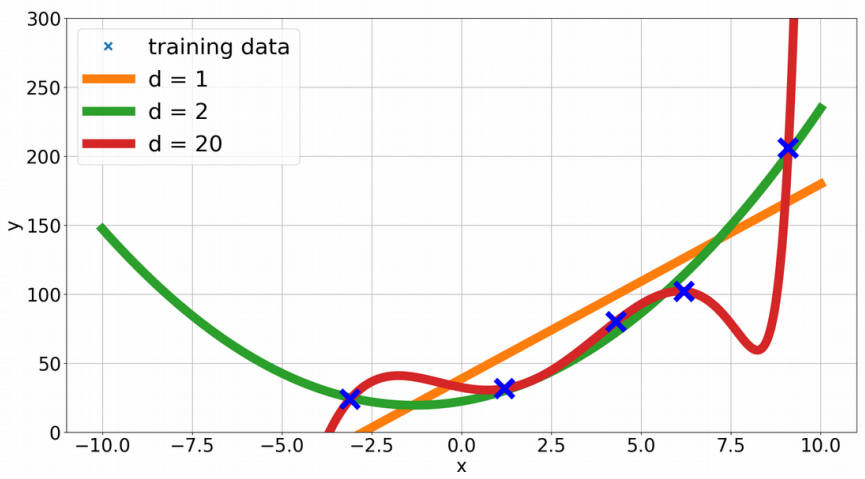
Exemple de régression : $N = 5$, $\mathbf{X} \in \mathbb{R}$ et $\mathbf{Y} \in \mathbb{R}$

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

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

Fonction de coût :

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



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 :

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

Avantages et inconvénients d'une approche paramétrique

Avantages

- Inférence efficace $\rightarrow$ pas d'accès à la base de données étiquetées.
- Temps d'inférence constant $\rightarrow$ ne dépend pas de la taille de la base de données étiquetées.

Inconvénients

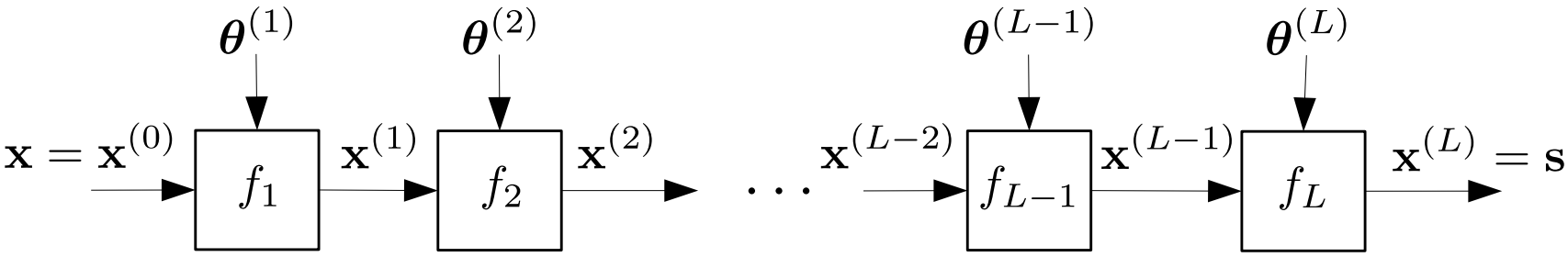
- Choix de la fonction paramétrique et de ses hyperparamètres.
- Étape d'apprentissage $\rightarrow$ souvent longue et gourmande en calculs.
- Difficile de modifier la base de données étiquetées $\rightarrow$ nécessite de faire un nouvel apprentissage.

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



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

Réseau de neurones = Composition de fonctions paramétriques

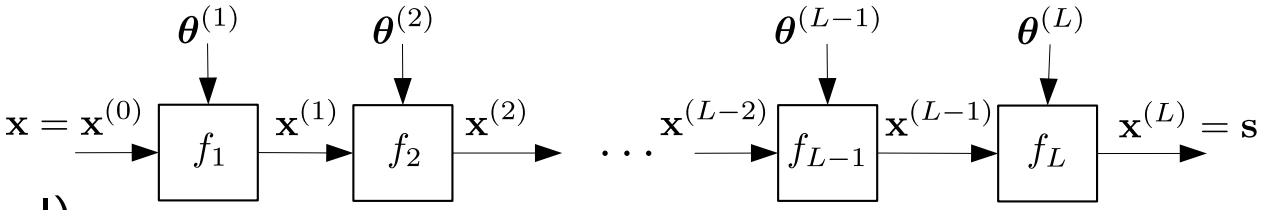
- Mathématique

$$s = f(\mathbf{x}; \boldsymbol{\theta}) = f_L(f_{L-1}(...f_2(f_1(\mathbf{x}, \boldsymbol{\theta}_1); \boldsymbol{\theta}_2)...; \boldsymbol{\theta}_{L-1}); \boldsymbol{\theta}_L)$$

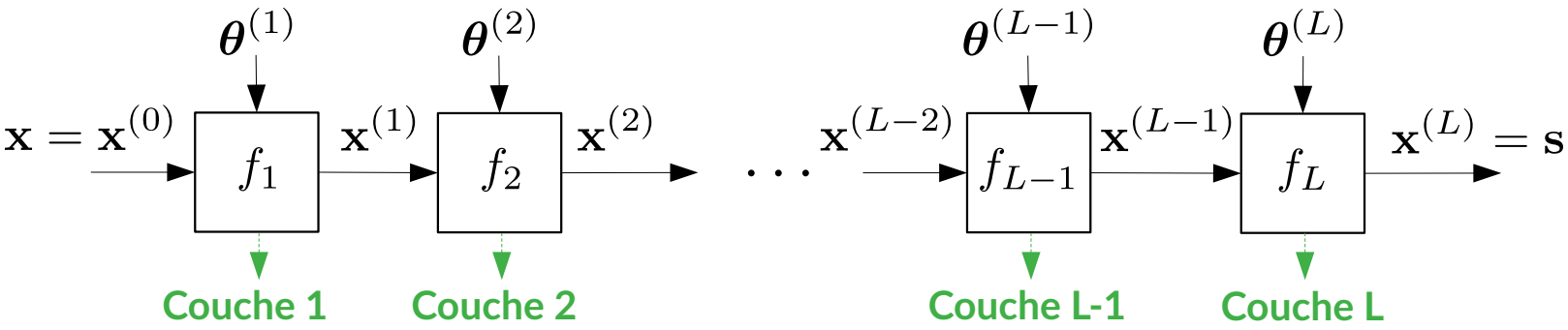
- Informatique (Python)

```python
def neuralNetwork_forward(x, theta, L):
    x1 = f1(x, theta[0])
    x2 = f2(x1, theta[1])
    ...
    s  = fL(..., theta[L-1])
    return s
```

- Graphique (Graphe de calcul)

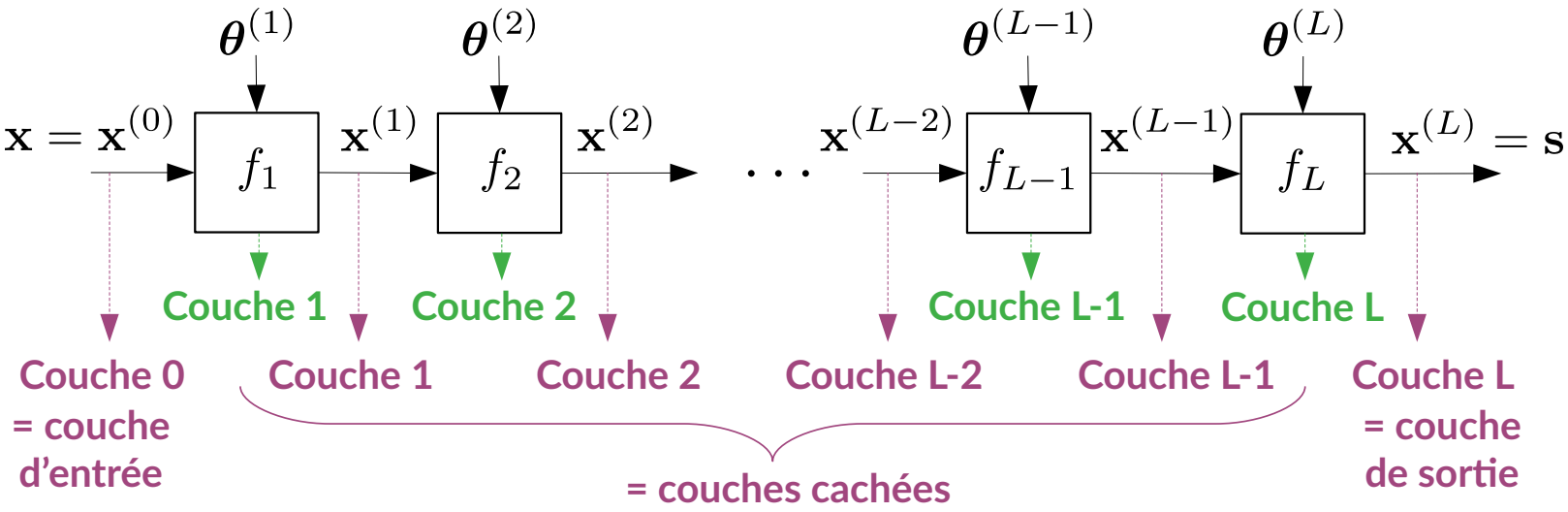


Réseaux de neurones : Terminologie



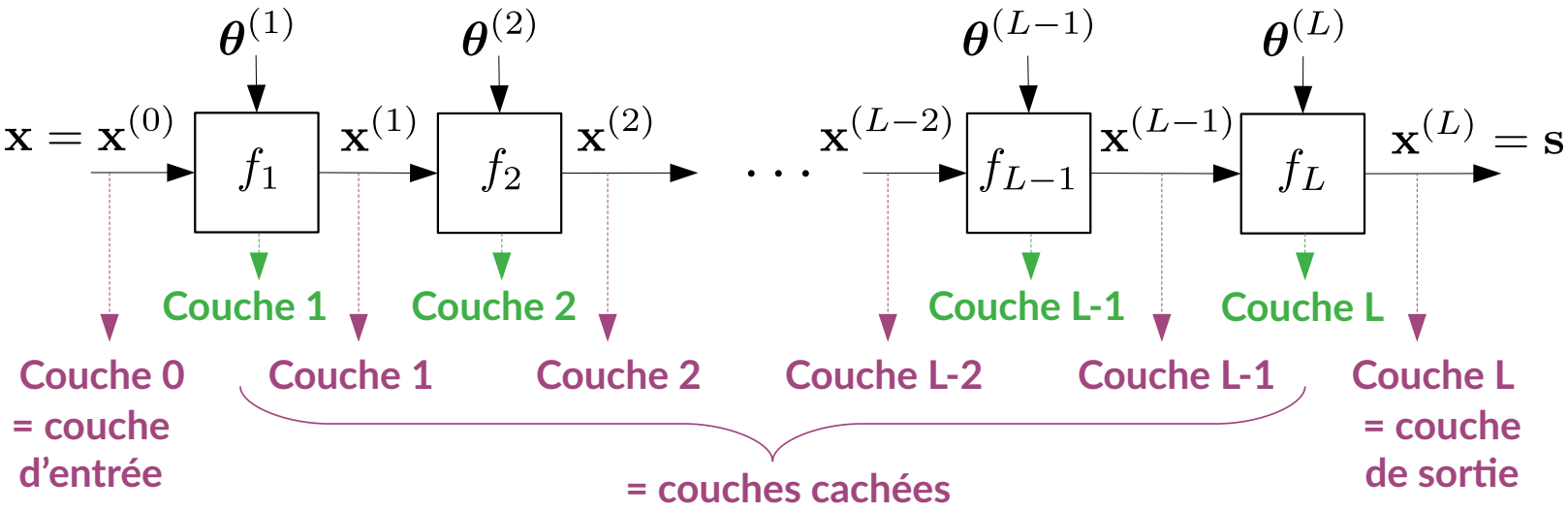
- Couche ("layer") :
  - — Sens 1 : Une fonction paramétrique du réseau

Réseaux de neurones : Terminologie



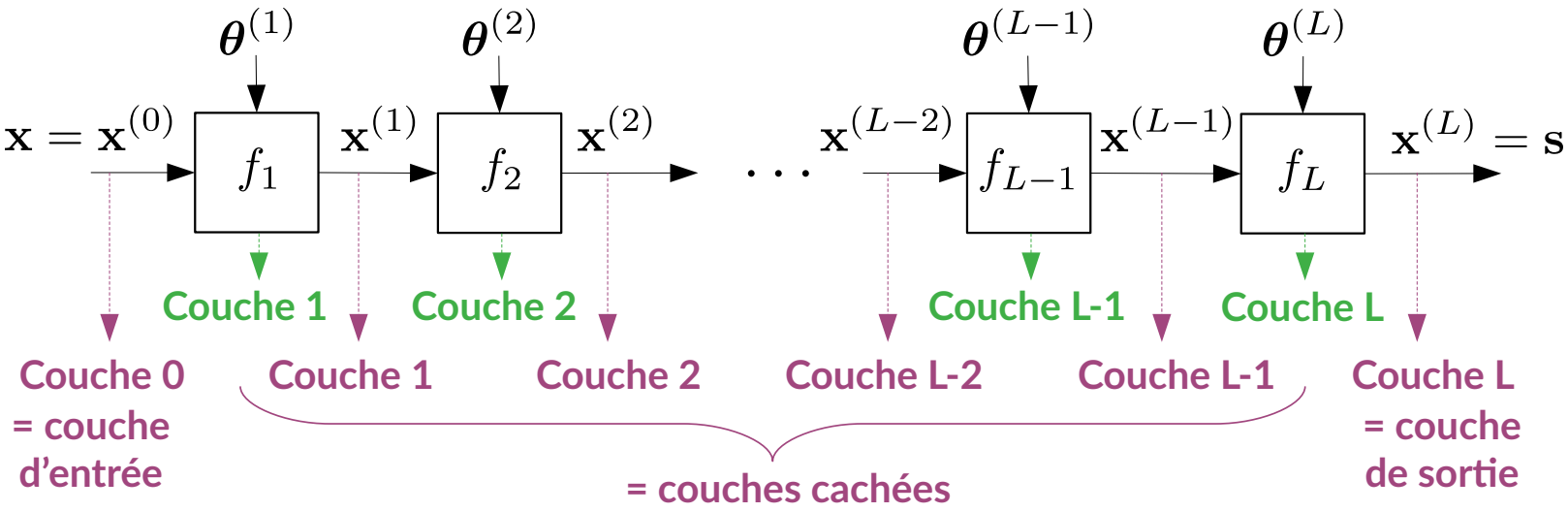
- Couche ("layer") :
  - — Sens 1 : Une fonction paramétrique du réseau
  - — Sens 2 : Un vecteur du réseau

Réseaux de neurones : Terminologie



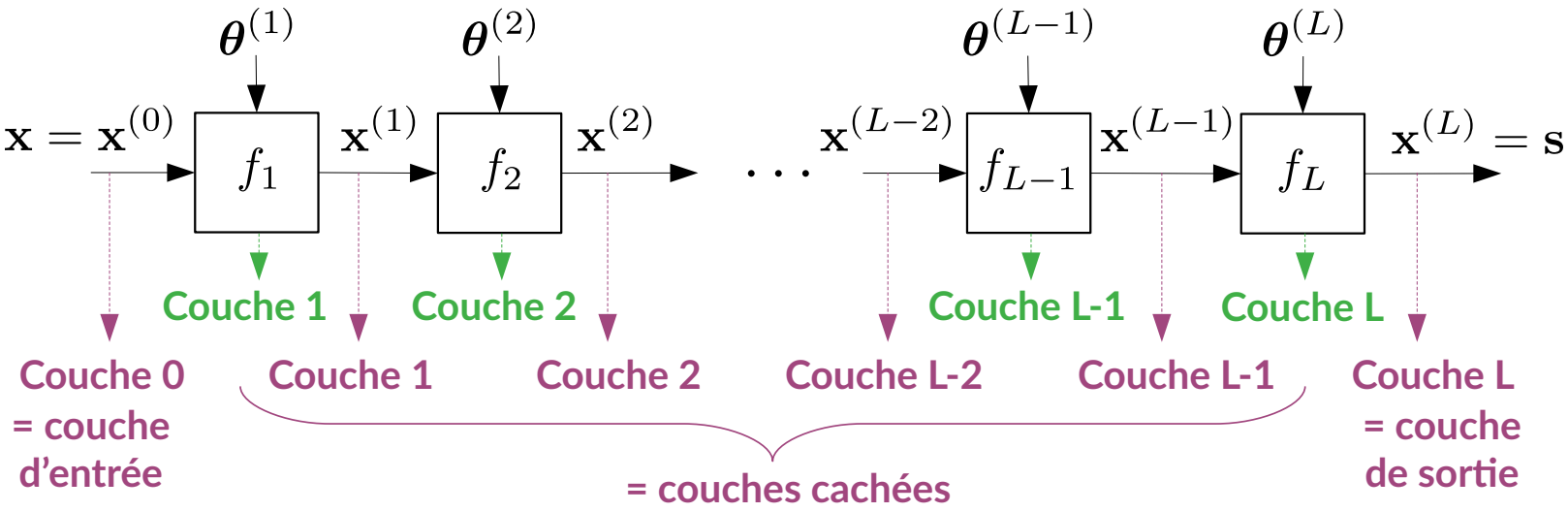
- Couche ("layer") :
  - — Sens 1 : Une fonction paramétrique du réseau
  - — Sens 2 : Un vecteur du réseau
- Profondeur du réseau de neurones = nombre de couches

Réseaux de neurones : Terminologie



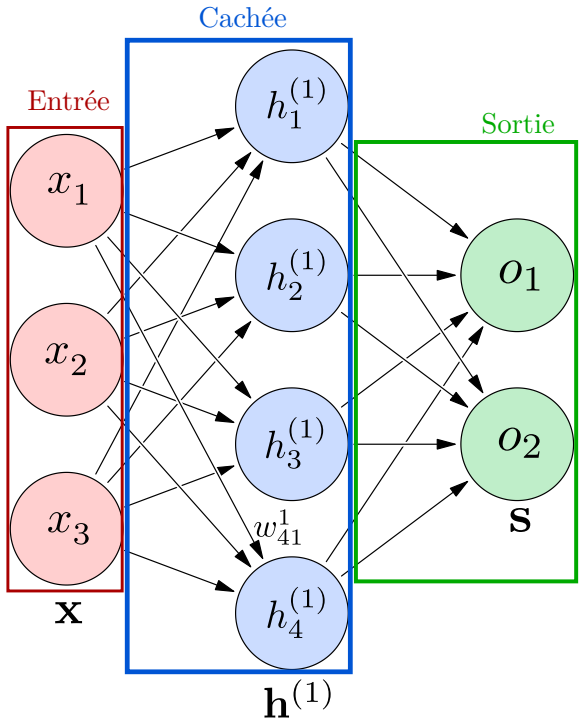
- Couche ("layer") :
  - — Sens 1 : Une fonction paramétrique du réseau
  - — Sens 2 : Un vecteur du réseau
- Profondeur du réseau de neurones = nombre de couches
- "Deep Neural Network" = réseau de neurones profond

Réseaux de neurones : Terminologie



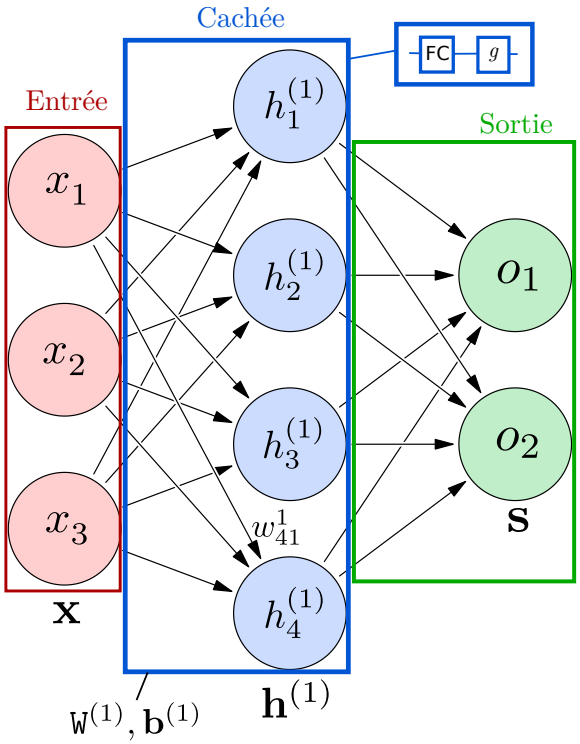
- Couche ("layer") :
  - — Sens 1 : Une fonction paramétrique du réseau
  - — Sens 2 : Un vecteur du réseau
- Profondeur du réseau de neurones = nombre de couches
- "Deep Neural Network" = réseau de neurones profond
- Architecture du réseau de neurones = choix du nombre de couches, du type de chaque couche et de ses hyperparamètres, etc.

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 $\mathbf{s} = \{s_1, s_2, \ldots, s_{N_s}\}$
- 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")

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

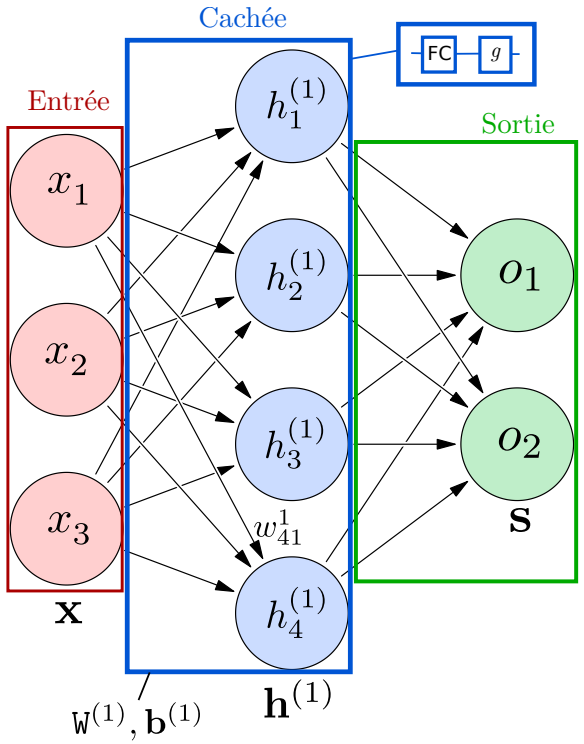
$$\mathbf{h}^{(1)} = \left(\mathtt{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right)$$

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

$\mathtt{W}^{(k)} = \{w_{ij}^k\}$ 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")

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

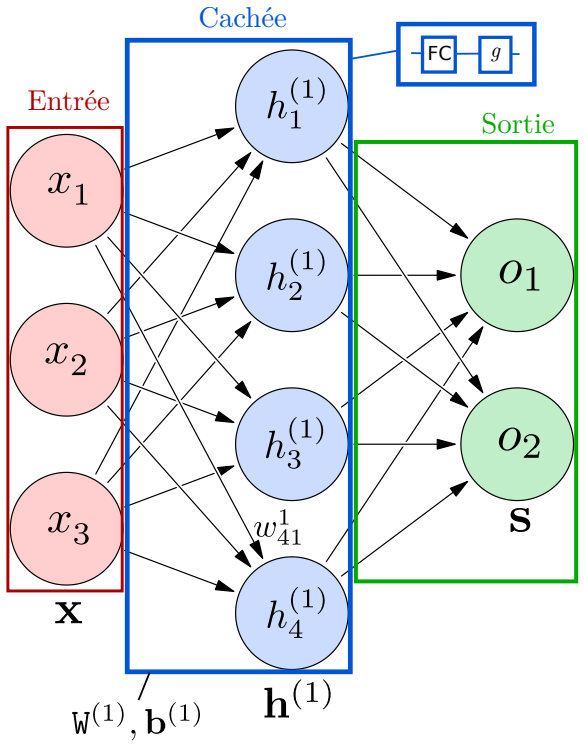
$$\mathbf{h}^{(1)} = \left(\mathtt{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right)$$

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

$\mathtt{W}^{(k)} = \{w_{ij}^k\}$ 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”)

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

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

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

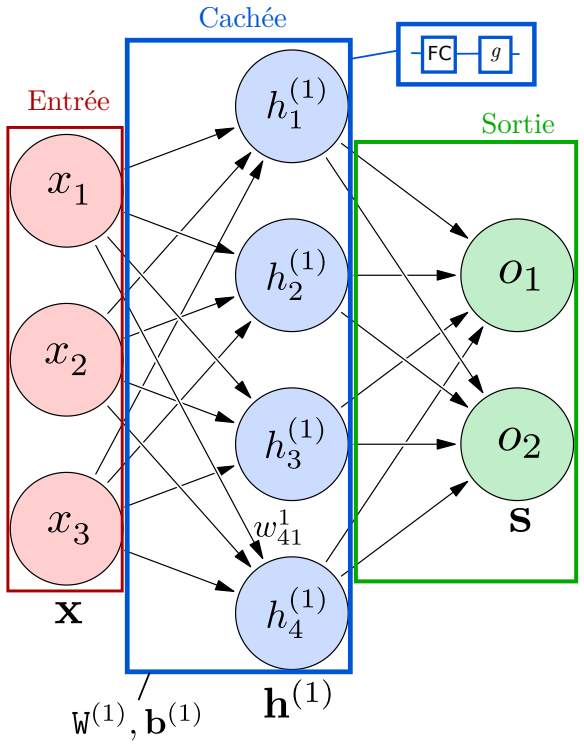
$$\begin{aligned}
h_1^{(1)} &= g_1\left(w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 + b_1^1\right)\\
h_2^{(1)} &= g_1\left(w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 + b_2^1\right)\\
h_3^{(1)} &= g_1\left(w_{31}^1 x_1 + w_{32}^1 x_2 + w_{33}^1 x_3 + b_3^1\right)\\
h_4^{(1)} &= g_1\left(w_{41}^1 x_1 +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

$\mathtt{W}^{(k)} = \{w_{ij}^k\}$ 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

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")

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

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

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

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

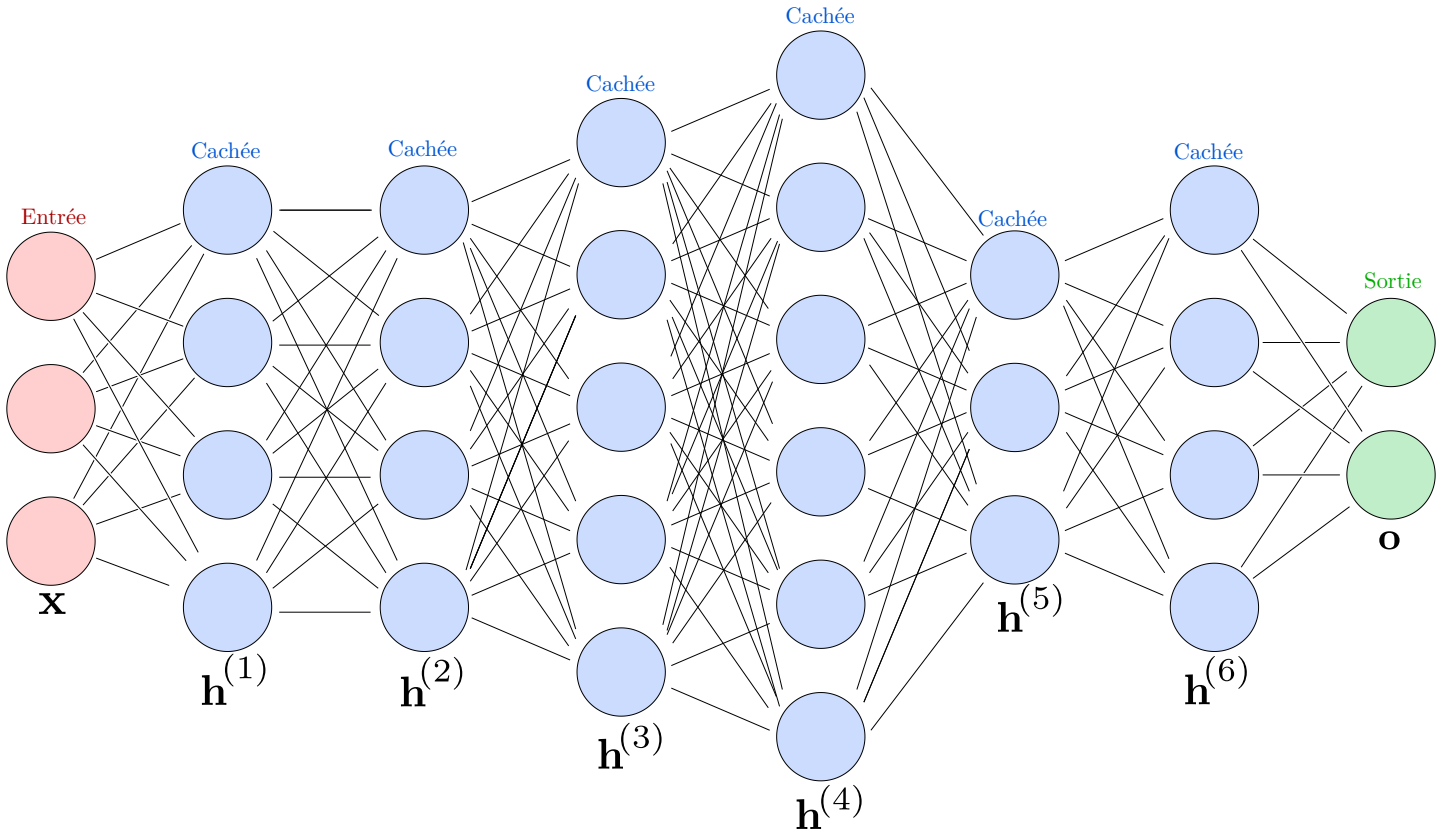
$\mathtt{W}^{(k)} = \{w_{ij}^k\}$ les **poids** entre le neurone précédent j et le suivant i à la couche k

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

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

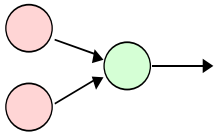


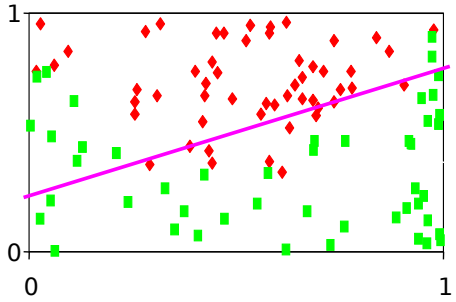
- Il peut avoir seulement une seule couche cachée (shallow network)
- Il peut avoir plusieurs couches cachées (deep network)
- Chaque couche (cachée ou de sortie) peut avoir une taille et une fonction d'activation différentes

Frontière de décision

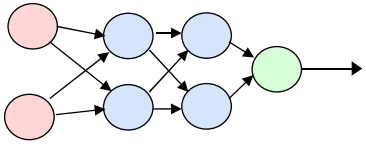
L'architecture du réseau définit la forme de la limite de décision :

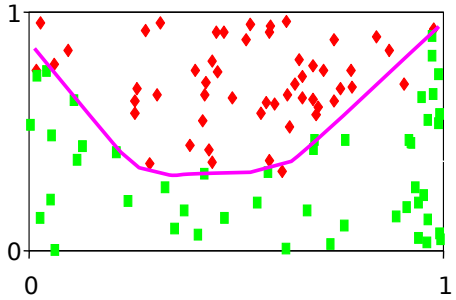
1 neurone



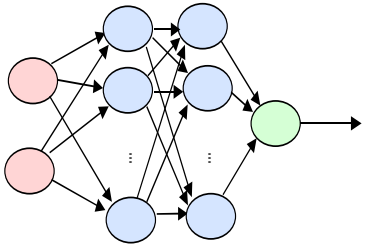


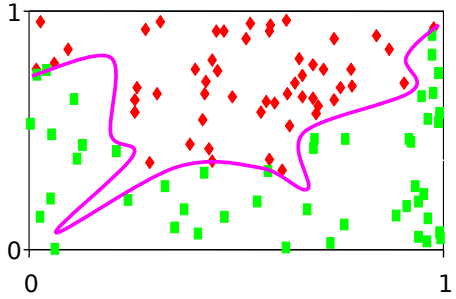
2+2+1 neurones





10+10+1 neurones





Complexité/capacité du réseau

$\Rightarrow$

Compromis entre généralisation et sur-apprentissage

Fonctions d'activation

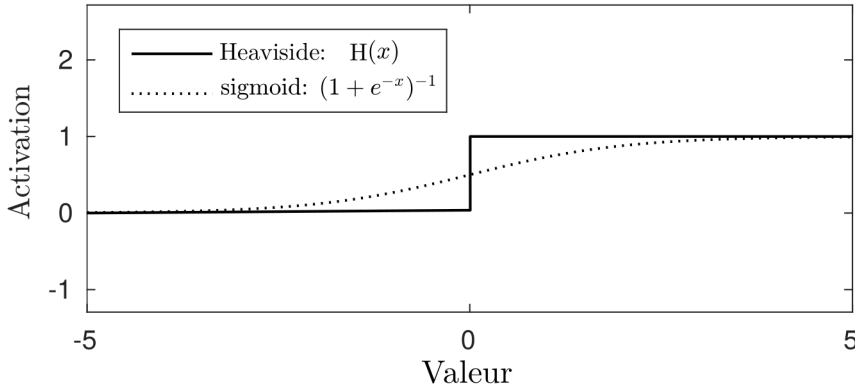
Fonction de seuil : par exemple la fonction de Heaviside (échelon)

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

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

Sigmoïde :

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

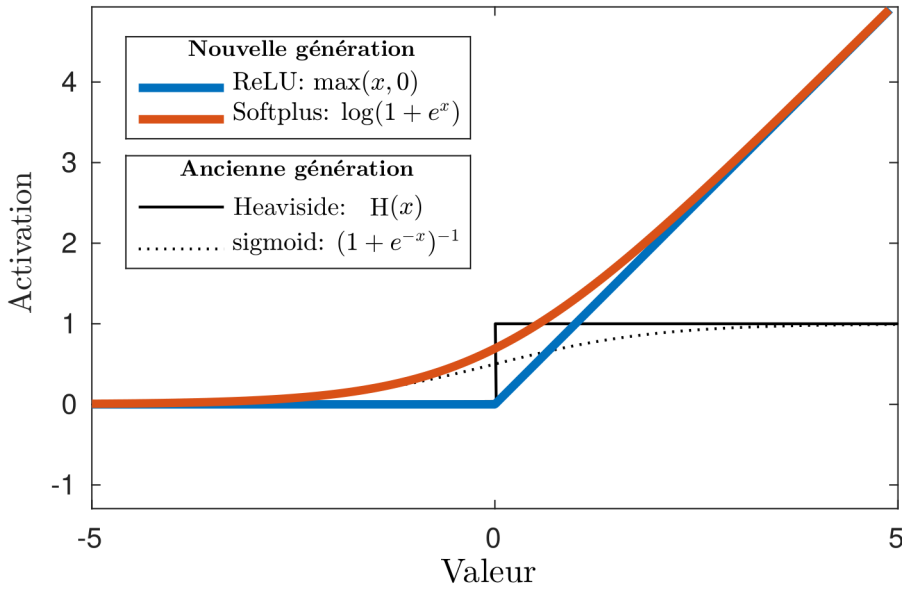


- Approximation différentiable de la fonction échelon
- Seuil/comportement linéaire pour les fortes/faibles valeurs

Fonctions d'activation

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

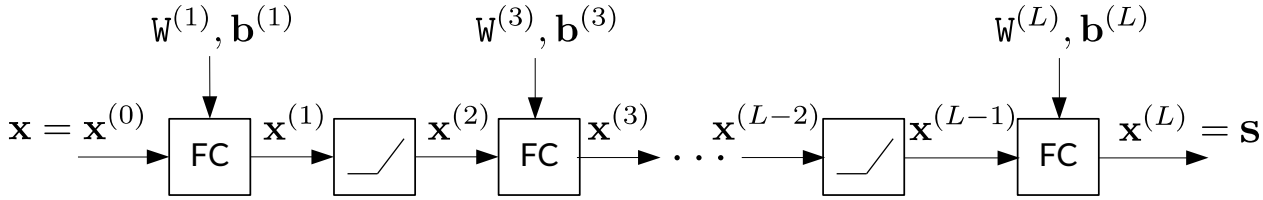
- Mathématique

$$f(\mathbf{x};\boldsymbol{\theta}) = \mathrm{MLP}\left(\mathbf{x};\boldsymbol{\theta} = \left\{\mathtt{W}^{(l)},\mathbf{b}^{(l)}\right\}_l\right) = \mathsf{FC}(\mathsf{ReLU}(...(\mathsf{ReLU}(\mathsf{FC}(\mathbf{x};\boldsymbol{\theta}_1))...);\boldsymbol{\theta}_L)$$

- Informatique (Python)

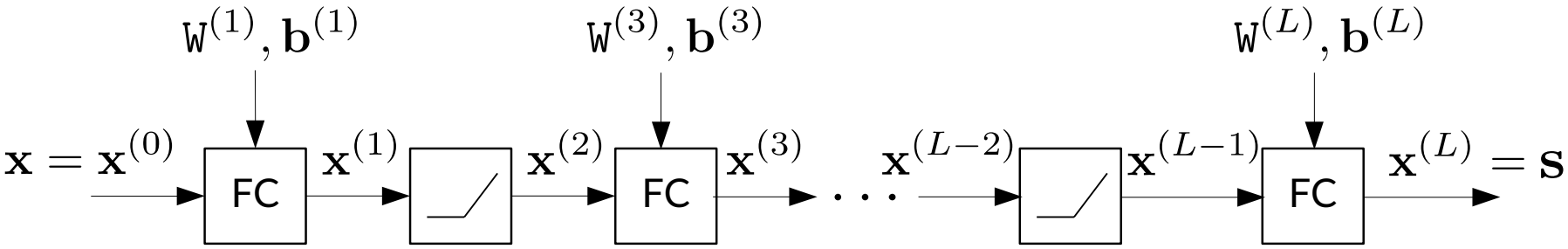
```
def MLP_forward(x, theta, L):
    x1 = FC(x, theta[0])
    x2 = ReLU(x1)
    x3 = FC(x2, theta[2])
    ...
    s  = FC(..., theta[L-1])
    return s
```

- Graphique (Graphe de calcul)



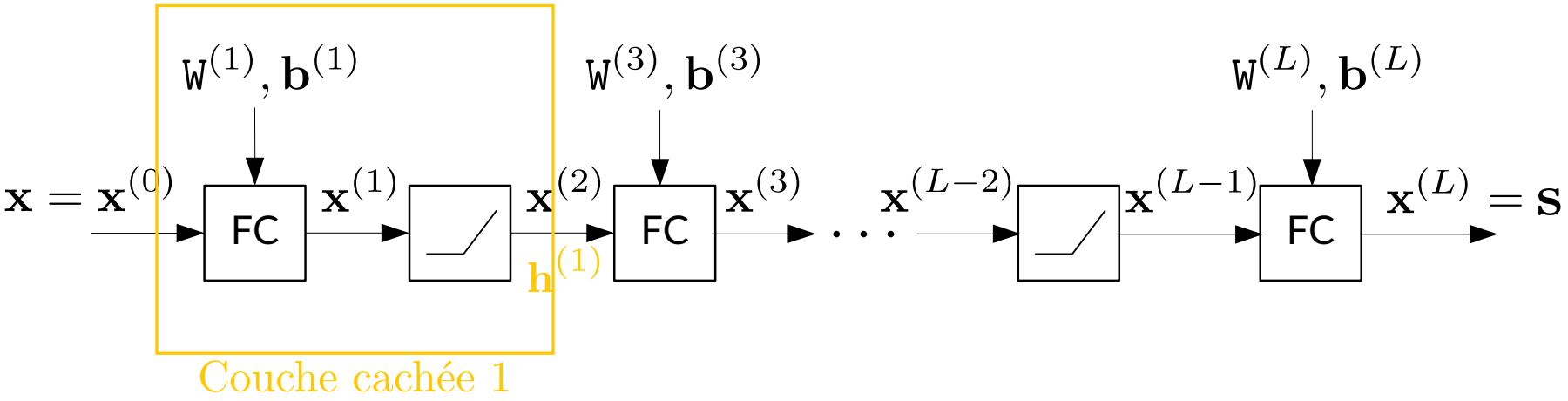
Composition de fonctions paramétriques : FC + ReLU

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



Composition de fonctions paramétriques : FC + ReLU

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



Attention !

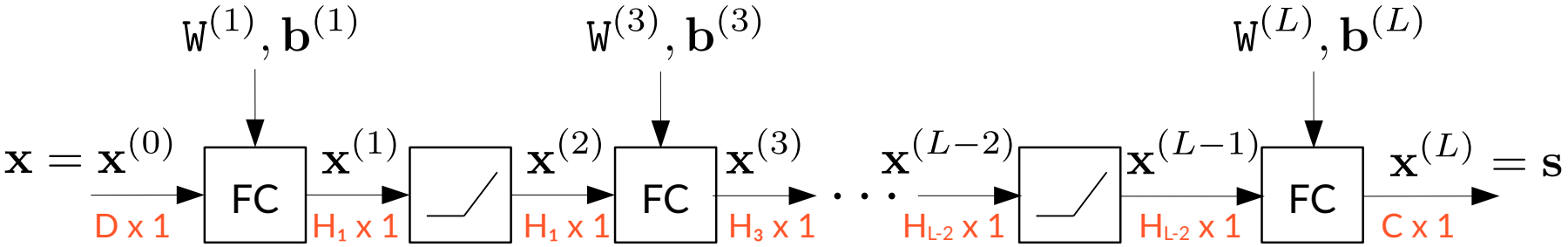
Dans cette représentation décomposant toutes les fonctions paramétriques :

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

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

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

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



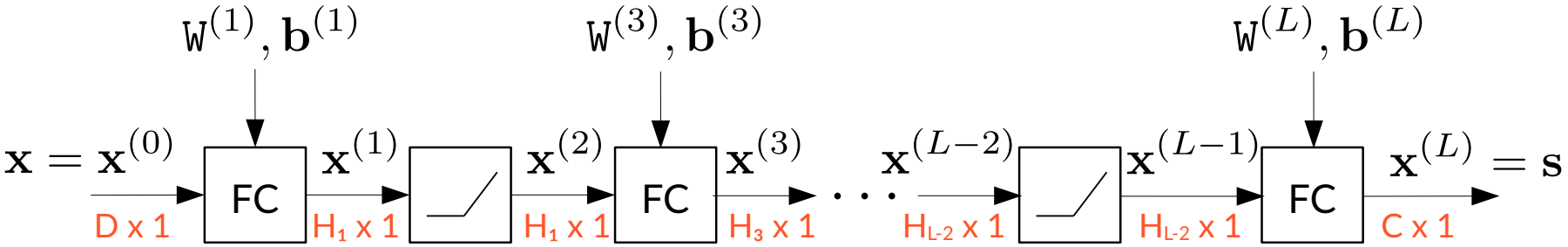
- Paramètres : $\left\{\mathtt{W}^{(2j-1)}, \mathbf{b}^{(2j-1)}\right\}_{j=1,\ldots,L/2}$
- Hyperparamètres : L (nombre de couches, contrôle la profondeur)
  $\{H_{2j-1}\}_{j=1,\ldots,(L-2)/2}$ (dimension, contrôle la largeur)

Perceptron multicouche (MLP)

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

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

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



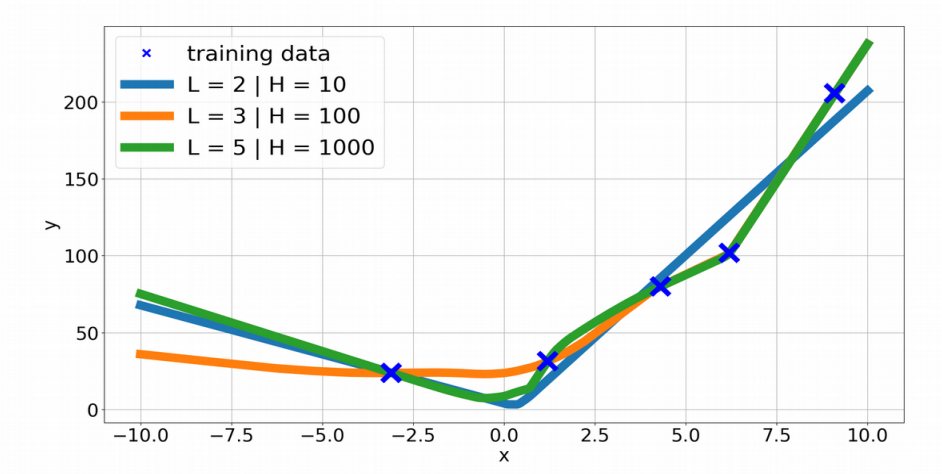
Fonction de coût :

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

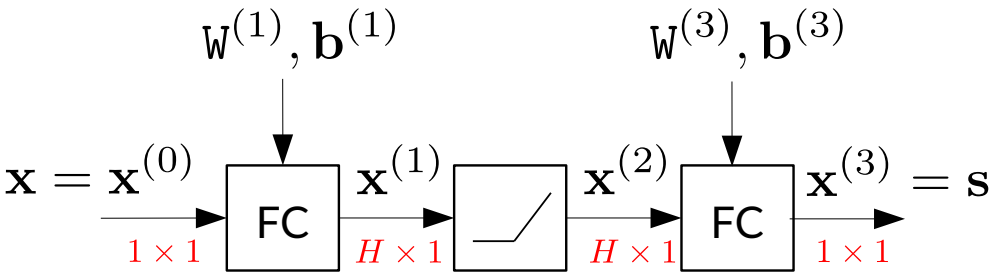
Optimisation :

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

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



MLP à une couche cachée en 1D



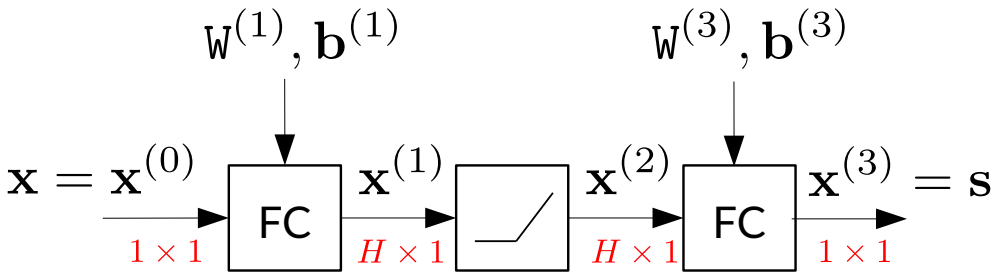
$$f(x) = \mathbf{w}_3^T \left(\mathrm{ReLU}\left(\mathbf{w}_1 x + \mathbf{b}_1\right)\right) + b_3$$

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

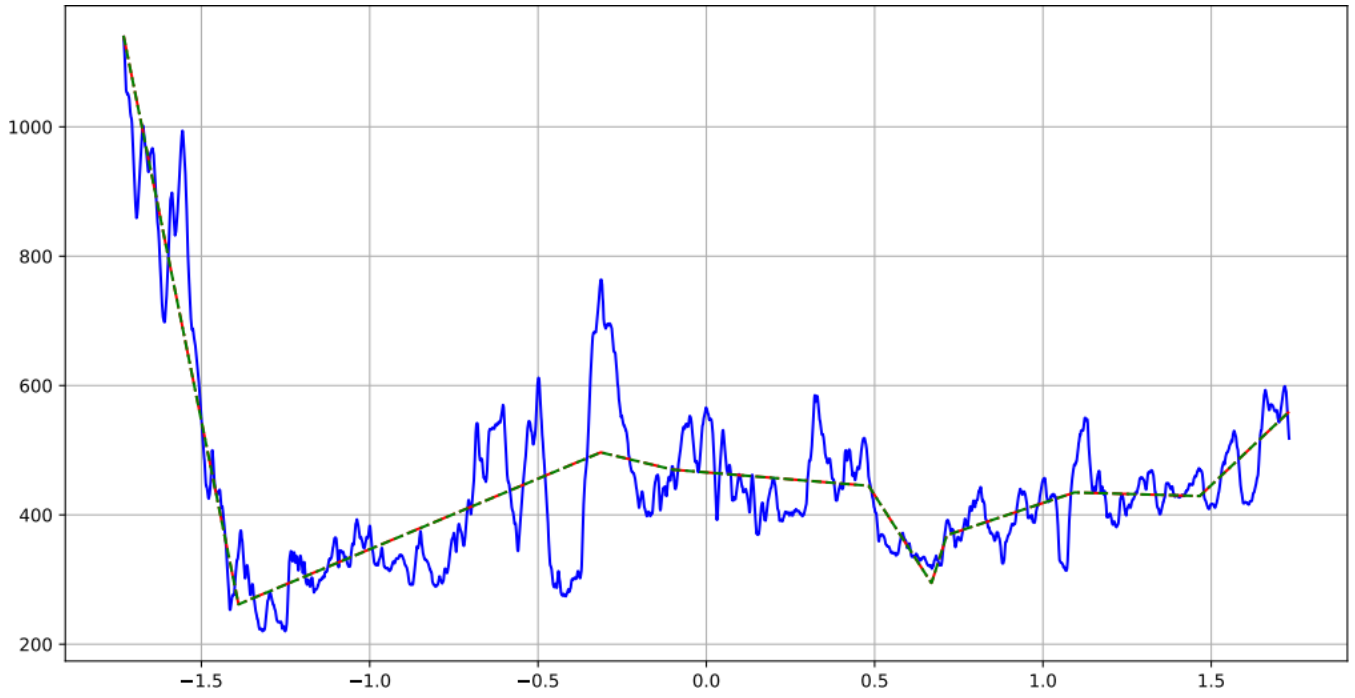
avec

$\mathtt{W}^{(1)} = \mathbf{w}_1$	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 : $3 \times \mathrm{H} + 1$

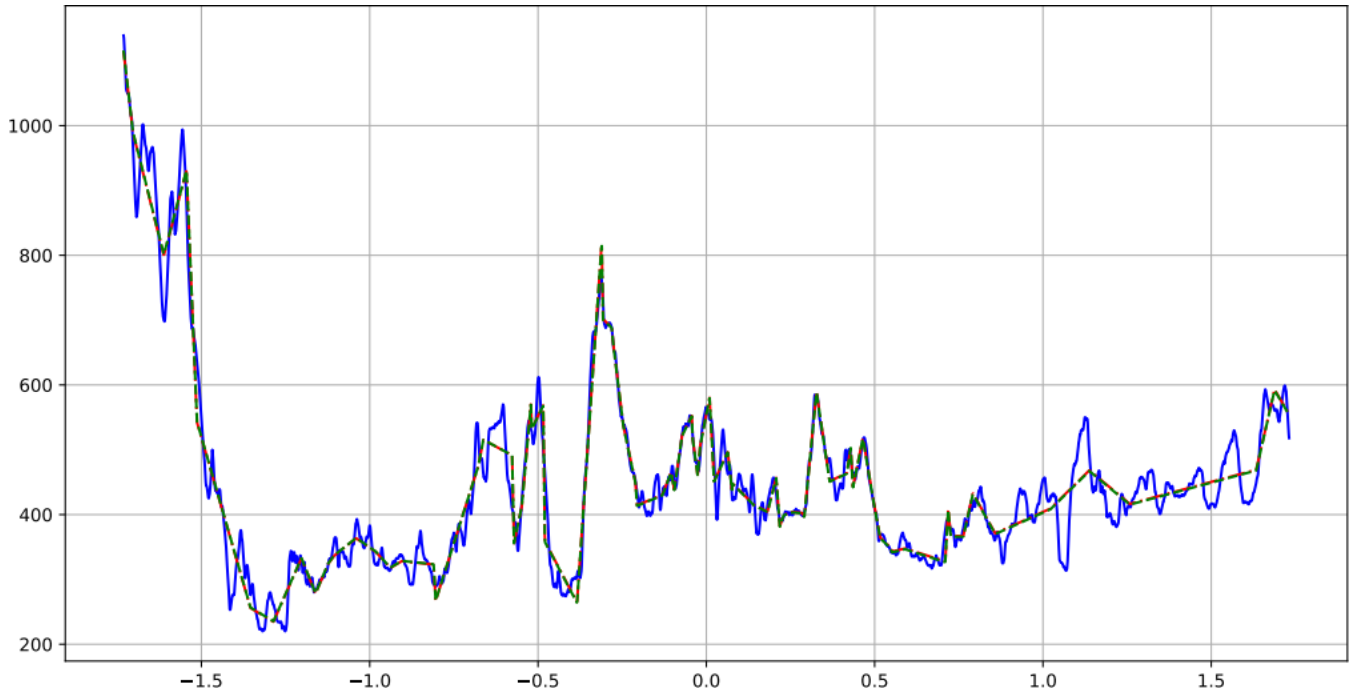
MLP à une couche cachée en 1D

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



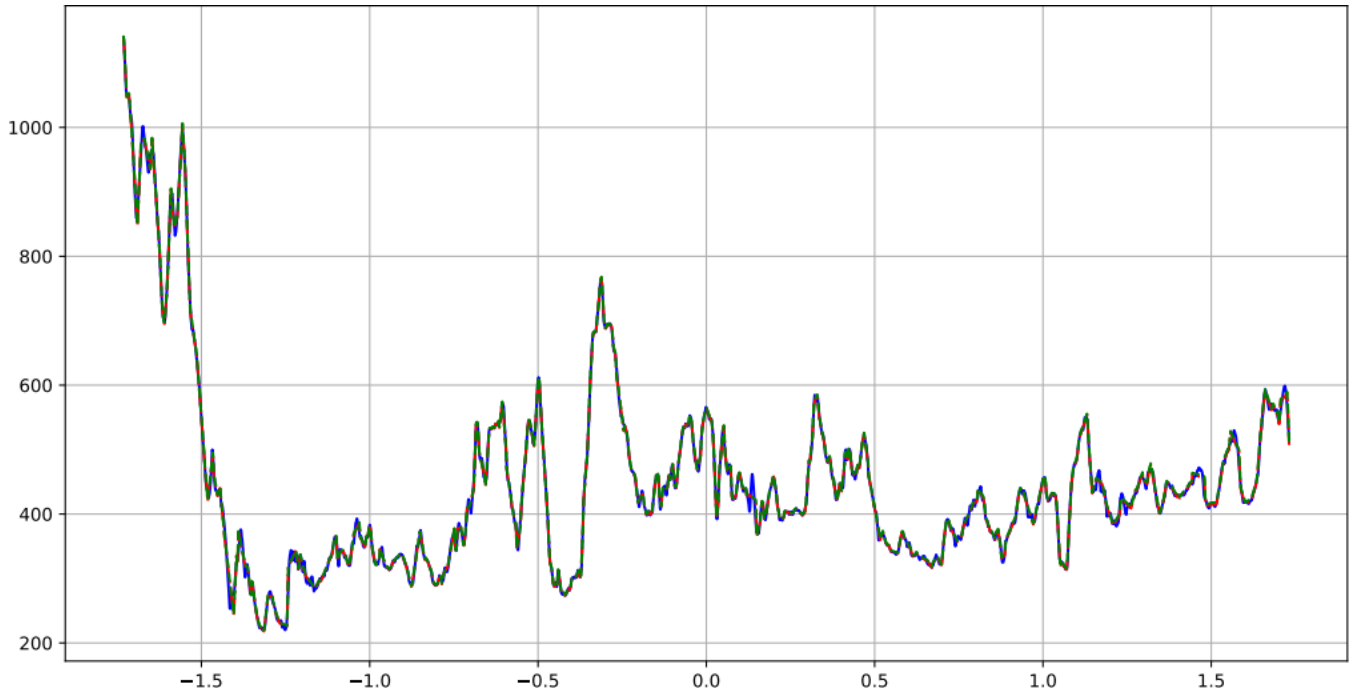
MLP à une couche cachée en 1D

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



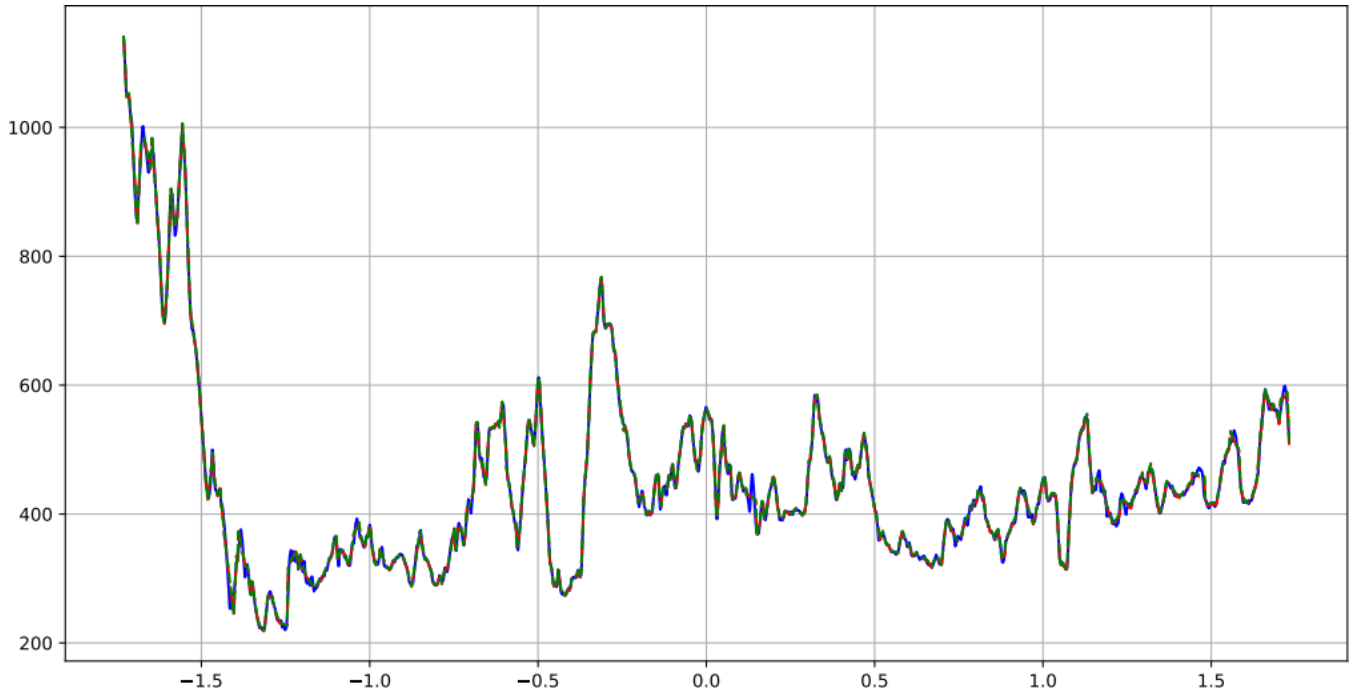
MLP à une couche cachée en 1D

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 : $\mathrm{H} = 1000$

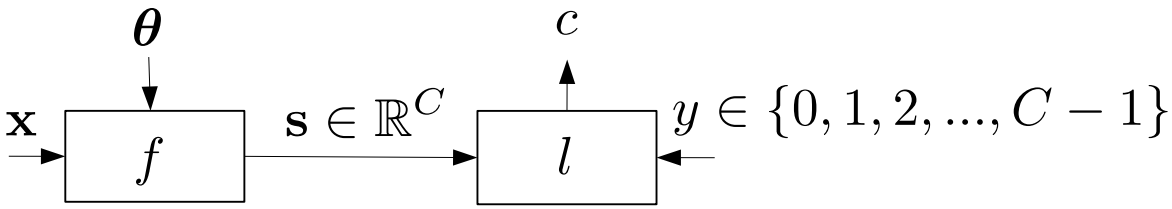


$\rightarrow$ Illustration du théorème d'approximation universelle

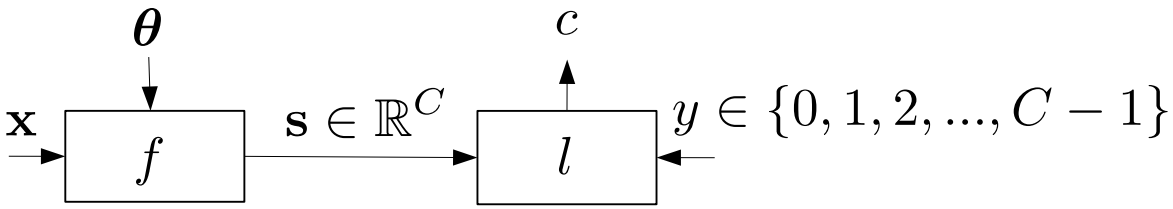
(Hornik et al, 1989; Cybenko, 1989)

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

Réseau de classification ?

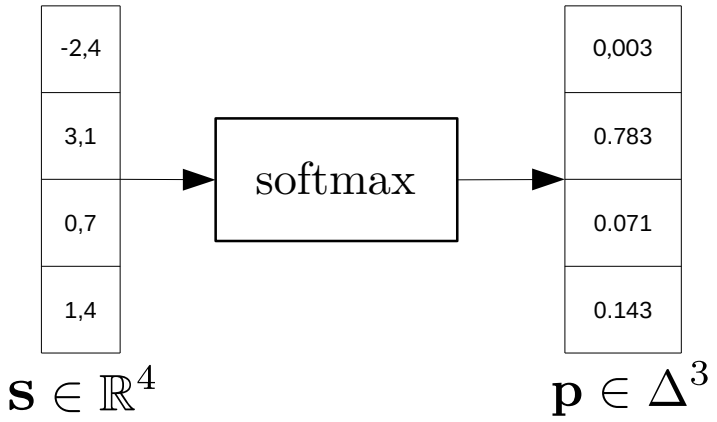


Réseau de classification ?



- **"Cross-entropy"** : $\mathrm{CE}(y, \mathbf{s}) = -\ln(\mathbf{p}_y)$

où $\mathbf{p}_y = \dfrac{\exp(\mathbf{s}[y])}{\sum_{c=0}^{C-1} \exp(\mathbf{s}[c])}$ ("softmax", approx. dérivable de la fonction argmax)



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

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

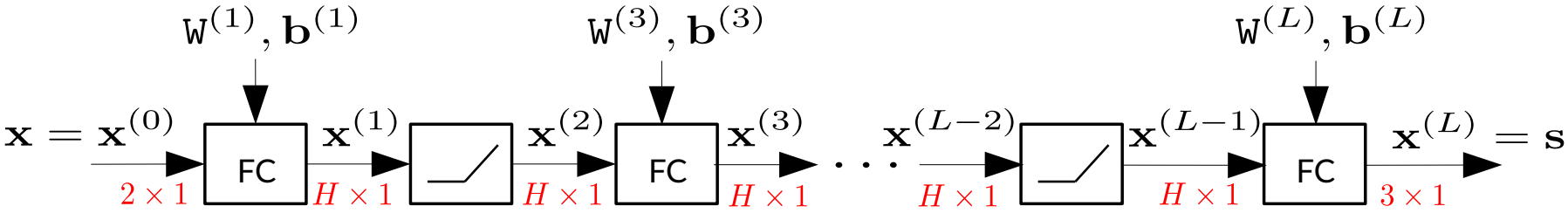
Ex. : Si $y = 2$, $L = -\ln(0.071) = 2.647$ (prédiction très mauvaise)

Perceptron multicouche (MLP)

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

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

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

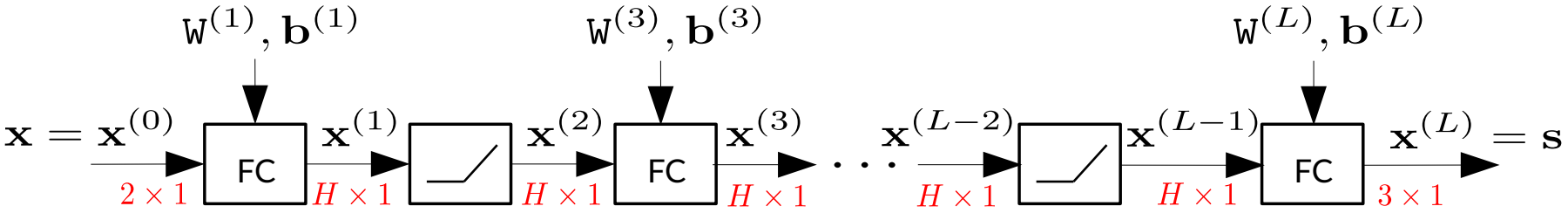


Perceptron multicouche (MLP)

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

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

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

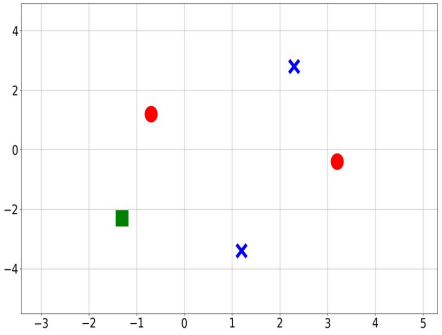


Fonction de coût :

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

Optimisation :

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

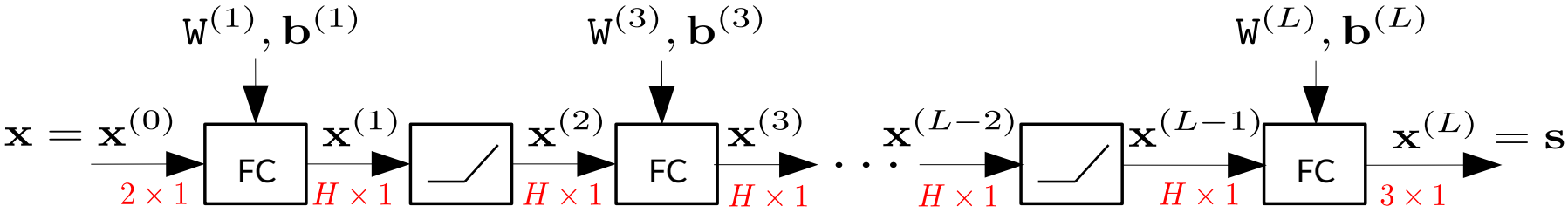


Inférence : $\mathbf{s}_{\text{test}} = f(x_{\text{test}}; \boldsymbol{\theta}^*)$ classe prédite : $\text{argmax}\ \mathbf{s}_{\text{test}}$

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

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

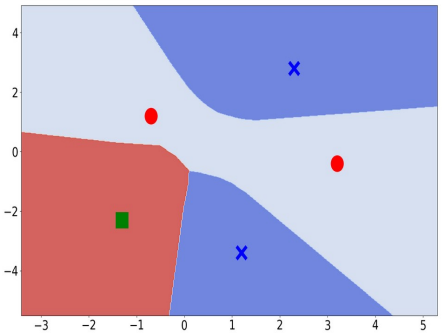
Fonction : $f(x; \boldsymbol{\theta}) = \text{MLP}\left(x; \boldsymbol{\theta} = \left\{\mathtt{W}^{(l)}, \mathbf{b}^{(l)}\right\}_l\right)$ hyperparamètres : H, L



Fonction de coût :

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

$L = 3, H = 30$

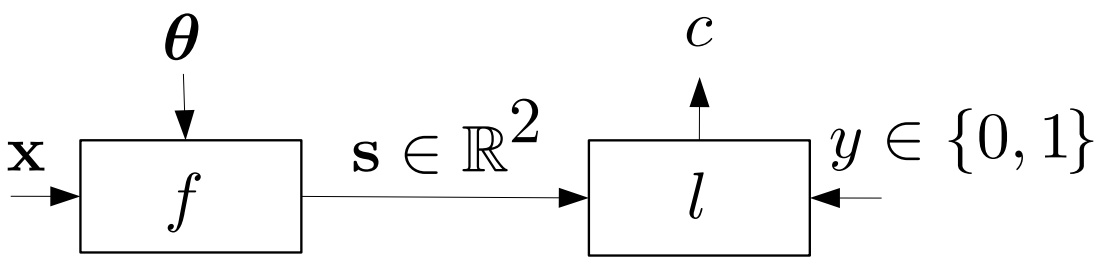


Optimisation :

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

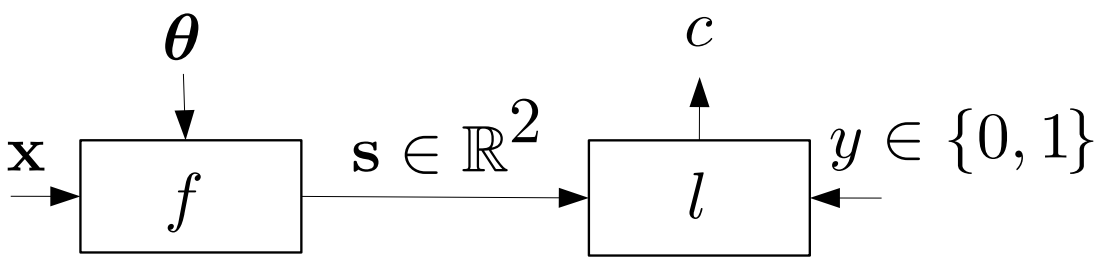
Inférence : $\mathbf{s}_{\text{test}} = f(x_{\text{test}}; \boldsymbol{\theta}^*)$ classe prédite : $\text{argmax}\ \mathbf{s}_{\text{test}}$

"Cross-entropy" à 2 classes vs "Binary cross-entropy"

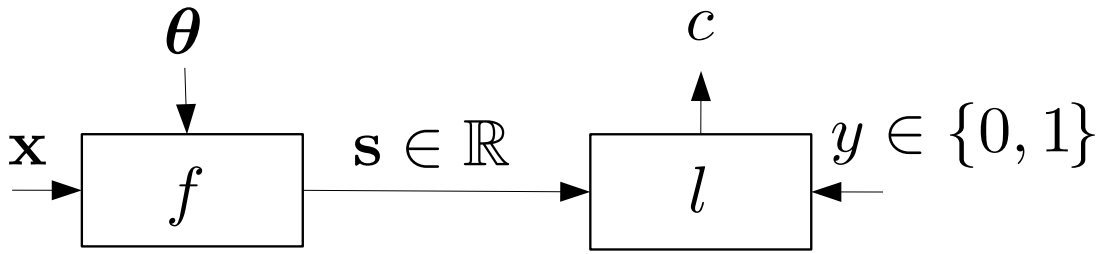


- **"Cross-entropy"** : $\mathrm{CE}(y, \mathbf{s}) = -\ln(\mathbf{p}_y)$ où $\mathbf{p}_y = \frac{\exp(\mathbf{s}[y])}{\sum_{c=0}^{C-1} \exp(\mathbf{s}[c])}$

"Cross-entropy" à 2 classes vs "Binary cross-entropy"

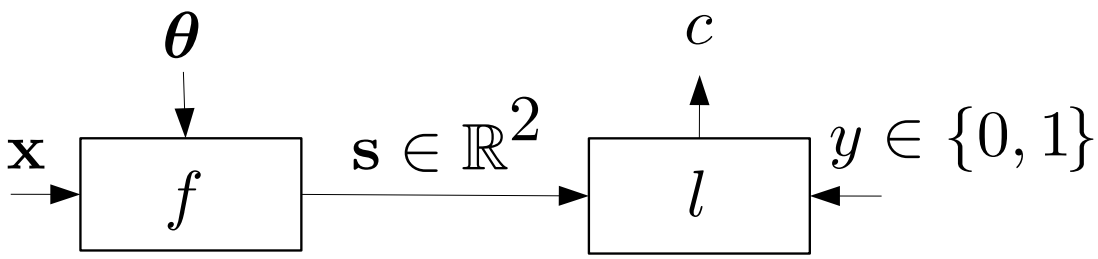


- **"Cross-entropy"** : $\mathrm{CE}(y, \mathbf{s}) = -\ln(\mathbf{p}_y)$ où $\mathbf{p}_y = \frac{\exp(\mathbf{s}[y])}{\sum_{c=0}^{C-1} \exp(\mathbf{s}[c])}$

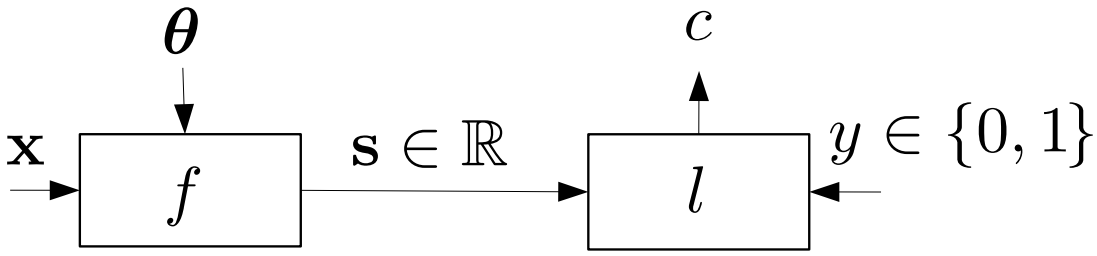


- **"Binary CE"** : $-y\ln(\mathbf{p}) - (1-y)\ln(1-\mathbf{p})$ où $\mathbf{p} = \frac{1}{1+\exp(-\mathbf{s})}$ (sigmoïde)

"Cross-entropy" à 2 classes vs "Binary cross-entropy"



- **"Cross-entropy"** : $\mathrm{CE}(y, \mathbf{s}) = -\ln(\mathbf{p}_y)$ où $\mathbf{p}_y = \frac{\exp(\mathbf{s}[y])}{\sum_{c=0}^{C-1} \exp(\mathbf{s}[c])}$



- **"Binary CE"** : $-y\ln(\mathbf{p}) - (1-y)\ln(1-\mathbf{p})$ où $\mathbf{p} = \frac{1}{1+\exp(-\mathbf{s})}$ (sigmoïde)

Équivalent ! Juste une question d'implémentation