

# Pratique de l'apprentissage statistique

## 9. Réseaux de neurones artificiels - Perceptron multicouche

V. Lefieux



École des Ponts

ParisTech

# Plan

Introduction

Neurone formel

Fonctions d'activation

Perceptron multicouche

Rétro-propagation

Pratique du perceptron multicouche

# Plan

Introduction

Neurone formel

Fonctions d'activation

Perceptron multicouche

Rétro-propagation

Pratique du perceptron multicouche

## Point de vue biologique I

Un **réseau de neurones** (artificiels) est un algorithme inspiré du fonctionnement des neurones du **cerveau humain**.

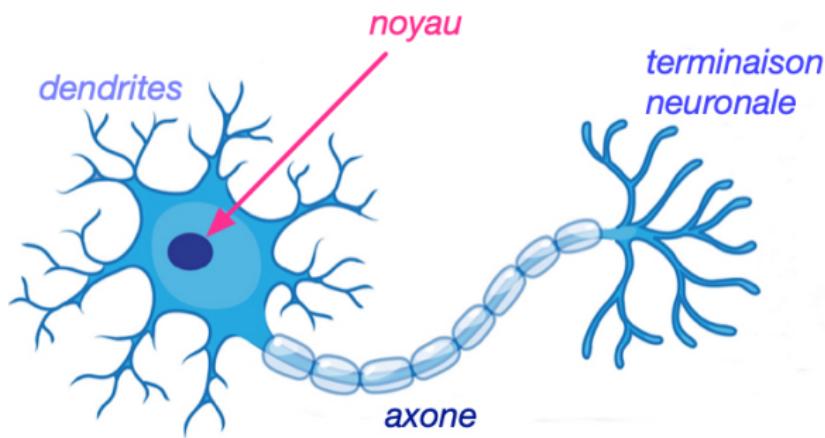


Figure – ©Getty / Science Photo Library - KTSDESIGN

## Point de vue biologique II

De manière extrêmement simplifiée, un neurone est une cellule cérébrale permettant de collecter, traiter et transmettre des signaux électriques. Il est notamment composé :

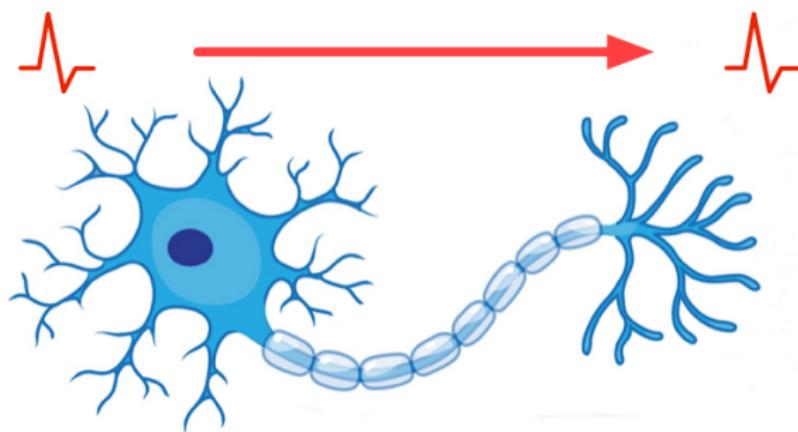
- ▶ de **dendrites** : les « entrées » du neurone,
- ▶ d'un **axone** : la « sortie » du neurone.



## Point de vue biologique II

De manière extrêmement simplifiée, un neurone est une cellule cérébrale permettant de collecter, traiter et transmettre des signaux électriques. Il est notamment composé :

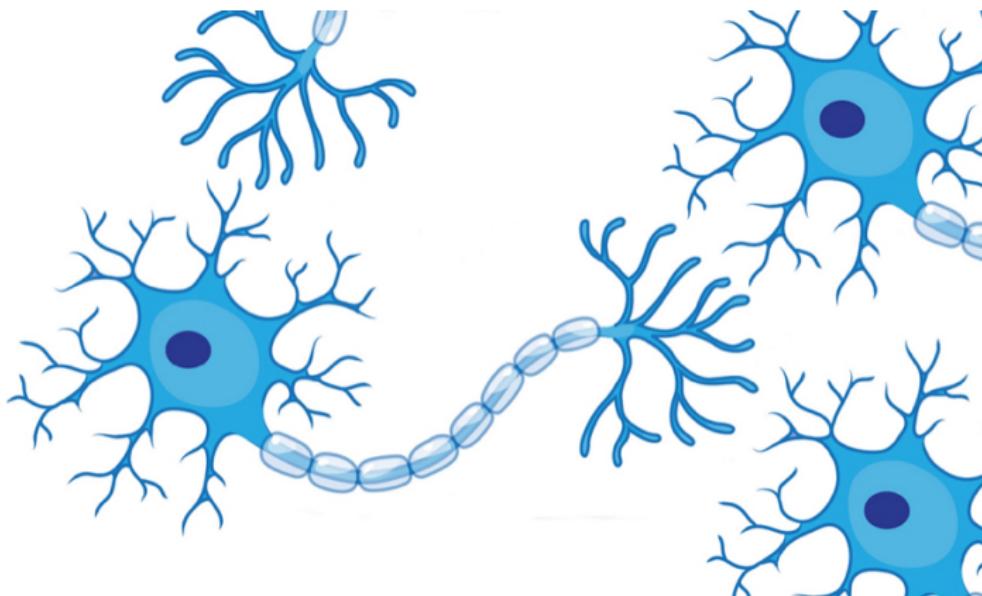
- ▶ de **dendrites** : les « entrées » du neurone,
- ▶ d'un **axone** : la « sortie » du neurone.



## Point de vue biologique II

De manière extrêmement simplifiée, un neurone est une cellule cérébrale permettant de collecter, traiter et transmettre des signaux électriques. Il est notamment composé :

- ▶ de **dendrites** : les « entrées » du neurone,
- ▶ d'un **axone** : la « sortie » du neurone.



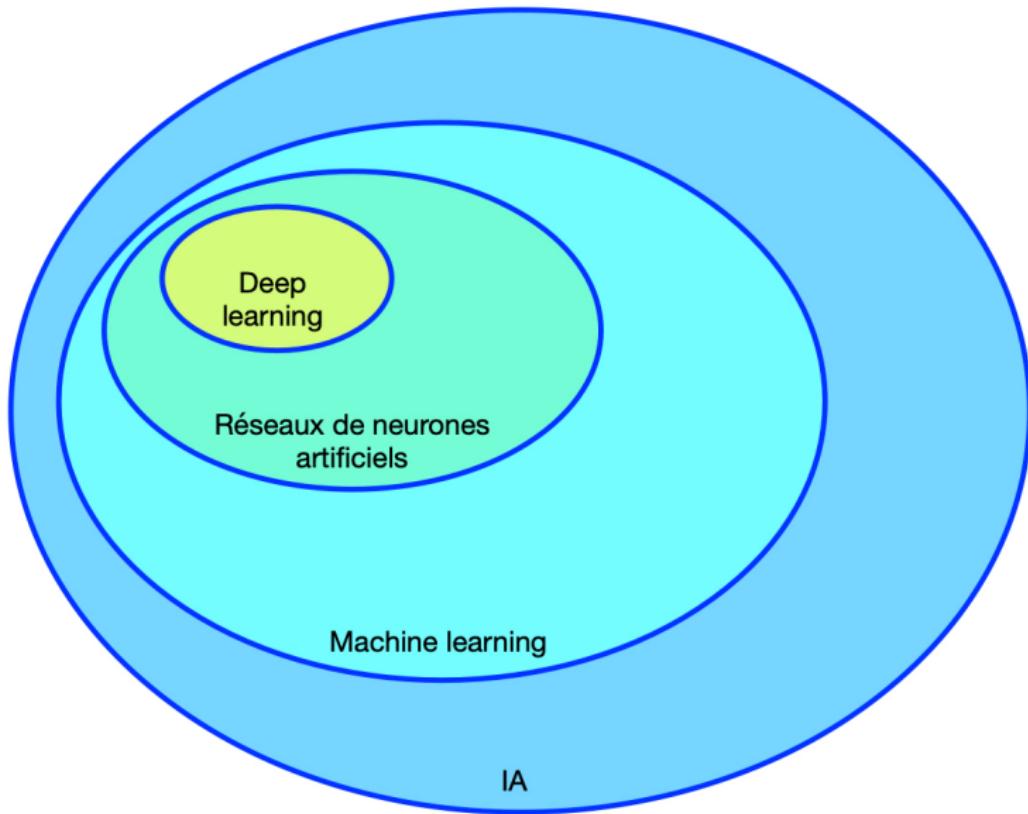
## Les réseaux de neurones (artificiels) en quelques dates

- ▶ 1943 : **neurone formel** - McCulloch (neurophysiologiste) et Pitts (logicien).
- ▶ 1958 : **perceptron** (simple) - Rosenblatt.
- ▶ 1974-1986 : **perceptron multicouche** et **rétro-propagation** - Rumelhart, Hinton et Williams, suite à des travaux de Werbos.
- ▶ Fin des années 1990 : **deep learning** - Bengio, Hinton et LeCun.

... et des « **hivers** » surmontés grâce à :

- ▶ de gros volumes de **données** (*big data*) labellisées,
- ▶ des **moyens de calculs** puissants (ex : GPU),
- ▶ des **algorithmes** plus performants.

# Réseaux de neurones et deep learning



## Données considérées

- ▶ On dispose d'un échantillon de  $(X, Y)$  :

$$\mathcal{D}_n = (X_i, Y_i)_{i \in \{1, \dots, n\}}.$$

On note :

$$d_n = (x_i, y_i)_{i \in \{1, \dots, n\}}.$$

- ▶ On considère dans la suite que :

- ▶  $X \in \mathbb{R}^p$  :

*Toutes les covariables sont considérés quantitatives.*

*Mais il est également possible de considérer des covariables qualitatives.*

- ▶  $Y \in \{0, 1\}$  dans le cas de la **classification supervisée binaire** et plus généralement  $Y \in \{1, \dots, K\}$  dans le cas de la **classification supervisée à  $K$  classes**.
  - ▶  $Y \in \mathbb{R}$  dans le cas de la **régression simple**.

# Plan

Introduction

Neurone formel

Fonctions d'activation

Perceptron multicouche

Rétro-propagation

Pratique du perceptron multicouche

## Définition

- ▶ Un neurone formel est composé :
  - ▶ d'entrées  $X^1, \dots, X^p$  dans  $\mathbb{R}$ , auxquelles sont associés des poids (synaptiques)  $\omega_1, \dots, \omega_p$  et un biais  $\omega_0$ ,
  - ▶ d'une sortie (réponse)  $Y$  dans  $\mathbb{R}$ .
- ▶ Plusieurs opérations sont successivement effectuées par le neurone :
  1. Somme pondérée des entrées avec les poids  $(\omega_j)_{j \in \{1, \dots, p\}}$ .
  2. Ajout du biais  $\omega_0$ .
  3. Transformation par une fonction d'activation  $\varphi$ .
- ▶ La réponse obtenue au final est :

$$Y = \varphi \left( \omega_0 + \sum_{j=1}^p \omega_j X^j \right).$$

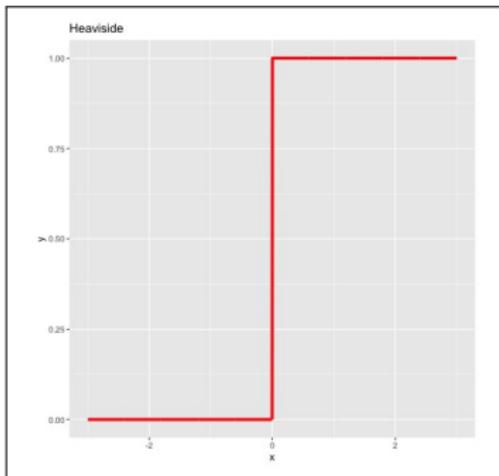
On note  $a$  le résultat des 2 premières étapes (sommation avec biais) :

$$a = \omega_0 + \sum_{j=1}^p \omega_j X^j.$$

## Fonction d'activation d'Heaviside

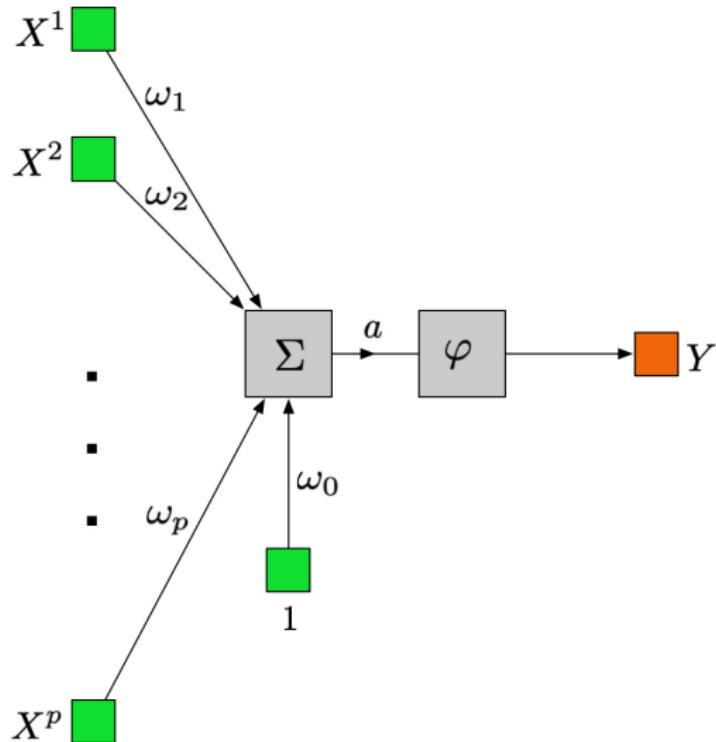
- ▶ Dans la première version du neurone formel, la fonction d'activation retenue était celle de **Heaviside** (ou échelon unité ou marche - *step*) :

$$\varphi(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{sinon} \end{cases}.$$



- ▶ Si  $\sum_{j=1}^p \omega_j X^j$  dépasse le seuil  $-\omega_0$ , la sortie du neurone vaut 1, sinon 0 (on parle alors de neurone éteint).

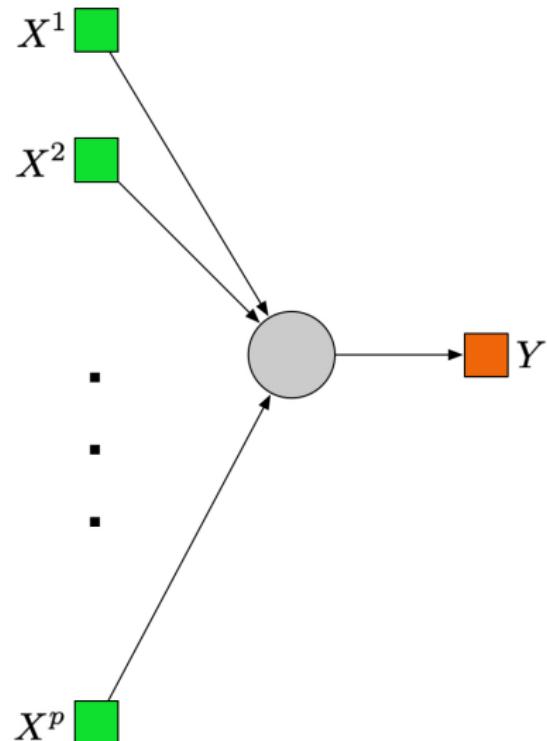
# Représentation I



**$p$  entrées**

**1 sortie**

## Représentation II



**$p$  entrées**

**1 sortie**

# Perceptron (simple) I

Le **perceptron** (simple) est un ensemble de neurones formels, reliés aux mêmes entrées auxquels, on adjoint une règle d'apprentissage pour les poids et les biais,

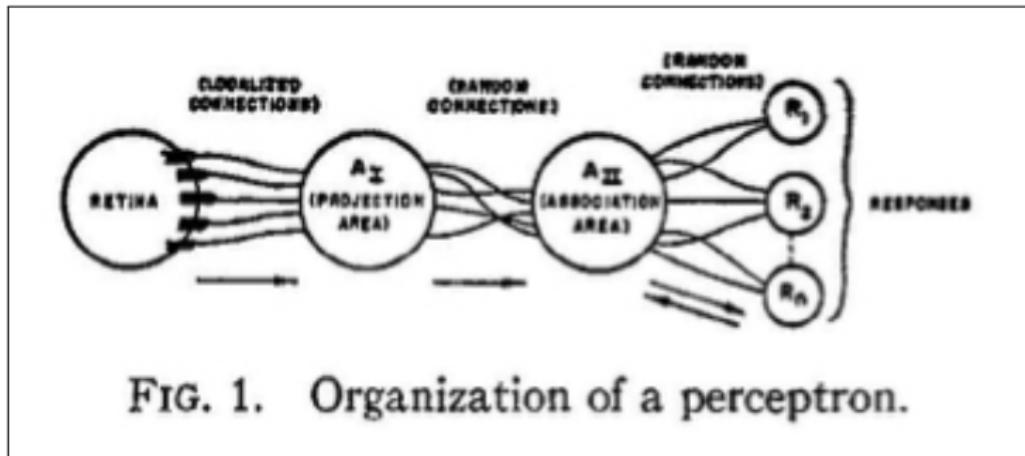
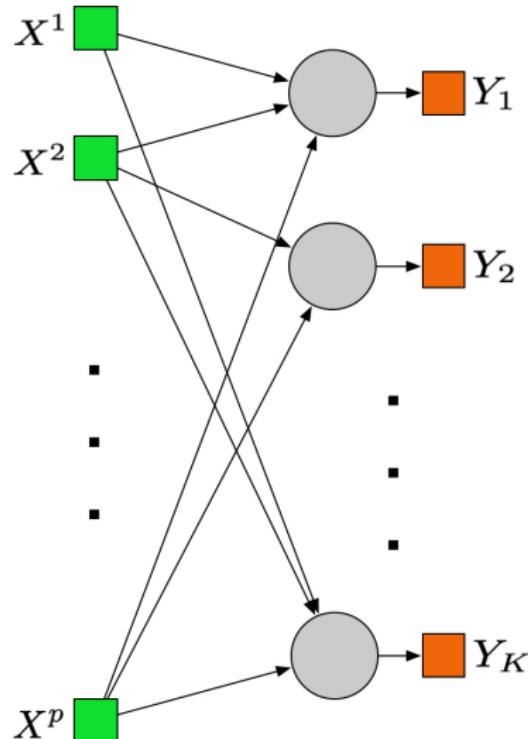


Figure – (Rosenblatt, 1958)

## Perceptron (simple) II



**$p$  entrées**

**$K$  sorties**

# Plan

Introduction

Neurone formel

Fonctions d'activation

Perceptron multicouche

Rétro-propagation

Pratique du perceptron multicouche

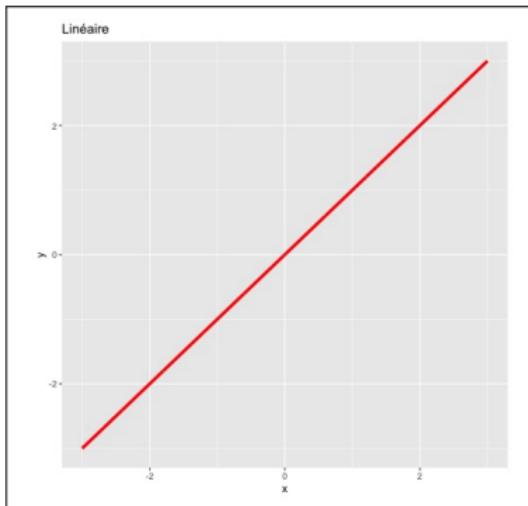
## Choix d'une fonction d'activation

- ▶ Le choix de la **fonction d'activation** est important, il est conditionné par des propriétés de régularité et d'interprétabilité.
- ▶ La facilité d'obtention de la **dérivée** est intéressante d'un point de vue calculatoire.
- ▶ Parmi les plus utilisées, on trouve la **sigmoïde**, la **tangente hyperbolique** et **ReLU**.
- ▶ Les **fonctions d'activation polynomiales** (incluant la **fonction d'activation linéaire**) limitent la portée du neurone, on les considère très peu en pratique.

## Fonction d'activation linéaire

- ▶ La fonction **linéaire** (ou identité) est :

$$\varphi(x) = x .$$



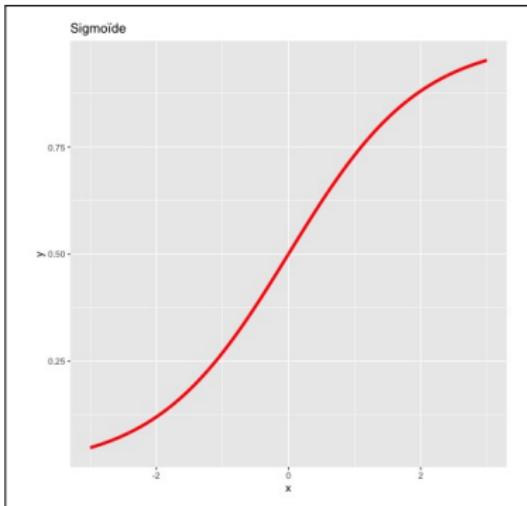
- ▶ A valeurs dans  $\mathbb{R}$ , elle est infiniment continûment dérivable, de dérivée :

$$\varphi'(x) = 1 .$$

## Fonction d'activation sigmoïde

- ▶ La fonction **sigmoïde** (ou logistique ou marche douce - *soft step*) est :

$$\varphi(x) = \frac{1}{1 + e^{-x}}.$$



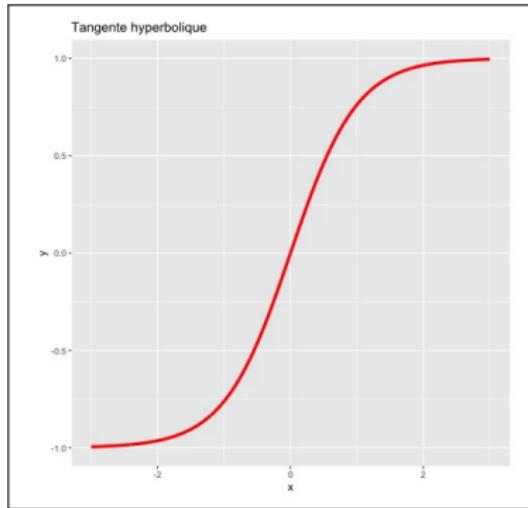
- ▶ A valeurs dans  $[0, 1]$ , elle est infiniment continûment dérivable, de dérivée :

$$\varphi'(x) = \varphi(x)[1 - \varphi(x)].$$

## Fonction d'activation tangente hyperbolique

- ▶ La fonction **tangente hyperbolique** est :

$$\varphi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} .$$



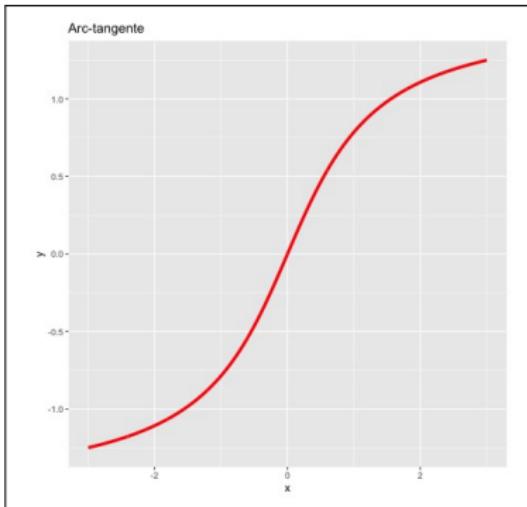
- ▶ A valeurs dans  $[-1, 1]$ , elle est infiniment continûment dérivable, de dérivée :

$$\varphi'(x) = 1 - \varphi^2(x) .$$

## Fonction d'activation arc-tangente

- ▶ La fonction **arc-tangente** est :

$$\varphi(x) = \tan^{-1}(x) .$$



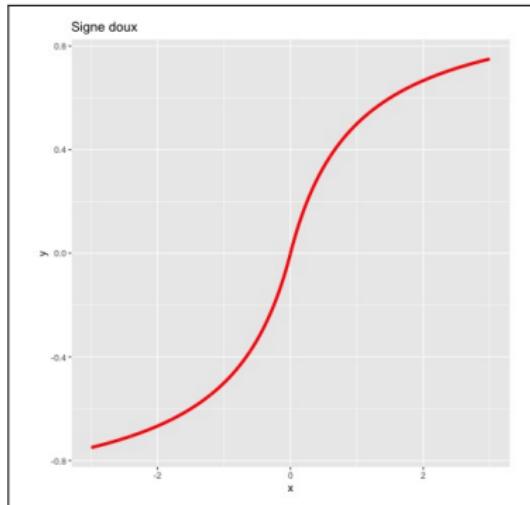
- ▶ A valeurs dans  $[-\frac{\pi}{2}, \frac{\pi}{2}]$ , elle est infiniment continûment dérivable, de dérivée :

$$\varphi'(x) = \frac{1}{1+x^2} .$$

## Fonction d'activation signe doux

- ▶ La fonction **signe doux** (*soft sign*) est :

$$\varphi(x) = \frac{x}{1 + |x|}.$$



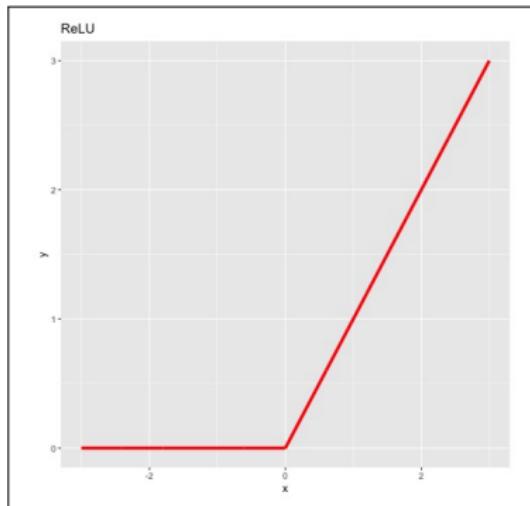
- ▶ A valeurs dans  $[-1, 1]$ , elle est continûment dérivable, de dérivée :

$$\varphi'(x) = \frac{1}{(1 + |x|)^2}.$$

## Fonction d'activation ReLU

- ▶ La fonction **ReLU** (*Rectified Linear Unit* ou rampe) est :

$$\varphi(x) = \begin{cases} x & \text{si } x \geq 0 \\ 0 & \text{sinon} \end{cases}.$$



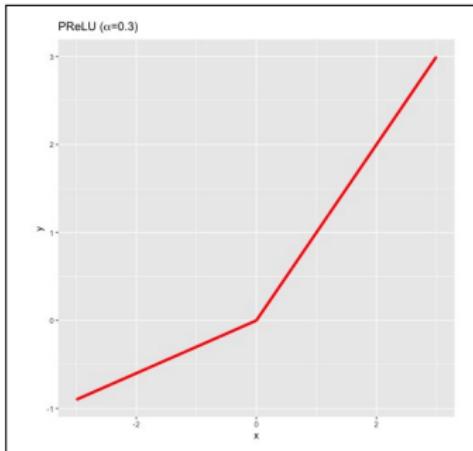
- ▶ A valeurs dans  $\mathbb{R}^+$ , sa dérivée vaut :

$$\varphi'(x) = \begin{cases} 1 & \text{si } x > 0 \ (\geq 0) \\ 0 & \text{si } x < 0 \end{cases}.$$

## Fonction d'activation PReLU

- ▶ La fonction **PReLU** (*Parametric Rectified Linear Unit*) est :

$$\varphi(x) = \begin{cases} x & \text{si } x \geq 0 \\ \alpha x & \text{si } x < 0 \end{cases} \quad \text{avec } \alpha \in \mathbb{R}^+.$$



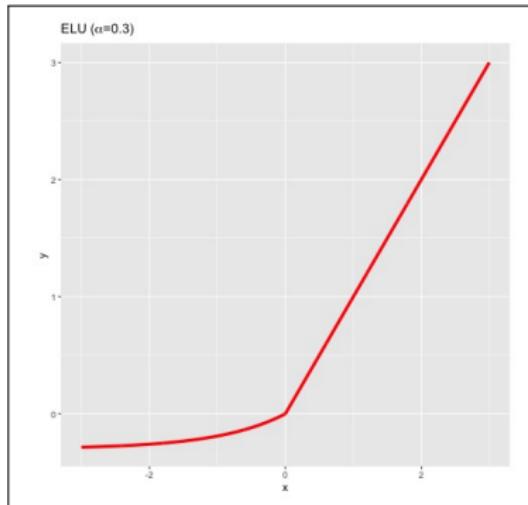
- ▶ A valeurs dans  $\mathbb{R}$ , sa dérivée vaut :

$$\varphi'(x) = \begin{cases} 1 & \text{si } x > 0 (\geq 0) \\ \alpha & \text{si } x < 0 \end{cases} .$$

## Fonction d'activation ELU

- ▶ La fonction **ELU** (*Exponential Linear Unit*) est :

$$\varphi(x) = \begin{cases} x & \text{si } x \geq 0 \\ \alpha(e^x - 1) & \text{sinon} \end{cases} \quad \text{avec } \alpha \in \mathbb{R}^+.$$



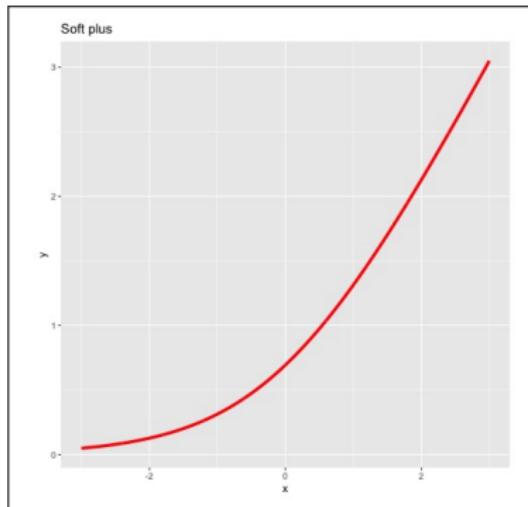
- ▶ A valeurs dans  $]-\alpha, +\infty[$ , sa dérivée vaut :

$$\varphi'(x) = \begin{cases} 1 & \text{si } x > 0 (\geq 0) \\ \varphi(x) + \alpha & \text{si } x < 0 \end{cases} .$$

## Fonction d'activation soft plus

- ▶ La fonction *soft plus* est :

$$\varphi(x) = \ln(1 + e^x) \quad \text{avec } \alpha \in \mathbb{R}^+.$$



- ▶ A valeurs dans  $\mathbb{R}^+$ , elle est infiniment continûment dérivable, de dérivée :

$$\varphi'(x) = \frac{1}{1 + e^{-x}} .$$

# Plan

Introduction

Neurone formel

Fonctions d'activation

**Perceptron multicouche**

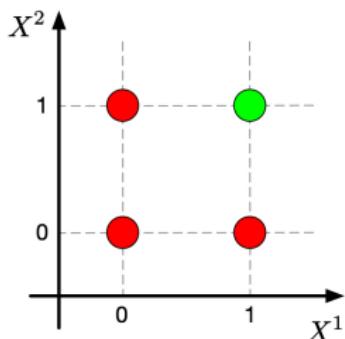
Rétro-propagation

Pratique du perceptron multicouche

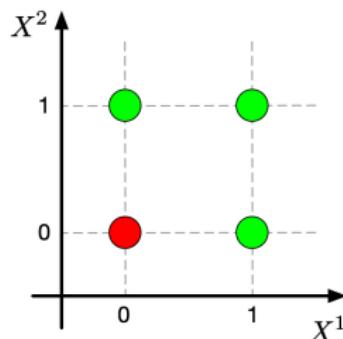
## Limites du neurone formel

- ▶ Le neurone formel ou le perceptron simple ne permettent de traiter correctement que des problèmes linéairement séparables.
- ▶ Afin de traiter des **problèmes non-linéairement séparables** (tels que que le XOR (« OU exclusif »)), on doit ajouter des couches aux réseaux de neurones.

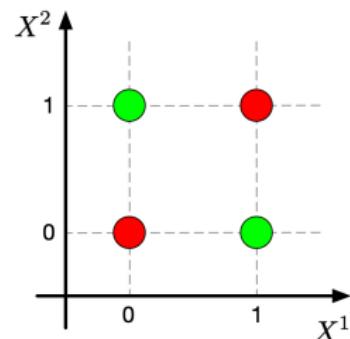
# Fonctions logiques



AND



OR

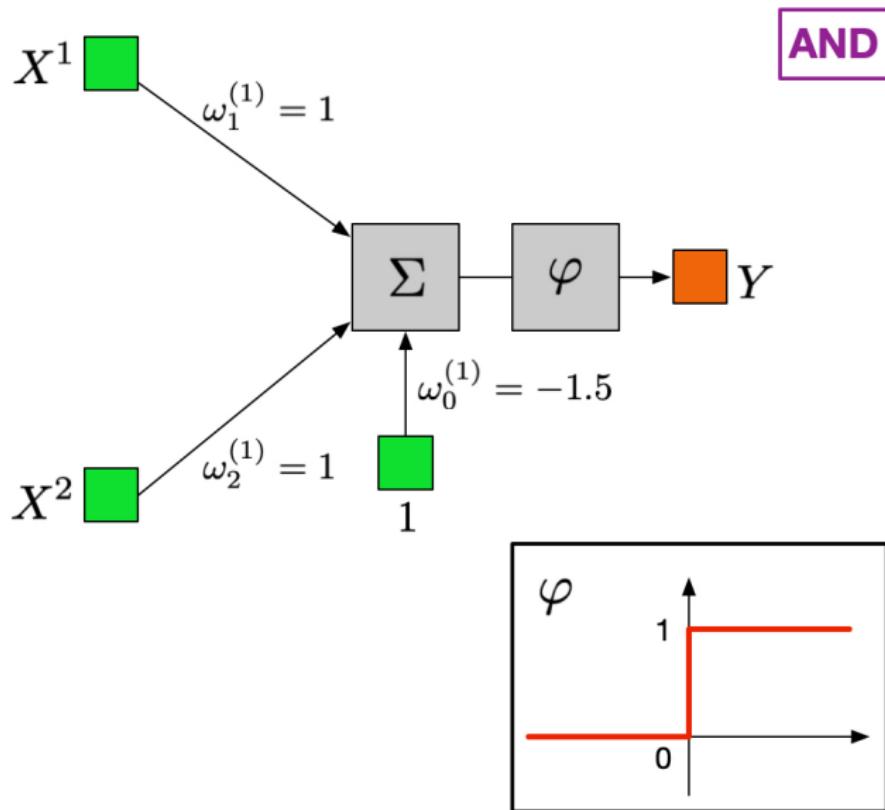


XOR

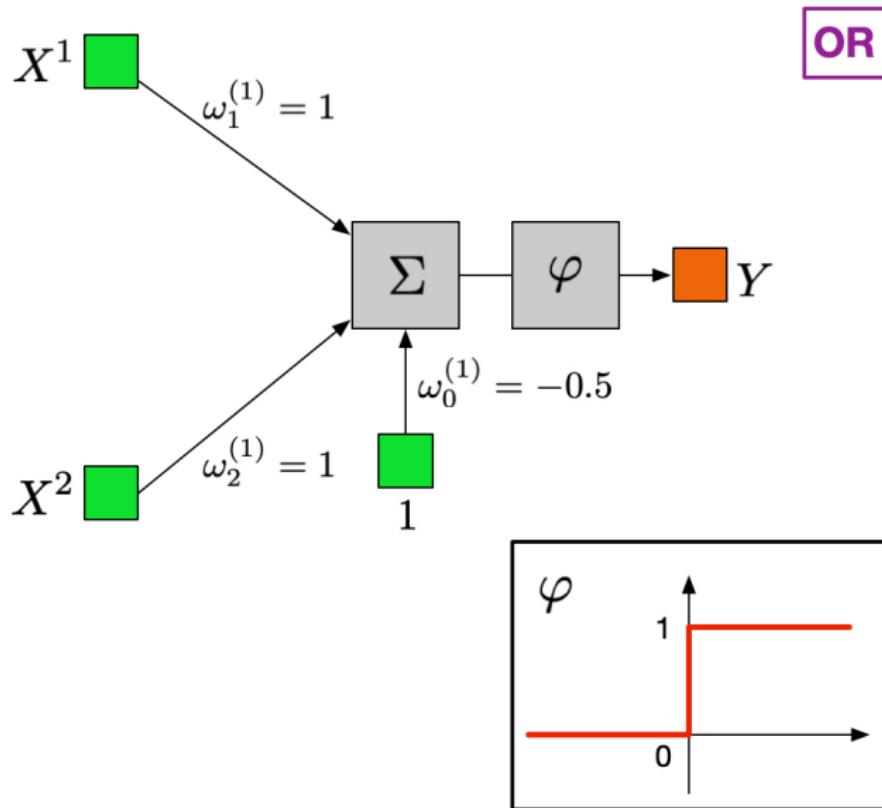
$Y = 0$

$Y = 1$

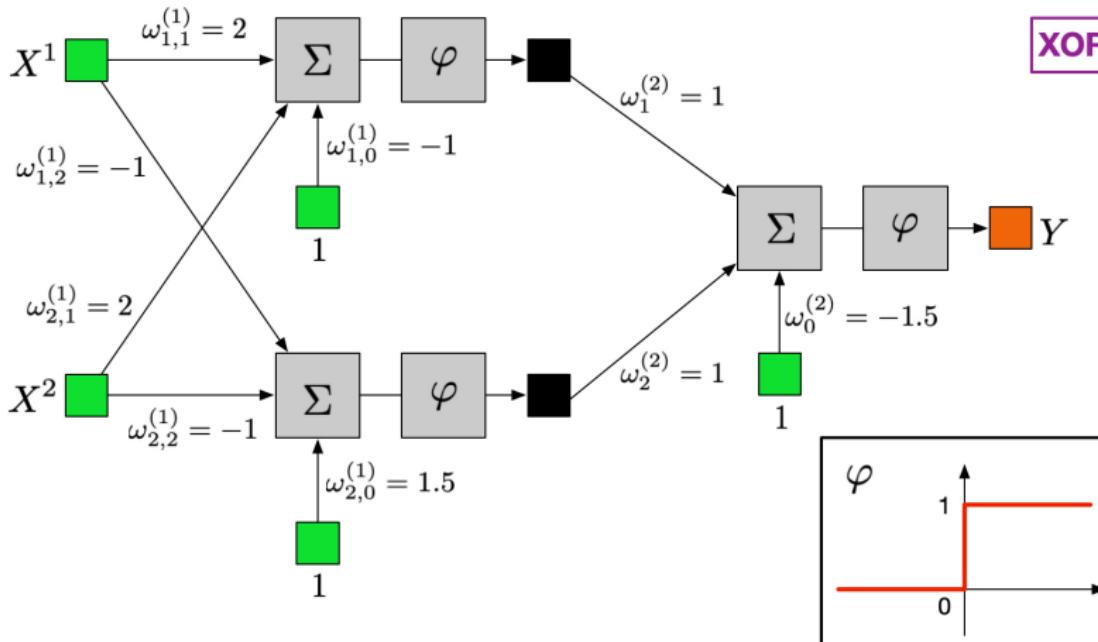
## Fonction AND



## Fonction OR



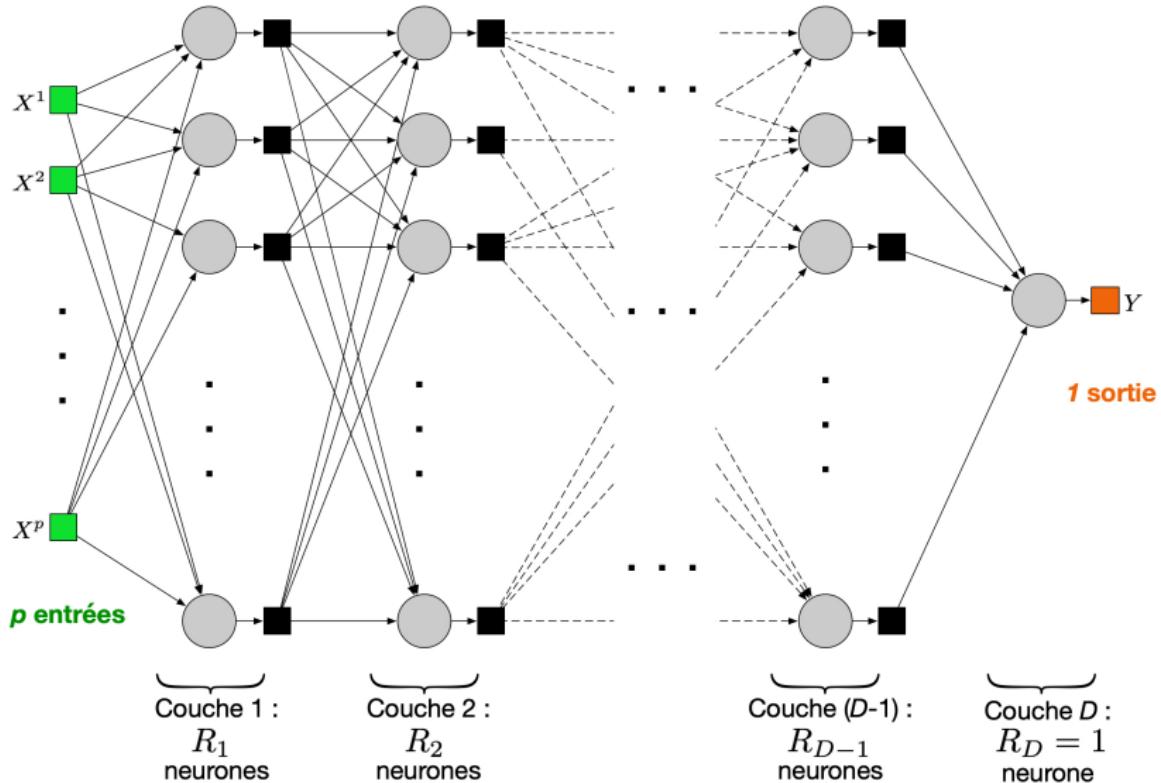
# Fonction XOR



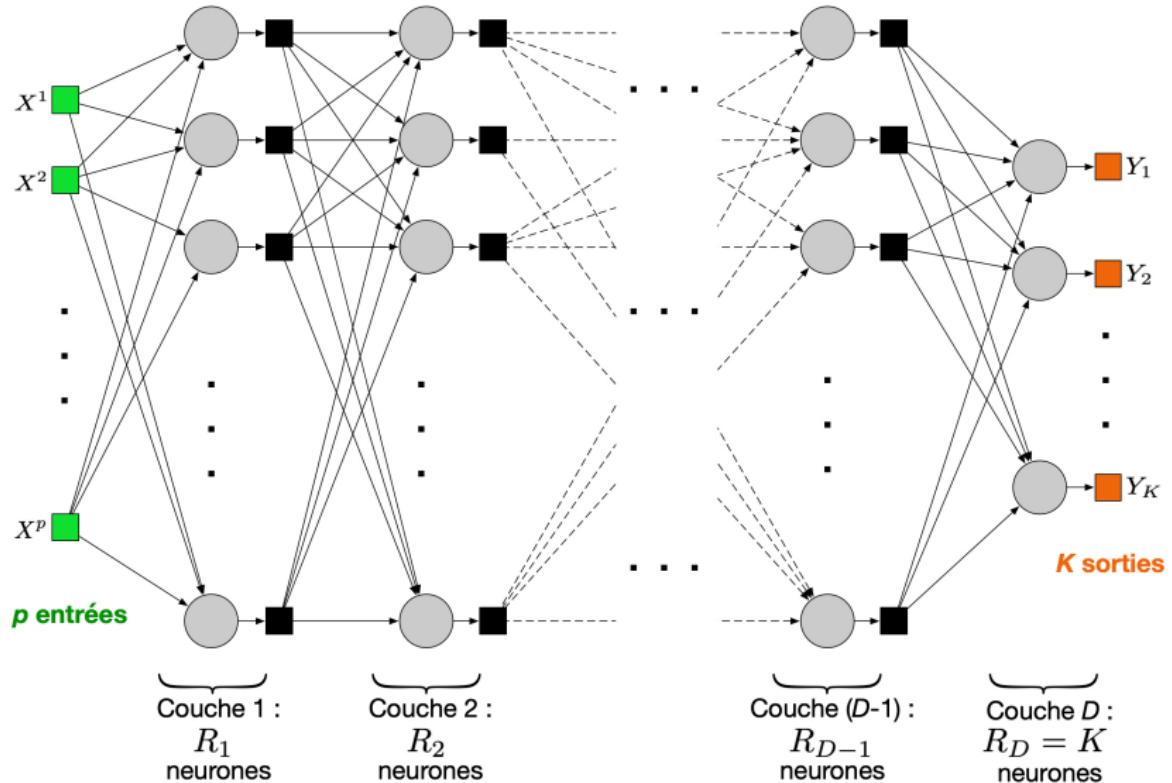
## Architecture

- ▶ Un **perceptron multicouche** (**MLP** : *MultiLayer Perceptron*) est constitué par des couches de neurones, chaque neurone d'une couche étant relié aux neurones de la couche adjacente.
- ▶ On parle d'architecture **feed-forward** (à propagation avant) : il n'y a pas de boucle de retour vers les couches plus basses du réseau.
- ▶ Les couches de neurones intermédiaires sont appelées **couches cachées** : plus leur nombre est important, plus le risque de sur-apprentissage est important.
- ▶ Pour les problématiques de traitement d'images, on peut trouver 30 couches cachées...

# Représentation I



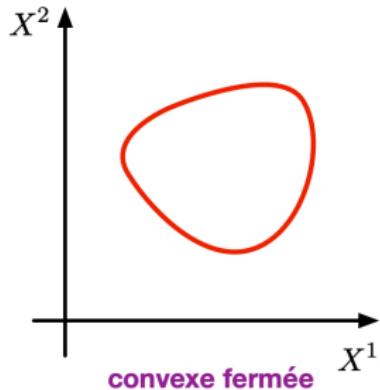
## Représentation II



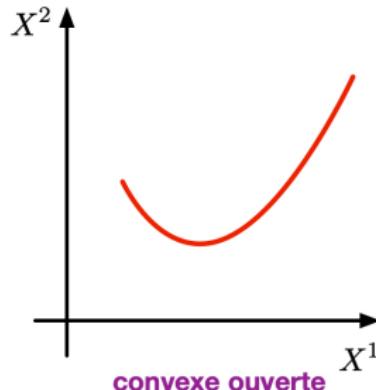
## Théorème d'approximation universelle

- ▶ Le théorème d'approximation universelle (Cybenko en 1989, Hirnik en 1991) indique que toute fonction continue peut s'approximer par un réseau de neurones avec une couche cachée.
- ▶ La complexité des problèmes peut néanmoins exiger beaucoup (trop) de neurones, conduisant à des réseaux de neurones « *fat* ». On leur préfère en pratique des réseaux de neurones profonds « *deep* ».

## Frontières de décision

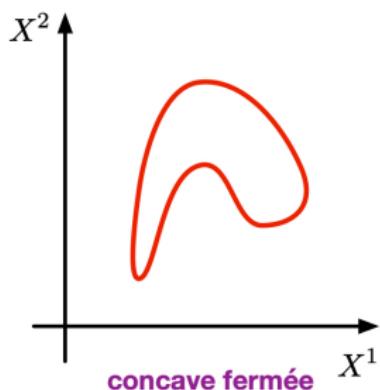


convexe fermée

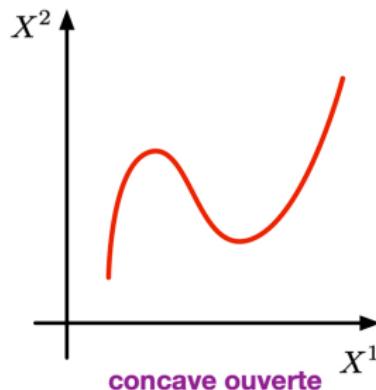


convexe ouverte

$D = 2$



concave fermée



concave ouverte

$D > 2$

## Estimation des poids et des biais

- ▶ L'apprentissage des poids et des biais s'effectue classiquement à l'aide de l'**algorithme de retropropagation** (*backpropagation*).
- ▶ Dans le domaine des réseaux de neurones, on emploie usuellement le terme d'**entraînement**.

# Plan

Introduction

Neurone formel

Fonctions d'activation

Perceptron multicouche

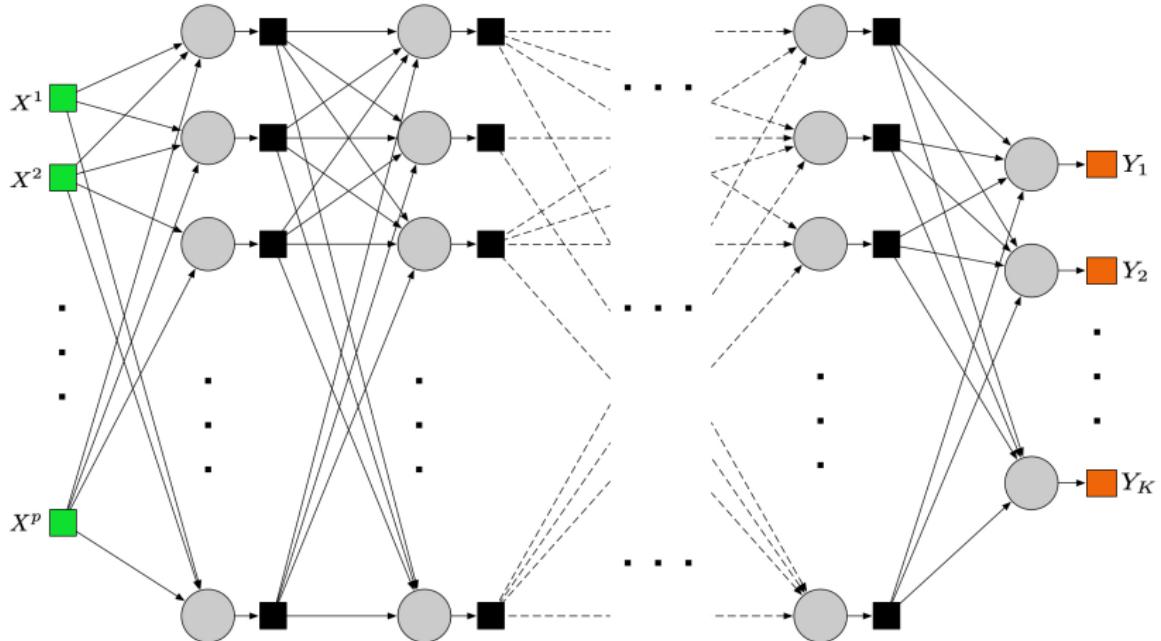
Rétro-propagation

Pratique du perceptron multicouche

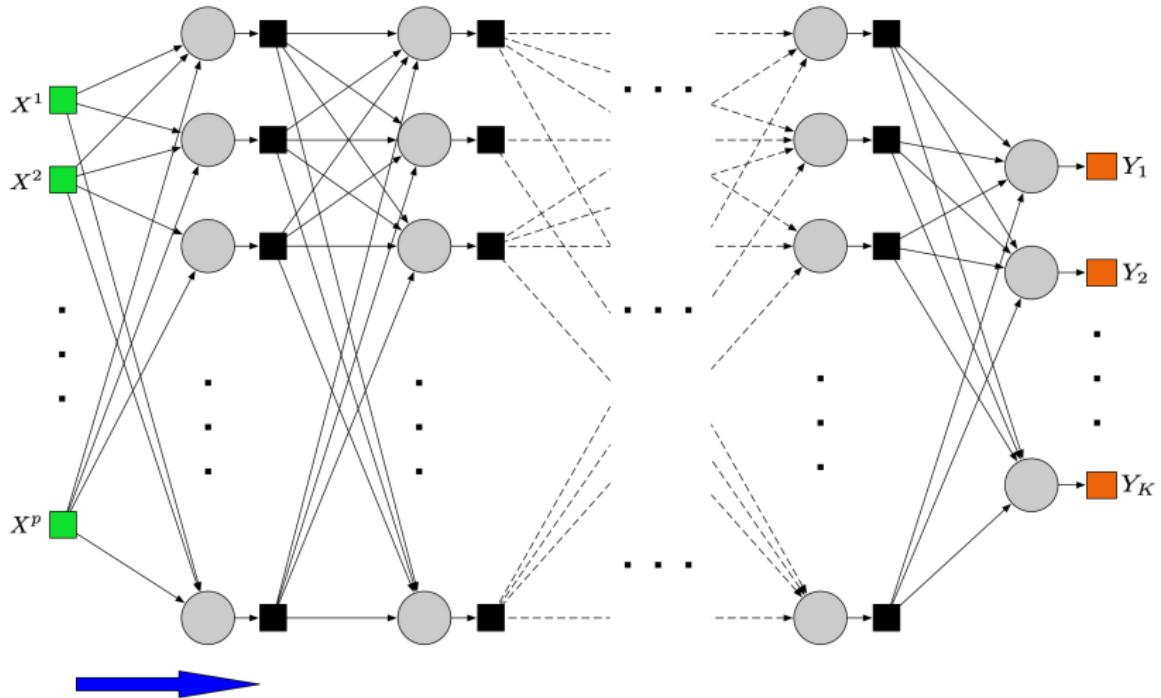
## Idée

- ▶ L'erreur sur un neurone de la couche cachée  $d \in \{1, \dots, D - 1\}$  dépend de l'erreur des neurones de la couche suivante ( $d + 1$ ).
- ▶ C'est ce constat qui est à l'origine de l'idée de **rétro-propagation** (*backward propagation* ou *backpropagation*) des erreurs : on amende les valeurs des poids et des biais de chaque neurone en rétro-propageant l'erreur observée sur la dernière couche.
- ▶ On applique la **descente du gradient** au risque empirique.
- ▶ La **propagation avant** (*forward propagation*) précède la rétro-propagation.

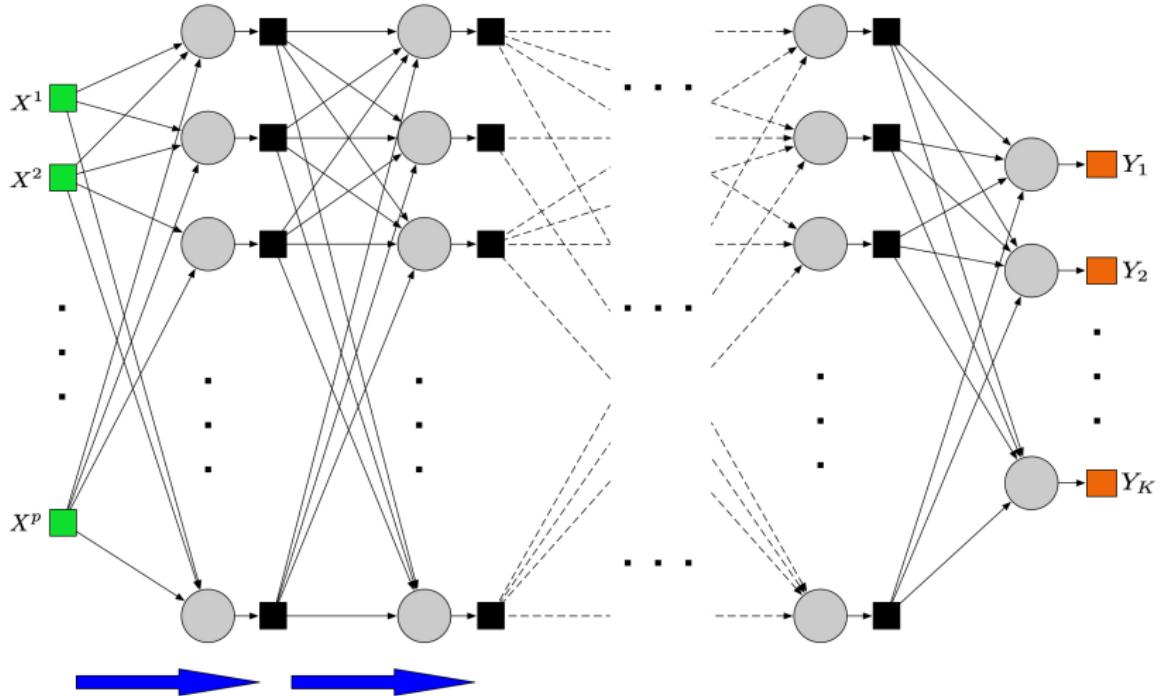
## Propagation avant...



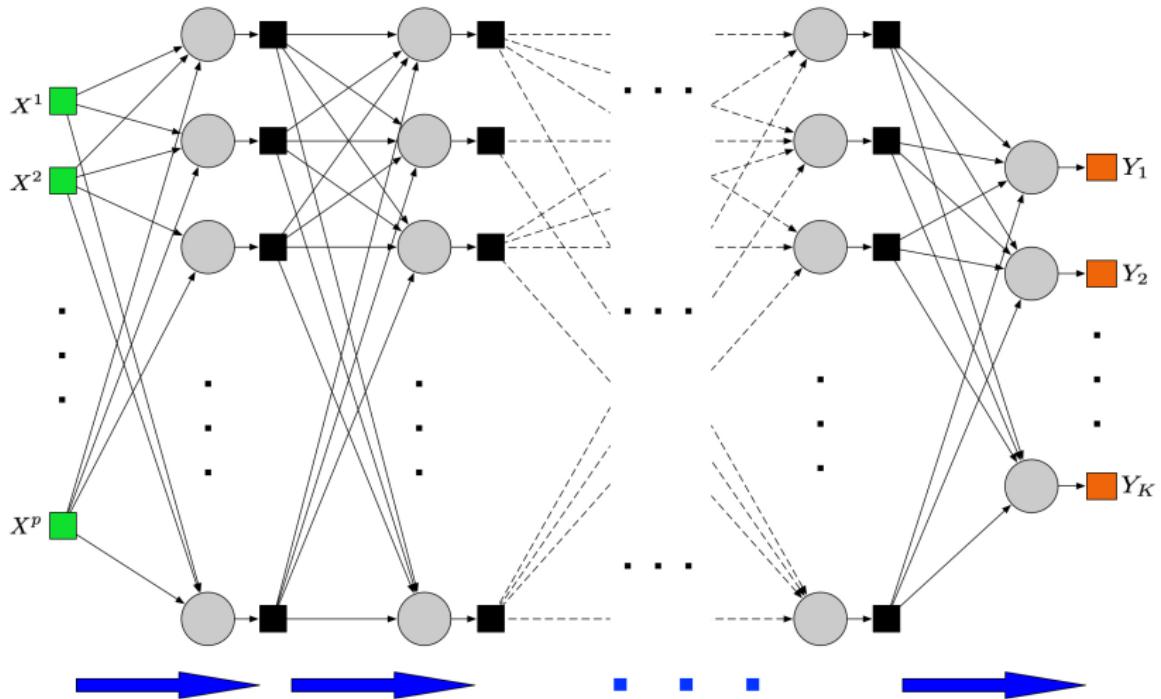
## Propagation avant...



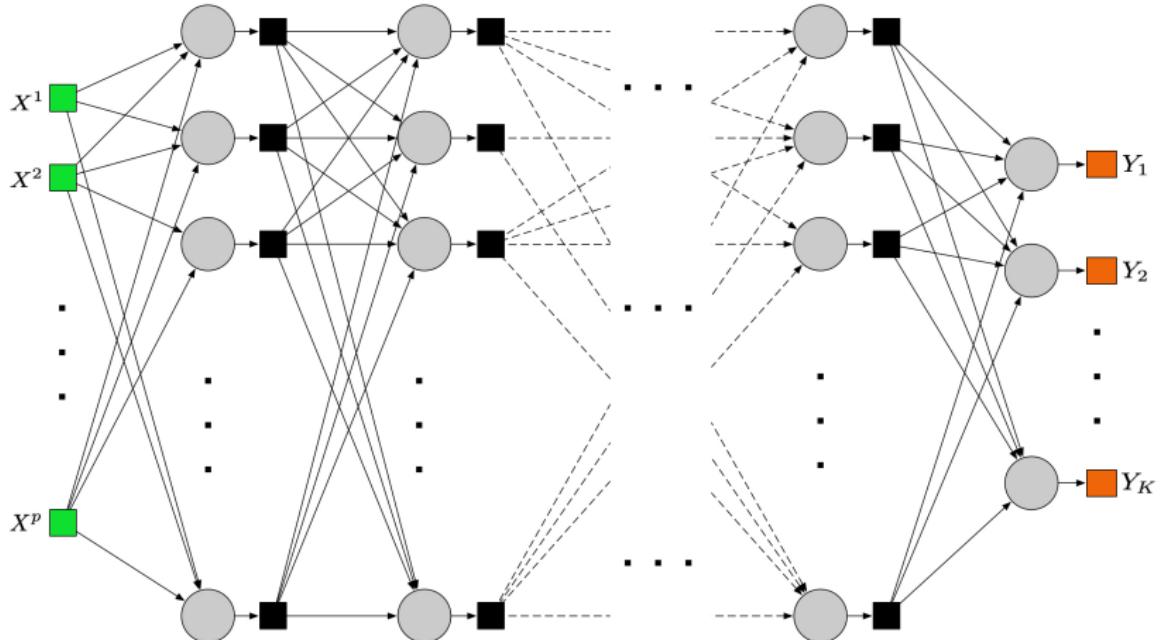
## Propagation avant...



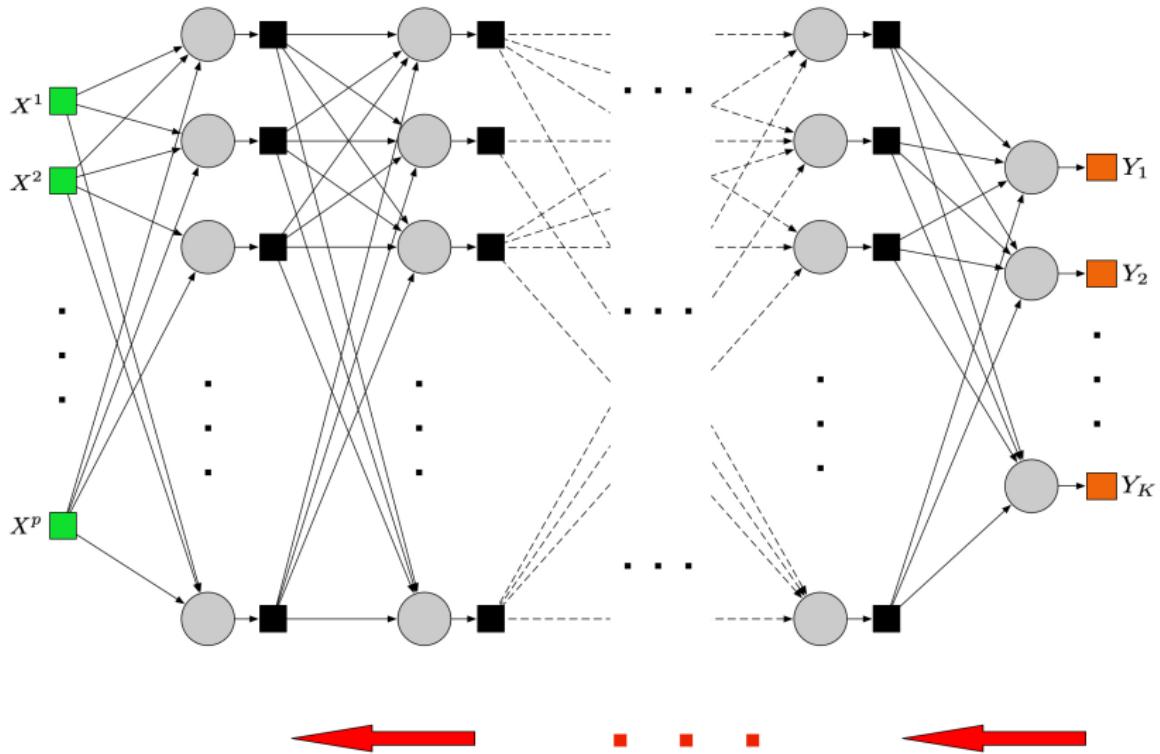
## Propagation avant...



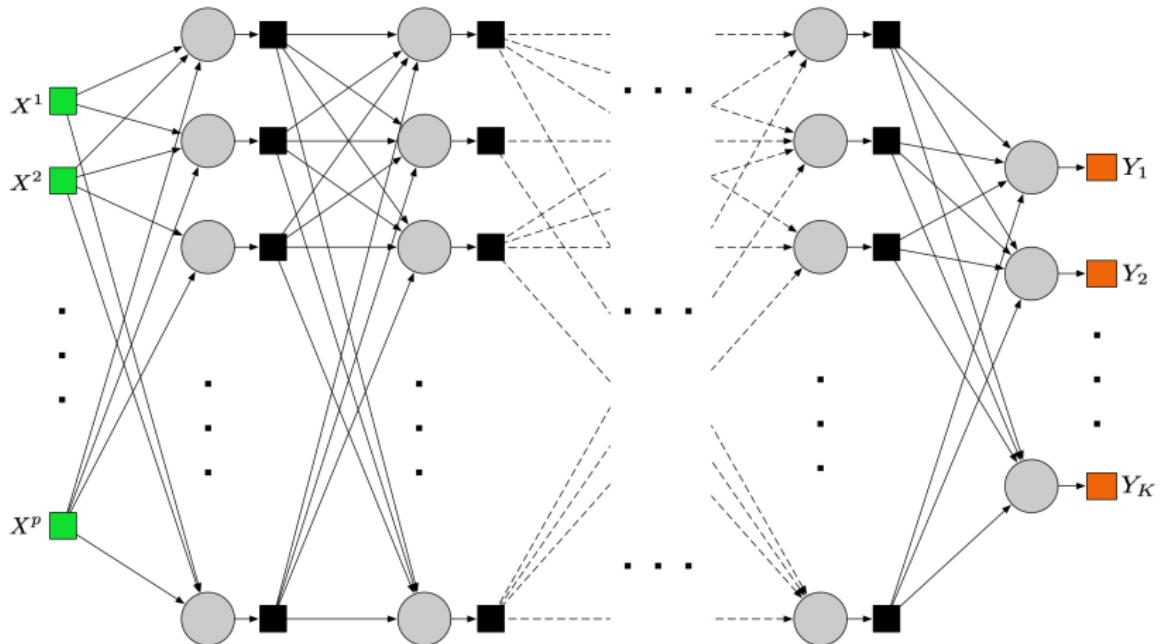
... puis rétro-propagation



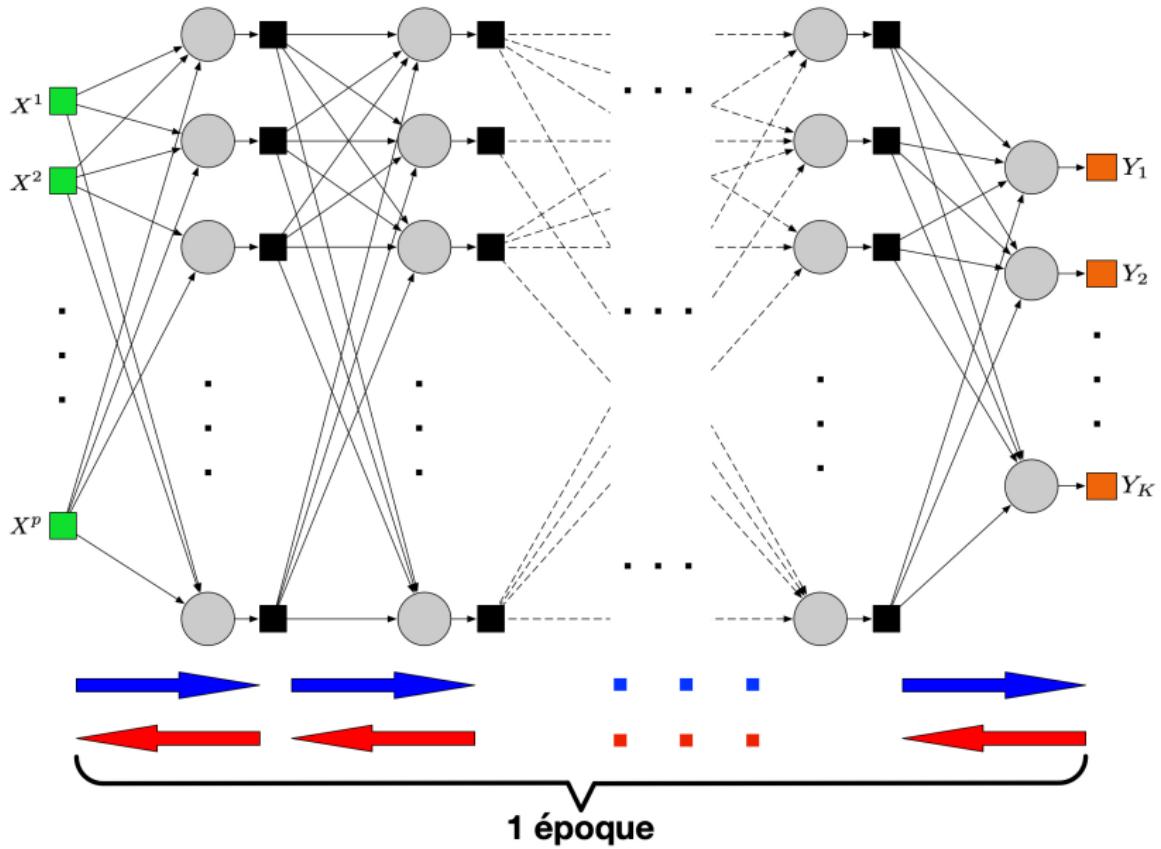
... puis rétro-propagation



... puis rétro-propagation



... puis rétro-propagation



## Evaluation du risque : cas d'une sortie

- ▶ Avec des poids, des biais et une fonction d'activation donnés, pour une entrée  $x_i = (x_i^1, \dots, x_i^p)^\top$ , on obtient  $\hat{y}_i$  (noté  $o^{(D),i}$  dans la suite) en sortie du perceptron multicouche.
- ▶ On considère :
  - ▶ l'**entropie croisée** dans le cas de **classification supervisée** :

$$R_n = - \sum_{i=1}^n y_i \ln (\hat{y}_i) ,$$

- ▶ l'**erreur quadratique** (divisée par un facteur 2 pour des commodités d'écriture) dans le cas de **régression** :

$$R_n = \frac{1}{2} \sum_{i=1}^n (y_i - \hat{y}_i)^2 .$$

## Evaluation du risque : cas de $K$ sorties

- ▶ Dans le cas de la **classification supervisée à  $K$  classes**, chaque neurone de sortie  $k \in \{1, \dots, K\}$  prend comme valeur 1 si l'observation  $i$  est dans la classe  $k$ , 0 sinon.
- ▶ Le risque empirique est alors la **somme des risques empiriques** des  $K$  neurones.

## Hypothèses et notations

- ▶ On se place dans le cadre de la **régression**, avec la **fonction d'activation sigmoïde**.
- ▶ Pour le neurone  $j \in \{1, \dots, R_d\}$  de la couche  $d \in \{1, \dots, D\}$ , et une entrée  $x_i$ , la sortie  $o_j^{(d),i}$  vaut :

$$o_j^{(d),i} = \varphi(a_j^{(d),i})$$

où :

$$a_j^{(d),i} = \omega_{0,j}^{(d),i} + \sum_{\ell=1}^{R_{d-1}} \omega_{\ell,j}^{(d)} o_{\ell}^{(d-1),i}.$$

## Correction des poids et biais : couche de sortie $D$ I

- ▶ On considère le neurone de la couche  $D$ .
- ▶ Pour l'observation  $i \in \{1, \dots, n\}$ , l'erreur en sortie du neurone vaut :

$$e^{(D),i} = y_i - o^{(D),i}$$

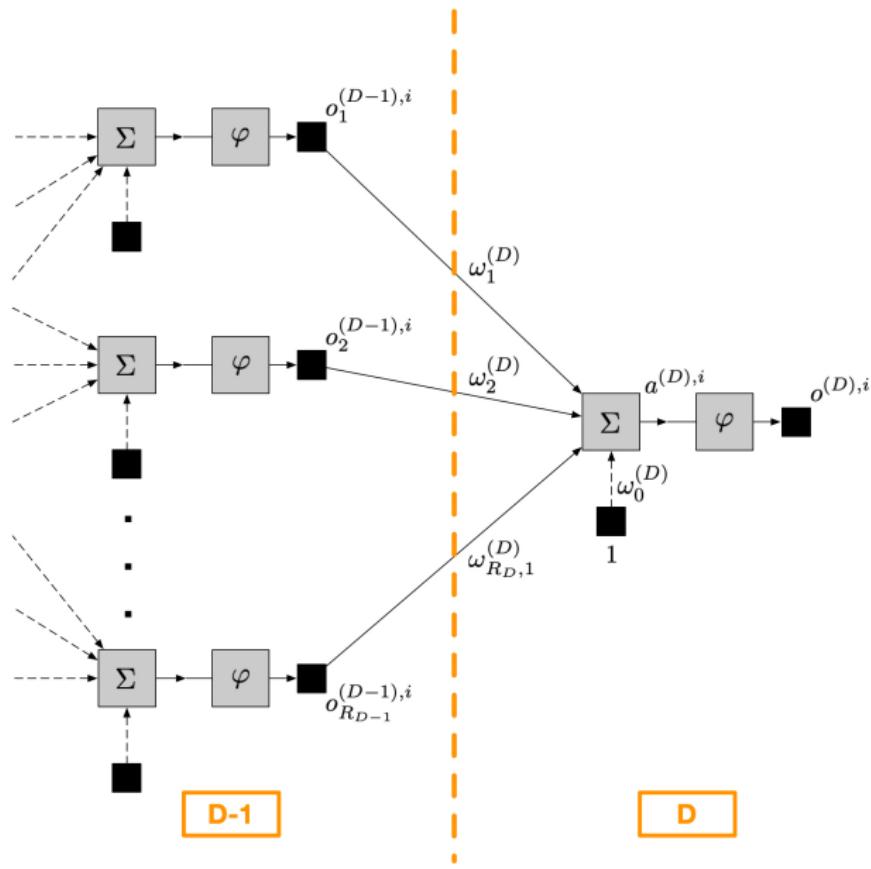
et le risque empirique (erreur quadratique) vaut :

$$R_n^{(D),i} = \frac{1}{2} \left( e^{(D),i} \right)^2 = \frac{1}{2} \left( y_i - o^{(D),i} \right)^2 .$$

- ▶ Pour l'échantillon  $d_n$ , le risque empirique vaut :

$$R_n^{(D)} = \frac{1}{n} \sum_{i=1}^n R_n^{(D),i} .$$

## Correction des poids et biais : couche de sortie $D$ II



## Correction des poids et biais : couche de sortie $D$ III

- ▶ La descente du gradient permet de considérer :

$$\begin{cases} \Delta\omega_0^{(D)} = -\eta \frac{\partial R_n^{(D)}}{\partial \omega_0^{(D)}} = -\frac{\eta}{n} \sum_{i=1}^n \frac{\partial R_n^{(D),i}}{\partial \omega_0^{(D)}} \\ \Delta\omega_\ell^{(D)} = -\eta \frac{\partial R_n^{(D)}}{\partial \omega_\ell^{(D)}} = -\frac{\eta}{n} \sum_{i=1}^n \frac{\partial R_n^{(D),i}}{\partial \omega_\ell^{(D)}} \end{cases}$$

où  $\eta$  est le **taux d'apprentissage**.

- ▶ La modification des poids de la couche de sortie dépend de l'erreur commise en sortie.
- ▶ Le théorème de **dérivation en chaîne** permet d'écrire :

$$\begin{cases} \frac{\partial R_n^{(D),i}}{\partial \omega_0^{(D)}} = \frac{\partial R_n^{(D),i}}{\partial o^{(D),i}} \frac{\partial o^{(D),i}}{\partial a^{(D),i}} \frac{\partial a^{(D),i}}{\partial \omega_0^{(D)}} \\ \frac{\partial R_n^{(D),i}}{\partial \omega_\ell^{(D)}} = \frac{\partial R_n^{(D),i}}{\partial o^{(D),i}} \frac{\partial o^{(D),i}}{\partial a^{(D),i}} \frac{\partial a^{(D),i}}{\partial \omega_\ell^{(D)}} \end{cases} .$$

## Correction des poids et biais : couche de sortie $D$ IV

On obtient :

$$\frac{\partial R_n^{(D),i}}{\partial o^{(D),i}} = - \left( y_i - o^{(D),i} \right) = -e^{(D),i} ,$$

$$\frac{\partial o^{(D),i}}{\partial a^{(D),i}} = \frac{\partial \varphi(a^{(D),i})}{\partial a^{(D),i}} = \varphi(a^{(D),i}) \left[ 1 - \varphi(a^{(D),i}) \right] = o^{(D),i} \left( 1 - o^{(D),i} \right) ,$$

$$\frac{\partial a^{(D),i}}{\partial \omega_0^{(D)}} = \frac{\partial}{\partial \omega_0^{(D)}} \left( \omega_0^{(D)} + \sum_{m=1}^{R_{D-1}} \omega_m^{(D)} o_m^{(D-1),i} \right) = 1 ,$$

$$\frac{\partial a^{(D),i}}{\partial \omega_\ell^{(D)}} = \frac{\partial}{\partial \omega_\ell^{(D)}} \left( \omega_0^{(D)} + \sum_{m=1}^{R_{D-1}} \omega_m^{(D)} o_m^{(D-1),i} \right) = o_\ell^{(D-1),i} .$$

## Correction des poids et biais : couche de sortie $D$ V

D'où :

$$\begin{cases} \frac{\partial R_n^{(D),i}}{\partial \omega_0^{(D)}} = -e^{(D),i} o^{(D),i} (1 - o^{(D),i}) \\ \frac{\partial R_n^{(D),i}}{\partial \omega_\ell^{(D)}} = -e^{(D),i} o^{(D),i} (1 - o^{(D),i}) o_\ell^{(D-1),i} \end{cases} .$$

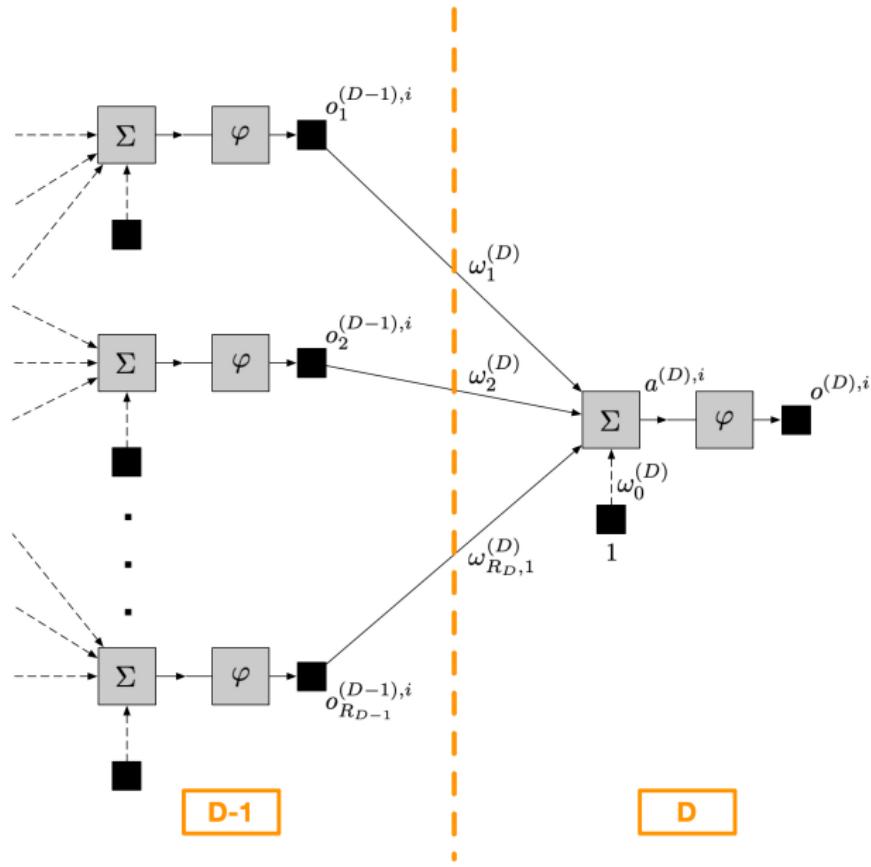
La correction des poids et biais apportée sur la (dernière) couche  $D$  est :

$$\begin{cases} \Delta \omega_0^{(D)} = \frac{\eta}{n} \sum_{i=1}^n \delta^{(D),i} \\ \Delta \omega_\ell^{(D)} = \frac{\eta}{n} \sum_{i=1}^n \delta^{(D),i} o_\ell^{(D-1),i} \end{cases}$$

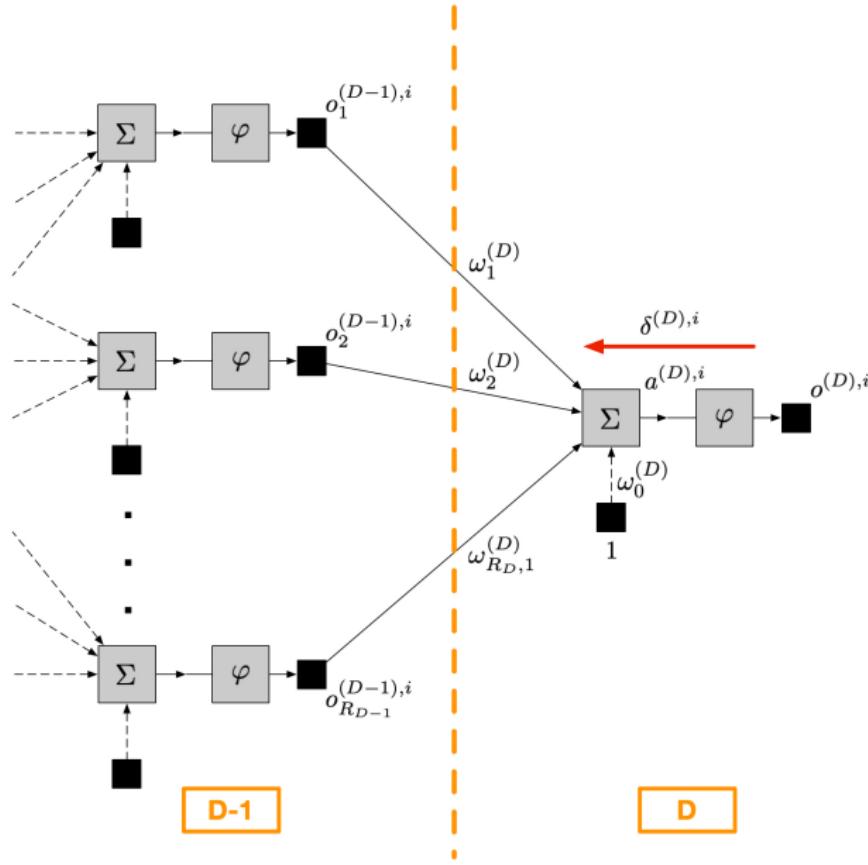
où :

$$\delta^{(D),i} = e^{(D),i} o^{(D),i} (1 - o^{(D),i}) .$$

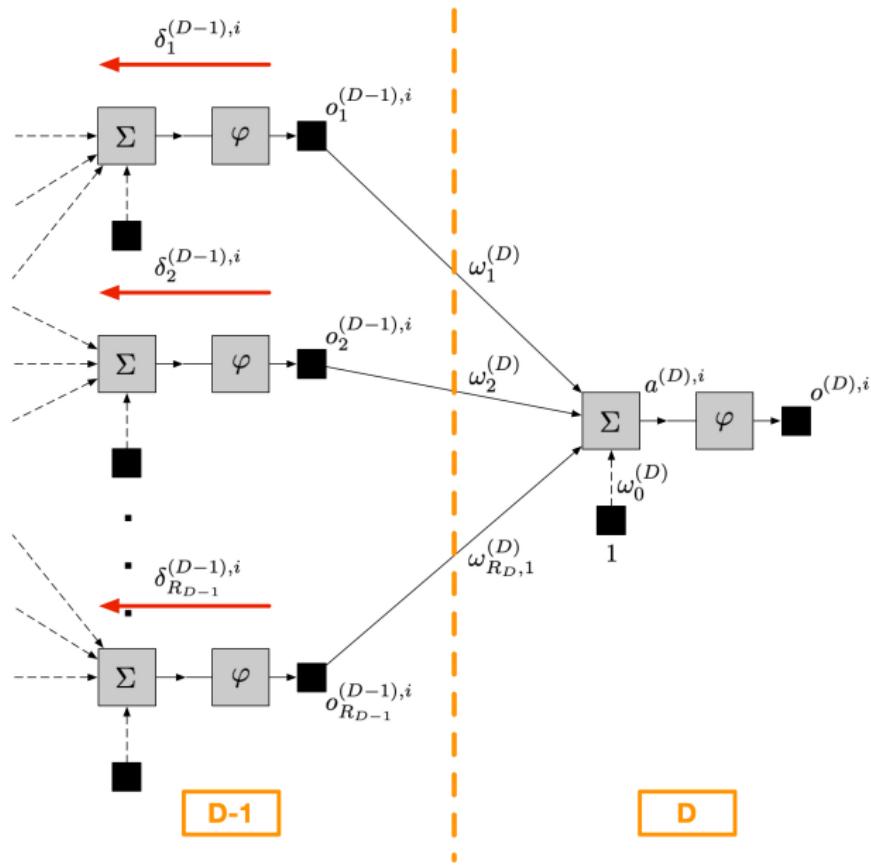
# Correction des poids et biais : couche de sortie $D$ VI



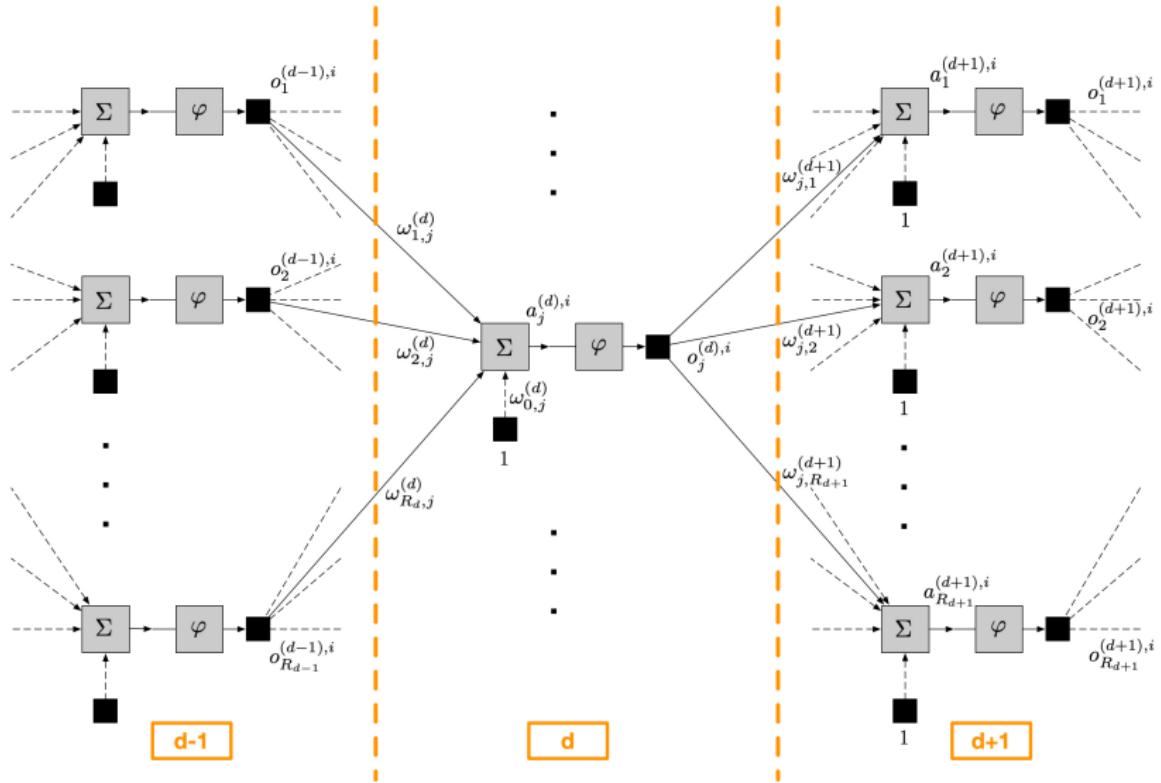
# Correction des poids et biais : couche de sortie $D$ VI



# Correction des poids et biais : couche de sortie $D$ VI



# Correction des poids et biais : couche cachée $d$ |



## Correction des poids et biais : couche cachée $d$ II

- ▶ On considère un neurone  $j \in \{1, \dots, d\}$  de la couche  $d$ .
- ▶ L'erreur sur le neurone  $j$  de la couche  $d$  dépend de l'erreur des  $R_{d+1}$  neurones de la couche  $d + 1$ .
- ▶ Pour l'observation  $i \in \{1, \dots, n\}$ , l'erreur en sortie du neurone  $m$  de la couche  $d + 1$  vaut :

$$e_m^{(d+1),i} = t_m^{(d+1),i} - o_m^{(d+1),i}$$

où  $t_m^{(d+1),i}$  est la valeur « objectif » (*target*) à la sortie du neurone  $m$ , et le risque empirique (erreur quadratique) vaut :

$$R_n^{(d+1),i} = \sum_{m=1}^{R_{d+1}} \frac{1}{2} \left( e_m^{(d+1),i} \right)^2 = \sum_{m=1}^{R_{d+1}} \frac{1}{2} \left( t_m^{(d+1),i} - o_m^{(d+1),i} \right)^2.$$

- ▶ Pour l'échantillon  $d_n$ , le risque empirique en sortie de la couche  $d + 1$  vaut :

$$R_n^{(d+1)} = \frac{1}{n} \sum_{i=1}^n R_n^{(d+1),i}.$$

## Correction des poids et biais : couche cachée $d$ III

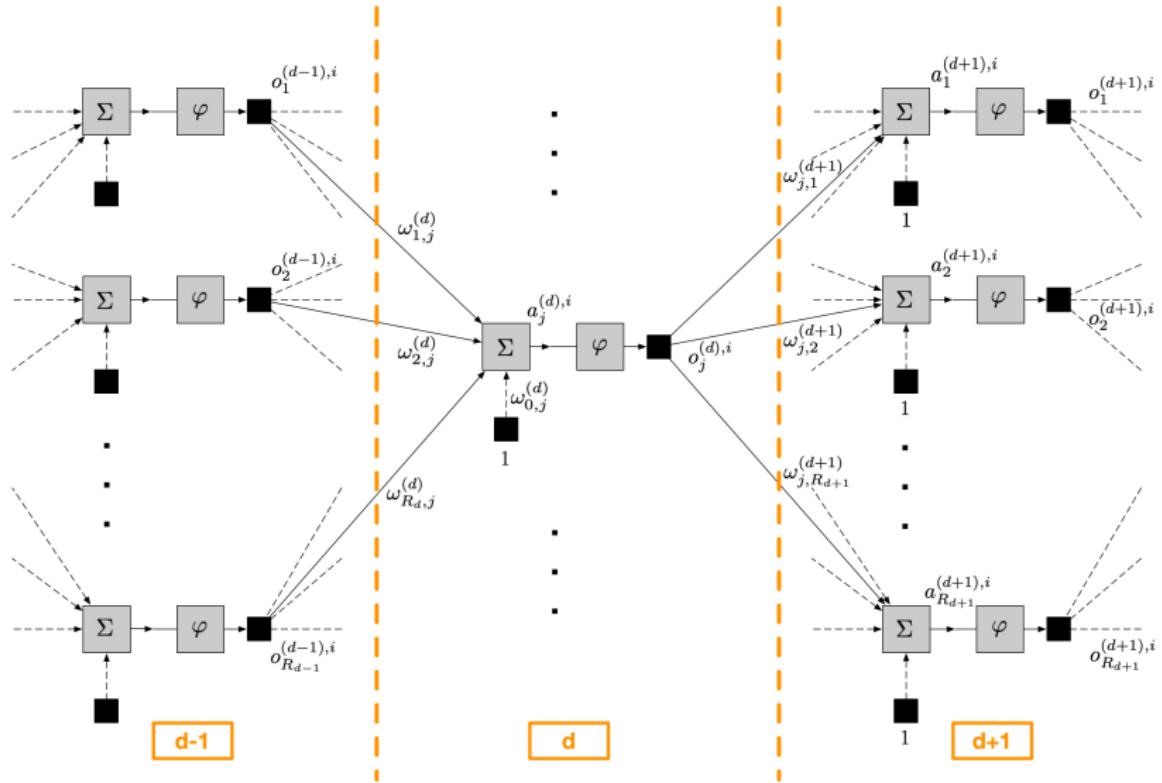
La correction des poids et biais apportée à la couche  $d$  est :

$$\begin{cases} \Delta\omega_{0,j}^{(d)} = \frac{\eta}{n} \sum_{i=1}^n \delta_j^{(d),i} \\ \Delta\omega_{\ell,j}^{(d)} = \frac{\eta}{n} \sum_{i=1}^n \delta_j^{(d),i} o_{\ell,j}^{(d-1),i} \end{cases}$$

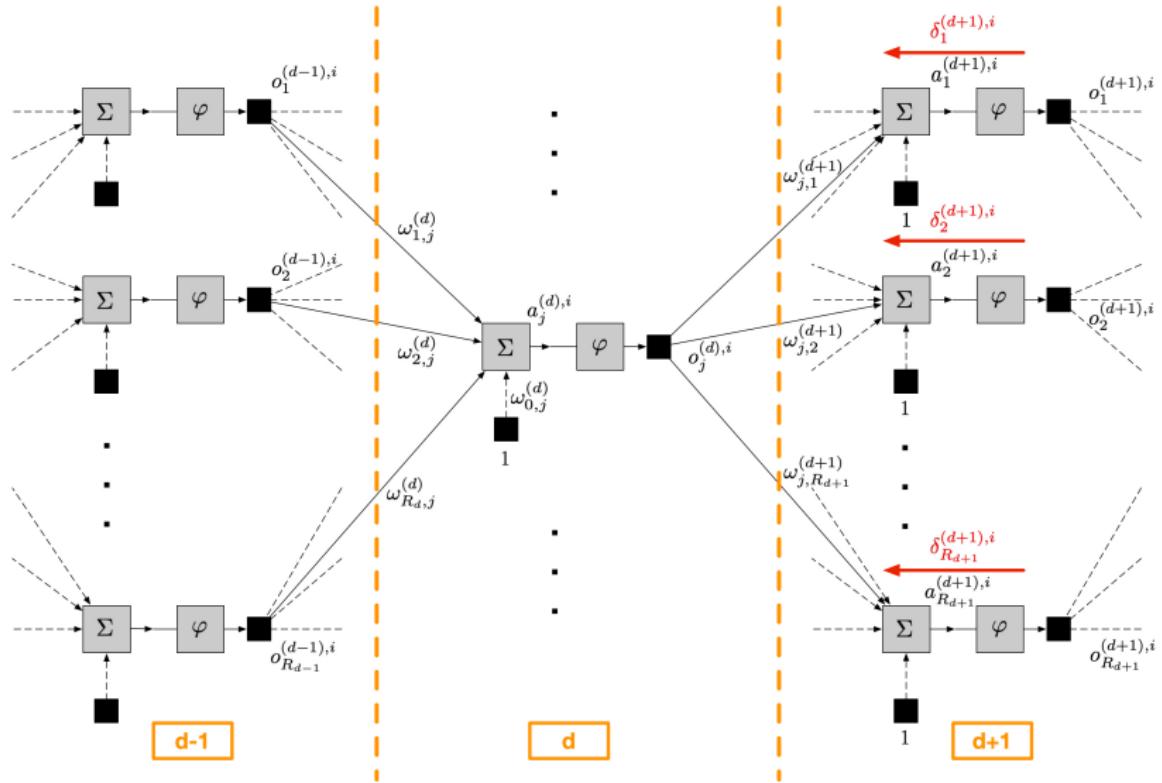
où :

$$\delta_j^{(d),i} = \left( \sum_{m=1}^{R_{d+1}} \delta_m^{(d+1),i} \omega_{j,m}^{(d+1)} \right) o_j^{(d),i} \left[ 1 - o_j^{(d),i} \right].$$

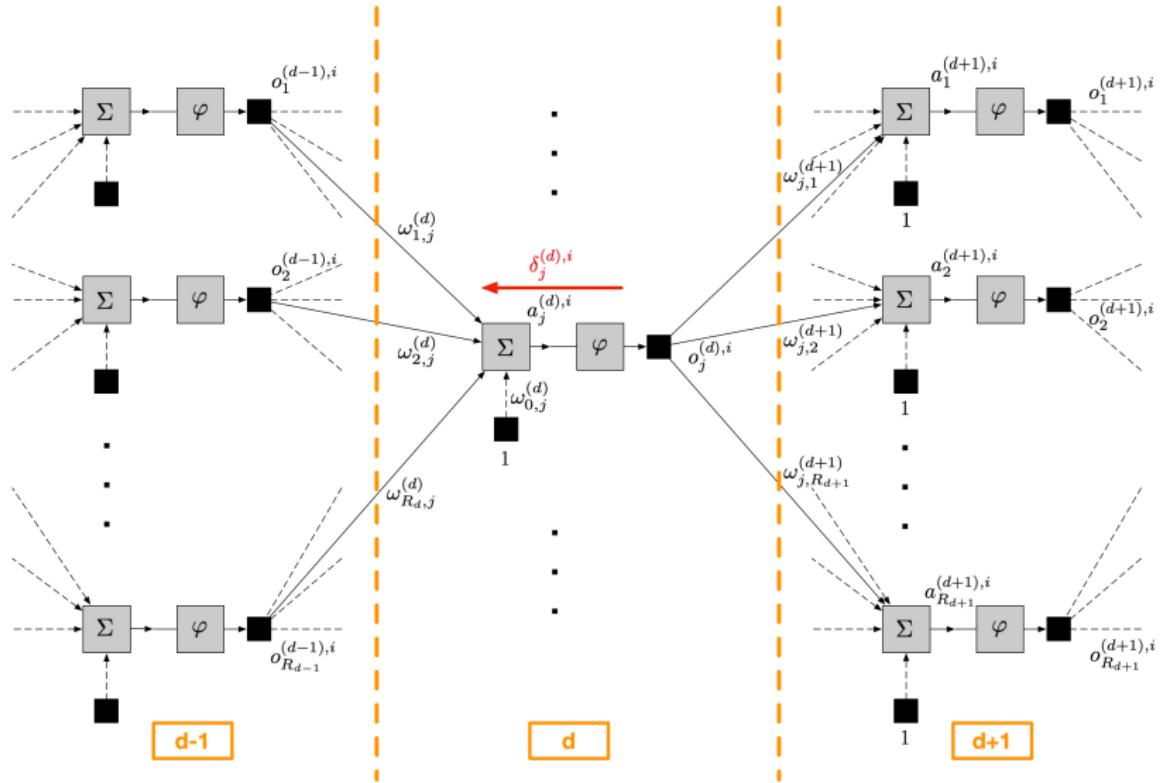
# Correction des poids et biais : couche cachée $d$ IV



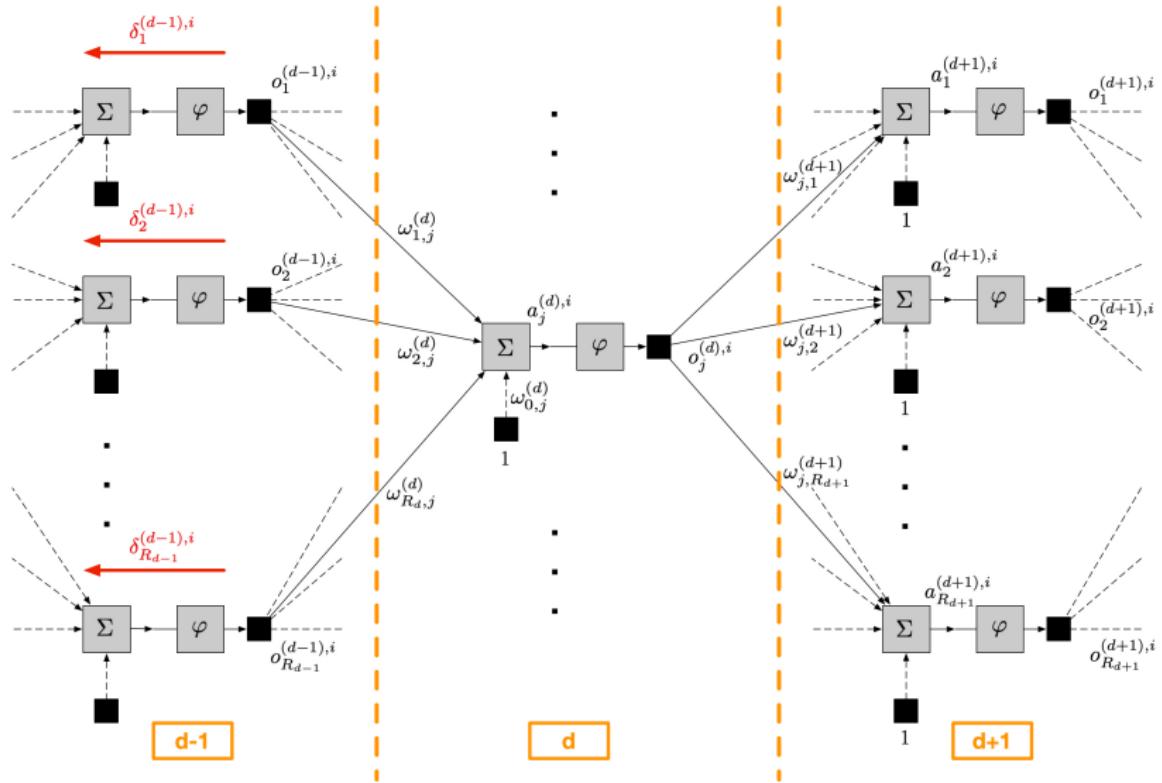
# Correction des poids et biais : couche cachée $d$ IV



# Correction des poids et biais : couche cachée $d$ IV



# Correction des poids et biais : couche cachée $d$ IV



## Correction des poids et biais : couche cachée $d$ (Calculs) I

- ▶ La descente du gradient permet de considérer pour  $\ell \in \{1, \dots, R_{d-1}\}$  :

$$\begin{cases} \Delta \omega_{0,j}^{(d)} = -\eta \frac{\partial R_n^{(d+1)}}{\partial \omega_{0,j}^{(d)}} = -\frac{\eta}{n} \sum_{i=1}^n \frac{\partial R_n^{(d+1),i}}{\partial \omega_{0,j}^{(d)}} \\ \Delta \omega_{\ell,j}^{(d)} = -\eta \frac{\partial R_n^{(d+1)}}{\partial \omega_{\ell,j}^{(d)}} = -\frac{\eta}{n} \sum_{i=1}^n \frac{\partial R_n^{(d+1),i}}{\partial \omega_{\ell,j}^{(d)}} \end{cases} .$$

- ▶ Le théorème de dérivation en chaîne permet d'écrire :

$$\begin{cases} \frac{\partial R_n^{(d+1),i}}{\partial \omega_{0,j}^{(d)}} = \frac{\partial R_n^{(d+1),i}}{\partial o_j^{(d),i}} \frac{\partial o_j^{(d),i}}{\partial a^{(D),i}} \frac{\partial a^{(D),i}}{\partial \omega_{0,j}^{(d)}} \\ \frac{\partial R_n^{(d+1),i}}{\partial \omega_{\ell,j}^{(d)}} = \frac{\partial R_n^{(d+1),i}}{\partial o_j^{(d),i}} \frac{\partial o_j^{(d),i}}{\partial a^{(D),i}} \frac{\partial a^{(D),i}}{\partial \omega_{\ell,j}^{(d)}} \end{cases} .$$

## Correction des poids et biais : couche cachée $d$ (Calculs) II

On obtient :

$$\frac{\partial o_j^{(d),i}}{\partial a_j^{(d),i}} = \frac{\partial \varphi(a_j^{(d),i})}{\partial a_j^{(d),i}} = \varphi(a_j^{(d),i}) [1 - \varphi(a_j^{(d),i})] = o_j^{(d),i} (1 - o_j^{(d),i}) ,$$

$$\frac{\partial a_j^{(d),i}}{\partial \omega_{0,j}^{(d)}} = \frac{\partial}{\partial \omega_{0,j}^{(d)}} \left( \omega_{0,j}^{(d)} + \sum_{\ell=1}^{R_{d-1}} \omega_{\ell,j}^{(d)} o_{\ell,j}^{(d-1),i} \right) = 1 ,$$

$$\frac{\partial a_j^{(d),i}}{\partial \omega_{\ell,j}^{(d)}} = \frac{\partial}{\partial \omega_{\ell,j}^{(d)}} \left( \omega_{0,j}^{(d)} + \sum_{m=1}^{R_{d-1}} \omega_{m,j}^{(d)} o_{m,j}^{(d-1),i} \right) = o_{\ell,j}^{(d-1),i} .$$

## Correction des poids et biais : couche cachée $d$ (Calculs) III

$$\begin{aligned}\frac{\partial R_n^{(d+1),i}}{\partial o_j^{(d),i}} &= \frac{\partial}{\partial o_j^{(d),i}} \left( \sum_{m=1}^{R_{d+1}} \frac{1}{2} \left( e_m^{(d+1),i} \right)^2 \right) \\ &= \sum_{m=1}^{R_{d+1}} \frac{\partial}{\partial o_j^{(d),i}} \left( \frac{1}{2} \left( e_m^{(d+1),i} \right)^2 \right) \\ &= \sum_{m=1}^{R_{d+1}} \frac{\partial}{\partial e_m^{(d+1),i}} \left( \frac{1}{2} \left( e_m^{(d+1),i} \right)^2 \right) \frac{\partial e_m^{(d+1),i}}{\partial o_j^{(d),i}} \\ &= \sum_{m=1}^{R_{d+1}} e_m^{(d+1),i} \frac{\partial e_m^{(d+1),i}}{\partial o_j^{(d),i}} \\ &= \sum_{m=1}^{R_{d+1}} e_m^{(d+1),i} \frac{\partial e_m^{(d+1),i}}{\partial a_m^{(d+1),i}} \frac{\partial a_m^{(d+1),i}}{\partial o_j^{(d),i}} .\end{aligned}$$

## Correction des poids et biais : couche cachée $d$ (Calculs) IV

$$\begin{aligned}\frac{\partial e_m^{(d+1),i}}{\partial a_m^{(d+1),i}} &= \frac{\partial}{\partial a_m^{(d+1),i}} \left( t_m^{(d+1),i} - o_m^{(d+1),i} \right) \\ &= -\frac{\partial}{\partial a_m^{(d+1),i}} \left( o_m^{(d+1),i} \right) \\ &= -\frac{\partial}{\partial a_m^{(d+1),i}} \left( \varphi \left( a_m^{(d+1),i} \right) \right) \\ &= -\varphi \left( a_m^{(d+1),i} \right) \left[ 1 - \varphi \left( a_m^{(d+1),i} \right) \right] \\ &= -o_m^{(d+1),i} \left( 1 - o_m^{(d+1),i} \right) ,\end{aligned}$$

$$\begin{aligned}\frac{\partial a_m^{(d+1),i}}{\partial o_j^{(d),i}} &= \frac{\partial}{\partial o_j^{(d),i}} \left( \omega_{0,m}^{(d+1),i} + \sum_{\ell=1}^{R_d} \omega_{\ell,m}^{(d+1)} o_{\ell}^{(d),i} \right) \\ &= \omega_{j,m}^{(d+1)} .\end{aligned}$$

## Correction des poids et biais : couche cachée $d$ (Calculs) V

Donc :

$$\begin{aligned}\frac{\partial R_n^{(d+1),i}}{\partial o_j^{(d),i}} &= - \sum_{m=1}^{R_{d+1}} e_m^{(d+1),i} o_m^{(d+1),i} \left(1 - o_m^{(d+1),i}\right) \omega_{j,m}^{(d+1)} \\ &= - \sum_{m=1}^{R_{d+1}} \delta_m^{(d+1),i} \omega_{j,m}^{(d+1)}\end{aligned}$$

où :

$$\delta_m^{(d+1),i} = e_m^{(d+1),i} o_m^{(d+1),i} \left(1 - o_m^{(d+1),i}\right).$$

## Correction des poids et biais : couche cachée $d$ (Calculs) VI

D'où :

$$\begin{cases} \frac{\partial R_n^{(d+1),i}}{\partial \omega_{0,j}^{(d)}} = - \left( \sum_{m=1}^{R_{d+1}} \delta_m^{(d+1),i} \omega_{j,m}^{(d+1)} \right) o_j^{(d),i} \left( 1 - o_j^{(d),i} \right) \\ \frac{\partial R_n^{(d+1),i}}{\partial \omega_{\ell,j}^{(d)}} = - \left( \sum_{m=1}^{R_{d+1}} \delta_m^{(d+1),i} \omega_{j,m}^{(d+1)} \right) o_j^{(d),i} \left( 1 - o_j^{(d),i} \right) o_{\ell,j}^{(d-1),i} \end{cases} .$$

Au final on obtient :

$$\begin{cases} \Delta \omega_{0,j}^{(d)} = \frac{\eta}{n} \sum_{i=1}^n \delta_j^{(d),i} \\ \Delta \omega_{\ell,j}^{(d)} = \frac{\eta}{n} \sum_{i=1}^n \delta_j^{(d),i} o_{\ell,j}^{(d-1),i} \end{cases}$$

où :

$$\delta_j^{(d),i} = \left( \sum_{m=1}^{R_{d+1}} \delta_m^{(d+1),i} \omega_{j,m}^{(d+1)} \right) o_j^{(d),i} \left[ 1 - o_j^{(d),i} \right] .$$

# Plan

Introduction

Neurone formel

Fonctions d'activation

Perceptron multicouche

Rétro-propagation

Pratique du perceptron multicouche

## Apprentissage en ligne et par lot I

- ▶ L'apprentissage par lot (*full batch learning*) considère l'ensemble des observations et corrige les poids (une fois par époque) à partir du risque empirique calculé sur tout l'échantillon.
- ▶ L'apprentissage en ligne (*online learning*) considère les observations l'une après l'autre et corrige les poids ( $n$  fois par époque) à partir du risque empirique calculé sur l'observation incorporée.

## Apprentissage en ligne et par lot II

- ▶ L'apprentissage par lot est plus **stable** que l'apprentissage en ligne mais nécessite de charger **toutes les données en mémoire**.
- ▶ Pour pallier en partie les instabilités de l'**apprentissage en ligne**, on présente les observations dans un **ordre aléatoire** à chaque étape (itération).
- ▶ Un compromis, utile face à de gros volumes de données, est l'**apprentissage par mini-lot** (*mini-batch learning*), dans lequel la taille de ces mini-lots est fixée par l'utilisateur.

## Apprentissage en ligne et par lot III

- ▶ Une **époque** correspond à un passage avant et arrière de **tout l'échantillon** à travers le réseau de neurones.
- ▶ Le nombre d'**itérations** est le nombre de lots nécessaires à une époque :
  - ▶ 1 pour l'apprentissage par lot,
  - ▶  $n$  pour l'apprentissage en ligne,
  - ▶  $L$  pour l'apprentissage par mini-lot, où  $L$  désigne le nombre de mini-lots (contenant approximativement  $\lceil \frac{n}{L} \rceil$  observations).

## Initialisation des poids et des biais

- ▶ Pour débuter la propagation avant, il faut disposer de **valeurs initiales** pour les **poids** et les **biais**.
- ▶ On les initialise à l'aide d'un **tirage aléatoire**, à partir par exemple d'une loi :
  - ▶  $\mathcal{U}(-[\frac{1}{2}, \frac{1}{2}])$ ,
  - ▶  $\mathcal{N}(0, 1)$ ,
  - ▶  $\mathcal{N}\left(0, \frac{1}{R_{d-1}}\right)$  pour les neurones de la couche  $d \in \{2, \dots, D\}$ .

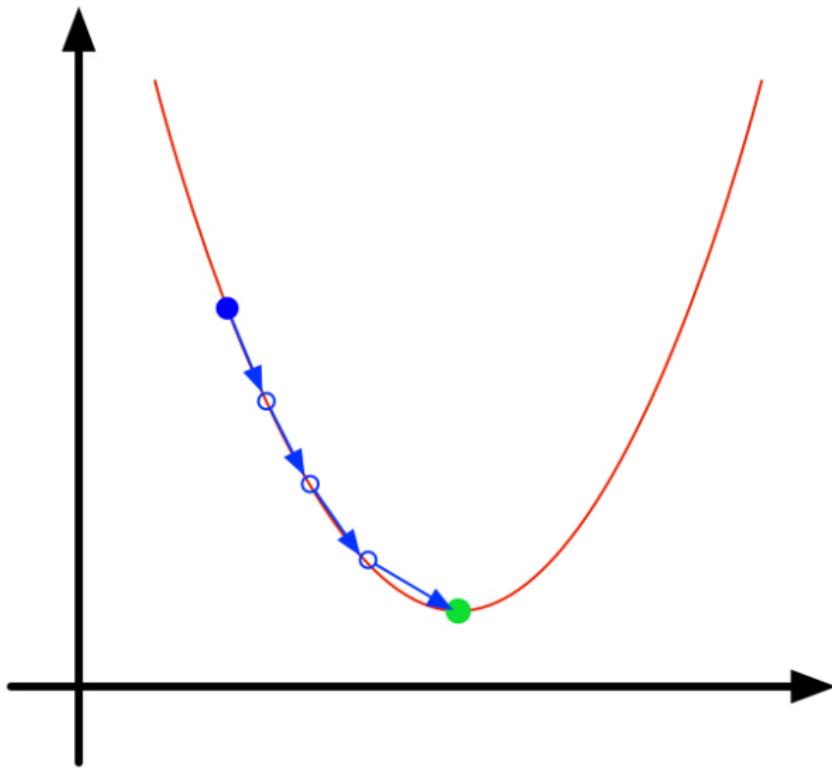
## Normalisation des données

- ▶ Pour la plupart des fonctions d'activation, leur gradient est quasiment nul pour de faibles ou grandes valeurs des entrées du neurone, ce qui conduit au mieux à un apprentissage très lent, au pire...  
On parle alors de **saturation** des neurones.
- ▶ On effectue donc une **normalisation des covariables** sur l'intervalle  $[-1, 1]$  via :
  - ▶ un centrage et une réduction,
  - ▶ une normalisation min-max.
- ▶ La normalisation sur les données de validation et de test doit être identique à celle effectuée sur les données d'apprentissage (avec les mêmes paramètres).
- ▶ Dans le cas de **classification supervisée**, on préfère également considérer comme valeurs de sortie  $\{0.05, 0.95\}$  plutôt que  $\{0, 1\}$ , toujours dans l'idée de pallier ce problème de saturation.

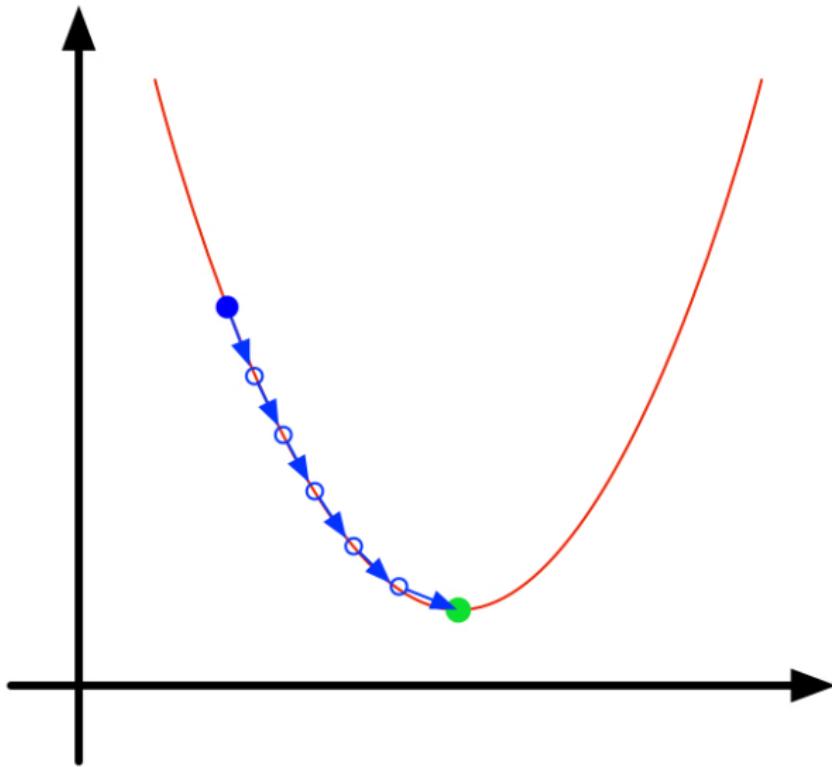
## Quelques problèmes liés à l'optimisation

- ▶ Le risque empirique considéré dans un perceptron multi-couche admet de nombreux **minima locaux** dans lesquels il est aisé de « tomber » si on n'y prête pas attention.
- ▶ On observe des phénomènes de **disparition du gradient** (*vanishing gradient*), ou à l'inverse, d'**explosion du gradient** (*exploding gradient*), dans les couches basses du réseau de neurones.
- ▶ Afin de pallier certains de ces problèmes, il est possible :
  - ▶ de choisir une **fonction d'activation adéquate** (ex : ReLU),
  - ▶ d'adopter un **taux d'apprentissage adaptatif**,
  - ▶ d'ajouter un terme d'inertie ou **momentum** dans la correction des poids : à chaque itération on modifie les poids en tenant compte en sus des changements de poids effectués à l'itération précédente.

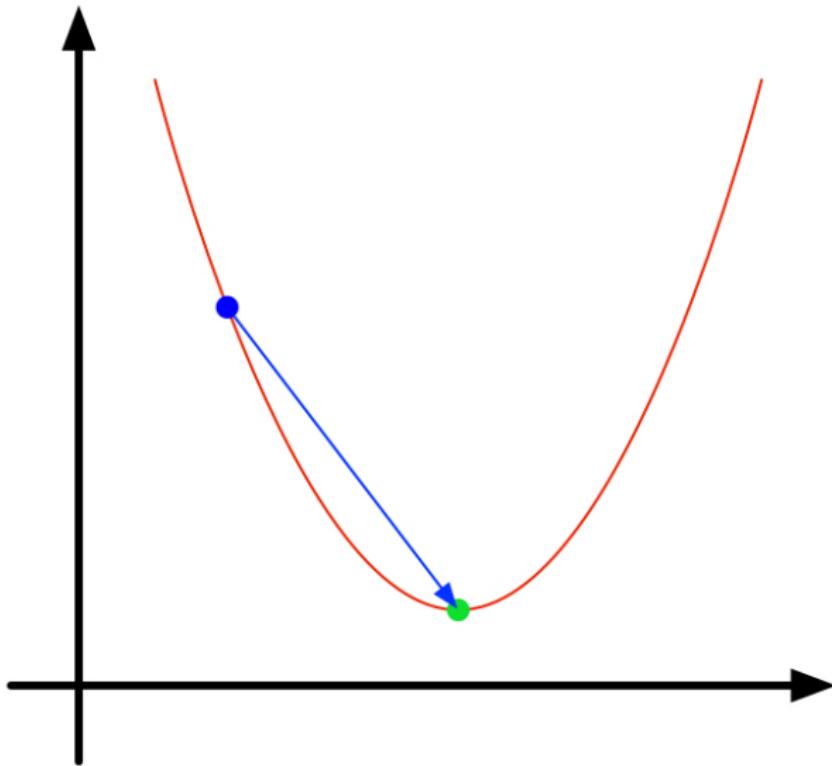
## Taux d'apprentissage



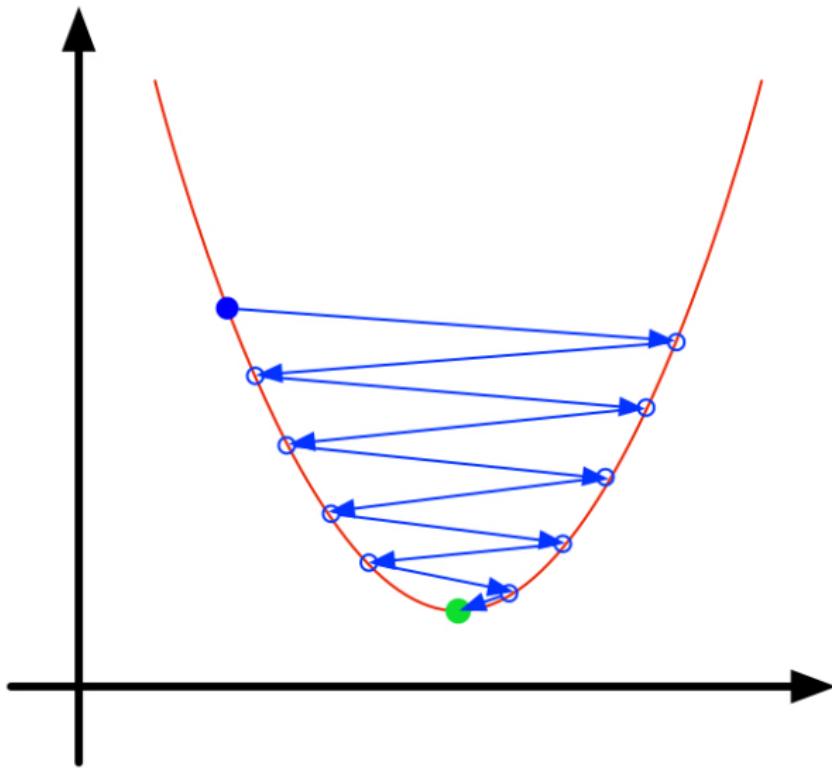
## Taux d'apprentissage



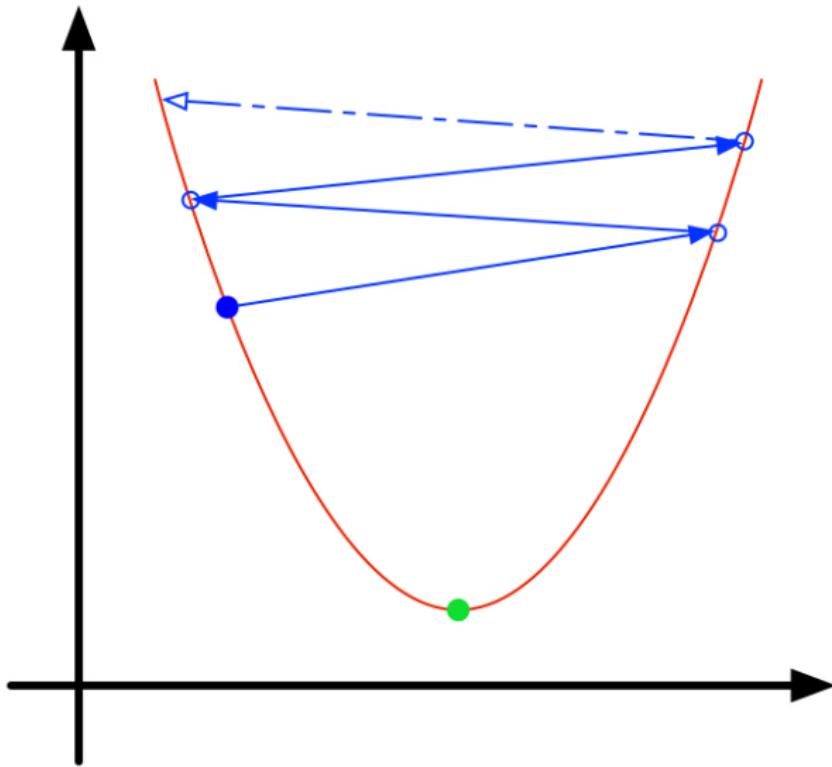
## Taux d'apprentissage



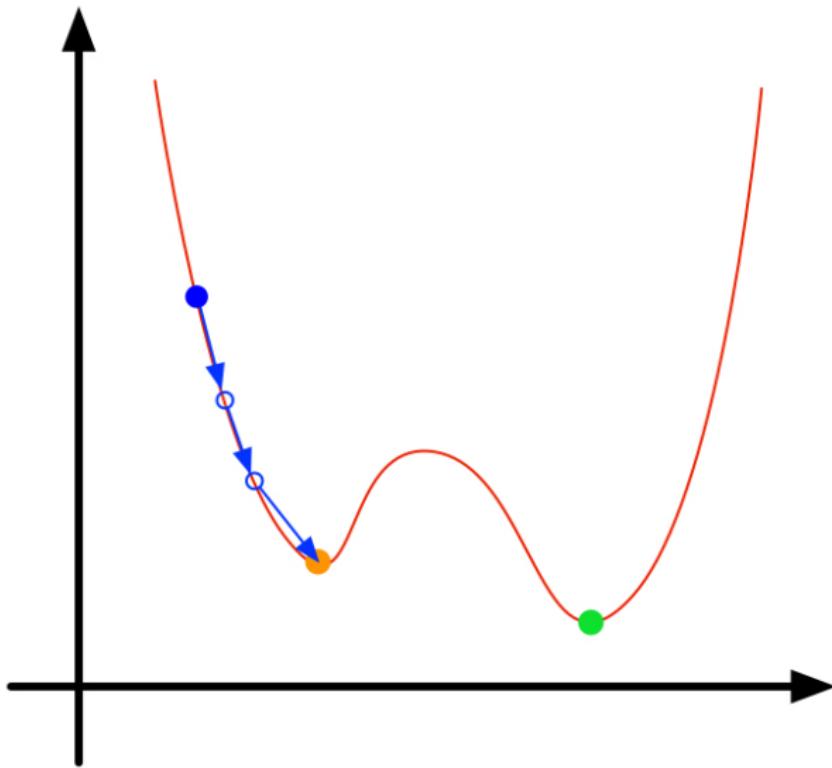
## Taux d'apprentissage



## Taux d'apprentissage

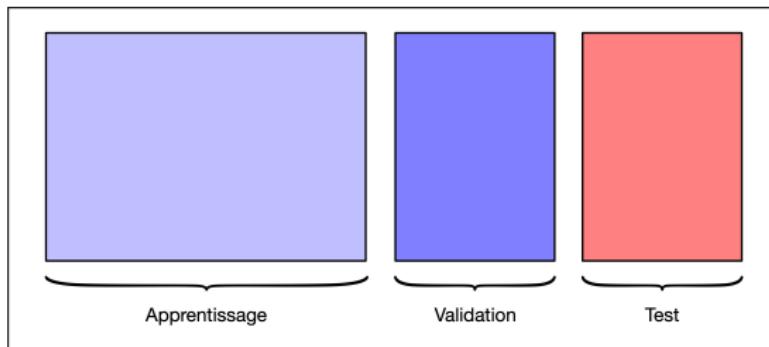


## Taux d'apprentissage



## Eviter le sur-apprentissage

- ▶ Afin de limiter le risque de **sur-apprentissage**, il faut disposer d'un échantillon de validation et contrôler :
  - ▶ le **nombre de neurones** (et de couches),
  - ▶ le **nombre d'époques**.



- ▶ Il est possible d'utiliser les techniques suivantes :
  - ▶ **Dégradation des poids** (*weight decay*) : régularisation du risque empirique avec une pénalité ridge ( $\lambda \sum_{\ell} \omega_{\ell}^2$ ).
  - ▶ **Inhibition de neurones** (*drop out*).

## Cas de la régression

- ▶ La structure recommandée est un réseau à **1 couche cachée**.
- ▶ On considère la **fonction d'activation linéaire** pour la **couche de sortie**.
- ▶ On considère la **fonction d'activation sigmoïde** pour la **couche cachée**.

## Références |

- Bishop, C. M. 1995, *Neural networks for pattern recognition*, Advanced Texts in Econometrics, Oxford University Press.
- Bishop, C. M. 2011, *Pattern recognition and machine learning*, Information Science and Statistics, Springer.
- Boyd, S. et L. Vandenberghe. 2003, *Convex optimization*, Cambridge University Press.
- Cybenko, G. 1989, «Approximation by superpositions of a sigmoidal function», *Mathematics of Control, Signals and Systems*, vol. 2, n° 4, p. 303–314.
- Goodfellow, I., Y. Bengio et A. Courville. 2016, *Deep learning*, Adaptive Computation and Machine Learning, MIT Press.
- Hornik, K. 1991, «Approximation capabilities of multilayer feedforward networks», *Neural Networks*, vol. 4, n° 2, p. 251–257.

## Références II

- McCulloch, W. et W. Pitts. 1943, «A logical calculus of ideas immanent in nervous activity», *Bulletin of Mathematical Biophysics*, vol. 5, p. 115–133.
- Rosenblatt, F. 1958, «The perceptron : probabilistic model for information storage and organization in the brain», *Psychological Review*, vol. 65, n° 6, p. 386–408.
- Rumelhart, D. E., G. E. Hinton et R. J. Williams. 1986, «Learning representations by back-propagating errors», *Nature*, vol. 323, p. 533–536.