

FASCICULE DE COURS

# DEEP LEARNING

2<sup>ème</sup> année Génie Informatique

Semestre 2 ♣





**DR. TAOUIK BEN ABDALLAH**



MAÎTRE-ASSISTANT À LA FS-BIZERTE  
UNIVERSITÉ DE CARTHAGE



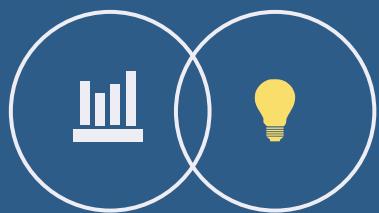
INSTRUCTEUR HUAWEI EN INTELLIGENCE ARTIFICIELLE



taoufik.benabdallah@iit.ens.tn



# | OBJECTIFS DU COURS



# Objectifs du cours

-  Présenter le domaine de l'apprentissage profond
-  Maîtriser les différentes étapes relatives à la définition, l'apprentissage, l'utilisation, et l'optimisation des réseaux de neurones profonds
-  Décrire les problèmes courants de l'apprentissage profond
-  Détailler quelques architectures prédéfinies de réseaux de neurones profond (CNN et RNN)
-  Maîtriser le concept de l'apprentissage profond par transfert
-  Introduire les réseaux de neurones génératifs

# Organisation du cours △



RÉPARTITION APPROXIMATIVE DU CHARGE HORAIRES=35H



COURS+TD	TRAVAUX PRATIQUES
17h	18h



EVALUATION

20%

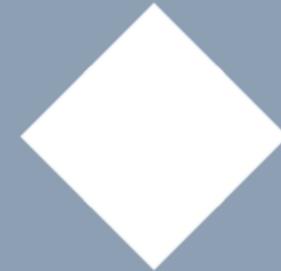
DS

25%

PROJET

55%

EXAMEN



# Plan du cours



## ▼ Introduction générale

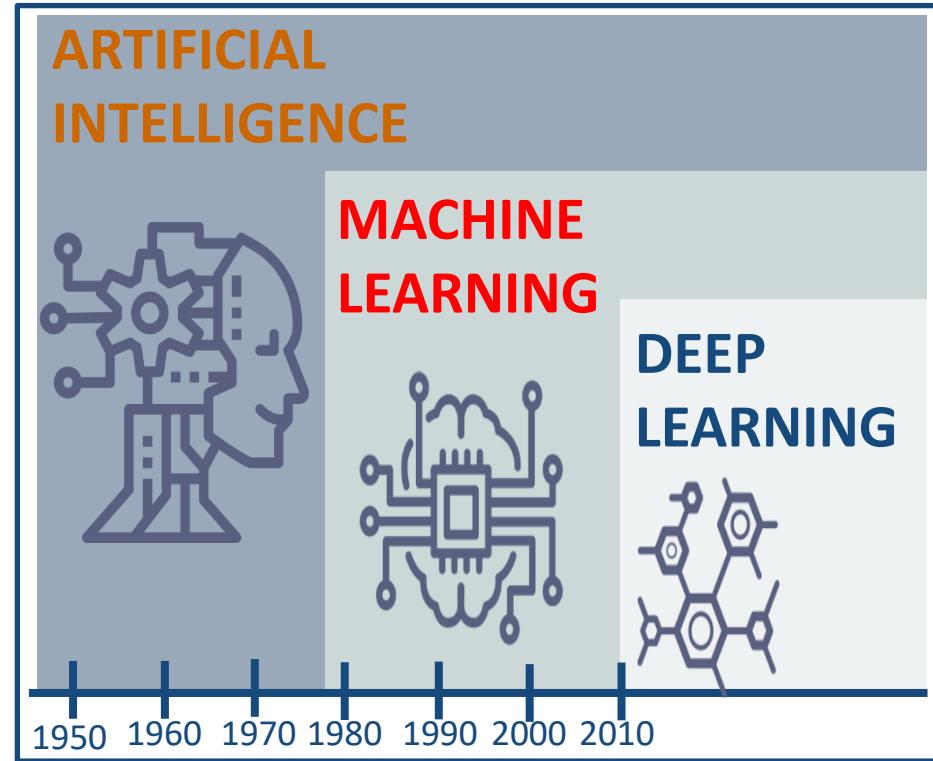
- 1- Apprentissage des réseaux de neurones profonds
- 2- Réseaux de neurones convolutifs : Classification des images
- 3- Solutions au problème de sur-apprentissage
- 4- Architectures CNN & Apprentissage par transfert
- 5- Réseaux de neurones récurrents : Traitement du langage naturel
- 6- Réseaux de neurones génératifs

# INTRODUCTION GÉNÉRALE

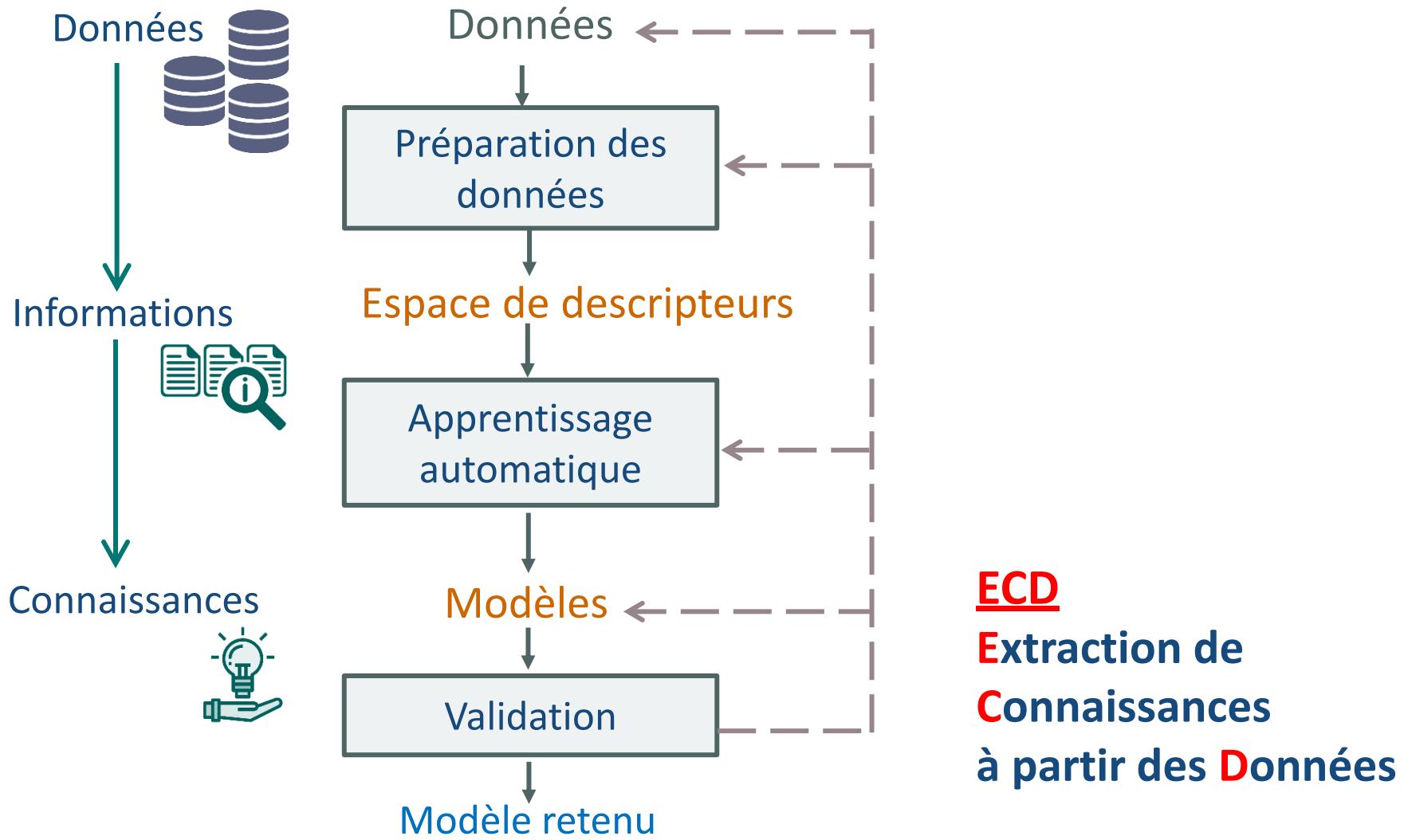
بَصَرٌ بِمِنْ عَالَمٍ



- **Artificial Intelligence (AI)** : vise à créer des systèmes informatiques capables de simuler des capacités humaines
- **Machine Learning (ML)** : permet aux ordinateurs d'apprendre à partir de données et de s'améliorer sans programmation explicite
- **Deep Learning (DL)** : se concentre sur l'entraînement de réseaux de neurones artificiels profonds



# Processus ECD **Trois phases majeurs!**



# Types d'apprentissage automatique



Apprentissage  
supervisé



Apprentissage  
non-supervisé



Apprentissage  
semi-supervisé



Apprentissage par  
renforcement

# Rappel/ Apprentissage supervisé

- Produire automatiquement des règles à partir d'une base de données d'apprentissage étiquetées
- Prédire la classe de nouvelles données observées

Observation	A1	A2	A3	Classe
1	1	0.001	12.134	Chat
2	0	0.023	12.001	Chien
3	1	0.456	9.0073	Chat
4	0	0.114	4.0017	Chat
5	0	0.555	19.771	Chien
6	1	0.551	6.9912	Chat
7	0	0.020	14.478	?
8	1	0.003	8.1238	?

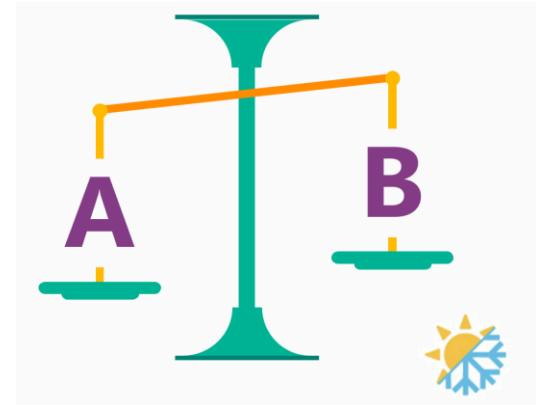
Training set

Test set

# Rappel/ Apprentissage supervisé

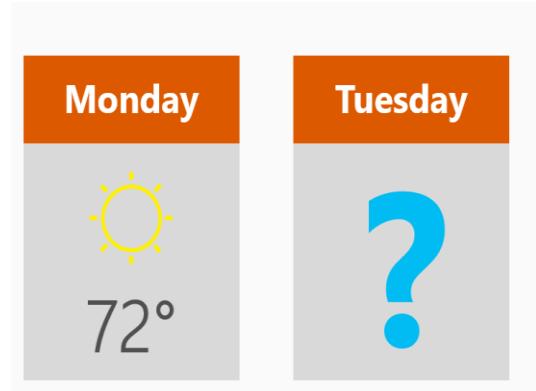
## Classification : Prédire des valeurs discrets

- Sera-t-il froid ou chaud demain? **Froid/ chaud**



## Régression : Prédire des valeurs continues

- Quelle est la température demain?



# Rappel/ Apprentissage non-supervisé

- Vise à découvrir des structures intrinsèques, des motifs ou des relations inhérentes aux données sans avoir d'informations préalables sur les sorties attendues
- Les classes sont inconnues → pas de cible (pas d'étiquette)

Observation	A1	A2	...	Classe
1	1	12.3	...	
2	0	14	...	
3	0	12	...	
4	0	11	...	
5	1	10.2	...	
6	1	22.1	...	
⋮	⋮	⋮	⋮	⋮

# Rappel/ Apprentissage non-supervisé **Utilisations**

## **Regroupement (Clustering)**

- Organisation des données en groupes (clusters)

## **Réduction de dimensionnalité**

- Sélection des descripteurs les plus discriminants
- Transformation des descripteurs dans un autre espace

## **Détection d'anomalies (Outliers detection)**

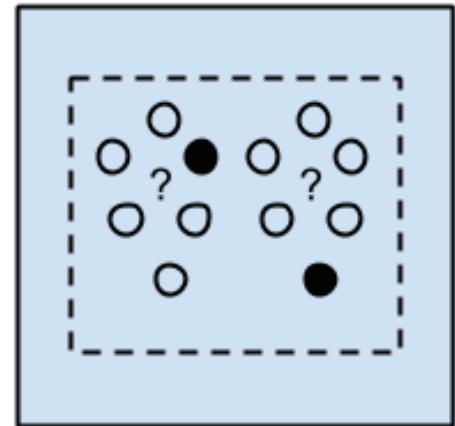
- Identification des observations aberrantes



# Rappel/ Apprentissage semi-supervisé

 Approche hybride qui combine des éléments de l'apprentissage supervisé et non supervisé

- Permet d'améliorer les performances du modèle en exploitant à la fois les informations des données étiquetées et non étiquetées
- Inclut souvent des techniques qui exploitent les relations entre les données étiquetées et non étiquetées, telles que l'auto-encodage, ou l'utilisation de modèles génératifs
- Peut être particulièrement utile dans des scénarios où les données d'entraînement sont limitées

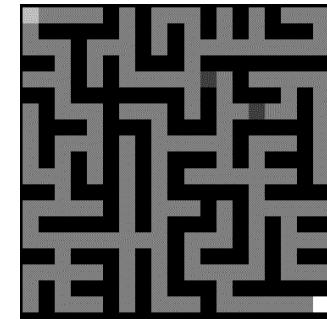


# Rappel/ Apprentissage par renforcement

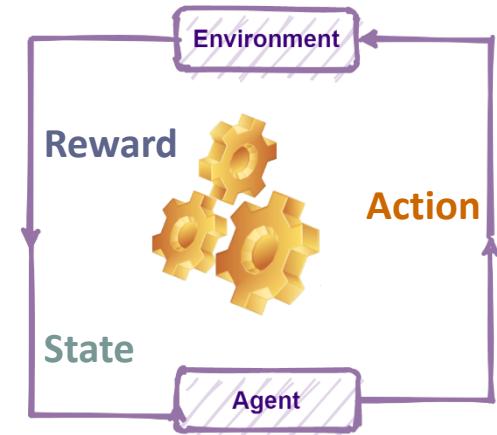
 Apprend à prendre des décisions en explorant l'environnement

 Maximiser une notion de récompense cumulée au fil du temps

 Implique un processus d'interaction dynamique entre l'agent et son environnement



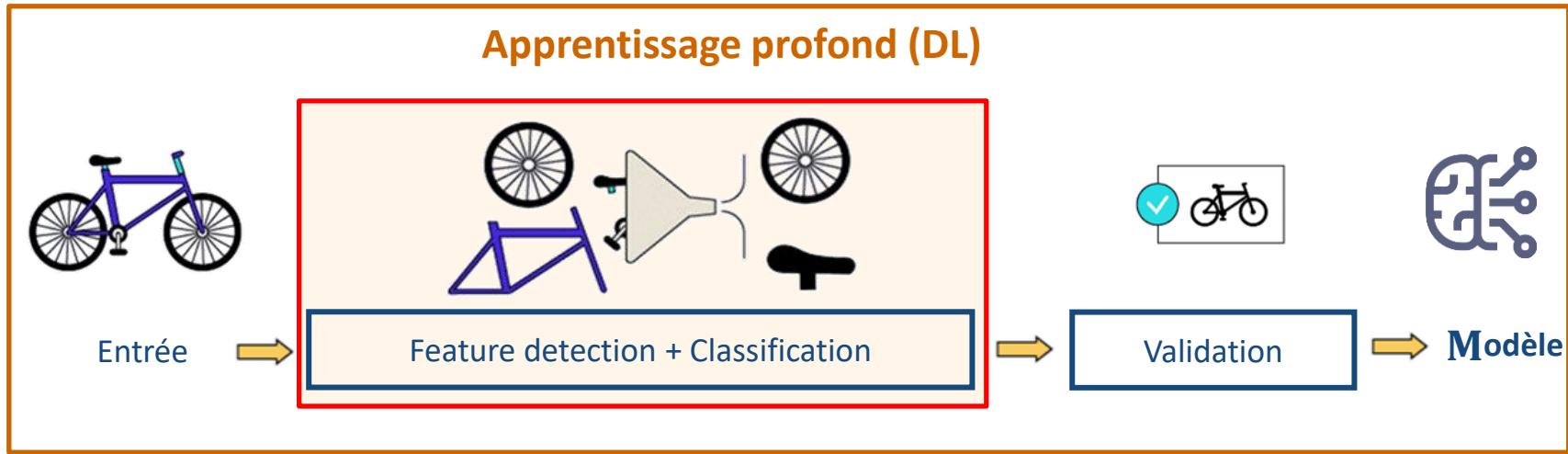
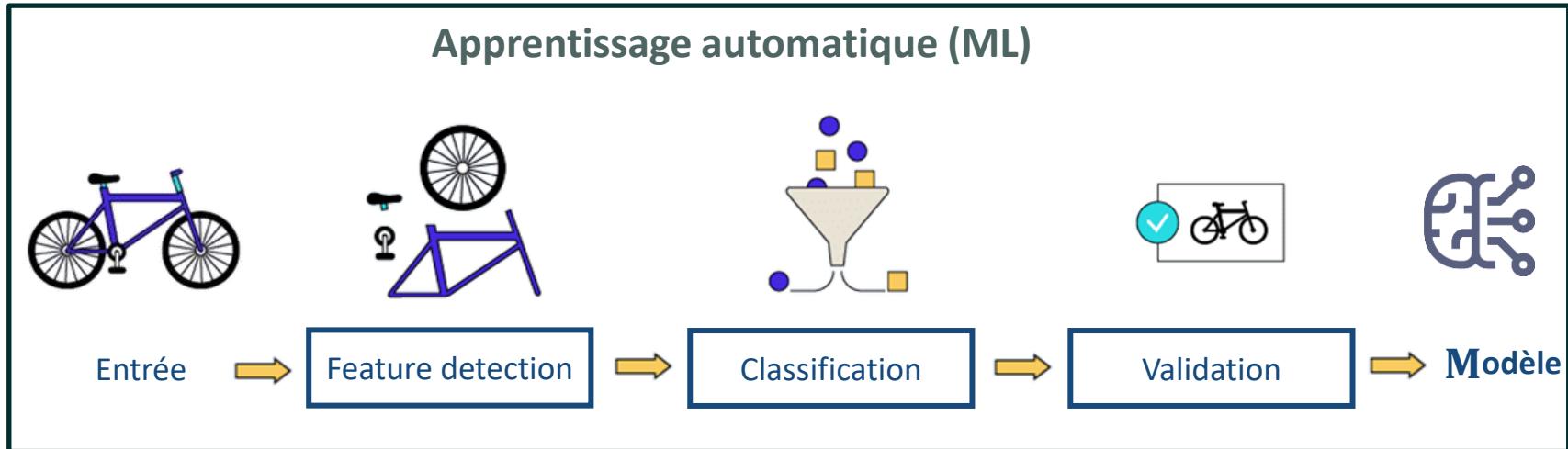
- **State**= représentation des caractéristiques actuelles de l'environnement
- **Action**= choix que l'agent peut faire à chaque étape (décision prise)
- **Reward**= valeur numérique qui mesure la qualité de la décision prise par l'agent



**DE MACHINE LEARNING VERS LE DEEP LEARNING**

**من التعلم الآلي إلى التعلم العميق**

# Apprentissage automatique vs. Apprentissage profond 1/2



# Apprentissage automatique vs. Apprentissage profond 2/2

	Apprentissage automatique (ML)	Apprentissage profond (DL)
Matériel (puissance de calcul)	CPU (Central Processing Unit)	GPU (Graphics Processing Unit))
Données	Des milliers d'observations	Big Data : des millions d'observation
Descripteurs	Handcrafted (bas niveau)	High-level (haut niveau)
Modélisation	Niveau par niveau	Bout en bout
Domaines d'applications	Classification & régression sur des données tabulaires	Classification & régression sur des données tabulaires, vision par ordinateur, NLP et audio

# Domaines d'application!

- Vision par ordinateur & reconnaissance d'images
- Traitement du langage naturel (NLP)
- Reconnaissance vocale



# Quelques applications ✓



## Vision par ordinateur & reconnaissance d'images

- Reconnaissance d'objets
- Reconnaissance d'actions
- Reconnaissance des visages et des émotions
- Segmentation d'images
- Ré identification des personnes



## Traitement du langage naturel & Reconnaissance vocale

- Traduction automatique
- Systèmes de recommandation intelligents
- Génération automatique de texte
- Chatbot conversationnelle
- Speech recognition



# Succès de Deep Learning



AlphaGo Zero  
2017



Amazon Rekognition  
2019



Google car  
2020



Google Smart Displays  
2021



Apple Live Text  
2021



Open AI DALL-E2  
2022



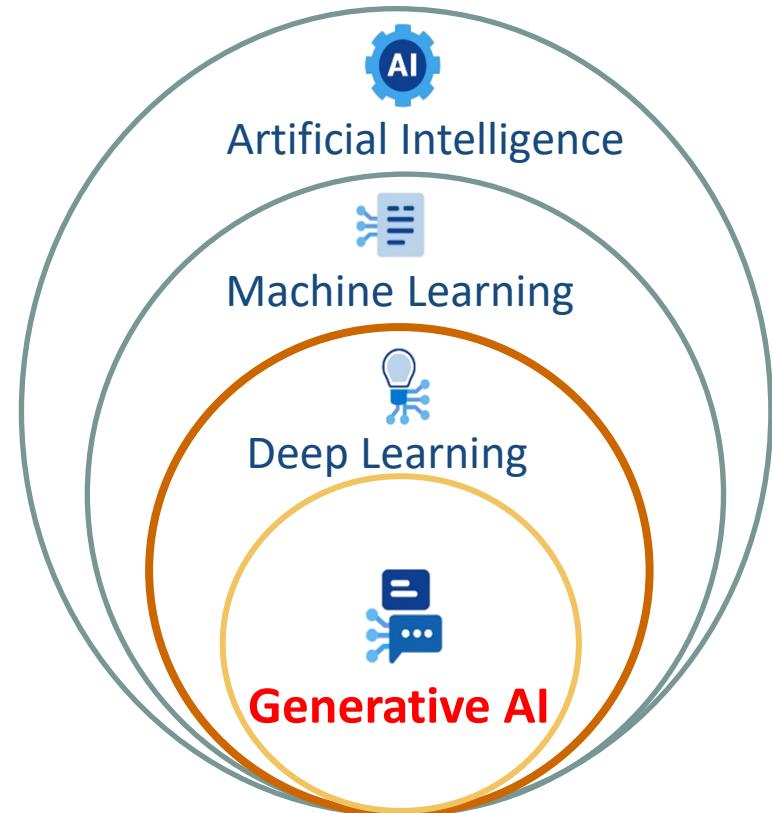
Open AI ChatGPT  
2022



Google Bard  
2023

# IA générative

- Fait référence à des modèles qui sont capables de générer de nouvelles données à partir de données existantes
-  Génération de texte, d'images et d'audio
  - Variational Autoencoders
  - Generative Adversarial Networks (GAN)
  - Les modèles basés sur les Transformers (LMM, BERT, GPT, ViT, etc.)



# Frameworks de DL



Caffe

The mxnet logo, which features a blue circle containing a white lowercase 'm' followed by the word "xnet" in a blue sans-serif font.

# Quiz

1. L'apprentissage profond intègre à la fois la détection des descripteurs de haut niveau et la classification. Il nécessite une quantité de données massives et un GPU (seule réponse)

Vrai

Faux

2. TensorFlow 2 . x (plusieurs réponses)

Intègre Keras, ce qui améliore considérablement la convivialité

Est un Framework Open Source

Est un Framework flexible

Est inventé par Facebook

3. Parmi les éléments suivants, lequel ne fait pas partie du Framework de développement de Deep Learning (seule réponse)

Caffe

Keras

sklit-learn

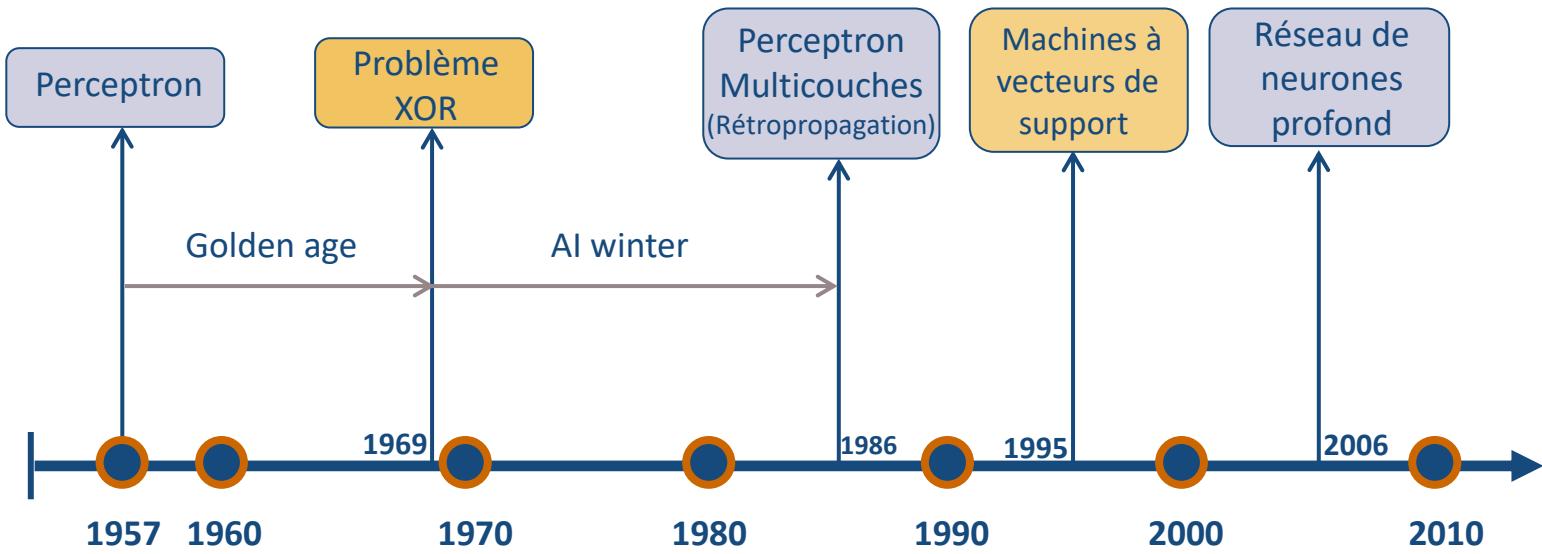
MXNet



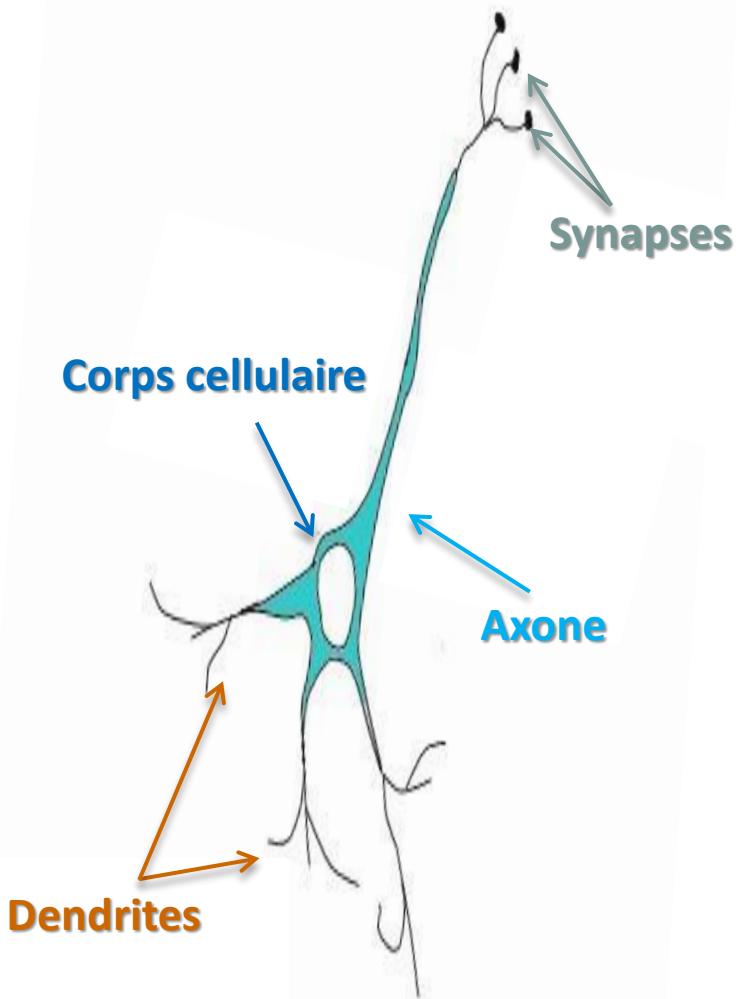
# CHAPITRE 1

## APPRENTISSAGE DES RÉSEAUX DE NEURONES PROFONDS

# Historique du développement des réseaux de neurones



# Neurone biologique

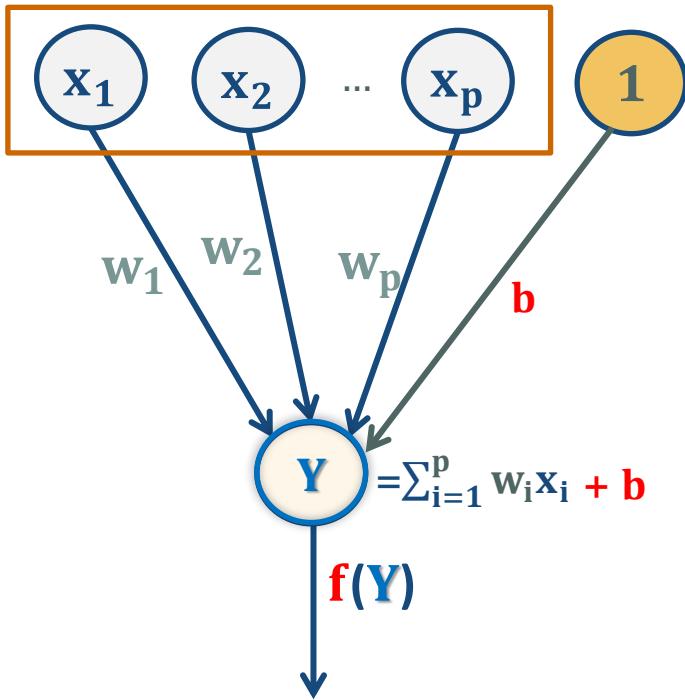


- Transient l'information venue de l'extérieur vers le corps cellulaire
- **=centre de contrôle** Fait la somme des informations qui lui parviennent
- Traite l'information et renvoie le résultat sous forme de signaux électriques, du corps cellulaire à l'entrée des autres neurones
- Permettent aux neurones de communiquer entre eux

$\simeq 10^{11}$  neurones dans le cerveau humain  
 $\simeq 10^4$  connexions (synapses + axones) / neurone

# PERCEPTRON SIMPLE

# Neurone Artificiel Perceptron simple 1/2



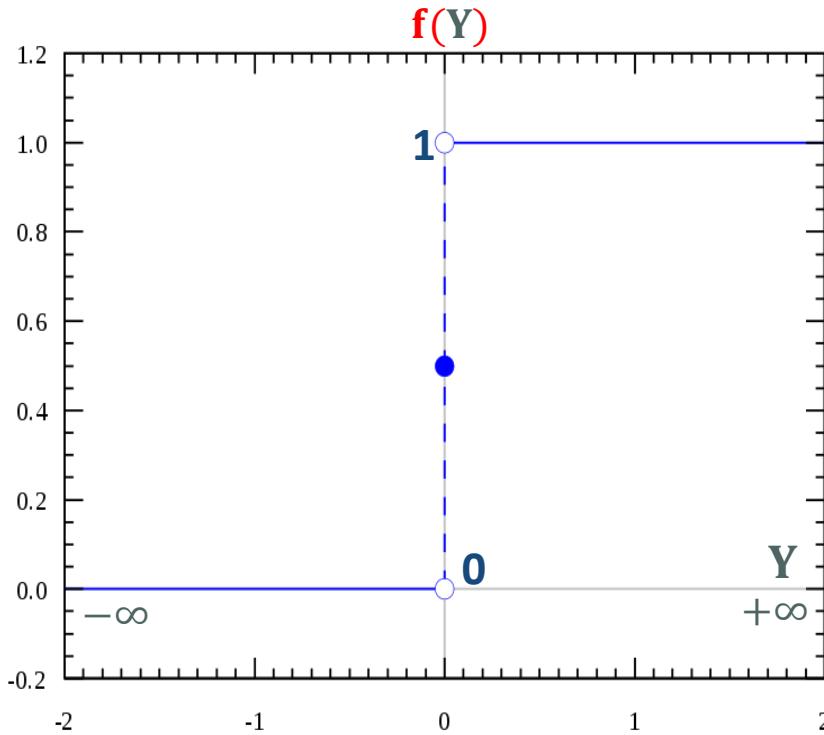
$x_1$	$x_2$	...	$x_p$	Classe_d

- **Dendrites** : Série d'entrées  $x_1 \dots x_p$
- **Synapses** : coefficients synaptiques  $w_1 \dots w_p$   
=> Chaque entrée est associée à un poids
- **Corps cellulaire** : Traitement (calcul de Y)
- **Axone** : Fonction d'activation f  
=> Résultat de neurones = sortie
- **Bias b =Régulateur**  
=> Aide à s'adapter au mieux aux données

# Neurone Artificiel **Perceptron simple** 1/2



## Fonction d'activation = **Heaviside (Marche)**



$$Y = \sum_{i=1}^p w_i x_i + b$$

$$f(Y) = 1 \text{ si } Y \geq 0$$

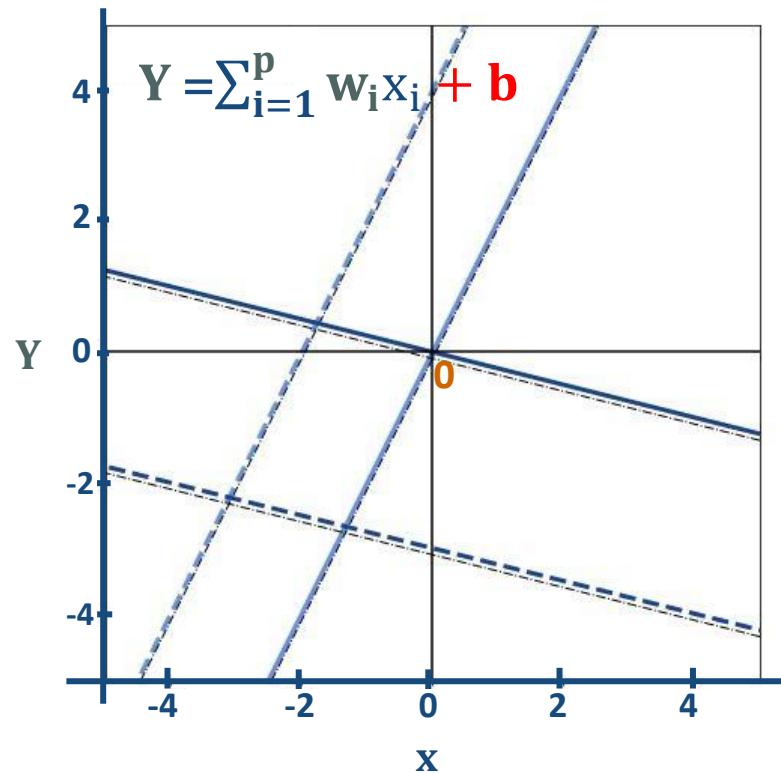
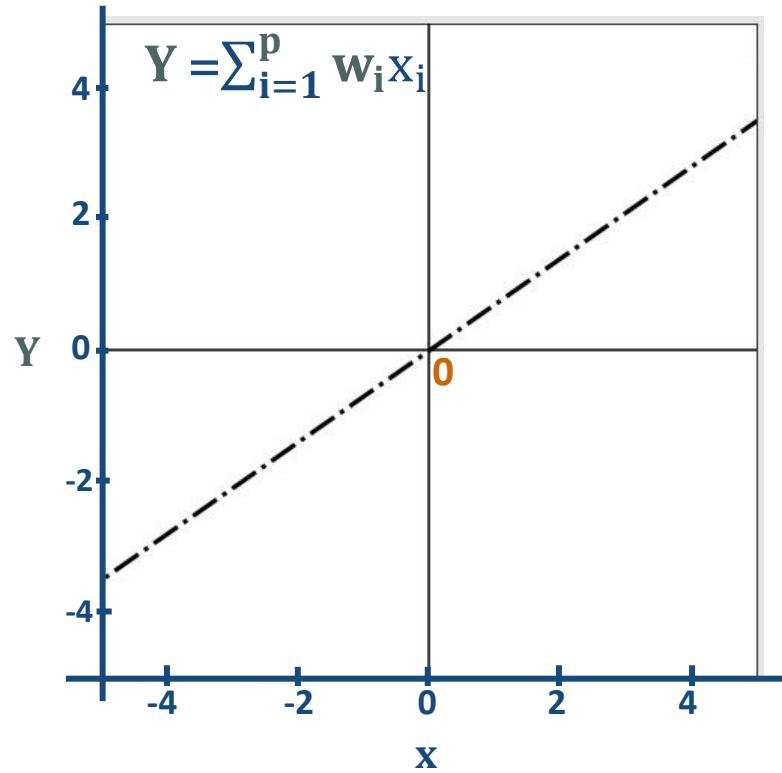
$$f(Y) = 0 \text{ sinon}$$

Deux valeurs : **0 ou 1**

Classification **binaire**

Problème linéairement séparable!

# Bias !



# Loi d'apprentissage = Ajustement des poids

## Loi de Widrow-Hoff (Delta rule)

$$w_i = w_i + \eta(d - \text{sortie})x_i$$

Erreur (loss)

$w_i$  =  $w_i + \eta(d - \text{sortie})x_i$

Classe désirée

Classe prédite =  $f(Y)$

Pas d'apprentissage  
=Learning rate  $\in ]0..1]$

Si  $\text{sortie} = d$ ,  $w_i$  sont inchangables

Si  $\text{sortie} < d$ ,  $w_i$  va diminuer

Si  $\text{sortie} > d$ ,  $w_i$  va augmenter

## Exemple introductif (1/4)

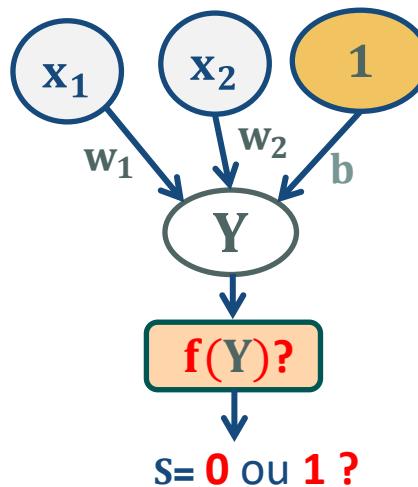
Étant donné le jeu de donnée d'apprentissage or\_logique

Variable à prédire (cible) :  $d \rightarrow$  classification binaire (0/ 1)

Variables explicatives :  $x_1$ , et  $x_2$

	$x_1$	$x_2$	$d$
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

\* Représenter l'architecture du perceptron associé à or\_logique?



## Exemple introductif (2/4)

💡 Objectif : ajuster les poids  $w_1$ ,  $w_2$ , et  $b$  !

❓ Etapes à suivre !

1- Initialiser aléatoirement les poids et Bias  $w_1$ ,  $w_2$ , et  $b$

$$w_1 = 1, w_2 = -1, b = 0$$

	$w_1$	$w_2$	$b$	$x_1$	$x_2$	$d$	$Y$	$f(Y)$	$w_1$	$w_2$	$b$
0	-	-	-	-	-	-	-	-	1	-1	0

2- Faire passer la première observation ( $x_1 = 0$ ;  $x_2 = 0$ ;  $d=0$ ) sur le réseau et mettre à jour  $w_1$ ,  $w_2$ , et  $b$  sachant que  $\eta=1$

	$w_1$	$w_2$	$b$	$x_1$	$x_2$	$d$	$Y$	$f(Y)$	$w_1$	$w_2$	$b$
1	1	-1	0	0	0	0	0	1	1	-1	-1

$$Y = \sum_{i=1}^2 w_i x_i + b = w_1 x_1 + w_2 x_2 + b = (1 \times 0) + (-1 \times 0) + 0 = 0$$

$f(Y)=1$  puisque  $Y \geq 0$

$$w_1 = w_1 + \eta(d - \text{sortie})x_1 = w_1 + \eta(d - f(Y))x_1 = 1 + 1(0 - 1)0 = 1$$

:

## Exemple introductif (3/4)

3- Faire passer les trois observations restantes une par une sur le réseau et mettre à jour  $w_1$ ,  $w_2$ , et  $b$  sachant que  $\eta=1$

	$w_1$	$w_2$	$b$	$x_1$	$x_2$	$d$	$Y$	$f(Y)$	$w_1$	$w_2$	$b$
2	1	-1	-1	0	1	1	-2	0	1	0	0
3	1	0	0	1	0	1	1	1	1	0	0
4	1	0	0	1	1	1	1	1	1	0	0

4-

Répéter

Repasser les observations une par une sur le réseau et mettre à jour  $w_1$ ,  $w_2$ , et  $b$  sachant que  $\eta=1$

Jusqu'à aucune correction n'est effectuée en passant toutes les observations (convergence)

:

## Exemple introductif (4/4)

⋮

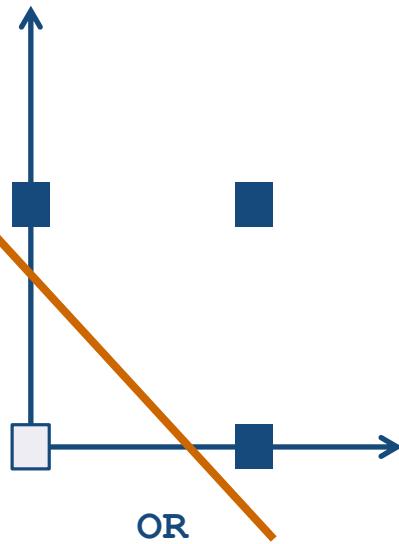
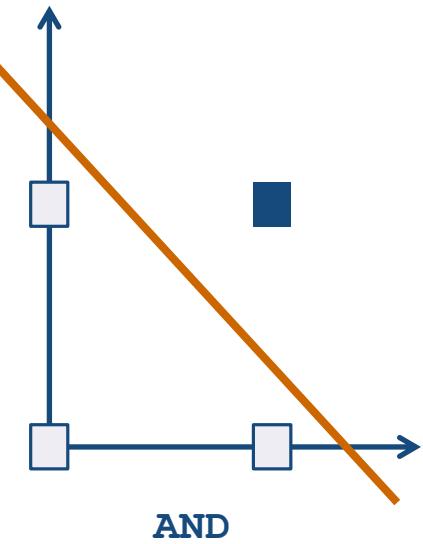
<del>w<sub>1</sub></del>	w <sub>1</sub>	w <sub>2</sub>	b	x <sub>1</sub>	x <sub>2</sub>	d	Y	f(Y)	w <sub>1</sub>	w <sub>2</sub>	b
0	-	-	-	-	-	-	-	-	1	-1	0
1	1	-1	0	0	0	0	0	1	1	-1	-1
2	1	-1	-1	0	1	1	-2	0	1	0	0
3	1	0	0	1	0	1	1	1	1	0	0
4	1	0	0	1	1	1	1	1	1	0	0
5	1	0	0	0	0	0	0	1	1	0	-1
6	1	0	-1	0	1	1	-1	0	1	1	0
7	1	1	0	1	0	1	1	1	1	1	0
8	1	1	0	1	1	1	2	1	1	1	0
9	1	1	0	0	0	0	0	0	1	1	-1
10	1	1	-1	0	1	1	0	1	1	1	-1
11	1	1	-1	1	0	1	0	1	1	1	-1
12	1	1	-1	1	1	1	1	1	1	1	-1
13	1	1	-1	0	0	0	-1	0	1	1	-1

ÉPOQUE  
(EPOCH)

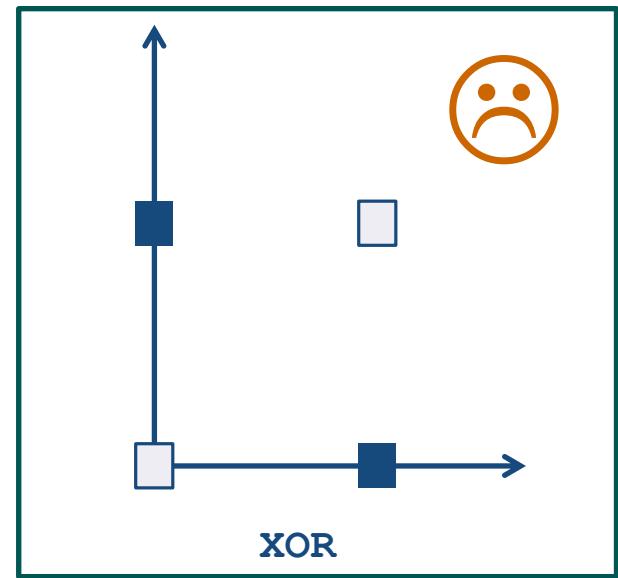
## Conditions d'arrêt

- Les poids sont stables → aucune mise à jour n'effectuée en passant une époque
- L'erreur globale ne diminue plus significativement
- Le taux d'erreur atteint un minimum fixé (seuil)
- Le nombre d'itérations atteint un maximum fixé
- Le nombre d'époques est atteint

# Perceptron & Problème XOR!



Incapable de classer les données non séparables linéairement



# Quiz

1. Le problème XOR peut être résolu en utilisant un perceptron simple  
(seule réponse)

- Vrai
- Faux

2. Étant donnée l'observation  $\mathbf{z} (4, 6, 7, 1)$  de 3 trois descripteurs et une classe (0 ou 1), le passage de cette observation sur un perceptron ne change pas les poids sachant que  $\eta = \text{Tous les poids}=1$  (seule réponse)

- Vrai
- Faux

3. Combien il existe de paramètres à apprendre au total à partir d'un perceptron simple entraîné sur un dataset comportant 200 observations et 20 descripteurs ?  
(seule réponse)

- 201
- 200
- 21
- 20

# Activité

Étant donné un jeu de données d'apprentissage de **3** descripteurs numériques **n<sub>1</sub>, n<sub>2</sub>, et n<sub>3</sub>** et une classe **d (oui et non)**

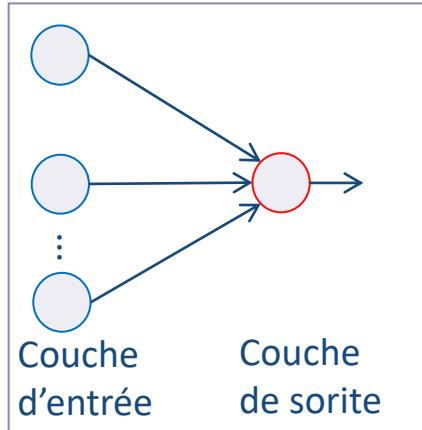
Le passage par **deux époques** (en respectant l'ordre des observations) sur un réseau de neurones de deux couches (entrée et sortie) engendre dans les **10 dernières itérations** les valeurs de poids suivants : **b= -0.3, w<sub>1</sub>=0.2, w<sub>2</sub>= 2w<sub>1</sub>et w<sub>3</sub>= -0.1**

- 1- Calculer la sortie de la 21<sup>ème</sup> itération**
- 2- Calculer les valeurs de poids associées au modèle dans la 21<sup>ème</sup> itération**
- 3- A quelle itération le processus d'ajustement de poids se converge ?**
- 4- Déduire l'accuracy du modèle associé aux données d'apprentissage**

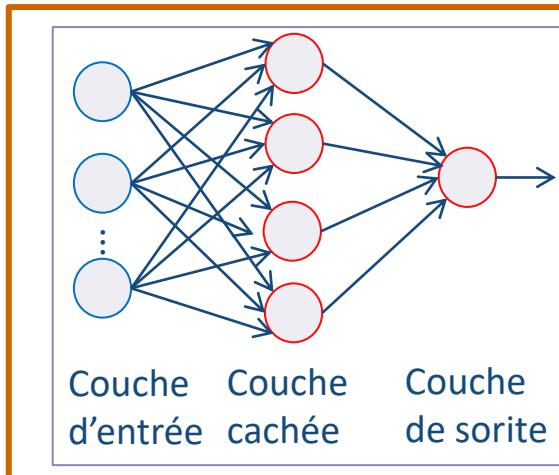
	<b>n<sub>1</sub></b>	<b>n<sub>2</sub></b>	<b>n<sub>3</sub></b>	<b>d</b>
<b>0</b>	2	0.8	1.2	Oui
<b>1</b>	0.7	1.9	0.1	Non
<b>2</b>	1.4	3.3	0.8	Oui
<b>3</b>	2.1	2.5	0.9	Non
<b>4</b>	1.1	2.9	2	Non
<b>5</b>	1.7	3	0.5	Oui
<b>6</b>	1.8	2.9	2	Oui
<b>7</b>	0.4	3	1.5	Non
<b>8</b>	2.1	2.8	1.4	Oui
<b>9</b>	1.6	0.3	2.7	Oui

# PERCEPTRON MULTICOUCHES (FULLY CONNECTED)

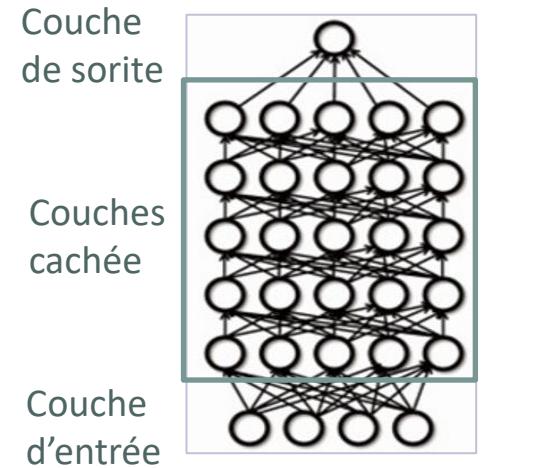
# Perceptron simple vs. multicouches



Perceptron  
simple

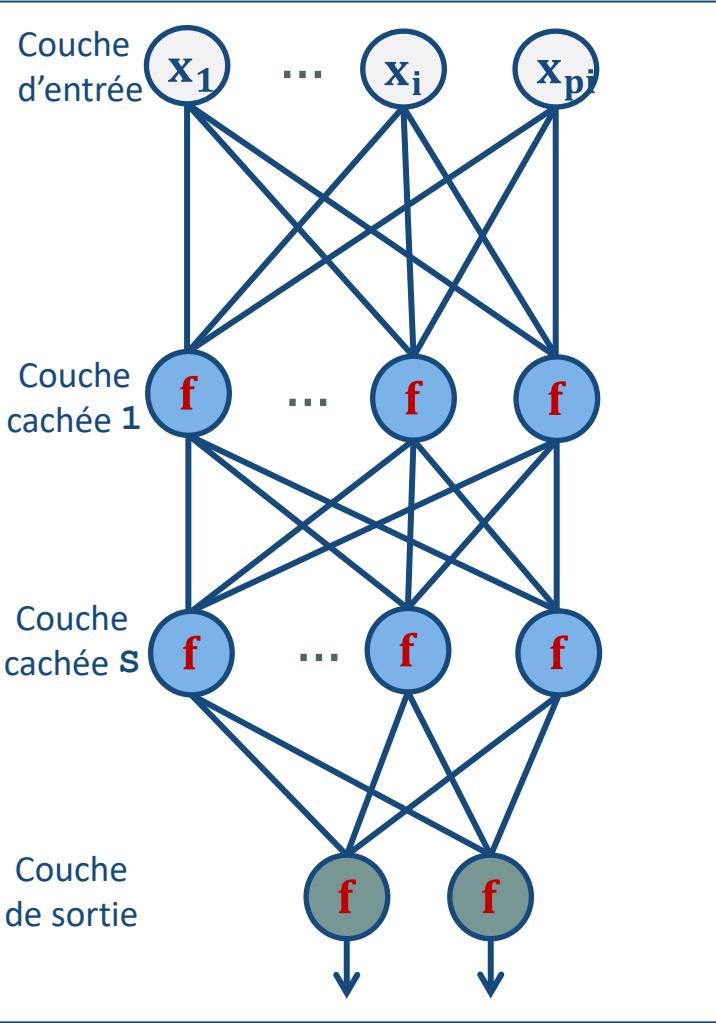


Perceptron multicouches (**MLP**)  
= Fully connected



💡 Le terme "**profond**" dans "apprentissage profond" fait référence au nombre de couches caché du réseau de neurones

# Perceptron multicouches



$x_1$	$x_2$	...	$x_p$	Classe_d
				C1
				C2
				C3

Couche d'entrée =  $p$  Neurones

**Une ou plusieurs couche(s) cachée(s)**

Couche cachée 1 =  $k$  Neurones

Couche cachée S =  $m$  Neurones

**Fully Connected & Séquentiel**

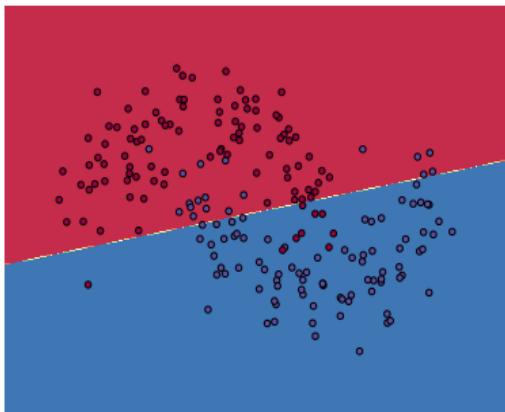
Couche de sortie=  $n$  Neurones= **Nombre de classe**

Fonctions d'activation **f non linéaire**

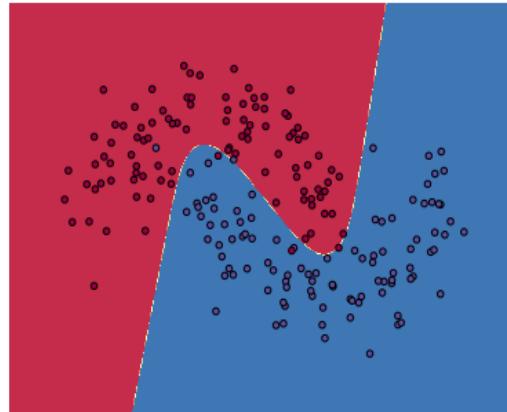
# Impacts des couches cachées!

Pouvoir séparateur : permet de classer des formes **non linéairement séparables** ✓

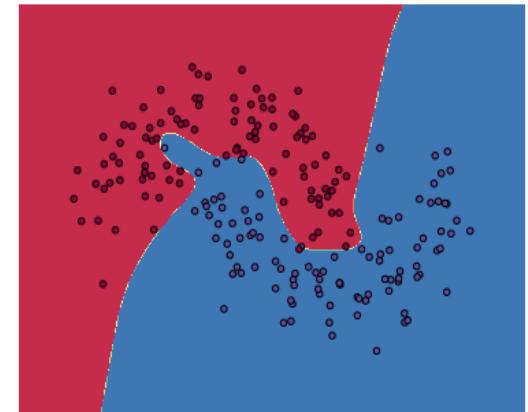
↗ nombre de couches & ↗ nombre de neurones accroît le pouvoir de séparation



0 Couche cachée



3 couches cachées  
chacun de 10 neurones



20 couches cachées  
chacun de 100 neurones

# Fonction d'activation non linéaire **Sigmoïde**

Étant donné  $p$  neurones ( $x_1, \dots, x_i, x_p$ ) dans la couche d'entrée et  $k$  neurones ( $Y_1, \dots, Y_j, Y_k$ ) dans la **couche cachée 1**

$$Y_j = \sum_{i=1}^p w_{ij} x_i + b_j$$

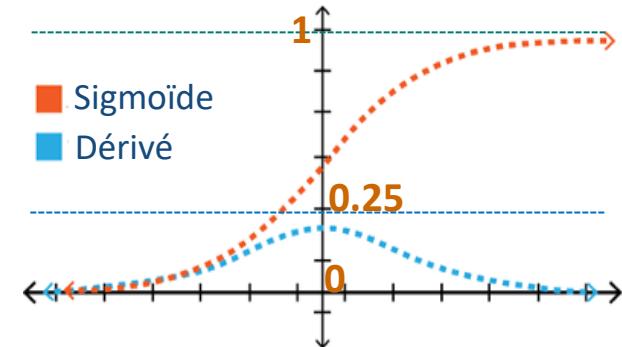
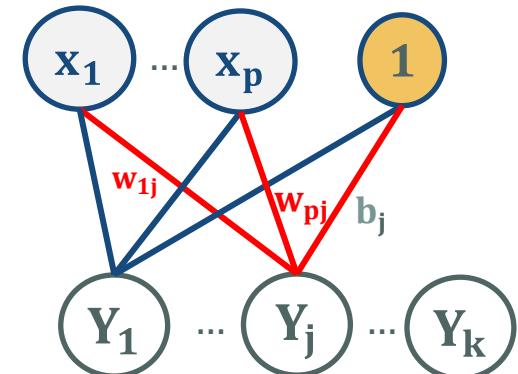
$$\text{Sigmoïde } f(Y_j) = \frac{1}{1+e^{-Y_j}} \quad ]0,1[$$

Dérivé de la fonction Sigmoïde

$$\frac{\partial f(Y_j)}{\partial Y_j} = f'(Y_j) = f(Y_j)(1 - f(Y_j)) \quad ]0,0.25[$$



Classification binaire



# Autres fonctions d'activations **Relu & ses variantes** 1/2

## Rectified Linear Unit (ReLU)

$$f(Y_j) = \max(0, Y_j) = \begin{cases} 0 & \text{si } Y_j < 0 \\ Y_j & \text{si } Y_j \geq 0 \end{cases} \quad [0, +\infty[$$

## Dérivé de la fonction ReLU

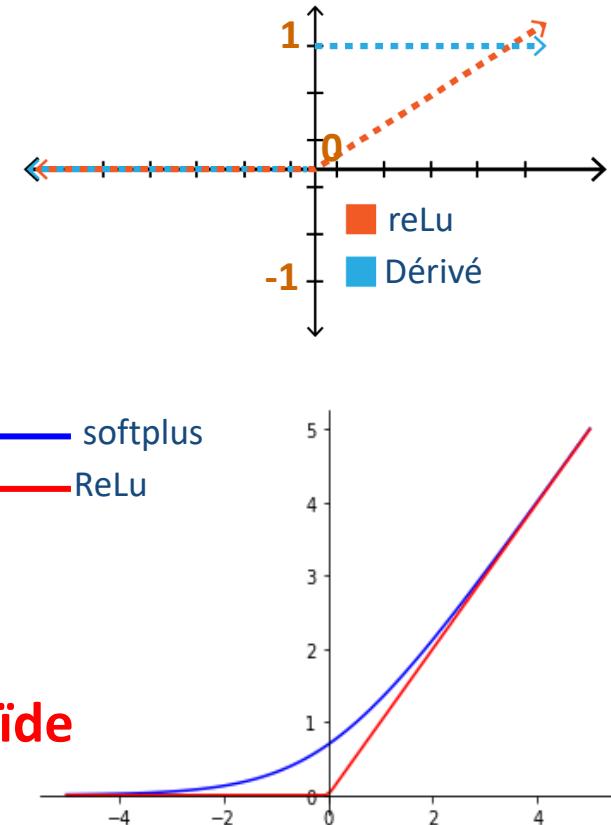
$$\frac{\partial f(Y_j)}{\partial Y_j} = f'(Y_j) = \begin{cases} 0 & \text{si } Y_j < 0 \\ 1 & \text{si } Y_j \geq 0 \end{cases} \quad 0 \text{ ou } 1$$

## softplus

$$f(Y_j) = \ln(e^{Y_j} + 1) \quad ]-\infty, +\infty[$$

## Dérivé de la fonction softplus= fonction Sigmoïde

$$\frac{\partial f(Y_j)}{\partial Y_j} = f'(Y_j) = \frac{1}{1+e^{-Y_j}} \quad ]0, 1[$$

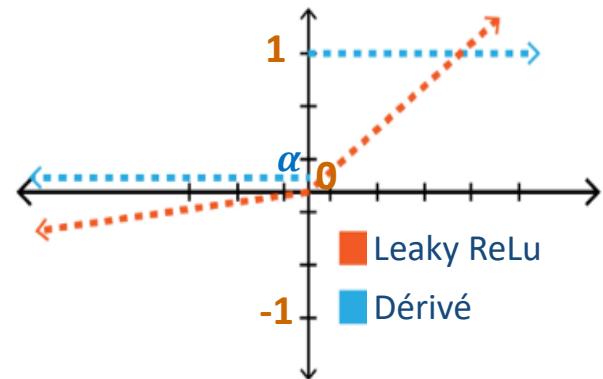


## Autres fonctions d'activations **Relu & ses variantes** 2/2

**Leaky ReLu**  $f(Y_j) = \begin{cases} \alpha Y_j & \text{si } Y_j < 0 \\ Y_j & \text{si } Y_j \geq 0 \end{cases} ]-\infty, +\infty[$

Dérivé de la fonction Leaky ReLu

$$\frac{\partial f(Y_j)}{\partial Y_j} = f'(Y_j) = \begin{cases} \alpha & \text{si } Y_j < 0 \\ 1 & \text{si } Y_j \geq 0 \end{cases} \quad \alpha \text{ ou } 1$$

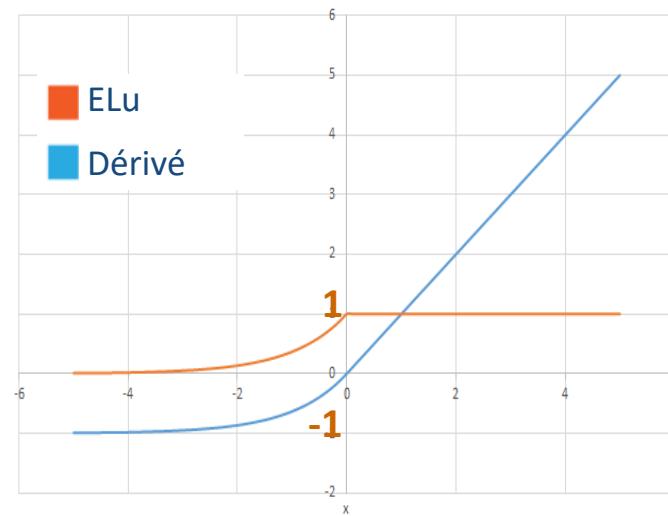


**Exponential Linear Unit (ELu)**

$$f(Y_j) = \begin{cases} \alpha (e^{Y_j} - 1) & \text{si } Y_j < 0 \\ Y_j & \text{si } Y_j \geq 0 \end{cases} ]-\infty, +\infty[$$

Dérivé de la fonction ELu

$$\frac{\partial f(Y_j)}{\partial Y_j} = f'(Y_j) = \begin{cases} f(Y_j) + \alpha & \text{si } Y_j < 0 \\ 1 & \text{si } Y_j \geq 0 \end{cases} ]-\infty, 1]$$



# Autres fonctions d'activations **Softmax**

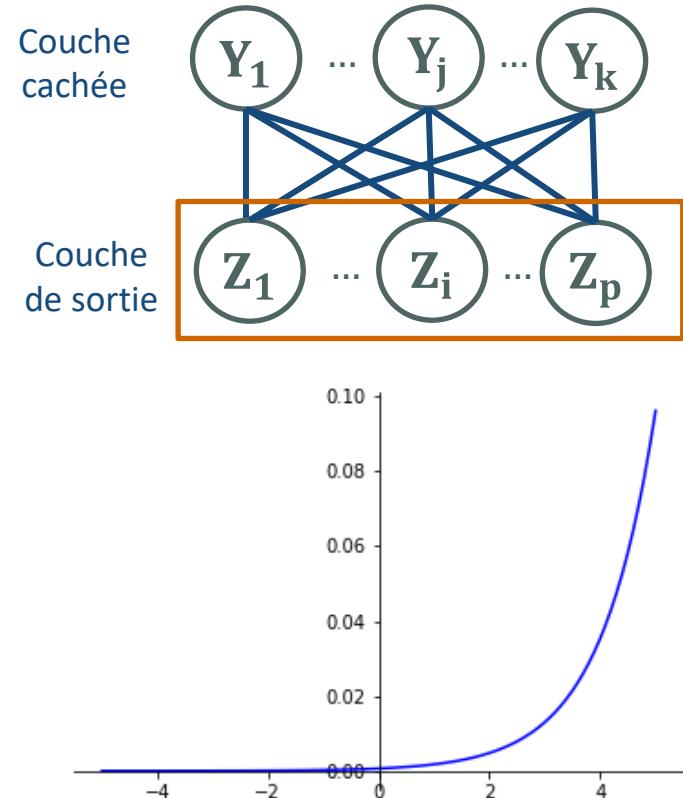
$$\text{softmax } f(Z_i) = \frac{e^{Z_i}}{\sum_{a=1}^p e^{Z_a}}$$

## Dérivé de la fonction softmax

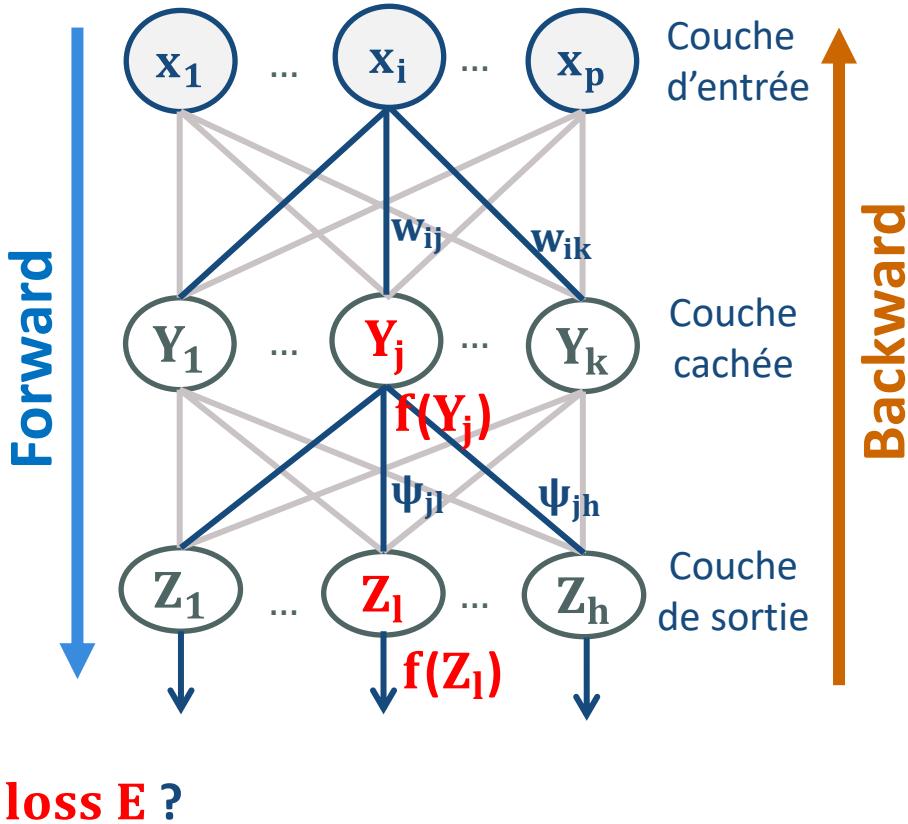
$$f'(Z_i) = \begin{cases} f(Z_i)(1 - f(Z_i)) & \text{si } i = j \\ -f(Z_i)f(Z_j) & \text{si } i \neq j \end{cases}$$

 Utilisée pour activer les neurones de la couche de sortie

 **Classification multi-classes**



# Rétropropagation du gradient Backward + Forward



## ■ Forward : Calcul d'erreur $E$

$$f(Y_j) = f\left(\sum_{i=1}^p w_{ij} x_i + b_j\right)$$

$$f(Z_l) = f\left(\sum_{j=1}^k \psi_{jl} f(Y_j) + b_l\right) = \text{sortie}$$

$E = ?$

## ■ Backward = Descente de gradient

Minimiser  $E$  & Ajustement des poids

$$w = w - \eta \frac{\partial E}{\partial w}$$

# Fonction du perte (loss E)

## Classification

Q Cross Entropy h classes ( $h > 2$ ) = h neurones dans la couche de sortie

$$E = \sum_{l=1}^h -d_l \ln(f(Z_l))$$

avec  $\begin{cases} h = \text{Nombre de classe} \\ d = \text{Classe désirée} \end{cases}$

Q Binary Cross Entropy 2 classes = 1 neurone dans la couche de sortie Z

$$E = -[d \ln(f(Z)) + (1 - d) \ln(1 - f(Z))]$$

Régression : Mean Square Error = 1 neurone dans la couche de sortie Z

$$E = \frac{1}{M} \sum_{i=1}^M (d^{(i)} - f(Z)^{(i)})^2$$

avec  $\begin{cases} M = \text{Nombre d'observation} \\ d^{(i)} = \text{Classe désirée de l'observation } i \\ f(Z)^{(i)} = \text{classe prédictive de l'observation } i \end{cases}$

# Descente de gradient Backward

Répéter

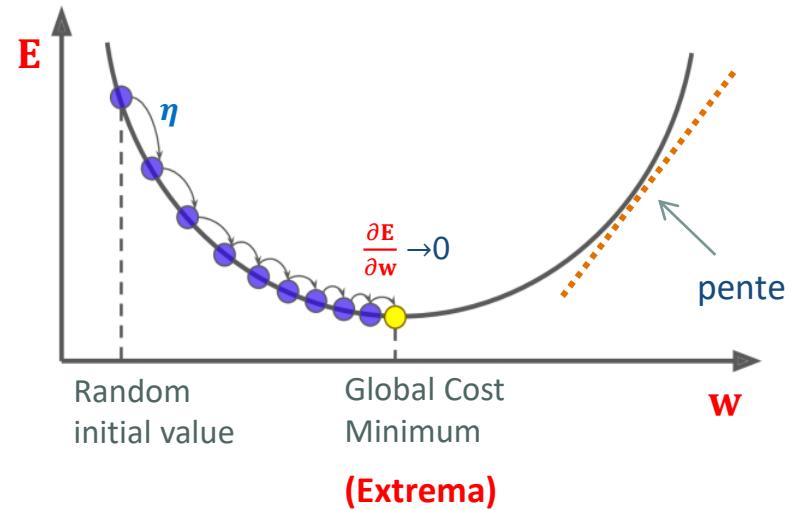
a. calculer le gradient (pente)

$$\nabla g = \frac{\partial E}{\partial w}$$

b. Mettre à jour les poids

$$w = w - \eta \frac{\partial E}{\partial w}$$

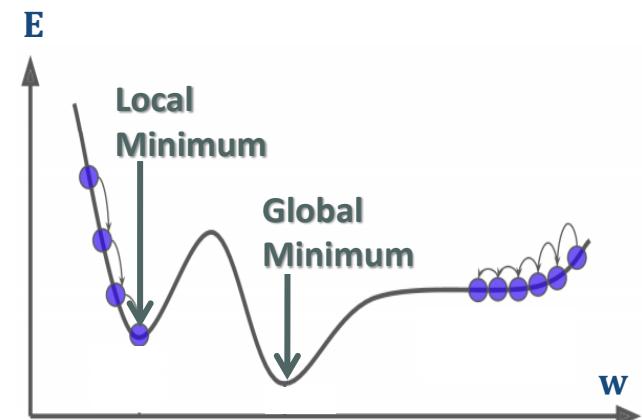
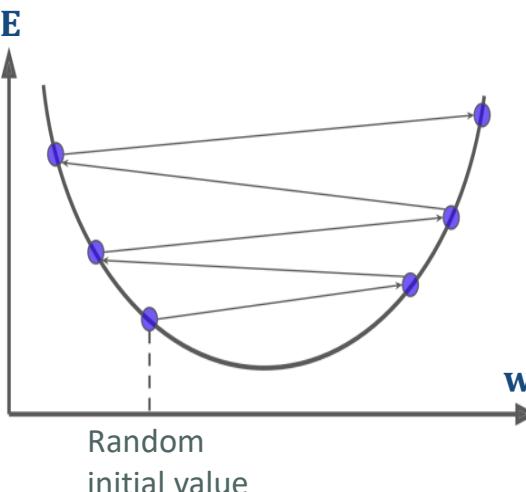
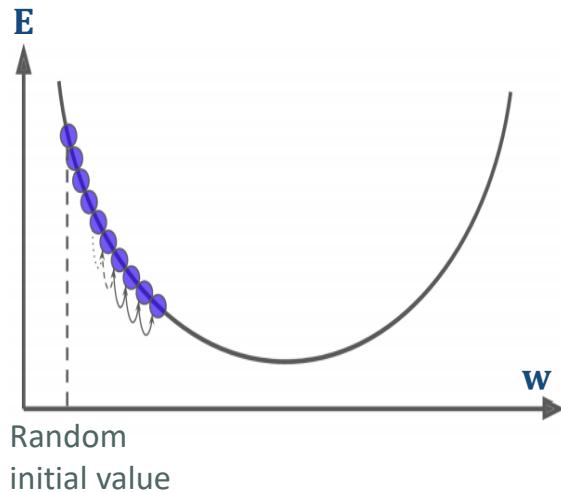
Jusqu'à Trouver le minimum d'erreur (convergence)



$\frac{\partial E}{\partial w} < 0 \Rightarrow$  pente descend vers la droite

$\frac{\partial E}{\partial w} > 0 \Rightarrow$  pente monte vers la droite

# Problèmes de descente de gradient



## $\eta$ EST TROP PETIT

- ⌚ Nombreuses itérations pour converger
- ⌚ Temps important de calcul

## $\eta$ EST TROP ÉLEVÉ

- ⌚ Sauter d'un côté à un autre
- ⌚ Diverger l'algorithme

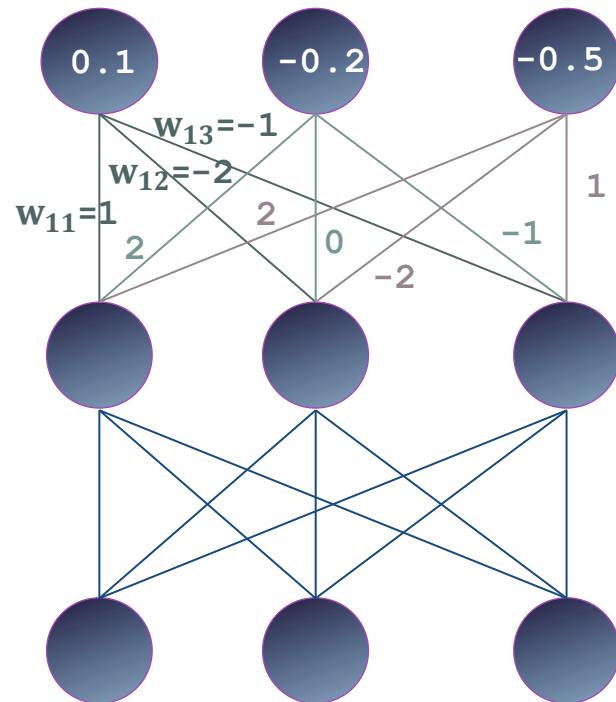
## TOMBER DANS LE MINIMUM LOCAL

- ⌚ Local Minimum > Global Minimum



Choix Learning rate  $\eta = 0.001$

# Formulation mathématique



Matrice de poids

	1	2	3
1	$w_{11}$ 1	$w_{12}$ -2	$w_{13}$ -1
2	$w_{21}$ 2	$w_{22}$ 0	$w_{23}$ -1
3	$w_{31}$ 2	$w_{32}$ -2	$w_{33}$ 1

Matrice des entrées

$$= \mathbf{f} \left( \begin{matrix} 0.1 \\ ? \\ -0.2 \\ ? \\ -0.5 \\ -? \end{matrix} \right)$$

(3, 3)      (3, 1)  
 Multiplication matricielle

# Activité (cas de classification binaire) 1/4

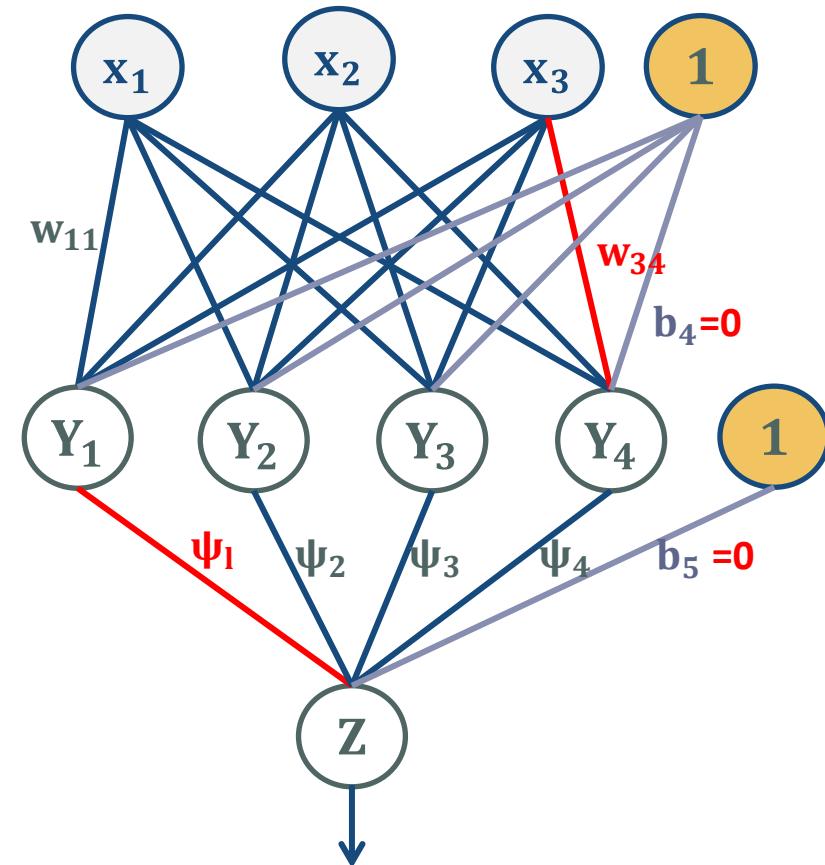
Soit un réseau de neurones de 3 couches:  
 entrée / 1 cachée de 4 neurones  $Y_1 \dots Y_4$  /  
 sortie de 1 neurone  $Z_1$

Tous les biais=0

$X_1$	$X_2$	$X_3$	$y$
5	6	6	0
5	5	6	0
7	8	10	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
7	7	9	1

$$w = \begin{pmatrix} 0.179 & -0.186 & -0.008 & -0.448 \\ 0.044 & -0.028 & -0.063 & -0.131 \\ 0.01 & -0.035 & -0.004 & \color{red}{0.088} \end{pmatrix}$$

$$\Psi = \begin{pmatrix} \color{red}{0.09} \\ -0.171 \\ 0.005 \\ -0.04 \end{pmatrix}$$



Ajuster les poids  $w_{34}$  /  $b_4$  /  $\psi_1$  /  $b_5$  en utilisant la 3<sup>ème</sup> observation ?

# Activité (cas de classification binaire) 2/4

## I- Forward (propagation avant)

### 1- Calcul de l'activation des neurones de la couche cachées

$$f(Y_1) = f\left(\sum_{i=1}^3 w_{i1} x_i + b_1\right) = \frac{1}{1 + e^{-Y_1}}$$

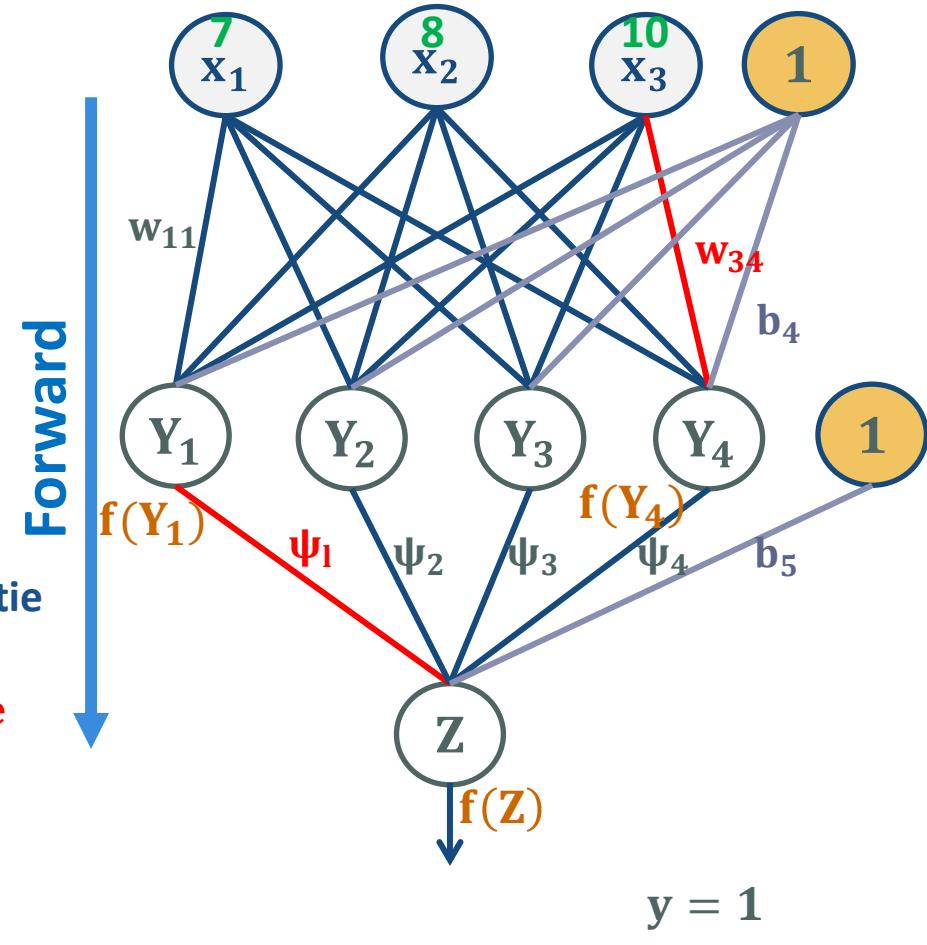
$$f(Y_4) = f\left(\sum_{i=1}^3 w_{i4} x_i + b_4\right) = \frac{1}{1 + e^{-Y_4}}$$

### 2- Calcul de l'activation du neurone de la C. Sortie

$$f(Z) = f\left(\sum_{i=1}^4 \psi_i f(Y_i) + b_5\right) = \frac{1}{1 + e^{-Z}} = \text{sortie}$$

### 3- Calcul de loss E (Binary cross Entropy)

$$E = -[y \ln(f(Z)) + (1 - y) \ln(1 - f(Z))]$$



# Activité (cas de classification binaire) 3/4

## II- Backward : Descente de gradient

$$\Psi_1? \quad \Psi_1 = \Psi_1 - \eta \frac{\partial E}{\partial \Psi_1}$$

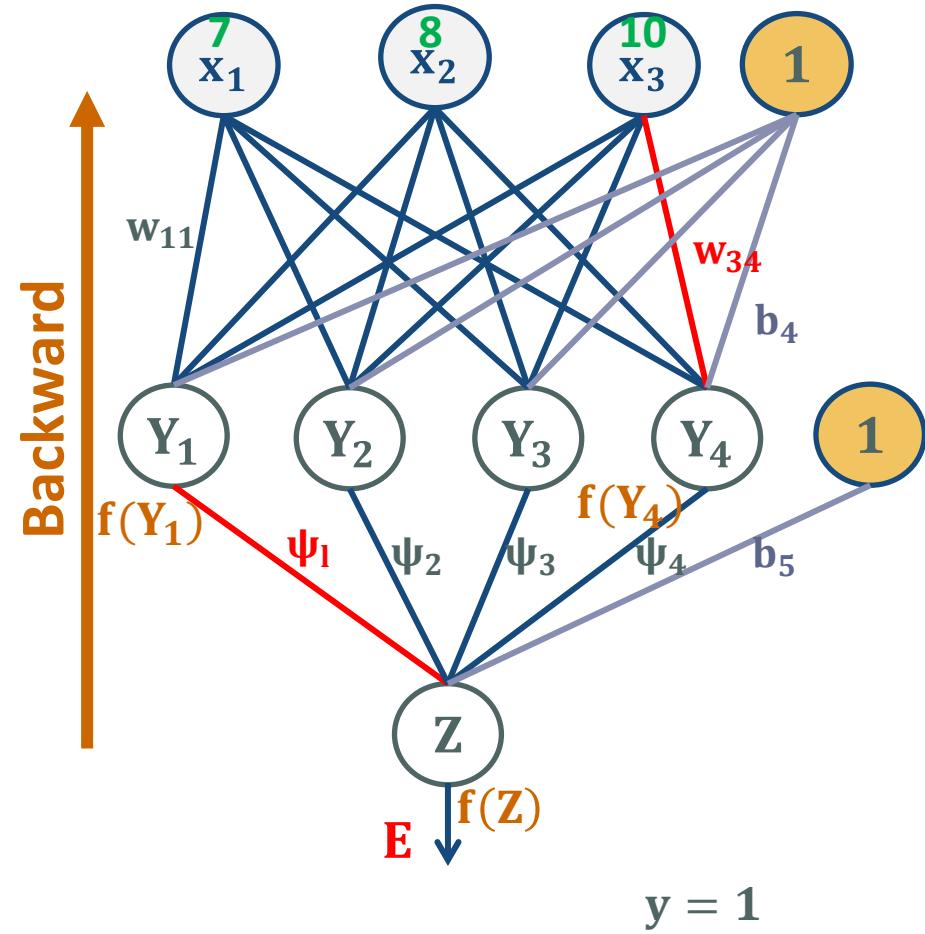
**Chain rule**

$$\frac{\partial E}{\partial \Psi_1} = \frac{\partial E}{\partial f(Z)} \frac{\partial f(Z)}{\partial Z} \frac{\partial Z}{\partial \Psi_1}$$

$$\frac{\partial Z}{\partial \Psi_1} = f(Y_1)$$

$$\frac{\partial f(Z)}{\partial Z} = f'(Z) = f(Z)(1 - f(Z))$$

$$\frac{\partial E}{\partial f(Z)} = ?$$



# Activité (cas de classification binaire) 4/4

$$\frac{\partial \ln(x)}{\partial x} = \frac{1}{x}$$

$$\frac{\partial \ln(1-x)}{\partial x} = -\frac{1}{1-x}$$

$$\frac{\partial E}{\partial f(Z)} = \frac{\partial [-y \ln(f(Z)) - (1-y) \ln(1-f(Z))]}{\partial f(Z)} =$$

$-y \frac{1}{f(Z)} + (1-y) \frac{1}{1-f(Z)}$

$$\frac{\partial E}{\partial f(Z)} = -\frac{y}{f(Z)} + \frac{1-y}{1-f(Z)}$$

**Chain rule**

$$\frac{\partial E}{\partial \psi_l} = \frac{\partial E}{\partial f(Z)} \frac{\partial f(Z)}{\partial Z} \frac{\partial Z}{\partial \psi_l}$$

$$\frac{\partial Z}{\partial \psi_l} = f(Y_1)$$

$$\frac{\partial f(Z)}{\partial Z} = f'(Z) = f(Z)(1 - f(Z))$$

$$\frac{\partial E}{\partial f(Z)} = ?$$

$$E = -[y \ln(f(Z)) + (1-y) \ln(1-f(Z))]$$

# Cas de classification multi-classes

$$\Psi_{11} = \Psi_{11} - \eta \frac{\partial E}{\partial \Psi_{11}}$$

$$E = \sum_{i=1}^3 -d_i \ln(f(T_i))$$

$$\frac{\partial E}{\partial \Psi_{11}} = \sum_{i=1}^3 \frac{\partial E}{\partial f(T_i)} \left( \sum_{j=1}^3 \frac{\partial f(T_i)}{\partial T_j} \frac{\partial T_j}{\partial f(Z_1)} \frac{\partial f(Z_1)}{\partial Z_1} \frac{\partial Z_1}{\partial \Psi_{11}} \right)$$

$$\frac{\partial Z_1}{\partial \Psi_{11}} = f(Y_1)$$

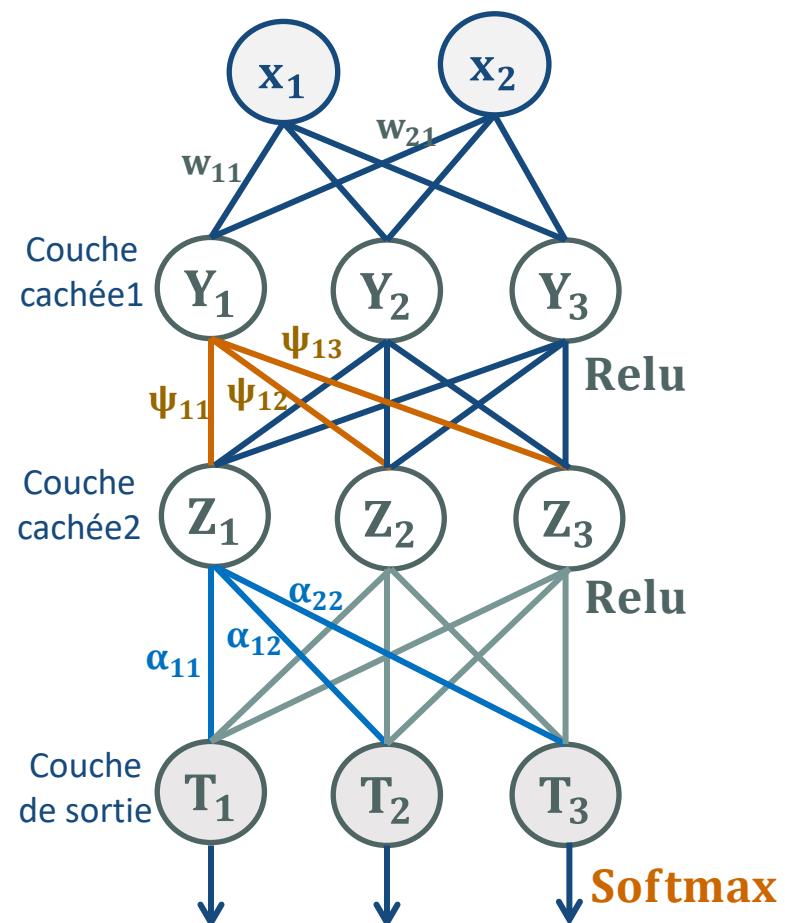
$$\frac{\partial f(z_1)}{\partial z_1} = \begin{cases} 0 & \text{si } z_1 < 0 \\ 1 & \text{si } z_1 \geq 0 \end{cases}$$

$$\frac{\partial T_j}{\partial f(z_1)} = \alpha_{1j}$$

$$\frac{\partial f(T_i)}{\partial T_j} = \begin{cases} f(T_i)(1 - f(T_i)) & \text{si } i = j \\ -f(T_i)f(T_j) & \text{si } i \neq j \end{cases}$$

$$\frac{\partial E}{\partial f(T_i)} = -\frac{d_i}{f(T_i)}$$

Backward



# Epoch, size\_batch & Iteration



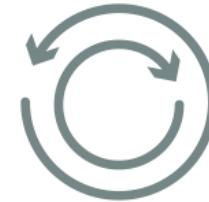
## ÉPOQUE (epoch)

Passage sur l'ensemble de toutes les **n** observations de la base d'apprentissage



## Size\_batch

Nombre total d'observations d'apprentissage présents dans un seul batch



## Iteration

Nombre des batches nécessaires pour compléter une époque

la taille de lots nombre d'observation

combien de lots nécessaires pour compléter les époques

\*Batch ≠ Size\_batch!

# Variantes de descente de gradient (1/2)



## Descente de gradient classique Batch Gradient Descent (BGD)

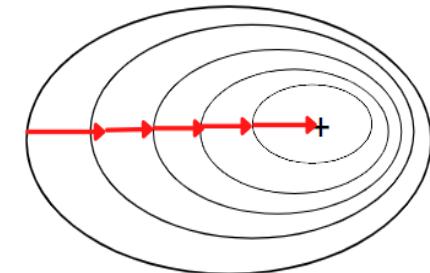
kif ikaml el  
epoc ibadel el  
poids

- Mettre à jour les poids après avoir fait passer la totalité des observations par époque

$$\mathbf{w}_i = \mathbf{w}_i - \eta \sum_{j=1}^N \frac{\partial E}{\partial w_i}$$

N= Nombre d'observation

- ☺ Stable
- ☹ Mise à jour lent des poids
- ☹ Facile à tomber dans le minimum local

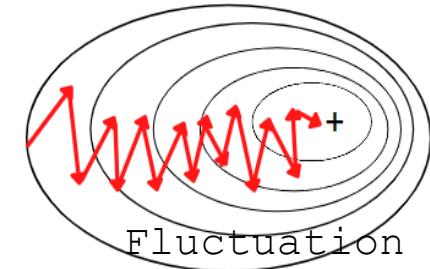


## Descente de gradient stochastique Stochastic Gradient Descent (SGD)

- Mettre à jour les poids au passage de chaque observation (indépendamment du nombre d'époque)

$$\mathbf{w}_i = \mathbf{w}_i - \eta \frac{\partial E}{\partial w_i}$$

- ☺ Mise à jour rapide des poids
- ☹ Instable
- ☹ Difficile de converger vers l'extrema



Fluctuation

apres chaque observation ibadel el poids

## Variantes de descente de gradient (2/2)



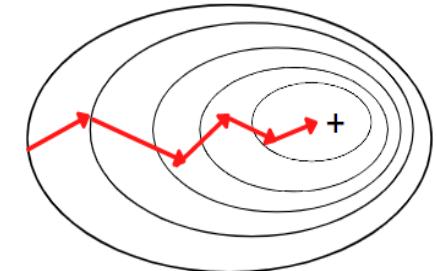
### Descente de gradient Mini-batch Mini-Batch Gradient Descent (MBGD)

— Mettre à jour par blocs d'observations (batch-size)

$$\mathbf{w}_i = \mathbf{w}_i - \eta \sum_{j=1}^K \sum_{j=1}^M \frac{\partial E}{\partial w_i}$$

K= Nombre de batch=Itération

M= Nombre d'observation dans un batch=batch-size



- ☺ Convergence plus rapide que BGD
- ☹ Sensibilité à la taille du lot (batch\_size)
- ☹ Facile à tomber dans le minimum local



# Quiz

1. L'erreur Binary cross entropy est adaptée aux problèmes de classification binaire (seule réponse)

- Vrai
- Faux

2. Lesquelles des affirmations suivantes sont correctes? (réponse multiple)

- L'algorithme de descente de gradient classique (BGD) est rapide et fiable
- L'algorithme BGD risque tomber dans le local minimum
- L'algorithme de descente de gradient stochastique est rapide mais instable
- Le traitement par lot accélère la convergence par rapport à la descente de gradient classique (BGD) grâce à une mise à jour fréquente des poids

3. Étant donné une base d'apprentissage de 100 observations avec la taille d'un batch est égal à 1, alors (seule réponse)

- Il faudra une seule itération pour compléter une époque
- Il faudra deux itérations pour compléter une époque
- Il faudra 100 itérations pour compléter une époque
- Aucune réponse n'est correcte

# Autres variantes de descente de gradient **Optimizeur**

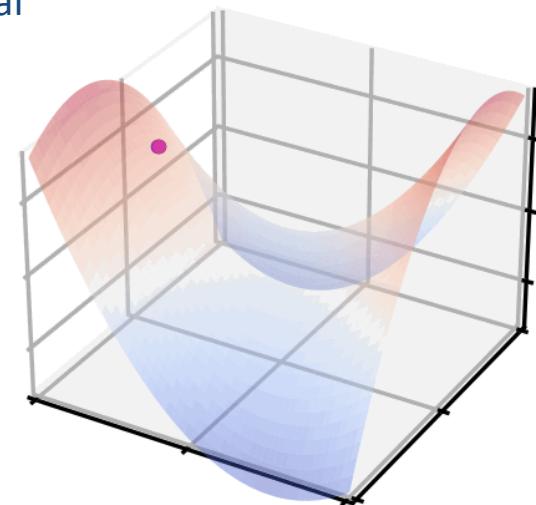


## Optimiser la manière d'ajustement de poids

- Accélérer la convergence tout en minimisant la fonction de perte
- Éviter ou dépasser les minimums locaux : atteindre le minimum global

### ① Optimiseurs les plus courants

- Batch Gradient Descent (BGD)
- Stochastic Gradient Descent (SGD)
- Mini-Batch Gradient Descent (MBGD)
- Adaptive Gradient Algorithm (Adagrad)
- Adaptive Learning Rate Method (Adadelta)
- Root Mean Square Propagation (RMSprop)
- Adaptive Moment Estimation (Adam)



# Optimiseur Adam

## 1- Initialisation

$$\mathbf{m}^{(t-1)} = \mathbf{0} \quad \mathbf{v}^{(t-1)} = \mathbf{0} \quad \beta_1 = 0.9 \quad \beta_2 = 0.999$$

## 2- Mise à jour de deux moments $\mathbf{m}^{(t)}$ et $\mathbf{v}^{(t)}$ ?

$$\mathbf{m}^{(t)} = \beta_1 \mathbf{m}^{(t-1)} + (1 - \beta_1) \frac{\partial E}{\partial \mathbf{w}^{(t)}} \quad \mathbf{v}^{(t)} = \beta_2 \mathbf{v}^{(t-1)} + (1 - \beta_2) \left( \frac{\partial E}{\partial \mathbf{w}^{(t)}} \right)^2$$

## 3- Mise à jour des poids !

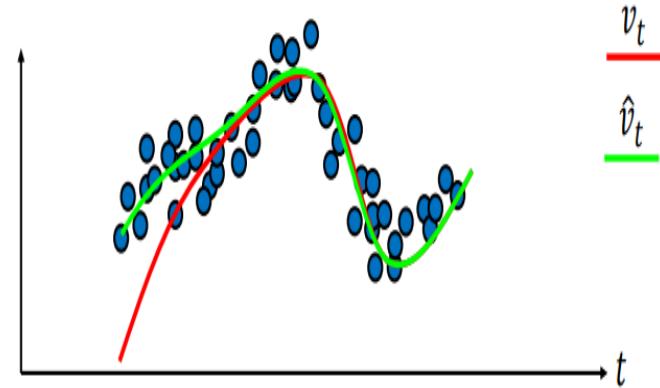
$$\hat{\mathbf{m}}^{(t)} = \frac{\mathbf{m}^{(t)}}{1 - \beta_1}$$

$$\hat{\mathbf{v}}^{(t)} = \frac{\mathbf{v}^{(t)}}{1 - \beta_2}$$

$$\mathbf{g}^{(t)} = \frac{\hat{\mathbf{m}}^{(t)}}{\sqrt{\hat{\mathbf{v}}^{(t)}} + \epsilon}$$

= $10^{-7}$  (constante pour éviter la division par 0)

$$\mathbf{w} = \mathbf{w}^{(t-1)} - \eta \mathbf{g}^{(t)}$$



# Adam/ Momentum/ Adagrad & Adadelta!

## Momentum

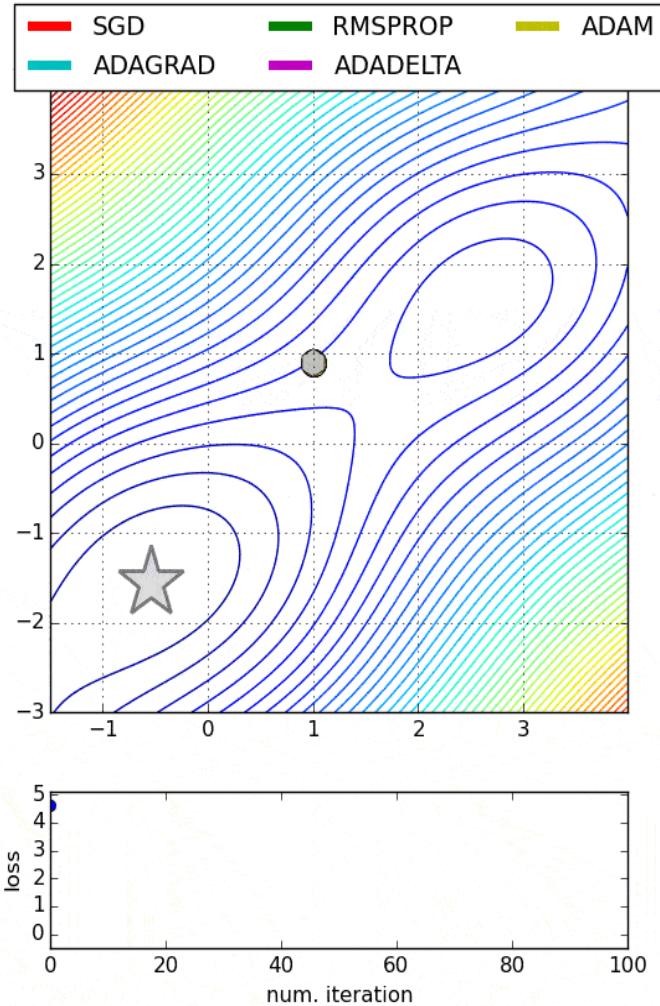
- 😊 Convergence très rapide
- 😢 Nécessité de la définition d'un nouveau hyper paramètre  $0 < \text{momentum} < 1$

## Adagrad & Adadelta

- 😊 Learning rate  $\eta$  modifiable
- 😊 Possible de s'entraîner sur des données peu nombreuses
- 😢 Coûteux en terme de calcul
- 😢 Learning rate  $\eta$  est toujours en diminution : Apprentissage lent

## Adam

- 😊 Convergence très rapide
- 😢 Coûteux en terme de calcul
- 😢 Nécessité de la définition de deux nouveaux hyper paramètres  $0 < \beta_1 < 1$  et  $0 < \beta_2 < 1$

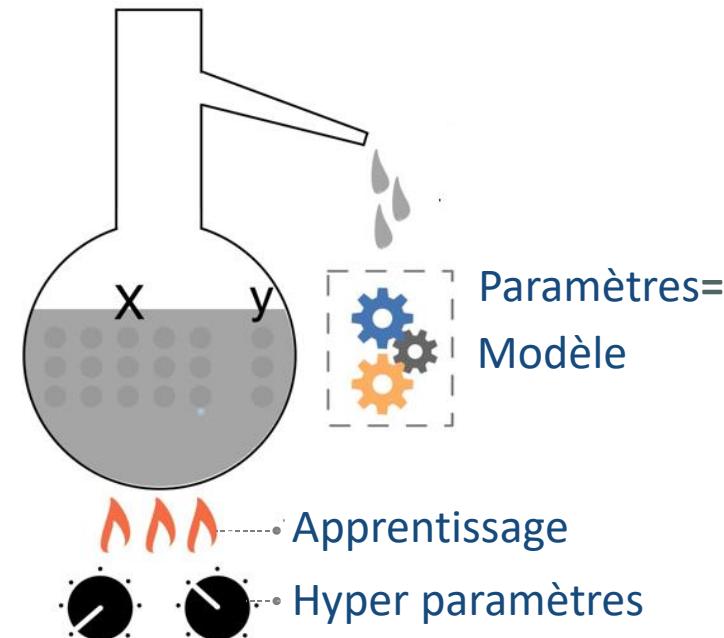


# Pour résumer Paramètres & Hyper paramètres!

**Paramètres** =automatiquement appris par les modèles  **Poids+ Biais**

**Hyper paramètres**=configurés manuellement

- Learning rate  $\eta$
- Nombre d'époques
- Taille d'un Batch
- Nombre de neurones dans chaque couche cachée
- Nombre de couches cachées
- Choix d'Optimizer



# ESTIMATION DES PERFORMANCES



# Evaluation d'un réseau de neurones

CM		Classe prédictive			
		$C_1$	...	$C_i$	$C_N$
Classe désirée	$C_1$	CM(1,1)		CM(1,i)	
	$\vdots$		$\vdots$		
	$C_i$			CM(i,i)	
	$C_N$				CM(N,N)

		Classe prédictive	
		yes	no
Classe désirée	yes	TP	FN
	no	FP	TN

Annotations:

- True Positive (TP): Cellule (yes, yes).
- False Positive (FP): Cellule (no, yes).
- True Negative (TN): Cellule (no, no).
- False Negative (FN): Cellule (yes, no).

$$\text{Accuracy} = \frac{\sum_{i=1}^N CM(i,i)}{n}$$

$$\text{recall}_{C_i} = \frac{CM(i,i)}{\sum_{j=1}^N CM(i,j)}$$

$$\text{precision}_{C_i} = \frac{CM(i,i)}{\sum_{j=1}^N CM(j,i)}$$

$$F1 - score_{C_i} = \frac{2 \times (\text{precision}_{C_i} \times \text{recall}_{C_i})}{\text{precision}_{C_i} + \text{recall}_{C_i}}$$

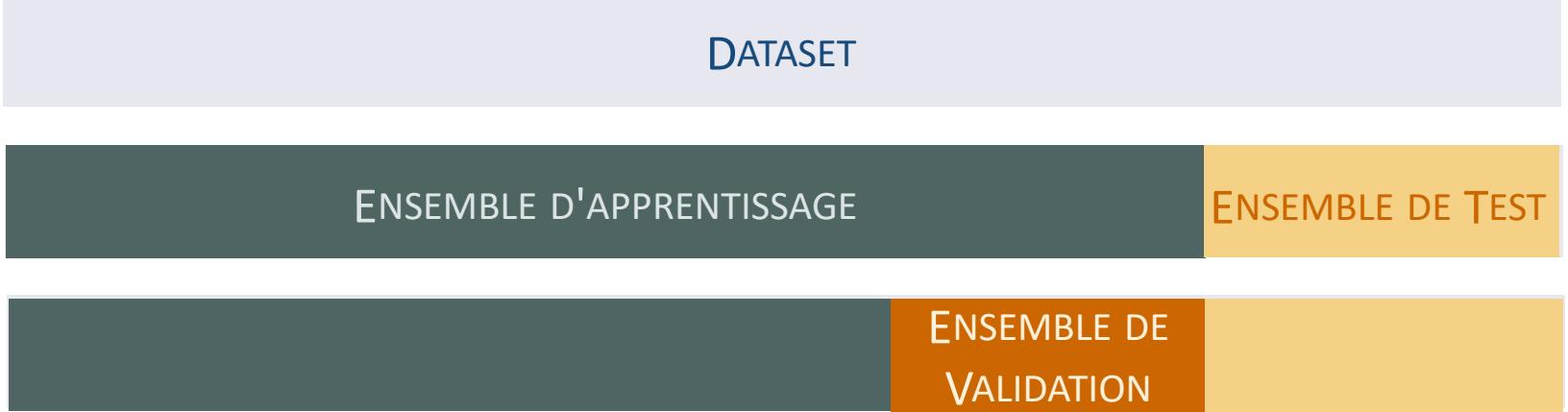
## Cas de 2 classes

recall = True Positive rate

$$= \text{Sensitivity} = \frac{TP}{TP+FN}$$

$$\text{True Negative rate} = \text{Specificity} = \frac{TN}{FP+TN}$$

# Échantillonnage!

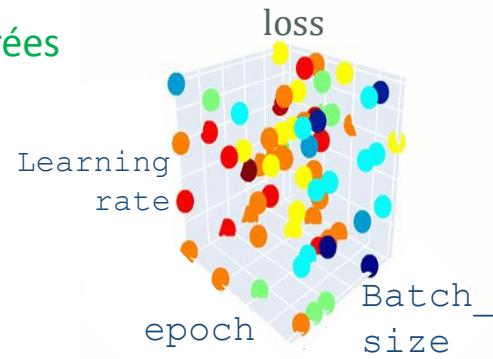


**ENSEMBLE DE VALIDATION**= utilisé pour déterminer les valeurs des **hyper paramètres** du modèle ✓

# Méthodes de recherche des hyper paramètres

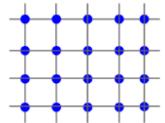
## Grid search

- Exploration exhaustive
- 😊 Garantit que toutes les combinaisons des hyperparamètres sont explorées
- 😢 Coûteux en termes de temps de calcul

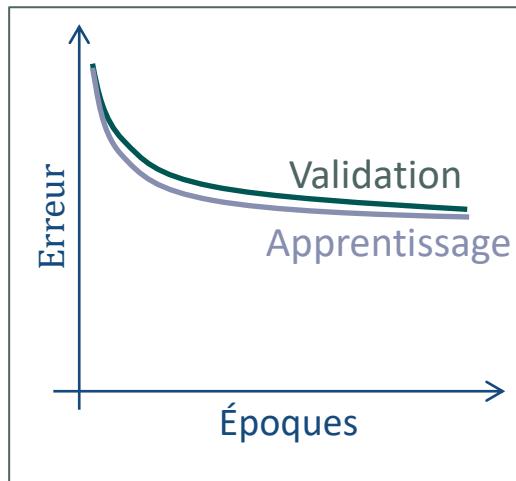


## Random search

- Exploration aléatoire
- 😊 Peut converger plus rapidement vers des combinaisons d'hyperparamètres performantes
- 😢 Ne garantit pas une exploration exhaustive de l'espace des hyperparamètres
- 😢 Certaines combinaisons peuvent ne pas être évaluées



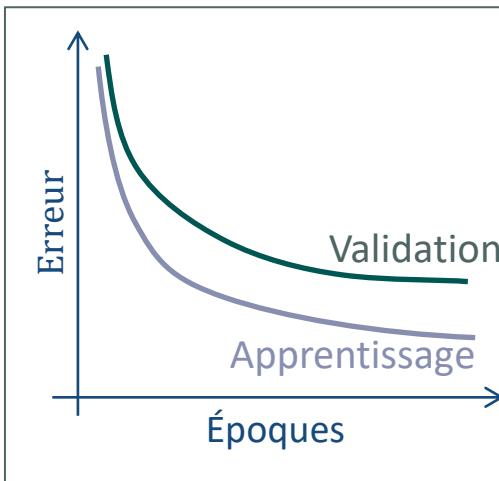
# Sous-apprentissage vs. Sur-apprentissage



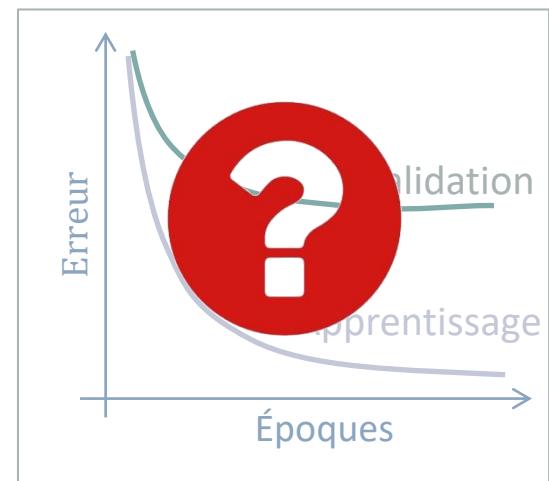
Sous-apprentissage  
(Underfitting)

biais élevé

il faut augmenter la complexité du modèle



Parfait  
(perfect)

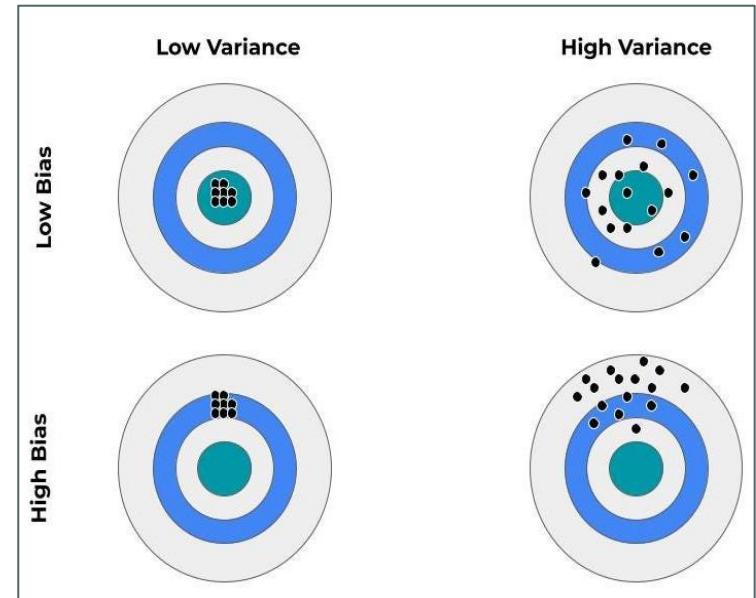


Sur-apprentissage  
(Overfitting)

variance élevé

# Erreur : Compromis Biais-variance 1/2

- **Biais** : mesure la différence entre les résultats des prédictions avec les valeurs désirées par rapport aux données d'apprentissage
- **Variance** : dispersion ou la variabilité d'un ensemble de données=mesure la sensibilité du modèle aux variations dans l'ensemble de données d'entraînement
- Erreur = **Biais<sup>2</sup>** + **Variance** + Erreur irreductible



# Erreur : Compromis Biais-variance 2/2

$$\text{Biais}^2 = \frac{1}{N} \sum_{i=1}^N (\bar{Y} - \text{desired}_i)^2$$

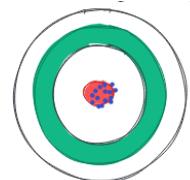
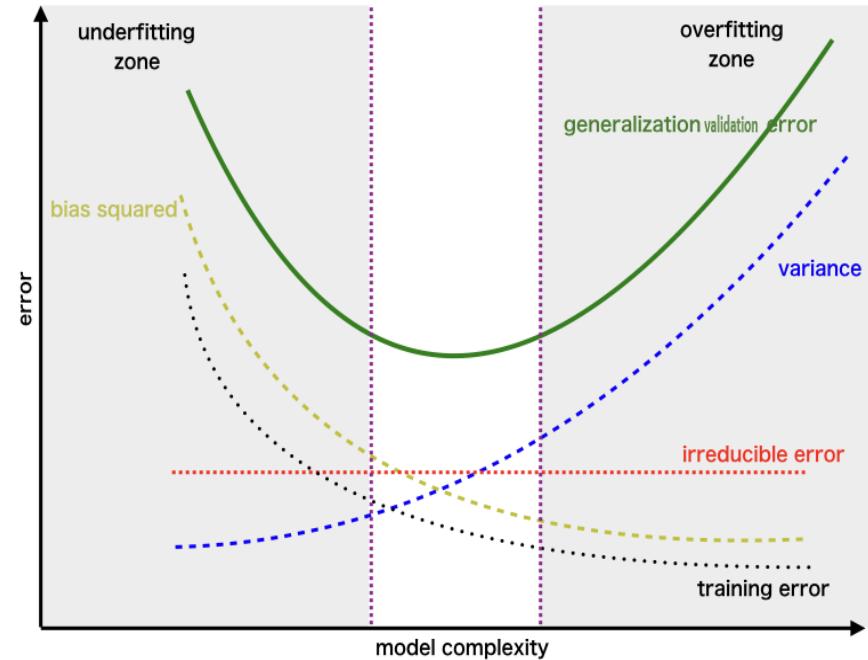
N = Nombre d'observation

$$\bar{Y} = \frac{1}{N} \sum_{i=1}^N \text{predicted}_i$$

$$\text{Variance} = \frac{1}{N-1} \sum_{i=1}^N (\text{predicted}_i - \bar{Y})^2$$

$$\text{Erreur irreductible}(\sigma^2) = \frac{1}{N} \sum_{i=1}^N (\varepsilon_i - \bar{\varepsilon})^2$$

$$\varepsilon_i = \text{Bruit dans les données} \quad \bar{\varepsilon} = \frac{1}{N} \sum_{i=1}^N \varepsilon_i$$





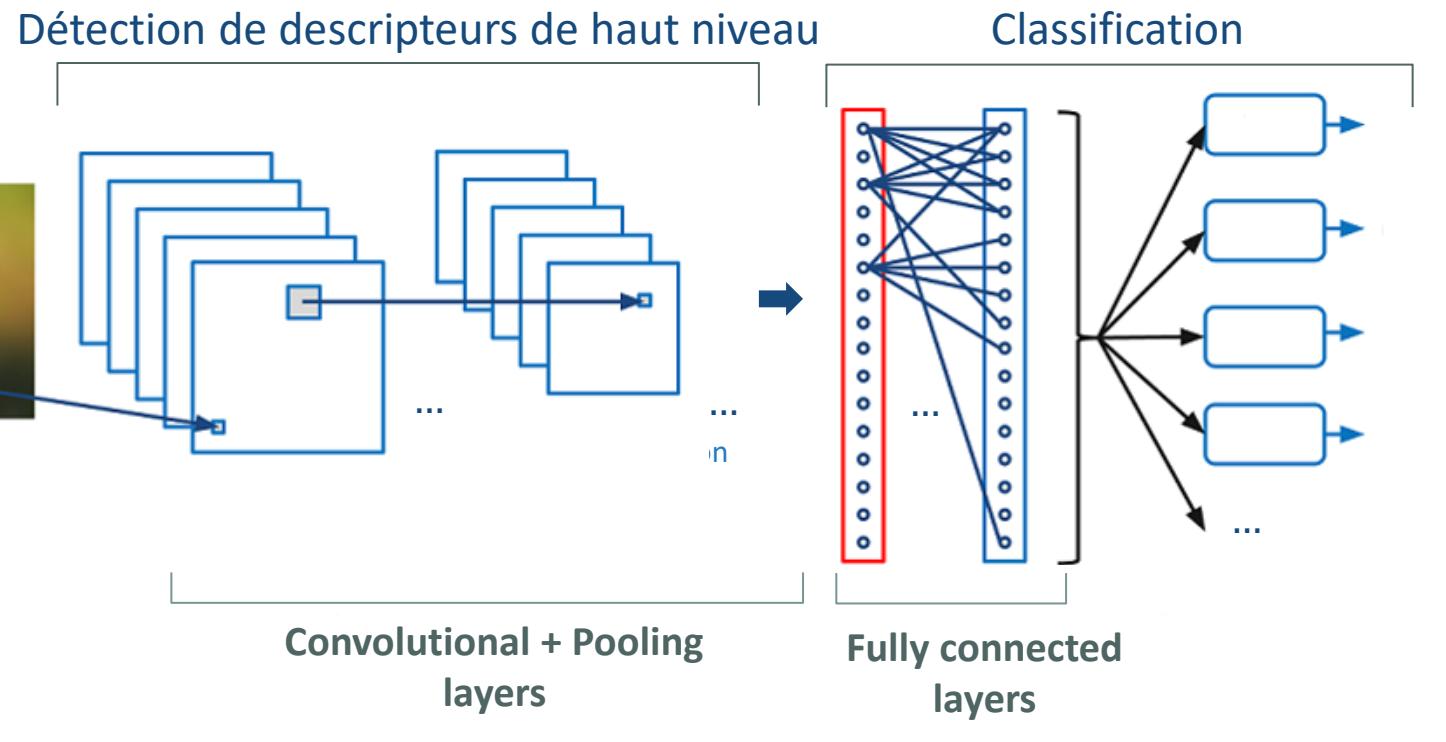
## CHAPITRE 2

# RÉSEAUX DE NEURONES CONVOLUTIFS CLASSIFICATION DES IMAGES

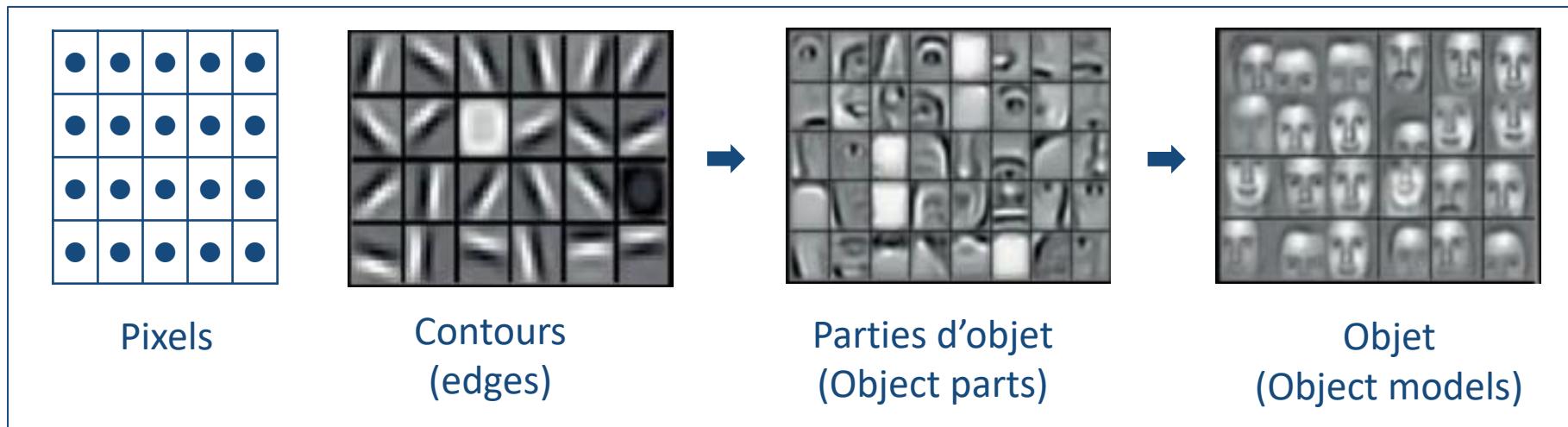


# Réseaux de neurones convolutifs

## Convolutional Neural Network (ConvNet/CNN)



# Descripteurs haut niveau!



Lettre (Character)	Mot (Word)	Groupe de mot= phrase (Sentences)	Article (Story)
-----------------------	---------------	-----------------------------------	--------------------

# Couches de CNN

CNN détermine les filtres adéquats qui maximisent la performance d'une façon automatique



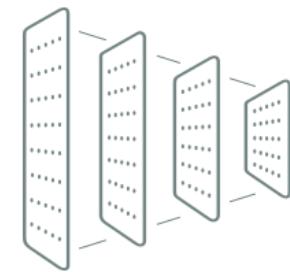
CONVOLUTION



POOLING



FLATTENING



FULLY CONNECTED

Appliquer des filtres pour extraire des descripteurs

les dimensions changent

Conserver les descripteurs importantes (réduction)

réduction de la dimensionnalité

Convertir les matrices de descripteurs en tableau 1D

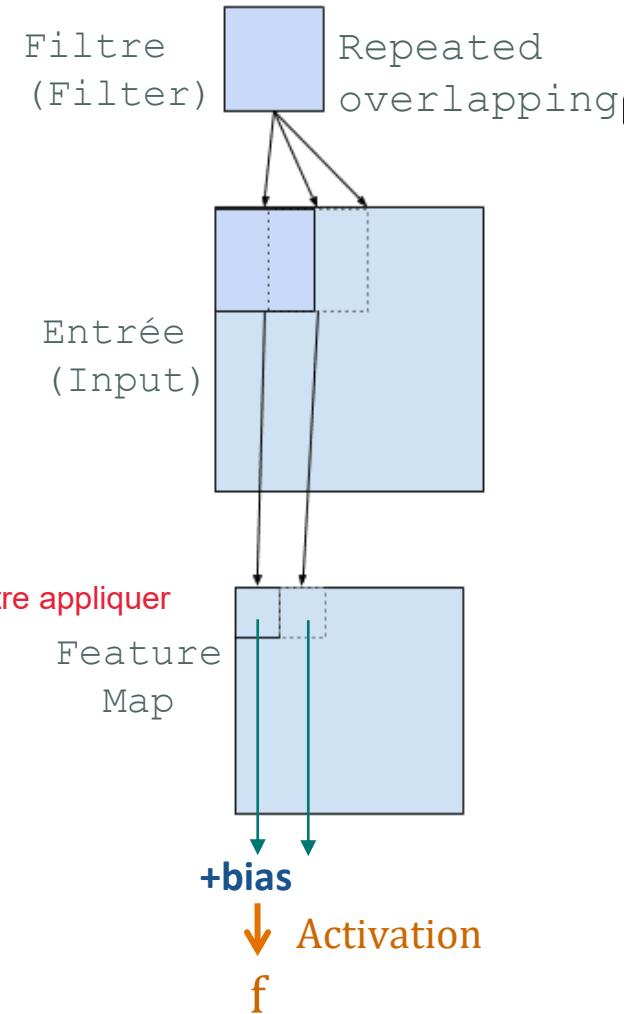
conversion nD --> 1D

Construire le modèle

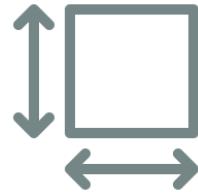
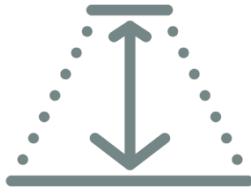
dimension

# Convolutional layers

- Application d'un filtre à une entrée qui entraîne une activation
- Filter=kernel=2D array of weights (poids)
- Le filtre est appliqué plusieurs fois à la matrice d'entrée
- Le résultat est un tableau 2D de valeurs de sortie: Feature Map  
— Le résultat doit être biaisé et activé par une fonction d'activation  $f$   
— Généralement **ReLU** ou l'une de ses variantes (**Elu**, **Leaky ReLU**, etc.)



# Convolutional layers



Depth (Volume)

kernel\_size

Stride

Padding

Nombre de filtres

Taille d'un filtre ou  
un noyau

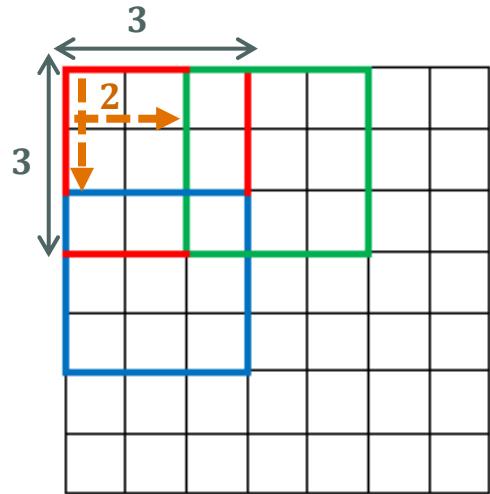
Détermine la façon  
dont le filtre est glissé  
sur l'entrée

3 max

Détermine le nombre de  
pixels remplis de zéros  
autour de la bordure

# Convolution simple depth = 1

💡 Depth = 1 → single-Filter

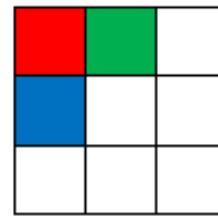


Entrée

Input =  $7 \times 7$

kernel\_size =  $3 \times 3$

Stride = 2



Sortie  $3 \times 3$  ?



# Convolution simple depth = 1

## ♣ Exemple de calcul

— Input= **5 × 5**

— kernel\_size= **3 × 3**

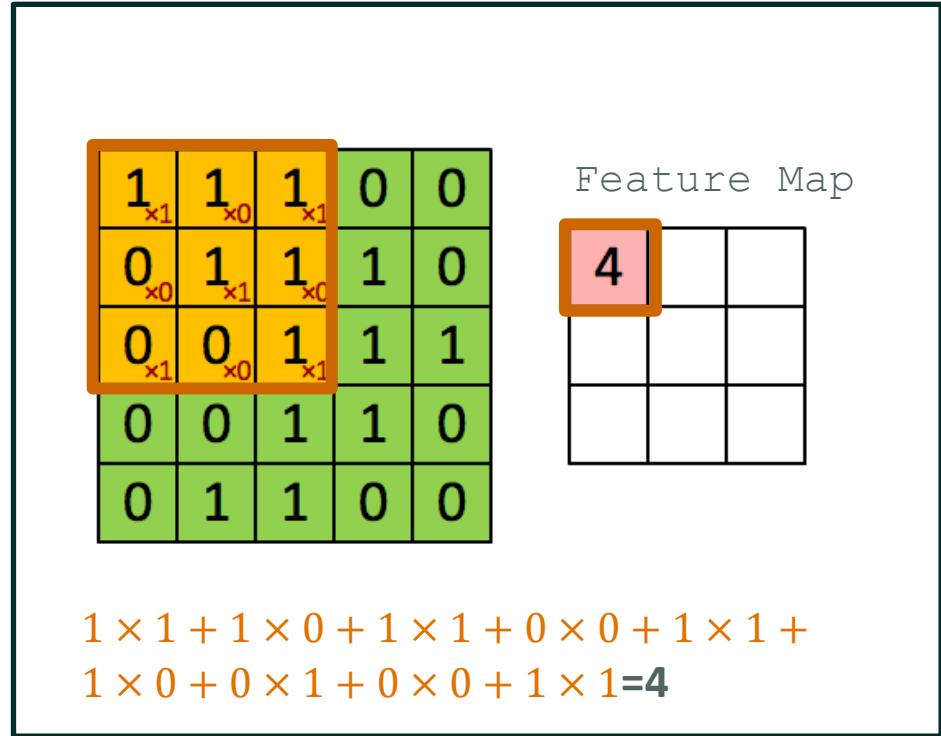
— Stride= **1**

Entrée

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filtre

1	0	1
0	1	0
1	0	1

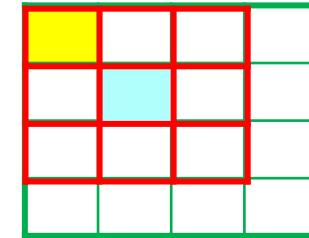


# Padding

## padding="valid"

- Pas de remplissage par des zéros
- La taille sera réduite : Ignorer certaines lignes et colonnes de la matrice d'entrée

padding="valid"



## padding="same"

- Des zéros sont ajoutés au cadre extérieur de l'image (droite/Gauche/Bas/Haut) pour permettre au filtre de couvrir plus d'espace dans l'image
- Taille d'entrée= Taille de feature map après convolution
- Analyse plus précise des images

padding="same"

0	0	0	0	0	0
0					0
0					0
0					0
0					0
0	0	0	0	0	0

# Convolution en volume

0	0	0	0	0	0	0	...
0	156	155	156	158	158	158	...
0	153	154	157	159	159	159	...
0	149	151	...	...	...	...	...
0	...	...	...	...	...	...	...
0	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...

Input Channel 1

-1	-1	-1
0	1	0
0	1	1

Kernel 1 Channel 1

=

308

0	0	0	0	0	0	0	...
0	167	166	167	169	169	169	...
0	164	165	168	170	170	170	...
0	160	161	...	...	...	...	...
0	...	...	...	...	...	...	...
0	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...

Input Channel 2

x

1	0	0
1	-1	-1
1	0	-1

Kernel 1 Channel 2

=

-489

0	0	0	0	0	0	0	...
0	163	162	163	165	170	170	...
0	160	161	164	165	166	166	...
0	156	158	...	...	...	...	...
0	...	...	...	...	...	...	...
0	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...

Input Channel 3

x

0	1	1
0	1	0
1	-1	1

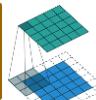
Kernel 1 Channel 3

=

164 + b = -25

Biais

Nb\_paramètres = (kernel\_size × Nb\_channels en entrée + 1) × Nb\_filtres



# Conv2D Formule générale

— Input =  $H \times L \times W$

—  $n$  Filtres (kernel\_size) =  $H_f \times L_f$

— Output =  $H_o \times L_o \times n$

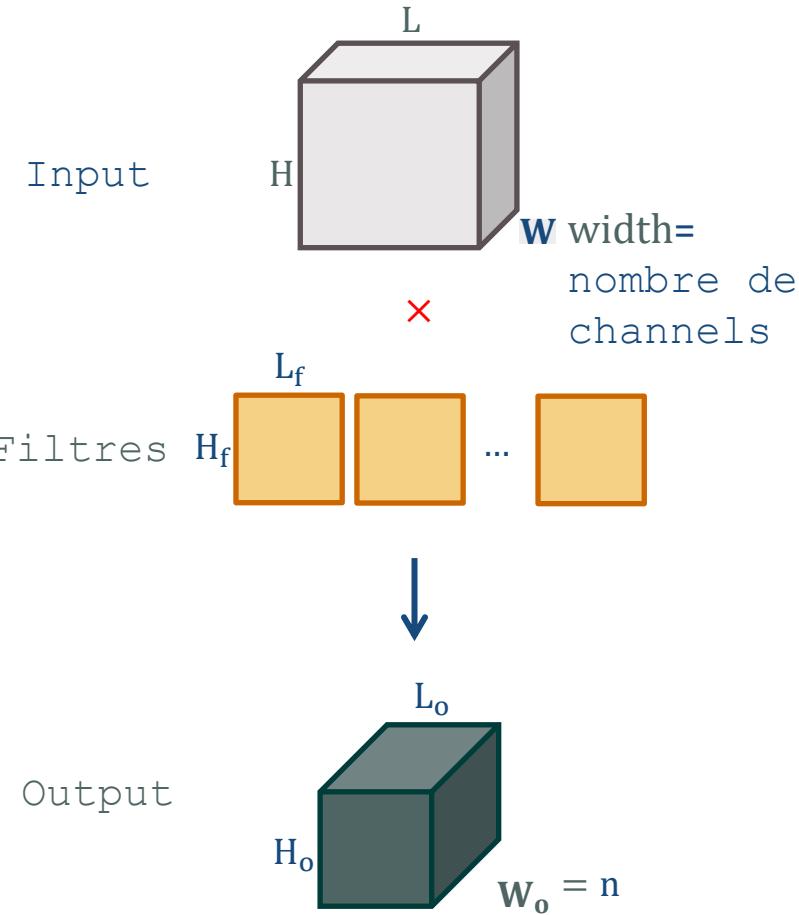
padding = "valid"

$$H_o = \text{ceil}\left(\frac{H - H_f + 1}{\text{stride}}\right) \quad L_o = \text{ceil}\left(\frac{L - L_f + 1}{\text{stride}}\right)$$

padding = "same"

$$H_o = \text{ceil}\left(\frac{H}{\text{stride}}\right) \quad L_o = \text{ceil}\left(\frac{L}{\text{stride}}\right)$$

Exemple:  $\text{ceil}(3.7) = 4$  /  $\text{ceil}(3.2) = 4$



# Conv2D tf.keras.layers.Conv2D

```
model = tf.keras.Sequential()  
model.add(tf.keras.layers.Conv2D(20, (3, 3), strides=(1,1),  
                               padding="same", activation="relu",  
                               input_shape=(64,64,3)))
```

Nombre de filters      kernel\_size

↓                        ↓

#Par défaut

```
tf.keras.layers.Conv2D(filters, kernel_size, strides=(1, 1),  
                      padding='valid', activation=None,  
                      kernel_initializer='glorot_uniform', bias_initializer='zeros',  
                      kernel_regularizer=None, bias_regularizer=None  
                      **kwargs)
```

# Pooling layers

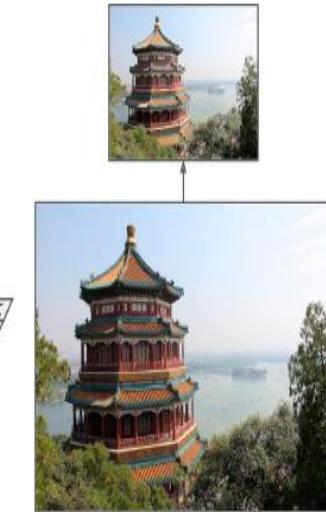
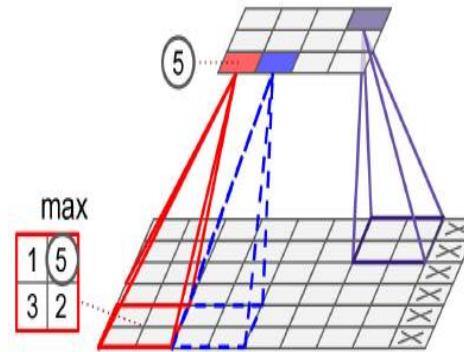
- Réduire la taille de l'entrée à la couche suivante (réduire les dimensions)

## 💡 MaxPooling / AveragePooling

- En général, la taille du noyau dans Pooling `pool_size` est  $2 \times 2$

- MaxPooling est le plus utilisé

- 😊 Prévenir le sur apprentissage
- 😊 Obtenir des données de longueur fixe
- 😊 Invariance



- `pool_size = 2 × 2`
- `padding = "valid" ou "same"`
- `Stride = 2`

# Pooling layers

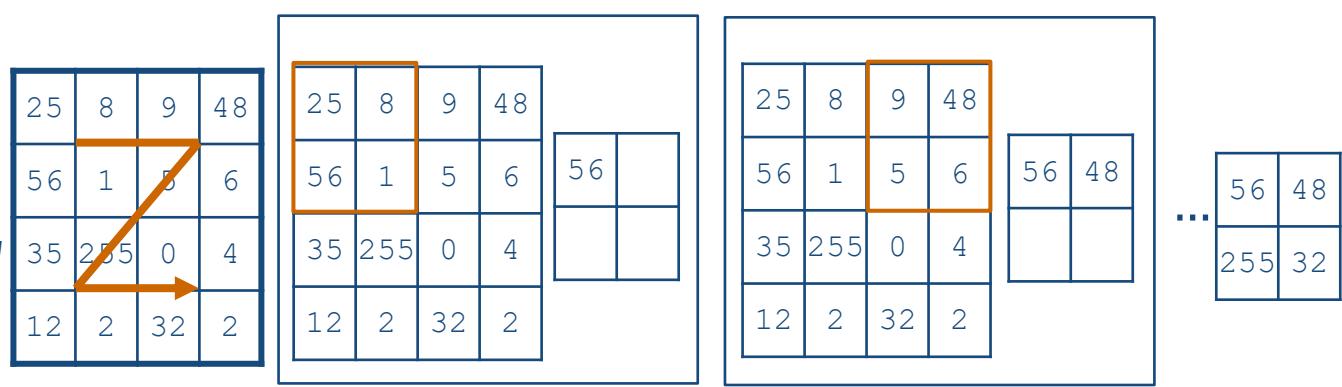
## ♣ Exemple

— MaxPooling

— pool\_size =  $2 \times 2$

— padding = "valid"

— Stride = 2



# Pooling layers Invariance

 Fait référence à la capacité du modèle à maintenir certaines caractéristiques importantes malgré des variations dans la position spatiale

— Reconnaître des objets même s'ils sont légèrement décalés ou positionnés différemment dans l'image

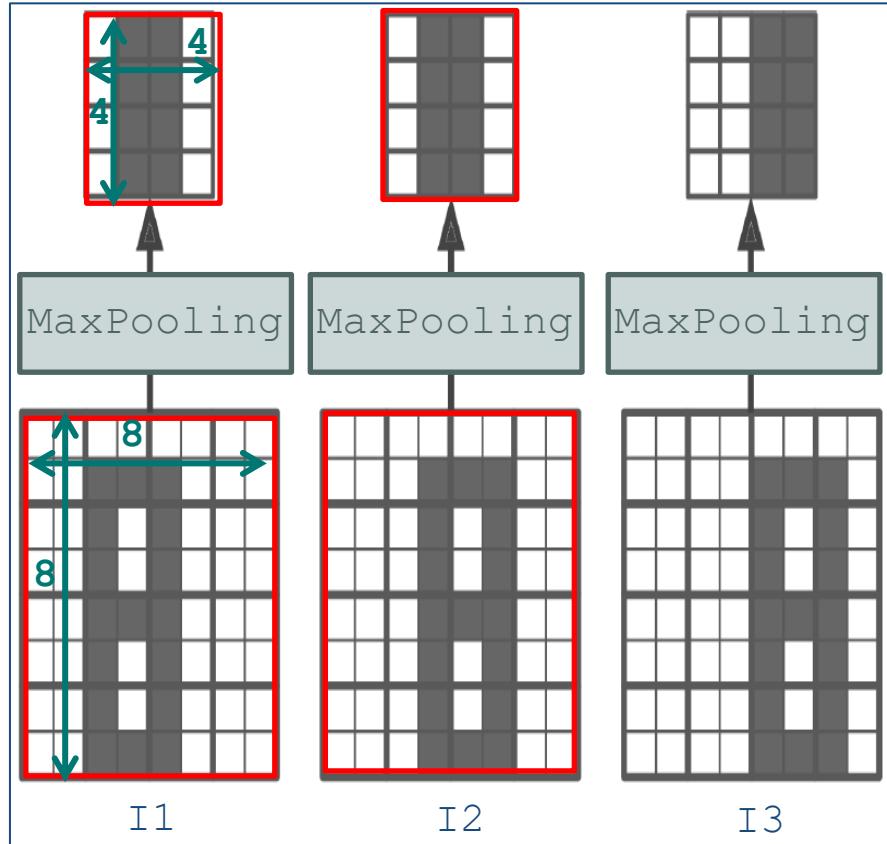
— MaxPooling

— pool\_size= **$2 \times 2$**

— padding="**valid**"



— Stride=2



I1, I2 et I3=Mêmes images mais chacune est décalée de 2 pixels vers la droite

# Pooling Formule générale

— Input =  $H \times L \times W$



padding = "valid"

$$H_o = \text{floor}\left(\frac{H - H_p}{\text{stride}}\right) + 1 \quad L_o = \text{floor}\left(\frac{L - L_p}{\text{stride}}\right) + 1$$

— pool\_size =  $H_f \times L_f$



padding = "same"

$$H_o = \text{floor}\left(\frac{H - 1}{\text{stride}}\right) + 1 \quad L_o = \text{floor}\left(\frac{L - 1}{\text{stride}}\right) + 1$$

Exemple:  $\text{floor}(3.7) = 3$  /  $\text{floor}(3.2) = 3$

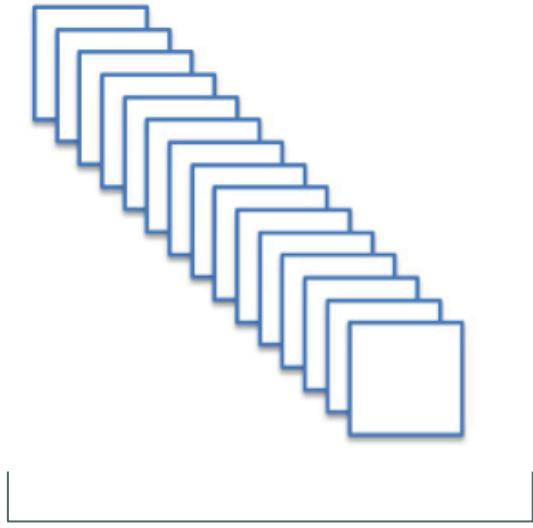
# MaxPool2D/ AveragePooling2D

```
model = tf.keras.Sequential()  
:  
  
model.add(tf.keras.layers.MaxPool2D(pool_size=(4, 4),  
                                    strides=(4,4), padding="valid"))  
  
model.add(tf.keras.layers.AveragePooling2D(pool_size=(4, 4),  
                                         strides=(4,4), padding="valid"))
```

#Par défaut

```
tf.keras.layers.MaxPool2D(pool_size=(2, 2), strides=None, padding='valid', **kwargs)  
  
tf.keras.layers.AveragePooling2D(pool_size=(2, 2), strides=None,  
                                 padding='valid', **kwargs)
```

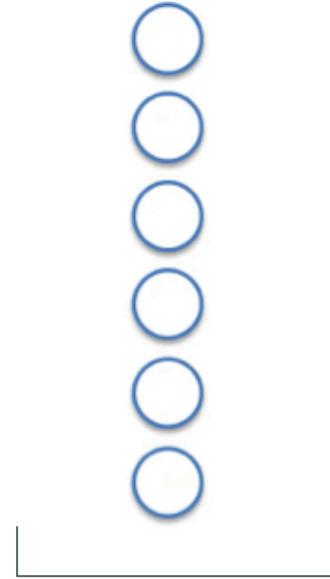
# Flattening



Last Pooling Layer

$$H \times L \times W$$

FLATTENING  
→

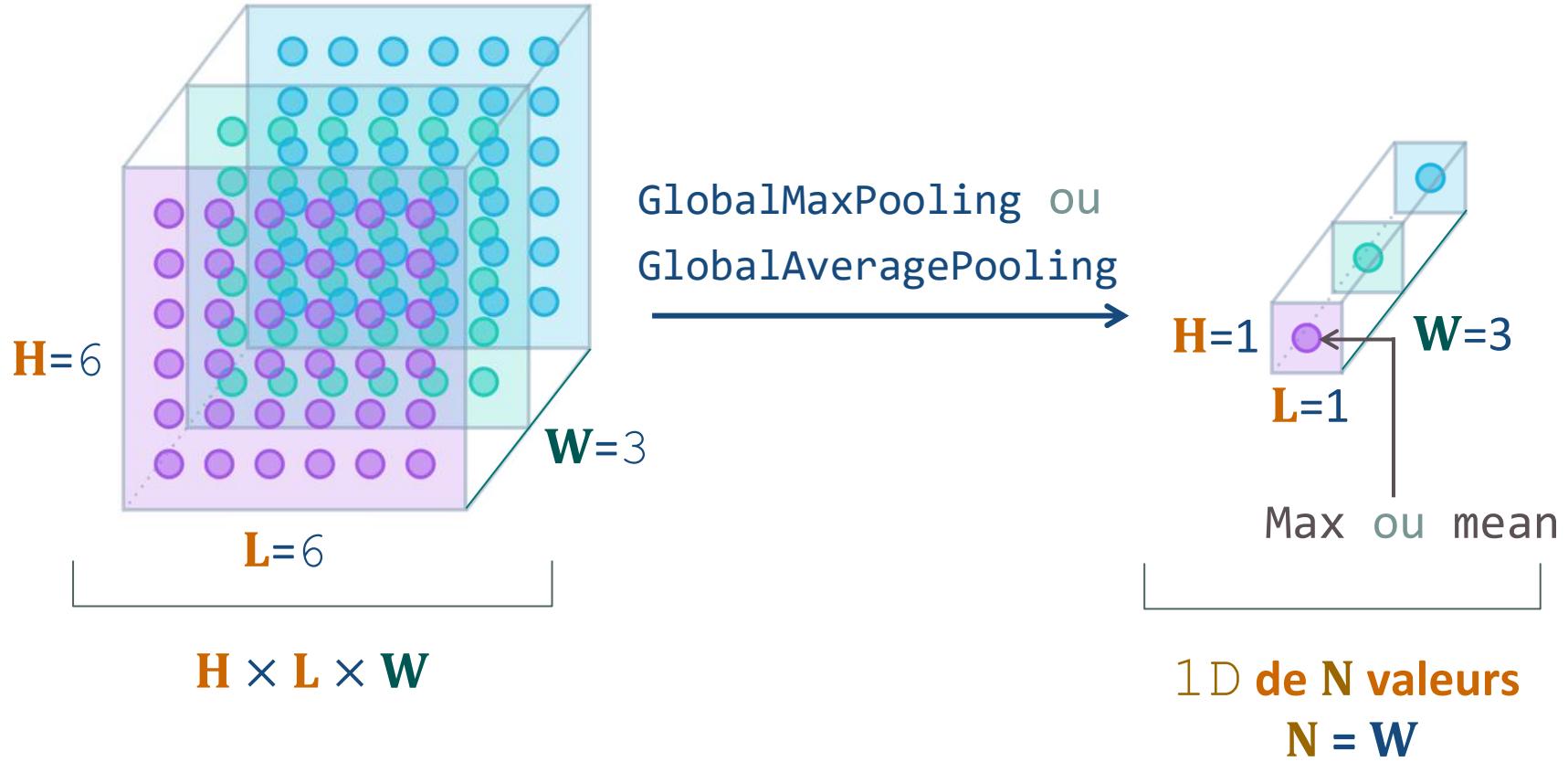


Input layer of Fully connected

1D de  $N$  valeurs

$$N = H \times L \times W$$

# Flattening GlobalMaxPooling/GlobalAveragePooling

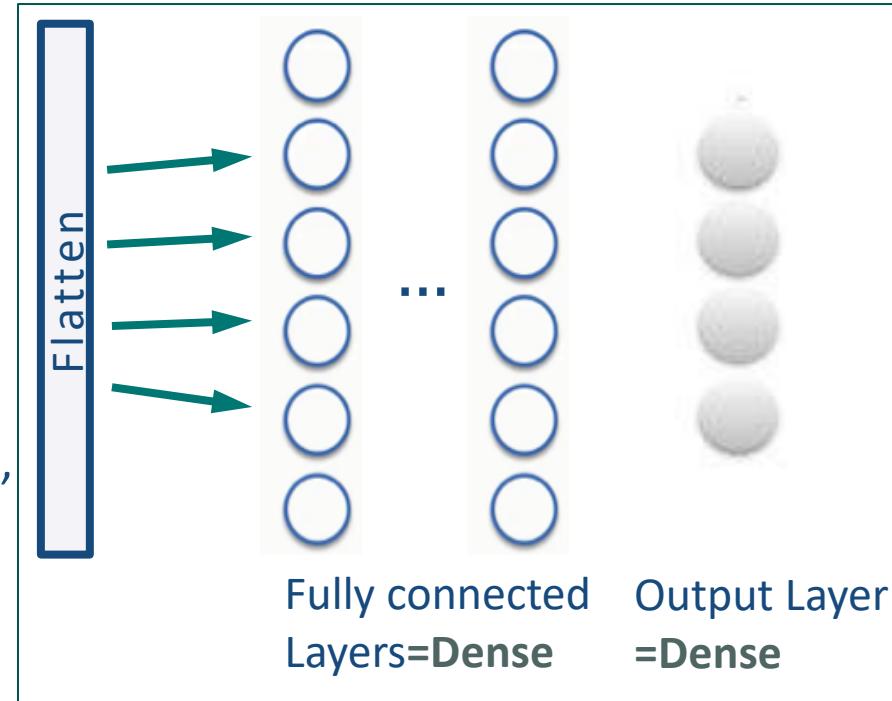


## Flatten/GlobalMaxPooling/GlobalAveragePooling

```
model = tf.keras.Sequential()  
:  
model.add(tf.keras.layers.Flatten())  
  
model.add(tf.keras.layers.GlobalMaxPool2D())  
  
model.add(tf.keras.layers.GlobalAveragePooling2D())
```

# Fully connected tf.keras.layers.Dense

```
model = tf.keras.Sequential()  
:  
model.add(tf.keras.layers.Dense(128,  
                               activation='relu'))  
  
#Par défaut  
tf.keras.layers.Dense(units, activation=None,  
                      use_bias=True,  
                      kernel_initializer='glorot_uniform',  
                      bias_initializer='zeros',  
                      kernel_regularizer=None,  
                      bias_regularizer=None,  
                      **kwargs)
```



# Quiz

1. Quel est l'intérêt d'ajouter une couche de Pooling lors de la création d'un réseau de neurones convolutif ? (seule réponse)

Réduire la taille de Feature maps

Extraire les descripteurs de l'image

Construire un modèle

Obtenir des données de longueur variable

2. Parmi les éléments suivants, lequel peut être l'entrée de la couche "Fully connected" (une seule réponse)

Un vecteur 2D de taille variable

Un vecteur 1D de taille variable

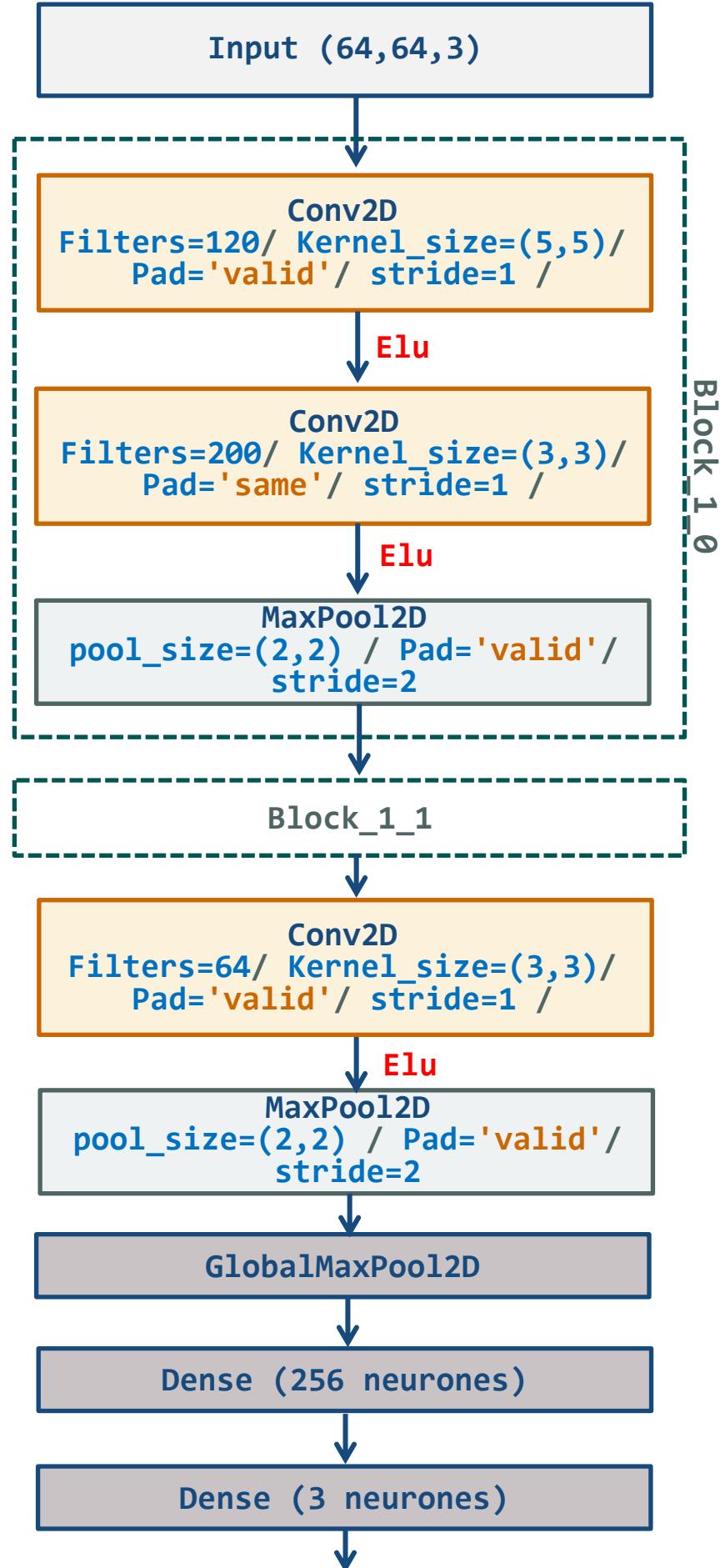
Un vecteur 2D de taille fixe

Un vecteur 1D de taille fixe

3. Les réseaux de neurones convolutifs permettent d'extraire automatiquement des descripteurs de haut niveaux à partir des images (Vrai/Faux)

Vrai

Faux



# Checkpointing .h5 (Poids ou Poids+structure)

```
model = tf.keras.Sequential()  
:  
callback = tf.keras.callbacks.ModelCheckpoint(filepath='/tmp/checkpoint.h5',  
                                              monitor='val_loss', save_best_only=True, save_weights_only=True, mode='min')  
history=model.fit(..., callbacks=[callback])
```

#Par défaut

```
tf.keras.callbacks.ModelCheckpoint(filepath, monitor='val_loss',  
                                  save_best_only=False, save_weights_only=False, mode='auto')
```

- **monitor='val\_loss'**, 'loss', 'val\_accuracy' ou 'accuracy'
- (**save\_best\_only=True**) Enregistrer uniquement le meilleur modèle selon **monitor**
- (**save\_weights\_only=True**) Seuls les poids du modèle seront enregistrés
- **mode='auto'** (selon monitor), 'max' (accuracy) ou bien 'min' (loss)

# Checkpointing pour chaque époque! .ckpt(Poids+structure)

```
model = tf.keras.Sequential()  
:  
filepath = '.../checkpoint{epoch:04d}.ckpt'  
  
callback = tf.keras.callbacks.ModelCheckpoint(filepath=filepath,  
                                              monitor='val_loss', mode='auto',  
                                              save_best_only=False, save_weights_only=False)  
  
history=model.fit(..., callbacks=[callback])
```



## CHAPITRE 3

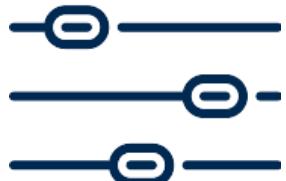
# SOLUTIONS AU PROBLÈME DE SUR-APPRENTISSAGE

# Solutions au problème de sur-apprentissage

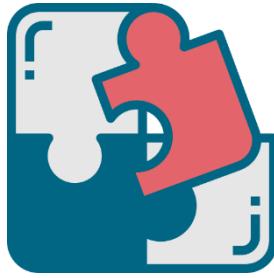


- ① Regularization
- ② Dataset Expansion (Data augmentation)
- ③ Early stopping
- ④ Batch Normalization

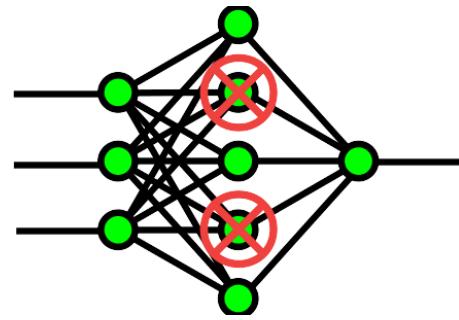
# 1- Regularization



$L_1$  Norm



$L_2$  Norm



Dropout

# 1- Regularization L<sub>1</sub> Norm, L<sub>2</sub> Norm

Regularization = Ajouter un terme de pénalité à la fonction loss E

L<sub>1</sub> Norm= LASSO regression (Least Absolute Shrinkage and Selection Operator)

$$E = E + \alpha_1 \|W\|_1$$

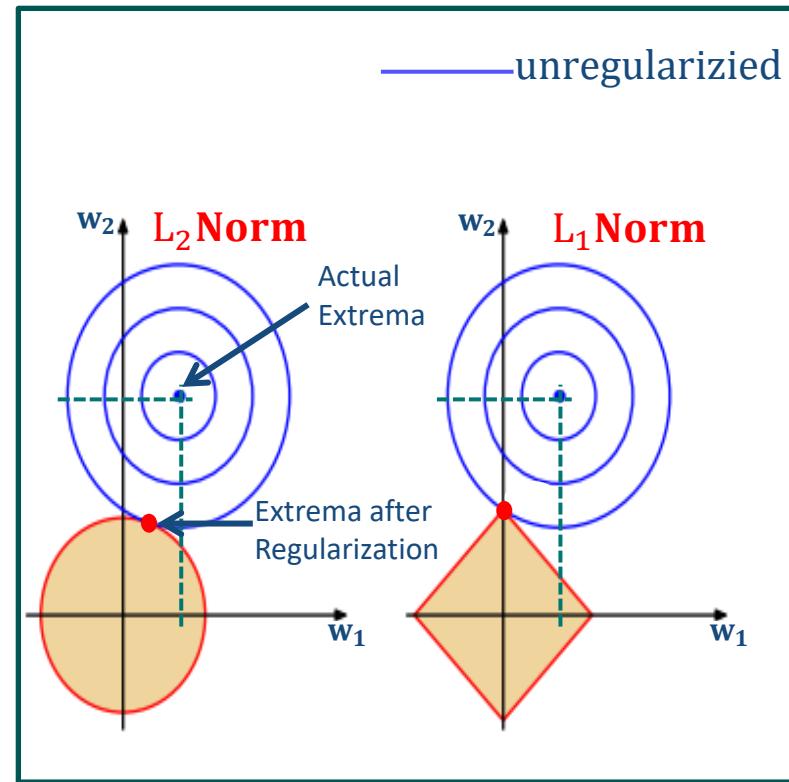
$$E = E + \alpha_1 \sum_{i=1}^k |w_i|$$

L<sub>2</sub> Norm= Ridge regression (squared magnitude)

$$E = E + \alpha_2 \|W\|_2$$

$$E(W) = E + \alpha_2 \sum_{i=1}^k w_i^2$$

k le nombre de poids



# 1- Regularization L<sub>1</sub> Norm, L<sub>2</sub> Norm

$$E = E + \alpha_1 L1 + \alpha_2 L2$$

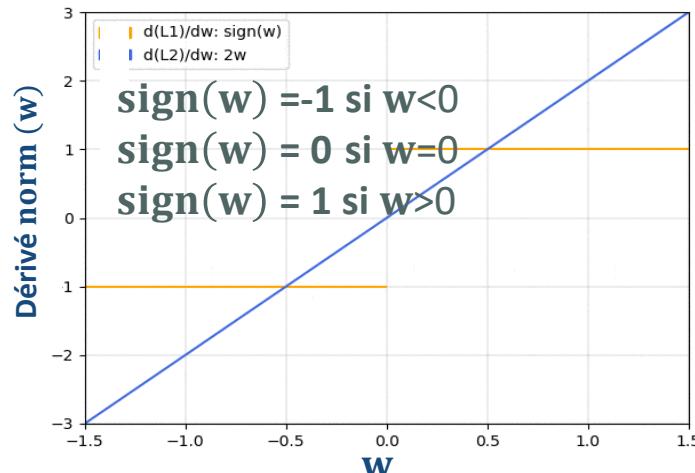
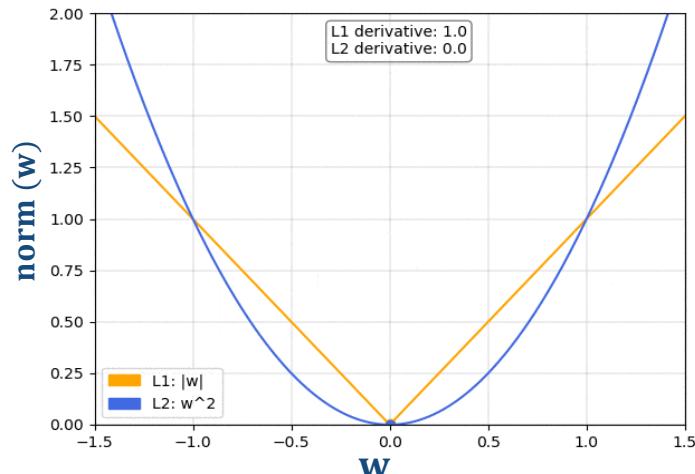
$$\begin{aligned} w &= w - \eta \frac{\partial E}{\partial w} \\ &= w - \eta \frac{\partial E}{\partial w} + \alpha_1 \frac{\partial L1}{\partial w} + \alpha_2 \frac{\partial L2}{\partial w} \\ &= w - \eta \frac{\partial E}{\partial w} + \alpha_1 \frac{\partial \sum_{i=1}^k |w_i|}{\partial w} + \alpha_2 \frac{\partial \sum_{i=1}^k w_i^2}{\partial w} \end{aligned}$$

$$\frac{\partial |x|}{\partial x} = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{si } x = 0 \\ -1 & \text{si } x < 0 \end{cases} \quad \frac{\partial x^2}{\partial x} = 2x \quad \frac{\partial ax+b}{\partial x} = a$$

$$= w - \eta \frac{\partial E}{\partial w} + \alpha_1 \text{sign}(w) + \alpha_2 2w$$

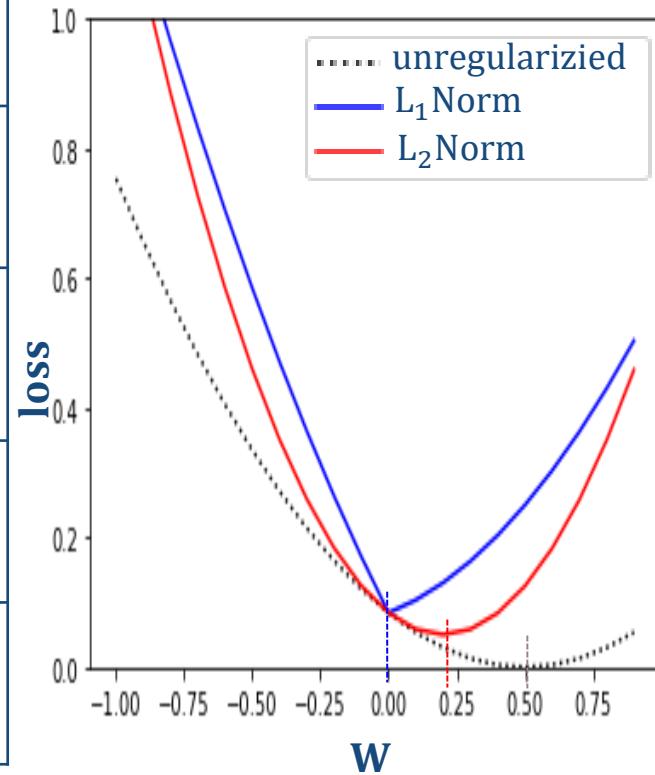
$\alpha_1, \alpha_2$ ? Hyperparamètres

$$\alpha_1 \in [0.0001 \dots 0.1] \quad \alpha_2 \in [0.0001 \dots 0.01]$$



# 1- Regularization L<sub>1</sub> Norm, L<sub>2</sub> Norm

	L <sub>1</sub> Norm	L <sub>2</sub> Norm
$W$	<ul style="list-style-type: none"><li>- Favorise la sparsité des poids</li><li>- Certains poids seront forcés à zéro</li></ul>	Tends vers 0 (mais pas exactement 0)
loss	↗ linéairement lorsque $w$ s'éloigne de 0	↗ de manière non linéaire $w$ s'éloigne de 0
Feature selection	✓	✗
Outlier detection	✓	✗



# 1- Regularization L<sub>1</sub> Norm, L<sub>2</sub> Norm

```
model = tf.keras.Sequential()  
:  
  
model.add(tf.keras.layers.Conv2D(..., kernel_regularizer=val,  
bias_regularizer=val, activity_regularizer=?))  
  
model.add(tf.keras.layers.Dense(..., kernel_regularizer=val,  
bias_regularizer=val, activity_regularizer=?))
```

- NB.
- val peut être **I1**, **I2** ou **I1\_I2** (par défaut  $\alpha=0.01$ )
  - val peut être **tf.keras.regularizers.I1(I1=0.01)** ou **tf.keras.regularizers.I2(I2=0.01)**
  - **activity\_regularizer** ?

# activity\_regularizer L<sub>1</sub>Norm, L<sub>2</sub>Norm

— **activity\_regularizer** = spécifier une fonction de régularisation à appliquer sur les sorties de couche

- ✓ **activity\_regularizer** est appliquée à chaque activation de neurone de la couche en question
- ✓ la somme des résultats est ajoutée à la fonction de perte globale du modèle

```
model.add(tf.keras.layers.Conv2D(activity_regularizer=val))
```

val peut être **tf.keras.regularizers.l1(l1=0.01)** ou **tf.keras.regularizers.l2(l2=0.01)**

# 1- Regularization Dropout

CNN block d'extraction des caractéristiques et bloc de classification

DROPOUT ou retire aléatoirement les neurones (block classification)

- Supprime de manière aléatoire  $M$  entrées au cours du processus d'apprentissage
- Les poids et biais correspondant aux entrées rejetées ne sont pas mis à jour

$n$  = nombre de neurones dans une couche d'entrée ou cachée

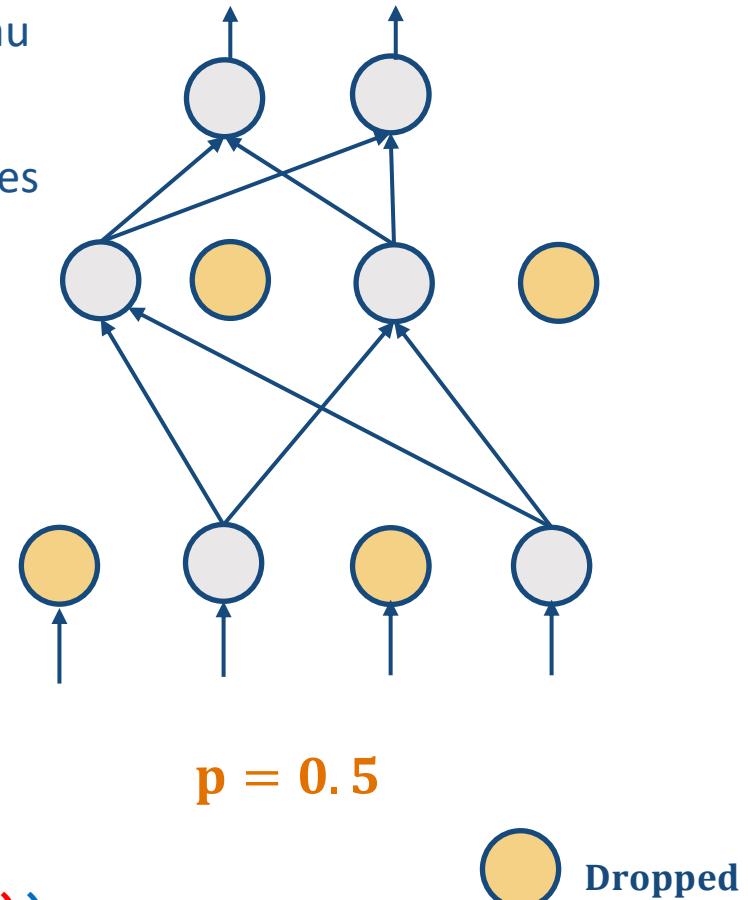
$p$  = Dropout rate ( $p \in [0..1]$ )

$$M = n \times p$$

```
model = tf.keras.Sequential()
```

:

```
model.add(tf.keras.layers.Dropout(0.5))
```



## 2- Data augmentation (DA)

Original



Horizontal Flipped



Vertically Flipped



+90 Rotation



-90 Rotation



+60 Rotation



Resized



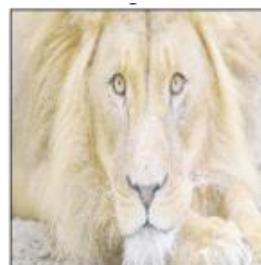
Rescaled



Cropped



Brighter



Darker



Noise added



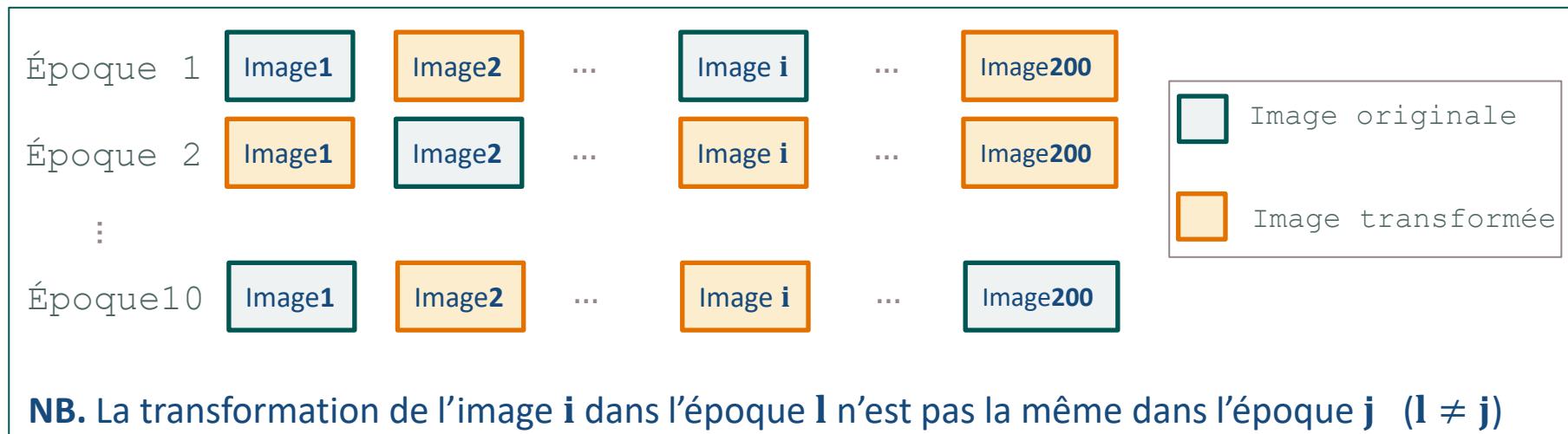
## 2- Data augmentation (DA)

```
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.Reshape(target_shape, **kwargs),
    tf.keras.layers.Resizing(height, width, **kwargs),
    tf.keras.layers.Rescaling(scale, offset=0.0, **kwargs),
    #scale=1./255, offset=0.0 ou scale=1./127.5, offset=-1
    tf.keras.layers.RandomFlip(mode="horizontal_and_vertical", **kwargs),
    #mode="horizontal", "vertical" ou bien "horizontal_and_vertical"
    tf.keras.layers.RandomRotation(factor, **kwargs) #factor ∈ [-0.2, 0.3]
    tf.keras.layers.RandomZoom(height_factor, width_factor = None, **kwargs),
    #height_factor ∈ [0.2, 0.3] ou height_factor ∈ [-0.3, 0.2]
    tf.keras.layers.RandomBrightness(factor, value_range = (0, 255), **kwargs) #factor ∈ [-1, 1]
    tf.keras.layers.RandomContrast(factor = val, **kwargs) #factor ∈ [1-val, 1+val]
    tf.keras.layers.RandomCrop(height, width, **kwargs)
])
```

## 2- Data augmentation (DA)

### Exemple:

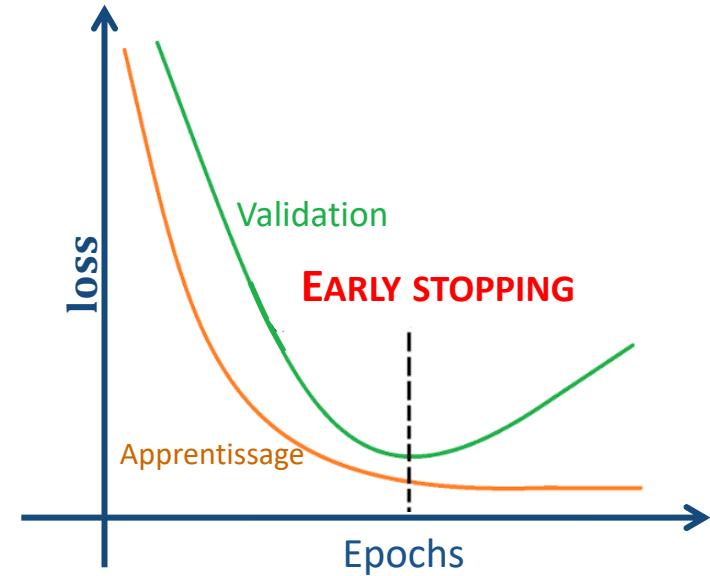
- Une base de donnée de 200 images
- Nombre d 'époques=10
- En appliquant **Data augmentation**, le nombre des images lors de l'apprentissage ne devient pas > 200 ?



### 3- Early stopping Nombre d'époques!

- Arrêter l'apprentissage une fois que la performance du modèle cesse de s'améliorer sur un ensemble de validation

**NB.** — Un grand nombre d'époques peut conduire à un sur-apprentissage  
— Un nombre insuffisant d'époques peut entraîner un sous-apprentissage



`tf.keras.callbacks.EarlyStopping` ?

## tf.keras.callbacks.EarlyStopping

```
model = tf.keras.Sequential()  
:  
  
callback= tf.keras.callbacks.EarlyStopping(monitor="val_loss", patience=2)  
  
history=model.fit(..., callbacks=[callback])
```

#Par défaut

```
tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=0, restore_best_weights=False)
```

—monitor = 'val\_loss', 'loss', 'val\_accuracy' ou 'accuracy'

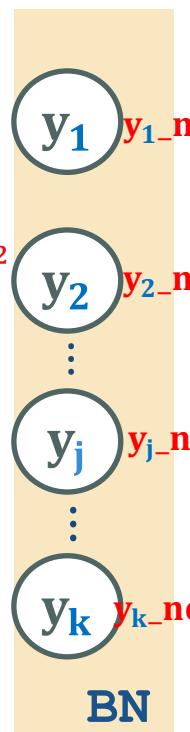
—patience = Nombre d'époques sans amélioration

## 4- Batch Normalization BN

- Introduit une couche supplémentaire dans le réseau de neurones, qui permet de normaliser les entrées d'une couche (avant ou après l'activation) pour **chaque mini-batch**

Étant donnée un mini-batch **B** de **m** observations / une couche de **k** neurones

- 1- Calcul de moyenne **mean\_B<sub>j</sub>** de mini-batch **B** sur la neurone **y<sub>j</sub>**  $\text{mean\_B}_j = \frac{1}{m} \sum_{i=1}^m y_j^{(i)}$
- 2- Calcul de variance **σ\_B<sub>j</sub>** de mini-batch **B** sur la neurone **y<sub>j</sub>**  $\sigma_B_j = \frac{1}{m} \sum_{i=1}^m (y_j^{(i)} - \text{mean\_B}_j)^2$
- 3- Normalisation de chaque **y<sub>j</sub>** de chaque observation **i** ( $i = 1..m$ ) de **B** **y<sub>j</sub><sup>(i)</sup>\_nor**  
$$y_j^{(i)}\_nor = \frac{y_j^{(i)} - \text{mean\_B}_j}{\sqrt{\sigma_B_j + \varepsilon}}$$
 avec  $\sqrt{\sigma_B_j}$  = Ecart type et  $\varepsilon = 10^{-3}$
- 4- Scale & shift  $y_j^{(i)}\_nor = \gamma \times y_j^{(i)}\_nor + \beta$        $\gamma?$     $\beta?$



## 4- Batch Normalization BN

$$\gamma_j = 1 \quad ; \quad \beta_j = 0$$

Pour chaque neurone  $j$   $y_j$

**moving\_mean<sub>j</sub>=0** #Initialisation de moyenne de batch  $B$  sur la neurone  $y_j$

**moving\_variance<sub>j</sub>=1** Initialisation de variance de de batch  $B$  sur la neurone  $y_j$

Pour chaque batch  $B$  de  $m$  observations

$$\text{mean\_B}_j = \frac{1}{m} \sum_{i=1}^m y_j^{(i)}$$

$$\sigma_B_j = \frac{1}{m} \sum_{i=1}^m (y_j^{(i)} - \text{mean\_B}_j)^2$$

$$\text{moving\_mean}_j = \text{momentum} \times \text{moving\_mean}_j + (1-\text{momentum}) \text{mean\_B}_j$$

$$\text{moving\_variance}_j = \text{momentum} \times \text{moving\_variance}_j + (1-\text{momentum}) \sigma_B_j$$

momentum=0 . 99



$$\text{gradient}_{\gamma_j} = \sum_{i=1}^m \frac{\partial E}{\partial \gamma_j} \quad \text{gradient}_{\beta_j} = \sum_{i=1}^m \frac{\partial E}{\partial \beta_j}$$

Pour chaque observation  $i$  ( $i = 1 \dots m$ ) de  $B$

$$\gamma_j = \gamma_j - \eta \text{ gradient}_{\gamma_j} \quad \beta_j = \beta_j - \eta \text{ gradient}_{\beta_j}$$

$$y_j^{(i)}_{\text{nor}} = \gamma \times \frac{y_j^{(i)} - \text{mean\_B}_j}{\sqrt{\sigma_B_j + \epsilon}} + \beta$$



## 4- Batch Normalization BN



### Learnable parameters $\gamma$ / $\beta$

- $\gamma$ = vecteur de poids initialisé à 1       $\gamma = \gamma - \eta \frac{\partial E}{\partial \gamma}$
- $\beta$ = vecteur de biais initialisé à 0       $\beta = \beta - \eta \frac{\partial E}{\partial \beta}$
- La mise à jour de  $\gamma$  et  $\beta$  se fait généralement après chaque mini-batch, et non après le parcours de toutes les mini-batches pour former une époque (Mini-Batch Stochastic Gradient Descent)



### Saved parameters `moving_mean/moving_variance`

- **`moving_mean`**= vecteur de moyennes des batch initialisé à 0
- **`moving_variance`**= vecteur de variances des batch initialisé à 1
- Utilisés lors de l'inférence (test)



## 4- Batch Normalization BN



BN peut être plus approprié après la fonction d'activation **sigmoïde**

**sigmoïde** ]0..1[



BN peut être plus approprié avant la fonction d'activation **relu ou ses variantes**

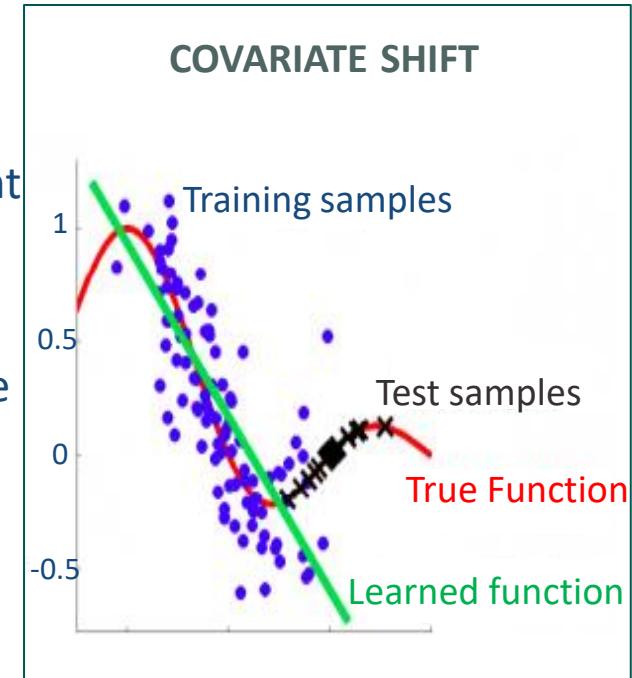
**relu** [0, +∞[

- Rééquilibrer la distribution des activations en centrant la distribution autour de zéro et en réduisant sa variance

## 4- Batch Normalization BN Avantages

vanishing gradient probleme qui empeche le modele converger

- Réduire le problème de Vanishing Gradient
- Améliorer la stabilité de l'apprentissage en accélérant ainsi la convergence de l'entraînement
- Améliorer les performances de généralisation du modèle
- Réduire l'instabilité des distributions (**COVARIATE SHIFT**)
- Réduire le sur-apprentissage



si les metrique de performance sont proche le modele converge

# tf.keras.layers.BatchNormalization

```
model = tf.keras.Sequential()  
:  
model.add(tf.keras.layers.BatchNormalization())
```

#Par défaut

```
tf.keras.layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001,  
        center=True, scale=True, beta_initializer='zeros', gamma_initializer='ones',  
        moving_mean_initializer='zeros', moving_variance_initializer='ones',  
        beta_regularizer=None, gamma_regularizer=None, **kwargs)
```

— **axis=-1** (dernier axe)

sachant que le tensor d'entrée a une forme (**batch\_size, height, width, channels**)

— **center=True** ⇒ Permettre l'apprentissage des biais **beta**

— **scale=True** ⇒ Permettre l'apprentissage des poids **gamma**

# Quiz

## 1. Qu'est-ce que la data augmentation ? (plusieurs réponses)

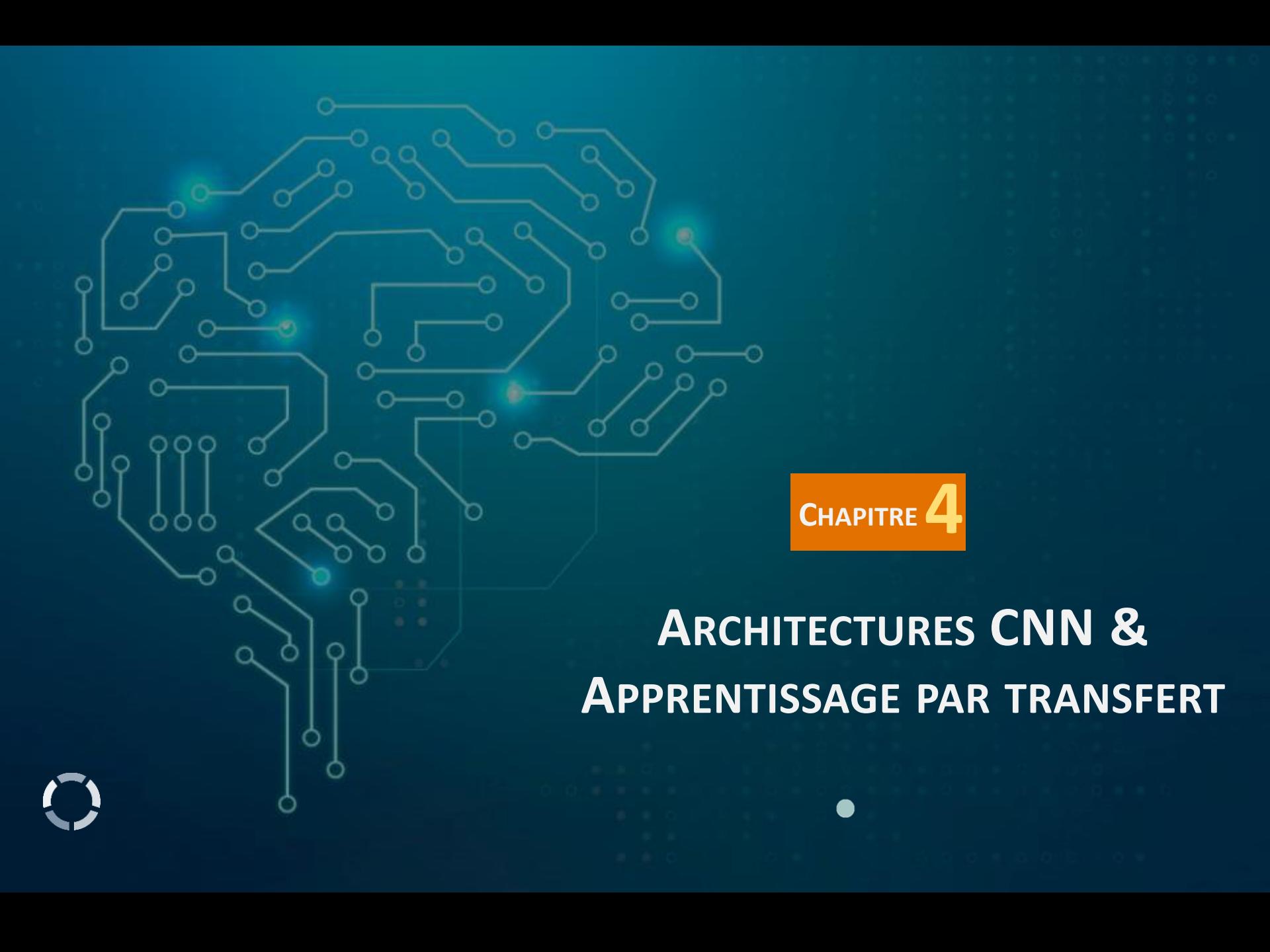
- L'ajout de bruit dans les données
- L'augmentation de la taille des données
- La modification des données existantes pour créer de nouvelles données
- L'amélioration de la généralisation

## 2. Parmi les affirmations suivantes, quelles sont celles qui sont correctes ? (plusieurs réponses)

- La régularisation L2 ajoute une pénalité égale à la somme des valeurs absolues des poids
- La régularisation L1 peut être utilisée pour la sélection de descripteurs
- Le Dropout est une méthode de régularisation pour les réseaux de neurones
- Le Dropout est principalement utilisé pour éviter le sur-apprentissage

## 3. La Batch Normalization peut aider à accélérer la convergence de l'apprentissage d'un réseau de neurones profond (une seule réponse)

- Vrai
- Faux



## CHAPITRE 4

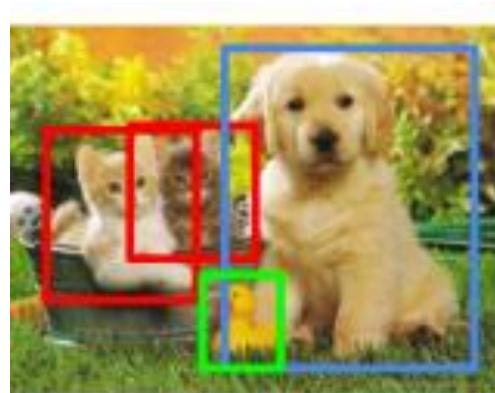
# ARCHITECTURES CNN & APPRENTISSAGE PAR TRANSFERT

# Utilisations de CNN!

## CLASSIFICATION



## OBJECT DETECTION



1 seul objet  
1 seule classe

## INSTANCE SEGMENTATION

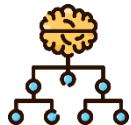


Localisation+ Classification  
1 ou plusieurs objets

Contour+ Mask

# Architectures CNN Classification

- LeNet-5
- AlexNet
- VGG16 / VGG19
- ResNet / ResNeXt
- SENet
- DenseNet
- GoogleNet (Inception)
- Inception-ResNet
- EfficientNet
- MobileNet



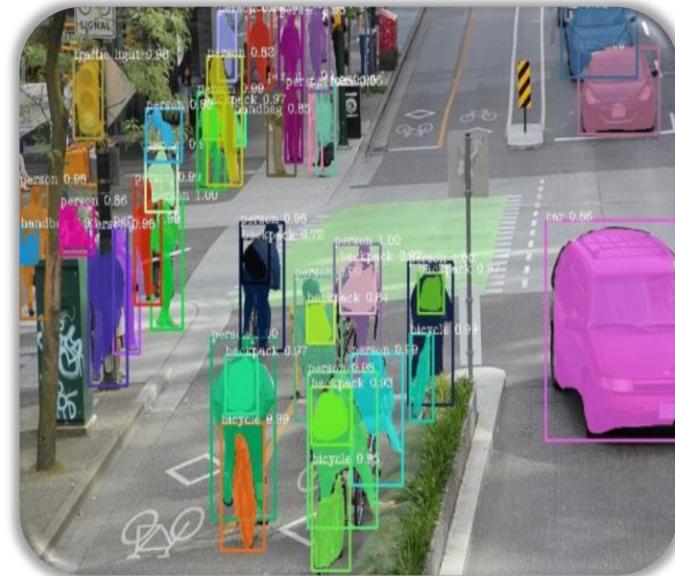
## Architectures CNN Object detection/Instance segmentation

# OBJECT DETECTION

- R-CNN
  - Fast R-CNN
  - Faster R-CNN
  - You Only Look Once (YOLO)

# OBJECT DETECTION+INSTANCE SEGMENTATION

- FCNs
  - Mask R-CNN
  - U-Net



# QUELQUES ARCHITECTURES CNN POUR LA CLASSIFICATION

# VGG16

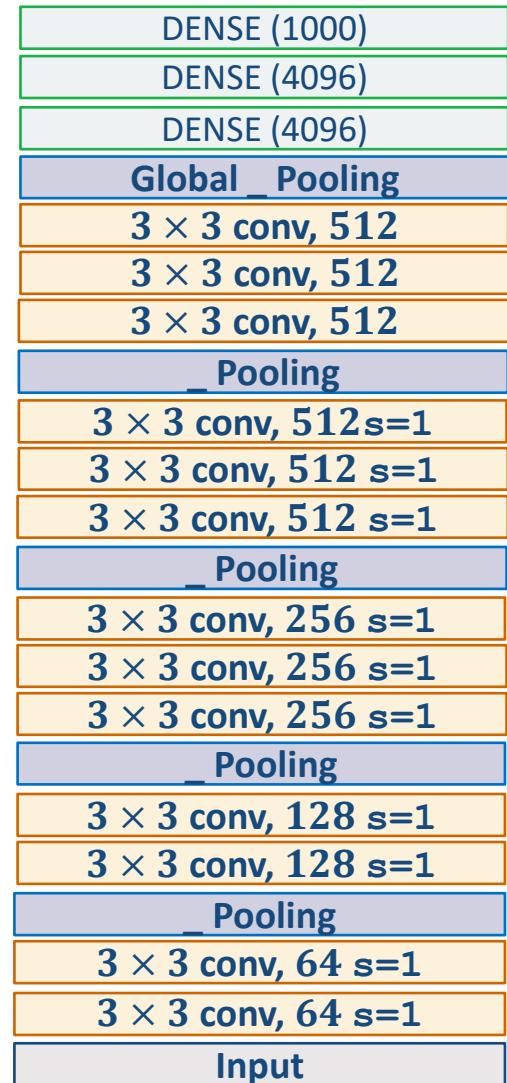
## 13 Couches de convolution

- 4 Couches de Pooling (MaxPooling ou AveragePooling)
- 1 Couche GlobalMaxPooling ou GlobalAveragePooling
- 3 Couches Fully Connected

#Par défaut

```
tf.keras.applications.vgg16.VGG16(include_top=True,  
weights='imagenet', input_tensor=None, input_shape=None,  
pooling=None, classes=1000, classifier_activation='softmax')
```

- **include\_top=True** Inclure 3 FCs Layers & **input\_shape** doit être (224,224,3)
- **weights='imagenet'** ou **None** (random initialization) ou '**path**'
- **pooling=None** si **include\_top=True** / **Global \_\_Pooling = Flatten**
- **pooling= 'avg'** ou **'max'** si **include\_top=False**
- **classes=1000** (à spécifier uniquement si **include\_top=True**)
- **classifier\_activation='softmax'** (à spécifier uniquement si **include\_top=True**)
- Toutes les couches de convolution ayant **padding="same"**



# Residual neural network (ResNet) Residual bloc



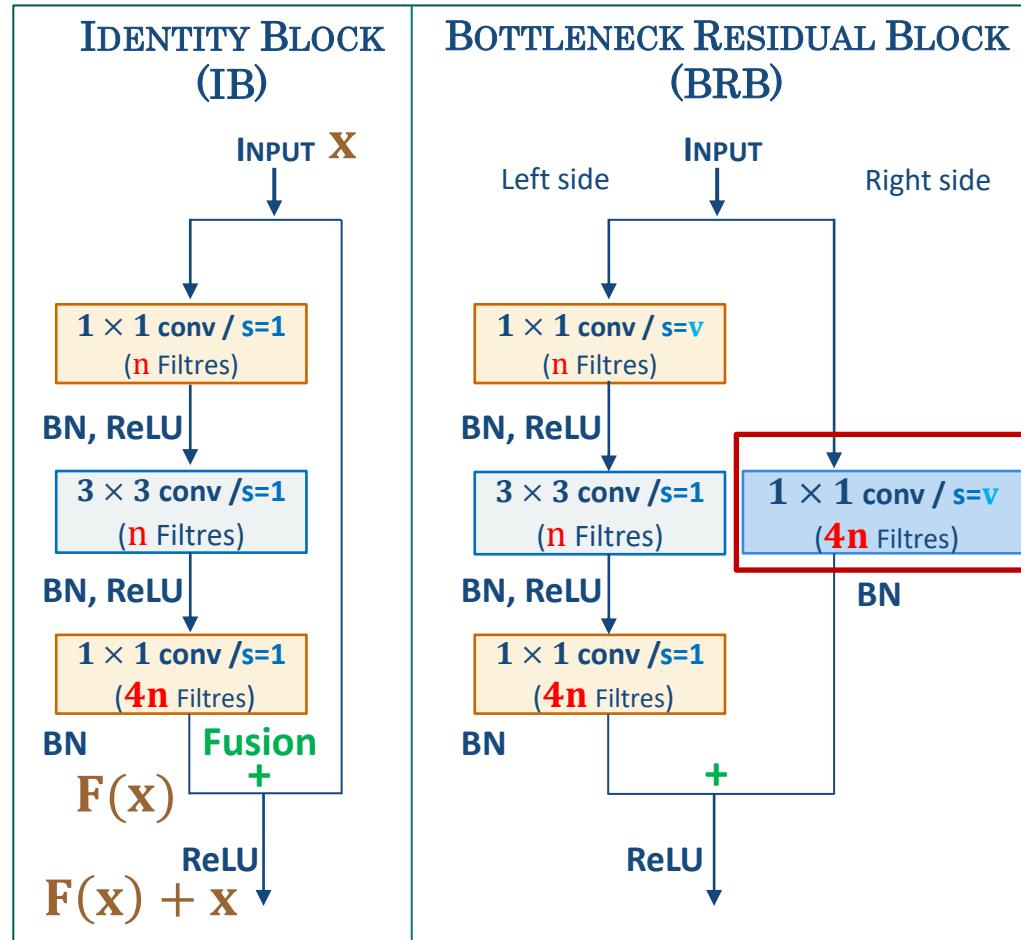
ResNet comprend à la fois des blocs résiduels de type :

— **IDENTITY BLOCK (IB)** : permet de propager l'information originale sans la modifier, tout en permettant au réseau

$x$  et  $F(x)$  doivent avoir la même taille, pour pouvoir s'additionner)

— **BOTTLENECK RESIDUAL BLOCK (BRB)** : réduit la dimensionnalité de l'entrée, permettant de réduire le coût computationnel tout en maintenant la représentation

— Toutes les couches de convolution ayant  
**padding="same"**

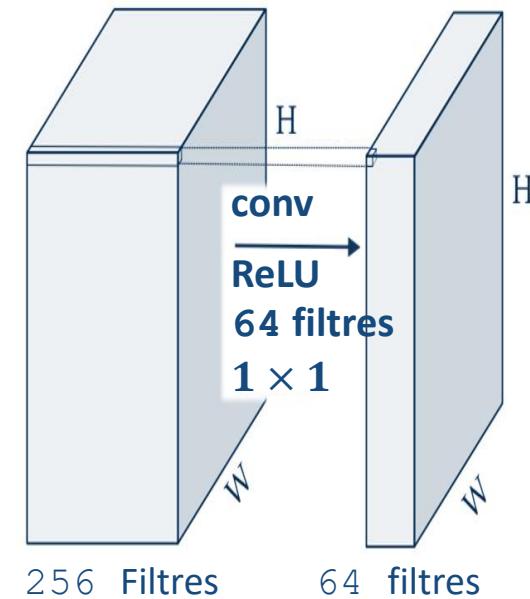


ResNet architecture decouche cache très profonde  $\geq 30$  éliminier d'aller en profondeur mais en cardinalité Métrique de performance Fops: nb d'opreation + au  $X$  par architecture ==> indication sur la complexité du modèle

# Convolution $1 \times 1$

- S'assurer que les dimensions de l'entrée et de la sortie sont compatibles pour l'opération d'addition
- Effectuent une opération de convolution sur chaque pixel de l'image en prenant en compte uniquement l'information de chaque canal, sans tenir compte des relations spatiales entre les pixels
- **Augmentation de la non-linéarité** : ajouter des couches d'activation non linéaires après les convolutions

**$1 \times 1$  conv / s=—  
— Filtres**

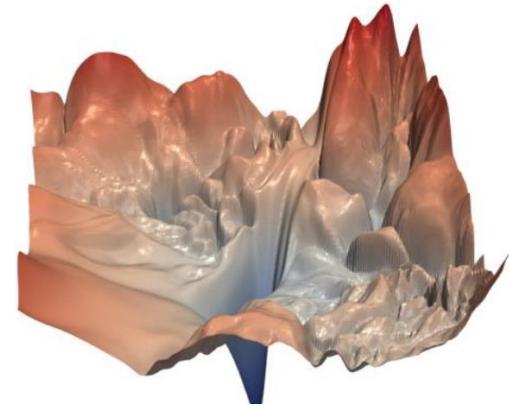


# ResNet Lissage de la fonction de coût!

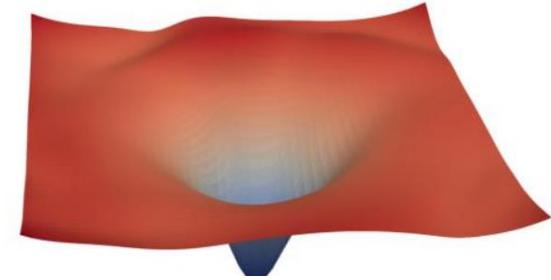


Residual bloc: permet au réseau de sauter des couches lors de la rétropropagation du gradient

- Réduit le risque de Vanishing gradient: contourner certaines transformations dans les blocs résiduels
- Stabiliser l'apprentissage dans les réseaux de neurones
- Eviter un apprentissage trop brusque ou instable, en rendant la fonction de coût plus régulière et plus lisse



Sans Convolution  $1 \times 1$



Avec Convolution  $1 \times 1$

# ResNet50/ResNet101/ResNet152

- ResNet50 : conv2D, Residual bloc\_1 (BRB + IB×2), Residual bloc\_2 (BRB + IB×3), Residual bloc\_3 (BRB + IB×5), Residual bloc\_4 (BRB + IB×2), FC

**tf.keras.applications.resnet50.ResNet50()**

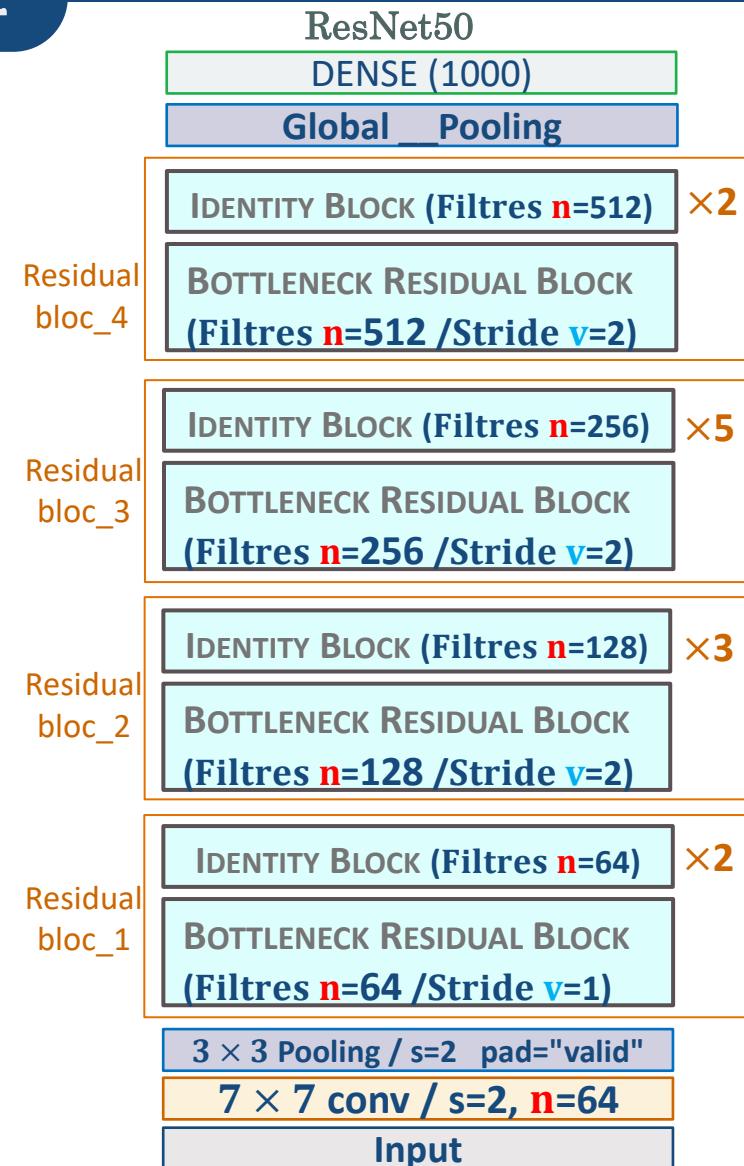
```
include_top=True,weights='imagenet', input_shape=None,  
pooling=None, classes=1000, classifier_activation='softmax')
```

- ResNet101 : conv2D, Residual bloc\_1 (BRB + IB×2), Residual bloc\_2 (BRB + IB×3), Residual bloc\_3 (BRB + IB×22), Residual bloc\_4 (BRB + IB×2), FC

**tf.keras.applications.resnet.ResNet101(...)**

- ResNet101 : conv2D, Residual bloc\_1 (BRB + IB×2), Residual bloc\_2 (BRB + IB×7), Residual bloc\_3 (BRB + IB×35), Residual bloc\_4 (BRB + IB×2), FC

**tf.keras.applications.resnet.ResNet152(...)**



# DenseNet Dense\_bloc



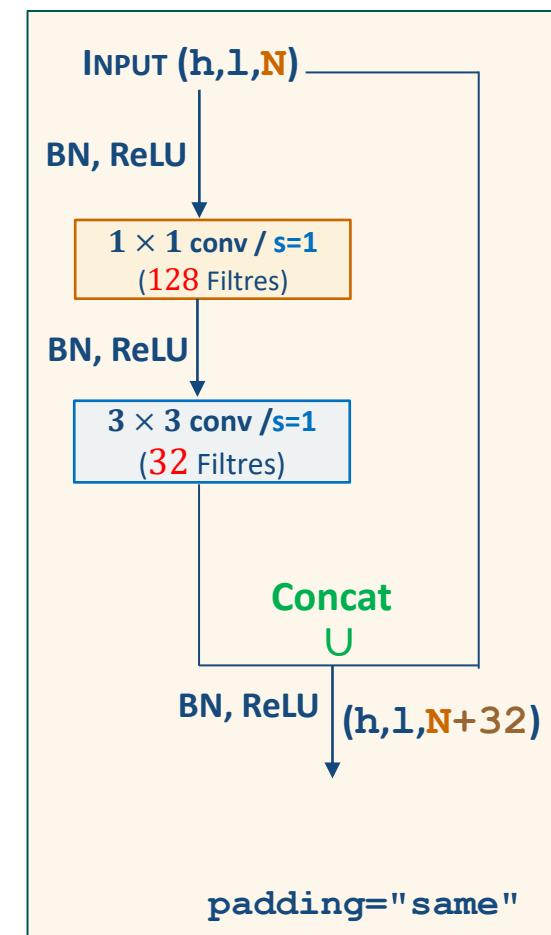
Comporte deux couches de convolution :

- Convolution  $1 \times 1$  avec 128 filtres : réduire la dimensionnalité des descripteurs spatiaux tout en préservant les informations importantes
- Convolution  $3 \times 3$  avec 32 filtres : Extraction des descripteurs
- Les feature maps résultantes de deuxième couche de convolution sont concaténées avec les feature maps d'entrée  $(h, l, N)$ . BN suivie d'une activation ReLU est appliquée entre les couches et à la sortie de la concaténation.



Dense\_bloc contribue à maximiser les échanges d'informations entre les couches, ce qui améliore la généralisation du modèle

Dense\_bloc



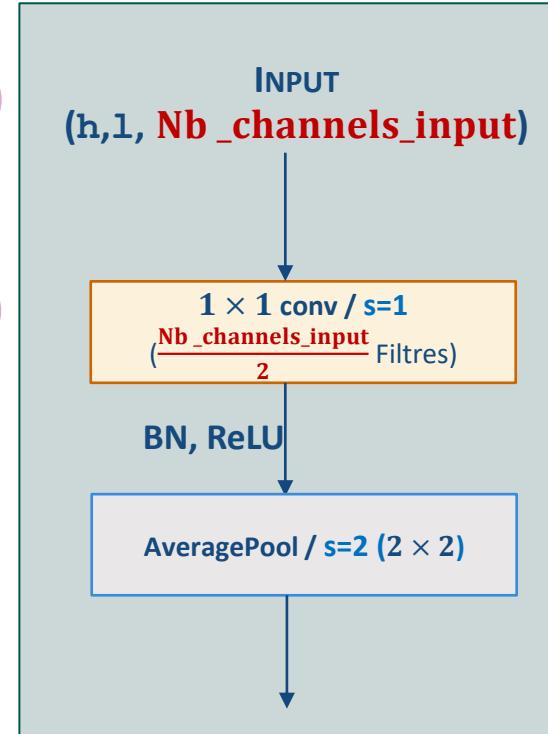
# DenseNet Transition\_layer



Comporte deux couches:

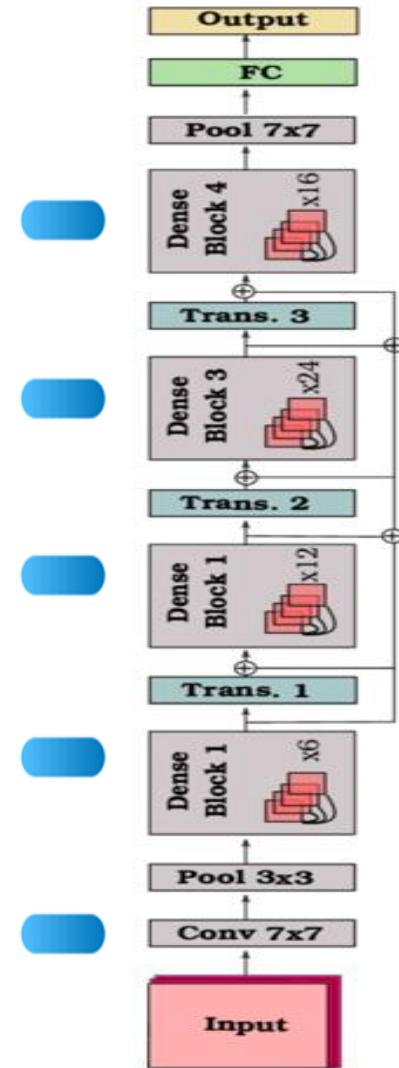
- Convolution  $1 \times 1$  avec  $\frac{\text{Nb\_channels\_input}}{2}$  filtres : réduire le nombre de canaux
- Couche de pooling Averagepool2D avec un pool\_size et un stride égal à  $(2, 2)$  : réduire à la dimension spatiale des caractéristiques
- Toutes les couches de convolution et pooling ayant padding="same"

## Transition\_layer



# DenseNet121/ 169/ 201/264

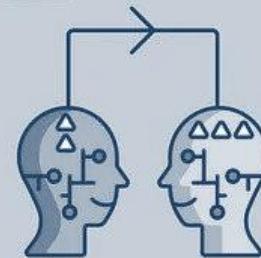
	DenseNet121	DenseNet169	DenseNet201	DenseNet264
Conv2D	$1 \times 1$ Conv/ stride=2			
MaxPool2D	$3 \times 3$ MaxPool/ stride=2			
Dense_bloc1	$1 \times 1$ Conv $\times 6$ $3 \times 3$ Conv	$1 \times 1$ Conv $\times 6$ $3 \times 3$ Conv	$1 \times 1$ Conv $\times 6$ $3 \times 3$ Conv	$1 \times 1$ Conv $\times 6$ $3 \times 3$ Conv
Transition_layer 1	$1 \times 1$ Conv/ stride=1			
	$2 \times 2$ AveragePool/ stride=2			
Dense_bloc2	$1 \times 1$ Conv $\times 12$ $3 \times 3$ Conv	$1 \times 1$ Conv $\times 12$ $3 \times 3$ Conv	$1 \times 1$ Conv $\times 12$ $3 \times 3$ Conv	$1 \times 1$ Conv $\times 12$ $3 \times 3$ Conv
Transition_layer 2	$1 \times 1$ Conv/ stride=1			
	$2 \times 2$ AveragePool/ stride=2			
Dense_bloc3	$1 \times 1$ Conv $\times 24$ $3 \times 3$ Conv	$1 \times 1$ Conv $\times 32$ $3 \times 3$ Conv	$1 \times 1$ Conv $\times 48$ $3 \times 3$ Conv	$1 \times 1$ Conv $\times 64$ $3 \times 3$ Conv
Transition_layer 3	$1 \times 1$ Conv/ stride=1			
	$2 \times 2$ AveragePool/ stride=2			
Dense_bloc4	$1 \times 1$ Conv $\times 24$ $3 \times 3$ Conv	$1 \times 1$ Conv $\times 32$ $3 \times 3$ Conv	$1 \times 1$ Conv $\times 32$ $3 \times 3$ Conv	$1 \times 1$ Conv $\times 48$ $3 \times 3$ Conv
Classification Layer	Global Average pooling			
	Dense (1000) / softmax			



`tf.keras.applications.densenet.DenseNetXXX(...)`

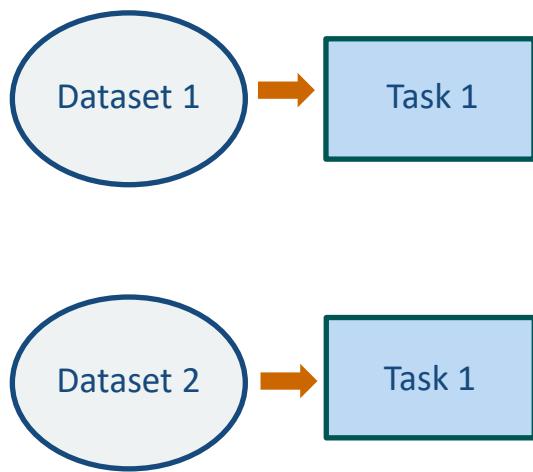
XXX= 121 ou 169 ou 201 ou 264

# TRANSFER LEARNING



# Apprentissage par transfert (Transfer Learning)

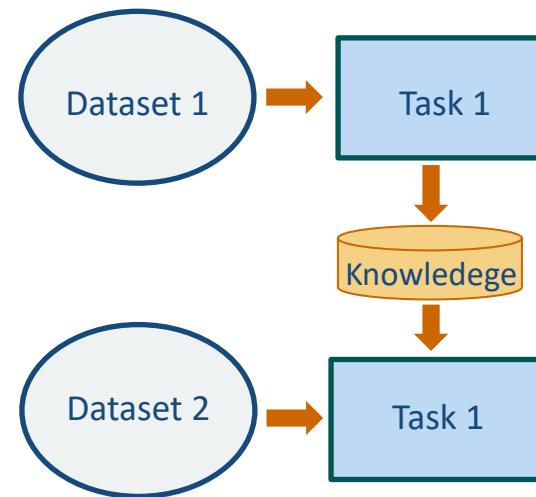
## Traditional ML



Les connaissances ne sont ni conservées ni accumulées

Transférer l'apprentissage d'un modèle à un autre modèle quand on utilise transfert learning  
-petit dataset  
-même besoin du modèle déjà développé

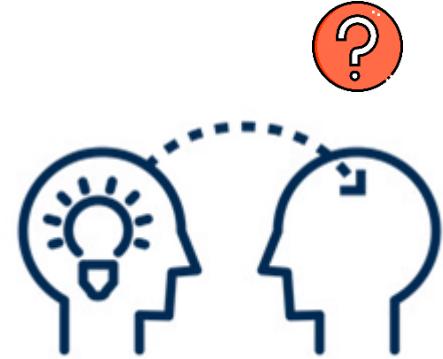
## Transfer Learning



Transférer des connaissances d'une tâche connexe qui a déjà été apprise

# Avantages de Transfer Learning

- 😊 Modèle déjà existant : même tâches
- 😊 Pas assez de données d'apprentissage **vs.** l'apprentissage "**from scratch**"
- 😊 Pas assez de puissance de calcul : Accélère considérablement l'apprentissage
- 😊 Gain de temps général ✓



# Deep Transfer Learning **Fine-tuning**



Frozen Pre-Trained Model

`trainable = false`



Unfrozen Pre-Trained Model

`trainable=true`  
les poid sa  
change

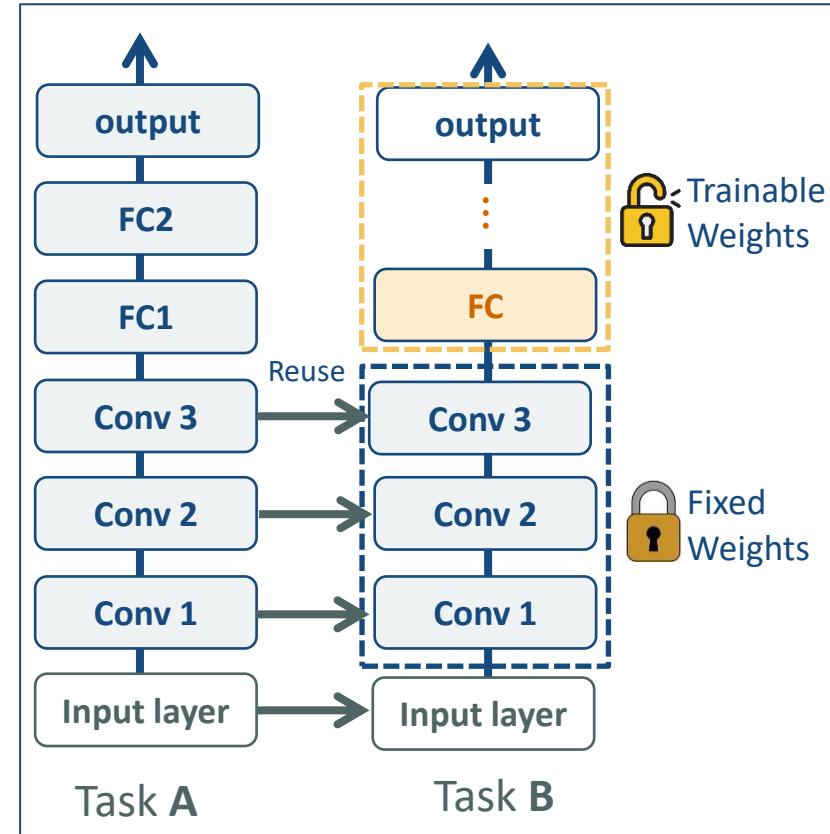
# Frozen Pre-Trained Model

- Charger les poids du modèle pré-entraîné
- Définir une ou plusieurs **couche(s) de Fully connected FC**

 Ne pas ajuster les poids des couches convolutives du modèle pré-entraîné

**model.trainable = False**

 Ajuster les poids des FC(s) en utilisant les nouveaux données

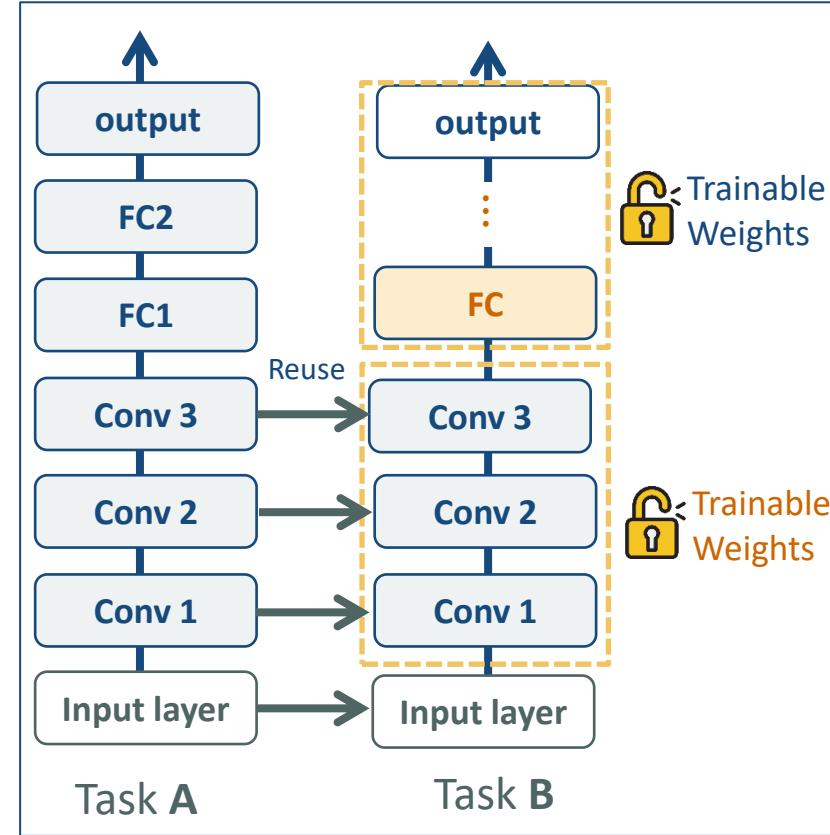


# Unfrozen Pre-Trained Model



Ajuster les poids en utilisant les nouveaux données

- des couches convolutifs du modèle pré-entraîné  
`model.trainable= True`
- des FC(s)

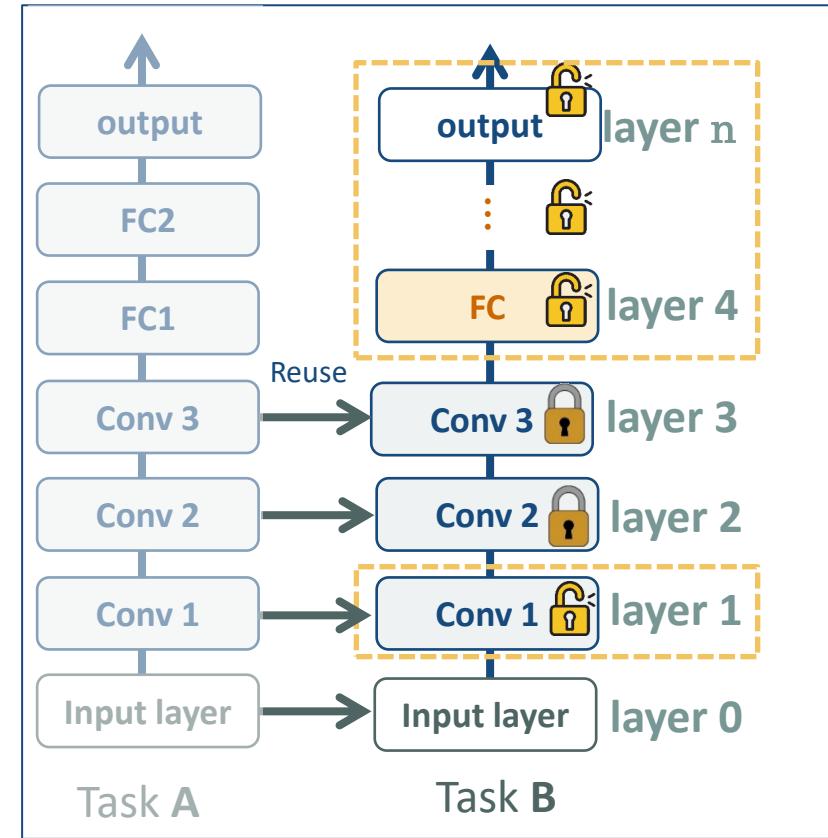


# Fine-Tuning sur quelques Layers!

`model.layers[1].trainable= True`

`model.layers[2:4].trainable= False`

`model.layers[4:n+1].trainable= True`



# Sequential Vs Functional API !



## Sequential API

Créer le modèle couche par couche

Le partage des poids (weights sharing) n'est pas autorisé

N'est pas possible d'avoir plusieurs entrées ou sorties

`tf.keras.Sequential()`



## Functional API

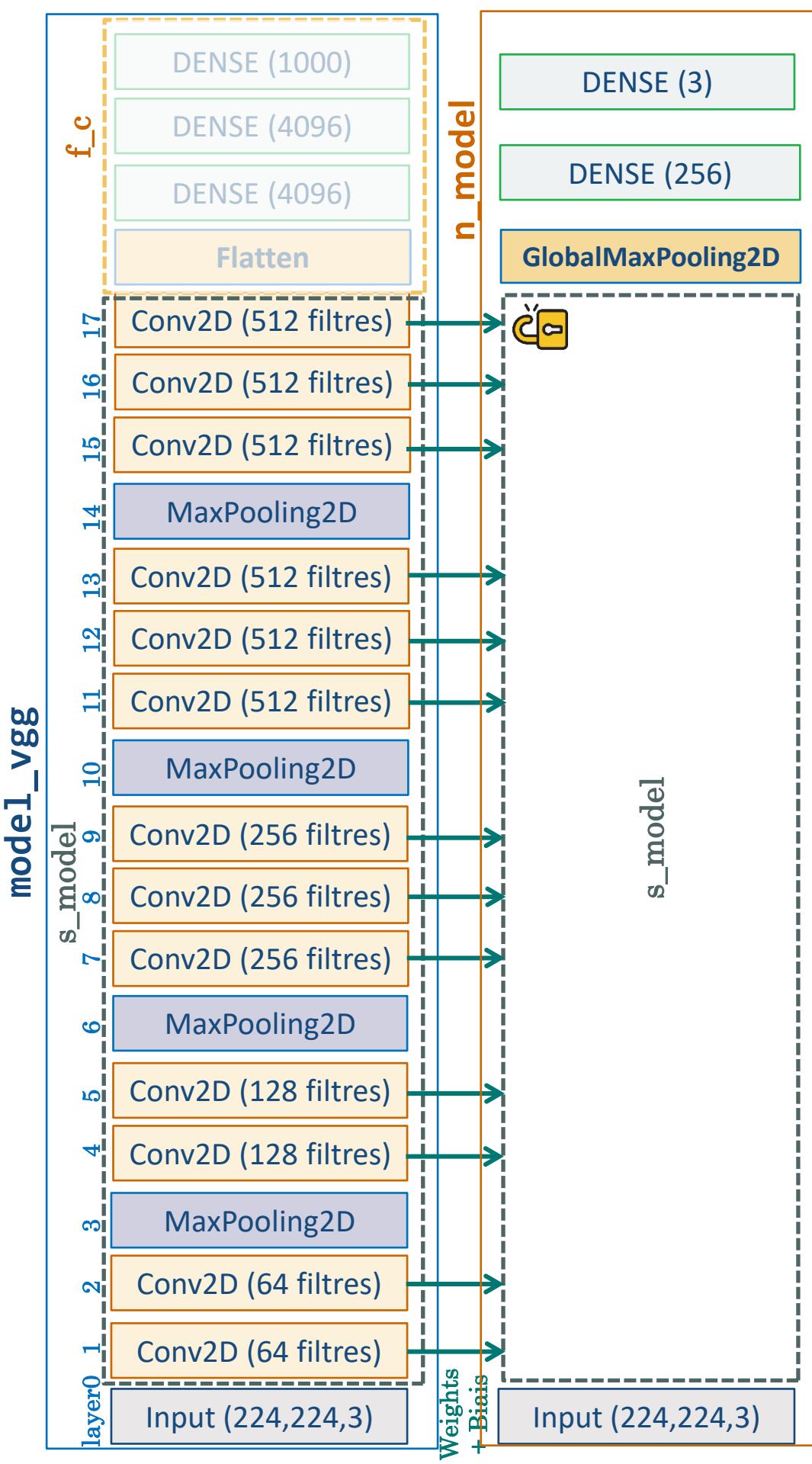
Définir chaque couche séparément

Branchement, entrées et sorties peuvent être multiples

Étendre et ajouter une ou plusieurs couches

`tf.keras.Model(inputs, outputs)`

# tf.keras.Model (1/2)



## tf.keras.Model (2/2)

```
model_vgg=tf.keras.applications.vgg16.VGG16(include_top=True, weights='imagenet')
```

### #Création de s\_model

```
s_model= tf.keras.Sequential()  
for layer in model_vgg.layers[1:18]:  
    s_model.add(layer)  
s_model.trainable=True
```



### #Génération du nouveau modèle n\_model

```
inputs = tf.keras.Input(shape=(224, 224, 3))  
x=s_model(inputs, training=True) ← Mode Apprentissage  
  
x=tf.keras.layers.GlobalMaxPooling2D()(x)  
x=tf.keras.layers.Dense(256, activation="relu")(x)  
outputs=tf.keras.layers.Dense(3, activation="softmax")(x)  
  
n_model=tf.keras.Model(inputs, outputs)
```

# Quiz

## 1. Qu'est-ce que l'apprentissage par transfert dans les réseaux de neurones ?(Une seule réponse)

- Consiste à réutiliser les poids d'un réseau pré-entraîné pour accélérer l'apprentissage d'un nouveau modèle
- Consiste à entraîner un modèle sur des données non étiquetées
- Consiste à utiliser des réseaux de neurones récurrents pour traiter des séquences de données
- Aucune de ces réponses n'est correcte

## 2. Pourquoi utiliser **tf.keras.Models** lors de la construction d'un réseau de neurones?

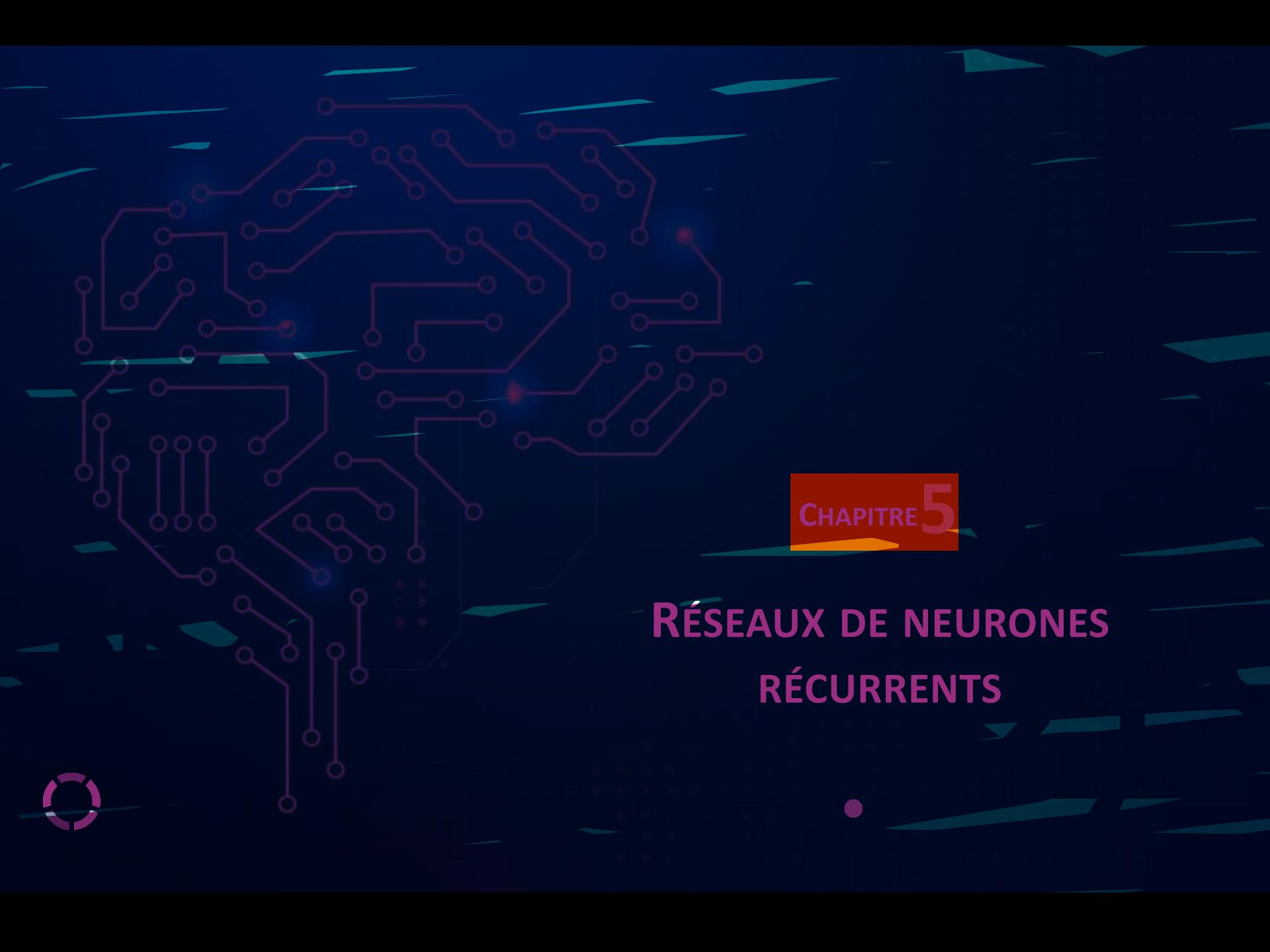
(Une seule réponse)

- Pour construire un modèle avec une structure de réseau de neurones simple et séquentielle
- Pour utiliser des couches de réseaux de neurones pré-entraînées
- Pour enregistrer le modèle entraîné
- Pour construire un modèle de réseaux de neurones personnalisé et complexe

## 3. Le ResNet-50 est un modèle de réseau de neurones convolutif pré-entraîné sur ImageNet et utilise des connexions résiduelles (Vrai/Faux)

- Vrai

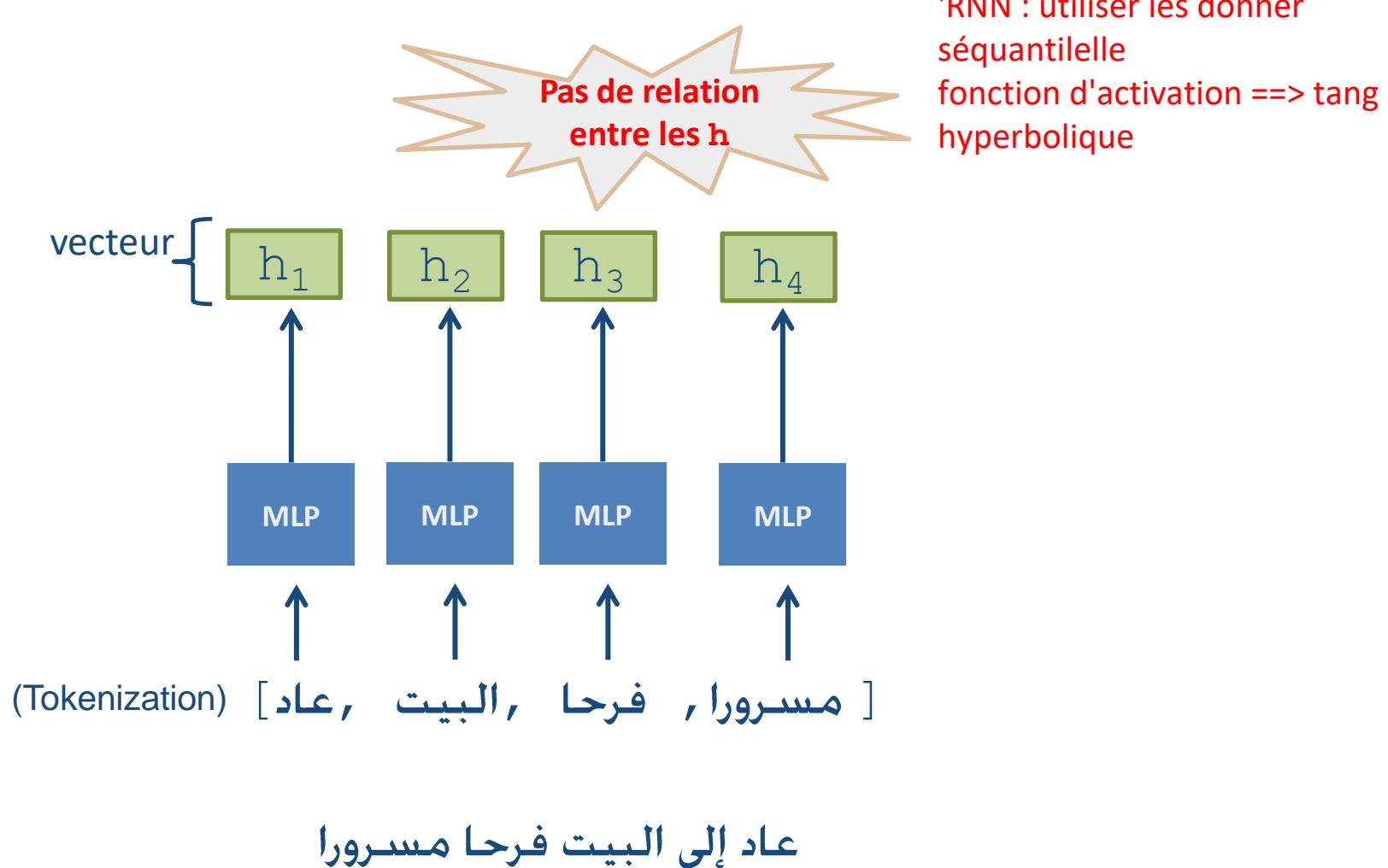
- Faux



CHAPITRE 5

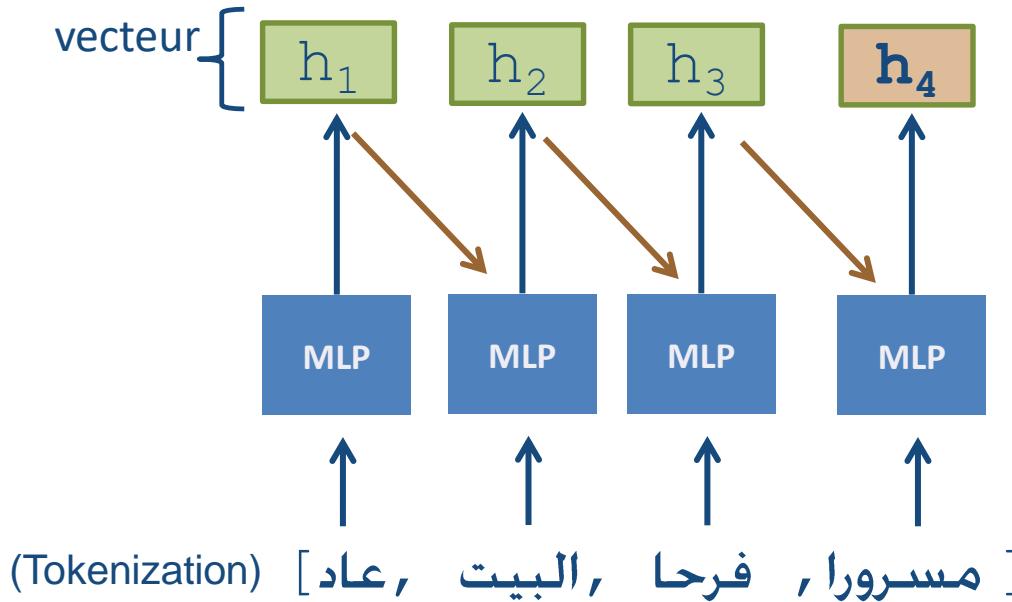
# RÉSEAUX DE NEURONES RÉCURRENTS

# Introduction



# Vers le réseau récurrent !

**h<sub>4</sub>** pourra contenir de l'information sur toute la séquence



عاد إلى البيت فرحا مسرورا

# Recurrent Neural Network RNN

=Réseau de neurones cyclique

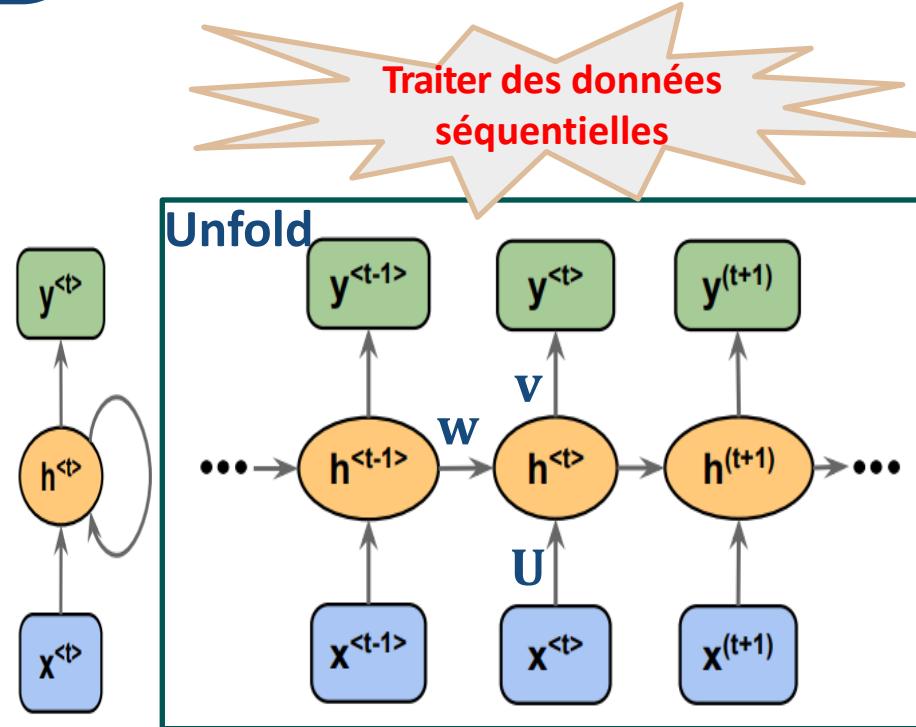
$x^{<t>}$ : l'entrée de la séquence d'entrée à l'instant t



**Unité de mémoire** : Conserver des informations sur les entrées précédentes et de les utiliser pour influencer les sorties futures



**Unité de sortie** : Responsables de la production des sorties du RNN à chaque pas de temps



$$h^{<t>} = \tanh(U \times x^{<t>} + W \times h^{<t-1>} + b_h)$$

$$y^{<t>} = \tanh(V \times h^{<t>} + b_y)$$

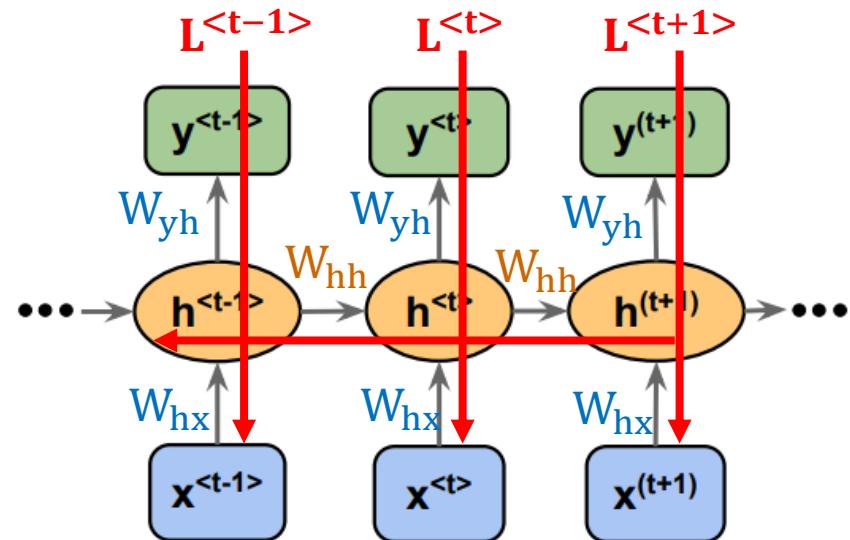
**U, W, V** : matrices de poids (sont apprises par le réseau pendant la phase d'apprentissage)

# Backpropagation Through Time (BPTT)

$$\frac{\partial L^{<t>}}{\partial W_{hh}} = \frac{\partial L^{<t>}}{\partial y^{<t>}} \frac{\partial y^{<t>}}{\partial h^{<t>}} \left( \sum_{k=1}^t \frac{\partial h^{<t>}}{\partial h^{<k>}} \frac{\partial h^{<k>}}{\partial W_{hh}} \right)$$

$$\frac{\partial h^{<t>}}{\partial h^{<k>}} = \prod_{i=k+1}^t \frac{\partial h^{<i>}}{\partial h^{<i-1>}}$$

$$L = \text{loss} = \sum_{t=1}^T L^{<t>}$$

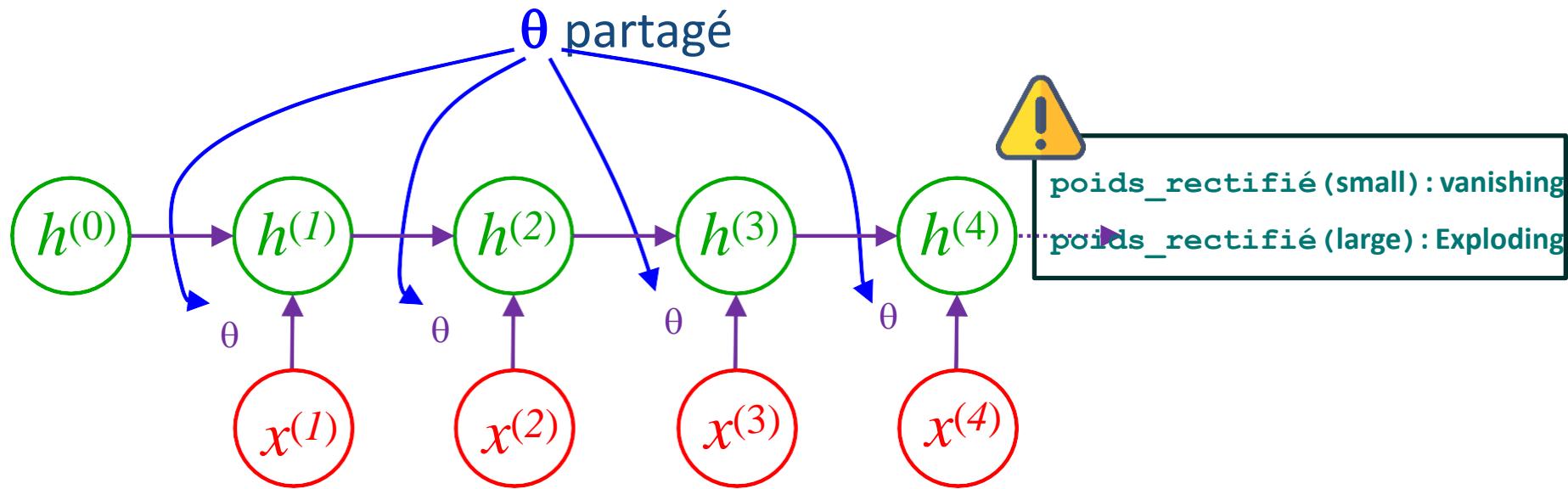


forward pass →  
 ← backprop →

# Recurrent Neural Network RNN $\theta$ partagé

$$\mathbf{h}^{(t)} = \mathbf{f}(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}, \theta)$$

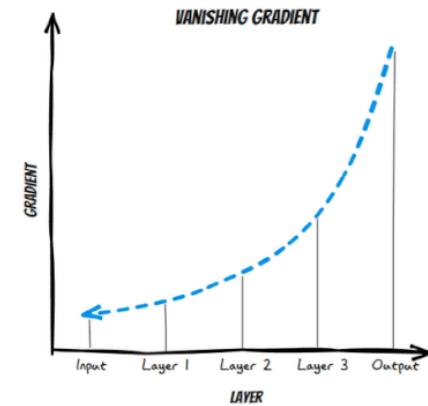
$\theta$  partagé



# Vanishing/ Exploding gradient

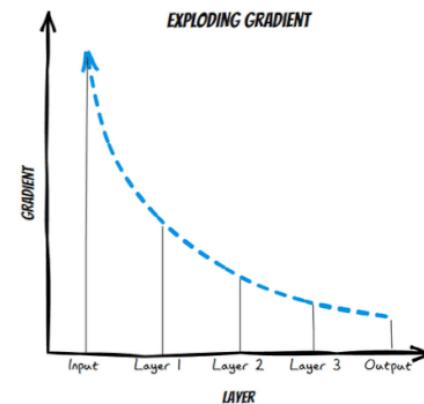
## Vanishing gradient

- Se produit lorsque les gradients deviennent très *petits* lors du backward
- Entraîner une stagnation ou une lenteur dans l'apprentissage du réseau
- Les poids des couches initiales du réseau sont mis à jour très peu par rapport aux couches finales
- Les poids des couches initiales ont peu d'impact sur la fonction de perte



## Explosion de gradient

- Se produit lorsque les gradients deviennent très *grands* lors du backward
- L'initialisation inappropriée des poids
- Entraîner une instabilité numérique et faire échouer l'entraînement du réseau

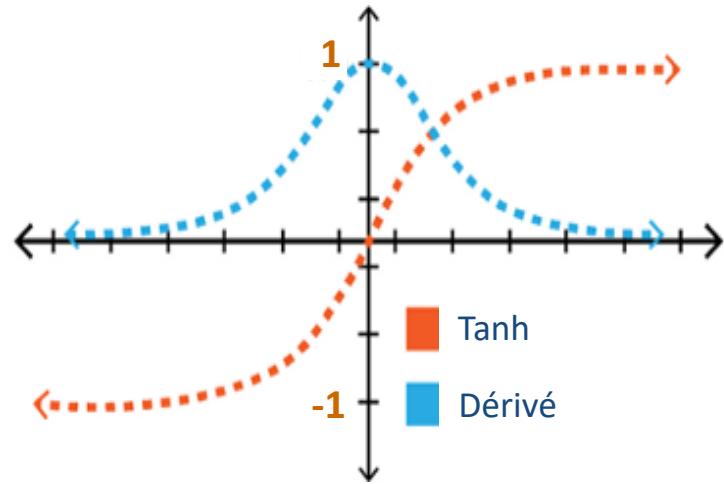


# Fonction d'activation tanh

$$\tanh(z) = \frac{e^{2z} - 1}{e^{2z} + 1} \quad ]-1,1[$$

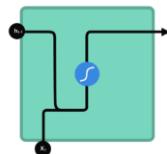
## Dérivé de la fonction tanh

$$\frac{\partial \tanh(z)}{\partial z} = 1 - f(z)^2 \quad ]0,1]$$



💡 Gérer les dépendances à long terme

💡 Éviter le problème de Vanishing gradient



# Typologies RNN

CNN

one to many

many to one

many to many

many to many

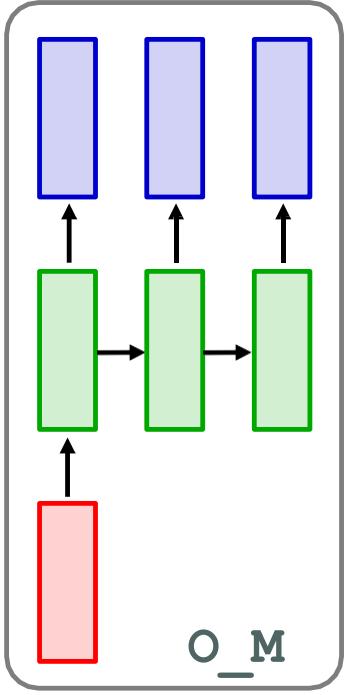
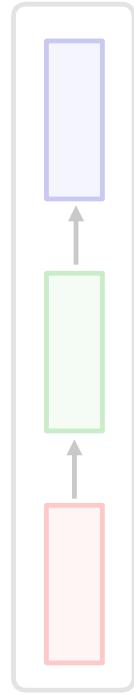
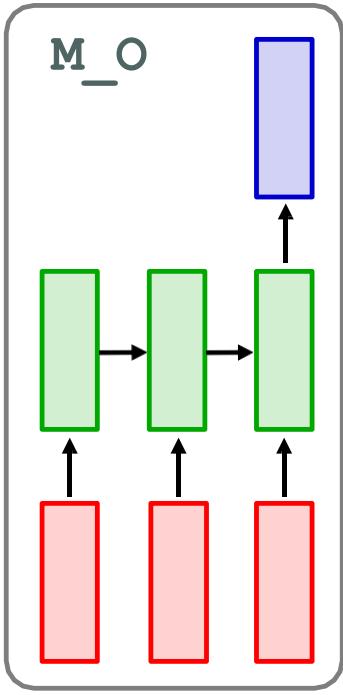
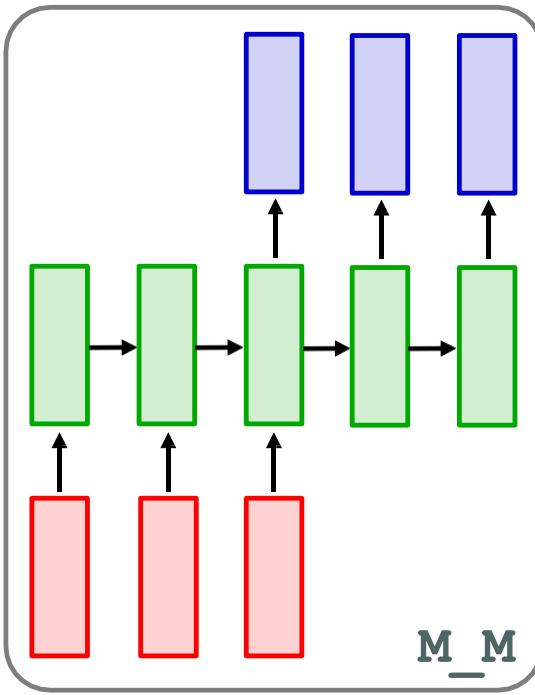


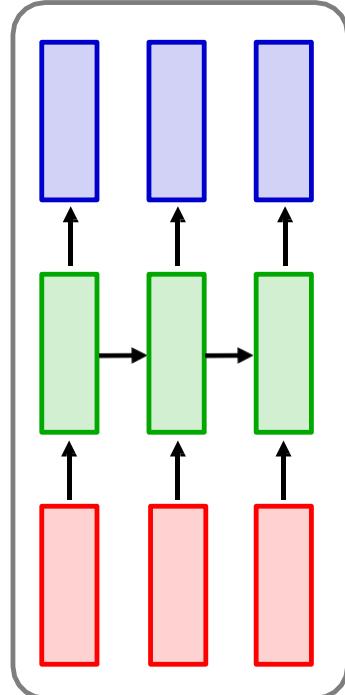
Image  
captioning



Classification de  
sentiment (texte)



Traduction,  
Réponse aux  
questions  
(tailles entrée/sortie  
variables)



Classification de  
trames vidéos

# Quelques applications de RNN!



Next word prediction



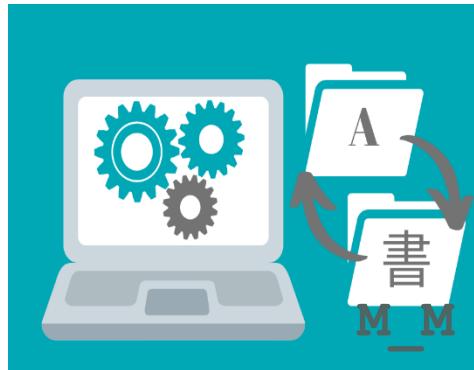
Music composition



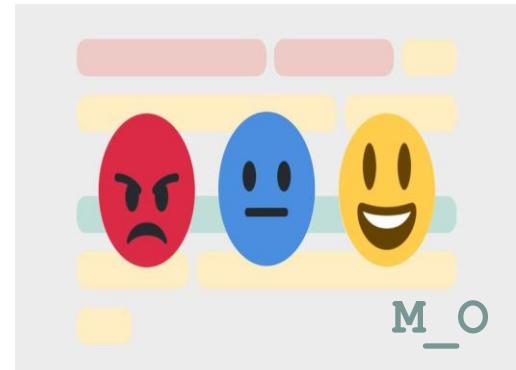
Image captioning



Speech recognition



Machine translation



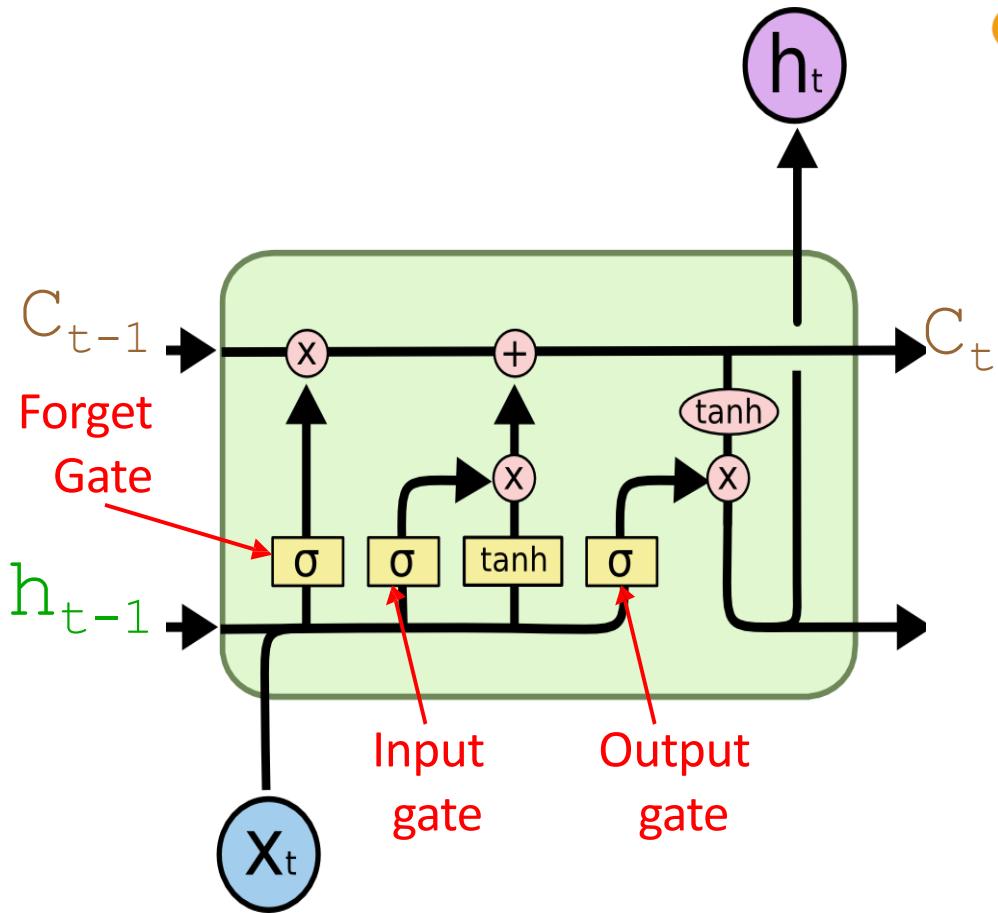
Sentiment analysis

# tf.keras.layers.SimpleRNN

```
model = tf.keras.Sequential()  
:  
  
model.add(tf.keras.layers.SimpleRNN(units, activation='tanh', use_bias=True,  
bias_initializer='zeros', kernel_regularizer=None, recurrent_regularizer=None,  
bias_regularizer=None, activity_regularizer=None, dropout=0.0,  
recurrent_dropout=0.0, return_sequences=False, **kwargs) )
```

- **units** : Dimensionnalité de l'espace de sortie
- **return\_sequences** indique si la couche RNN doit renvoyer des séquences complètes ou uniquement la dernière sortie
- **return\_sequences=False** signifie que seule la dernière sortie sera renvoyée

# Long Short-Term Memory (LSTM) (1/3)



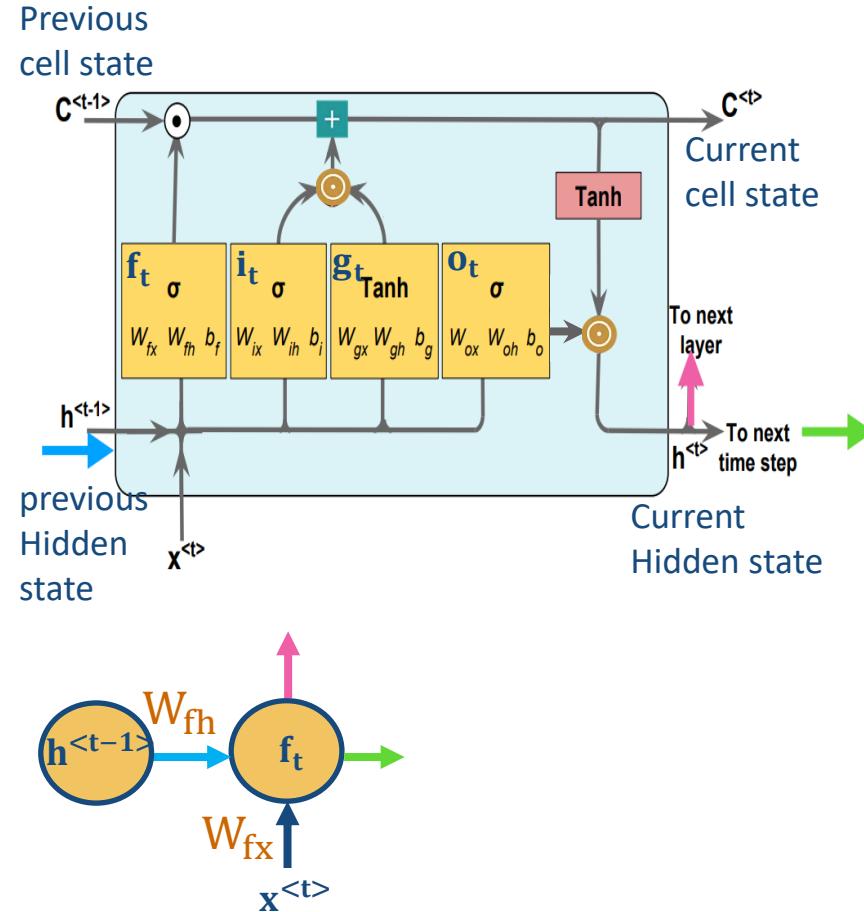
**h (hidden state)**

- Prend en compte les informations précédentes et les mélange avec les nouvelles informations
- Hidden state final encapsule alors une représentation complète de des données

**C (cell state)**

- Mis à jour en ajoutant ou en supprimant des informations pertinentes
- Comme  $h$ ,  $C$  prend en compte les informations précédentes et les nouvelles informations
- $C$  garde une trace plus persistante de l'information à long terme

# Long Short-Term Memory (LSTM) (2/3)



## 1- Forget gate $f_t$

mémoriser que l'important mémoriser

Détermine les informations à oublier

$$f_t = \sigma(W_{fx}x^{t-1} + W_{fh}h^{t-1} + b_f)$$

## 2- input gate $i_t$

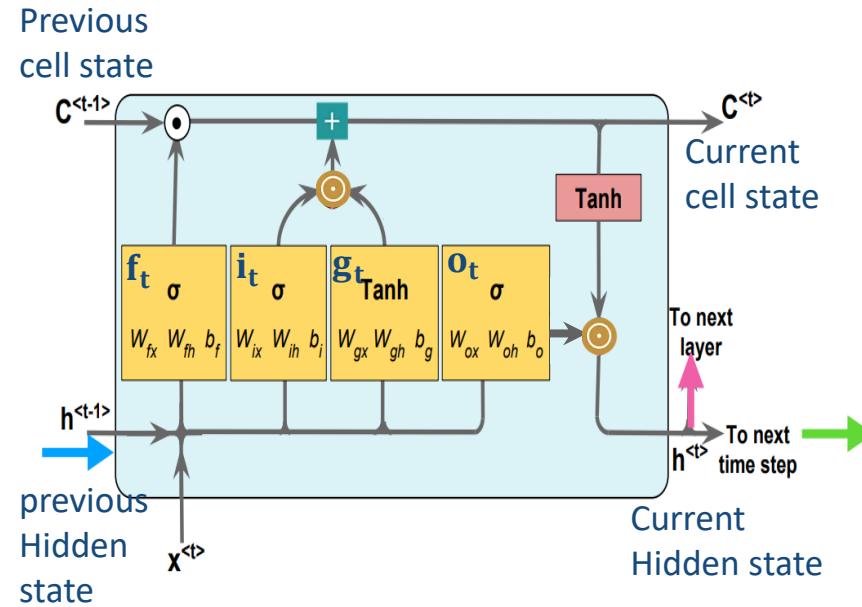
Détermine les nouvelles informations à ajouter à cell state

$$i_t = \sigma(W_{ix}x^{t-1} + W_{ih}h^{t-1} + b_i)$$

## New memory network $g_t$

$$g_t = \tanh(W_{gx}x^{t-1} + W_{gh}h^{t-1} + b_g)$$

# Long Short-Term Memory (LSTM) (3/3)



## 3- Output gate $o_t$

Contrôle l'activation de la mémoire interne pour produire la sortie

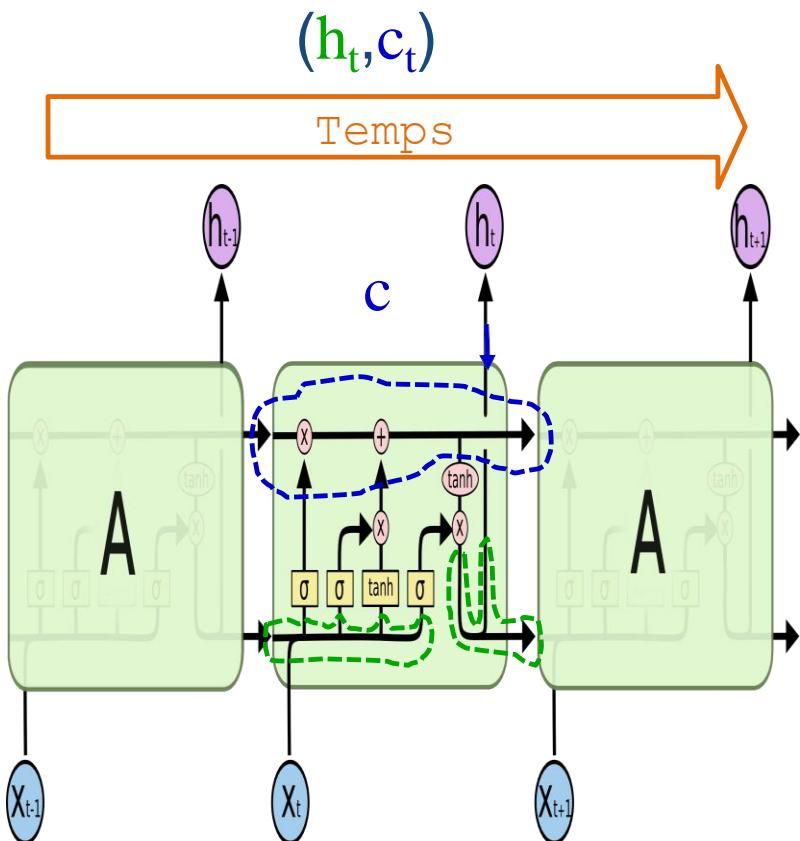
$$o_t = \sigma(W_{ox}x^t + W_{oh}h^{t-1} + b_o)$$

$$\text{Current cell state } C^t = (C^{t-1} \odot f_t) + (i_t \odot g_t)$$

$$\text{Current Hidden state} = h^t = o_t \odot \tanh(C^t)$$

Point Wise Addition	Sigmoïde	Point Wise Multiplication	tanh

# LSTM : récursivité déroulée



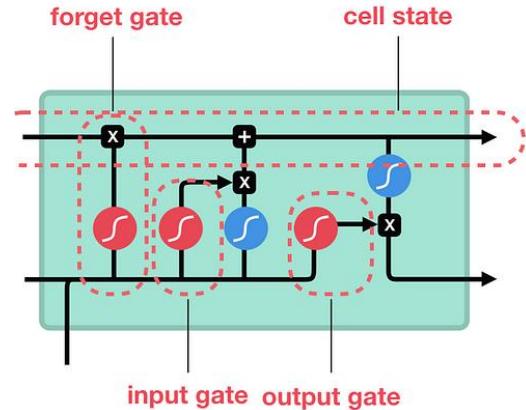
- Décomposer le réseau récurrent sur une certaine longueur de séquence en une série de couches identiques, chacune représentant un pas de temps
- Au lieu de visualiser le réseau comme une boucle récurrente, il est représenté comme une séquence de couches empilées horizontalement

# tf.keras.layers.LSTM

```
model = tf.keras.Sequential()
```

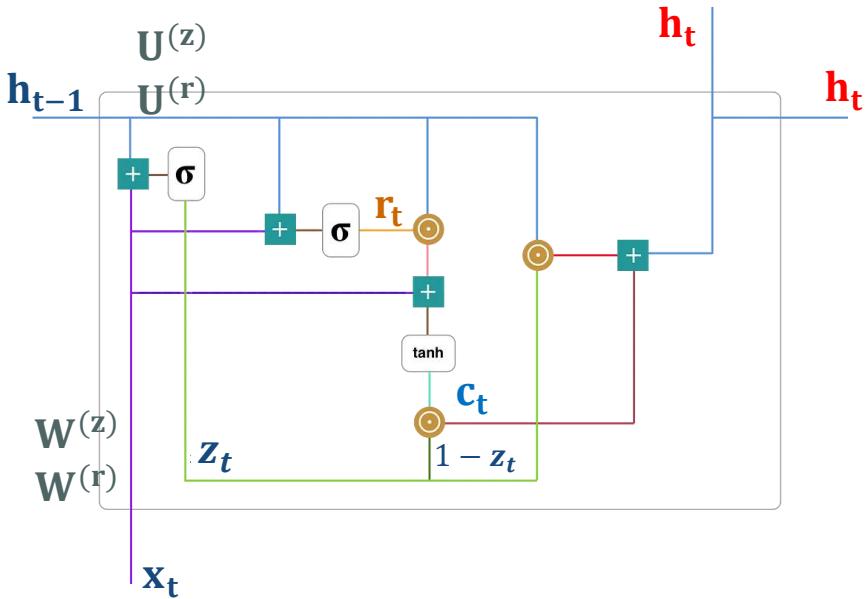
```
:
```

```
model.add(tf.keras.layers.LSTM(units, activation='tanh',
                                recurrent_activation='sigmoid', use_bias=True,
                                bias_initializer='zeros', kernel_regularizer=None, recurrent_regularizer=None,
                                bias_regularizer=None, activity_regularizer=None, return_sequences=False,
                                return_state=False, **kwargs) )
```



- **activation** : fonction d'activation appliquée à la sortie de chaque cellule LSTM
- **recurrent\_activation** : fonction d'activation appliquée aux portes de la cellule LSTM  
(**forget gate**, **input gate** et **output gate**)

# Gated Recurrent Unit (GRU) (1/2)



## 1- Update gate $z_t$

=mise à jour des informations dans l'étape actuelle

=aide le modèle à déterminer dans quelle mesure les informations passées doivent être transmises vers le futur

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1} + b_z)$$

## 2- Reset gate $r_t$

=décide l'importance de l'information passée (à oublier ou à conserver)

=aide le modèle à gérer les dépendances à long terme

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1} + b_r)$$



Point Wise  
Addition



Sigmoïde

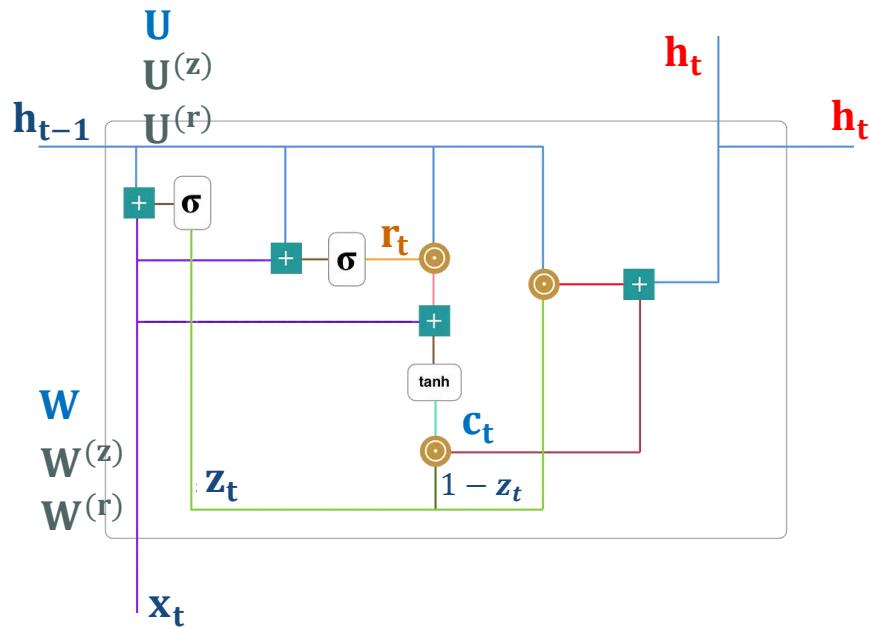


Point Wise  
Multiplication



tanh

# Gated Recurrent Unit (GRU) (2/2)



## 3- Current memory content $c_t$

=Introduire un nouveau contenu de mémoire qui utilisera **reset gate** pour stocker les informations pertinentes du passé

$$c_t = \tanh(W^{(c)}x_t + r_t \odot U^{(c)}h_{t-1} + b_c)$$

## 4- Final memory $h_t$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot c_t$$

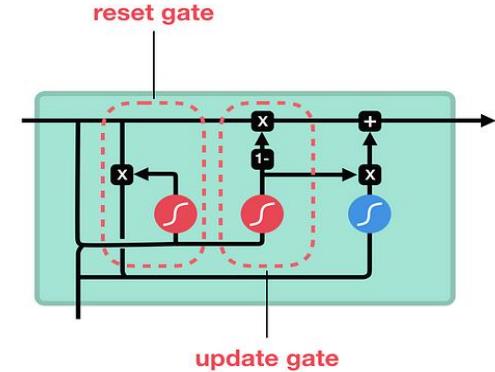
Point Wise Addition	Sigmoïde	Point Wise Multiplication	tanh

# tf.keras.layers.GRU

```
model = tf.keras.Sequential()
```

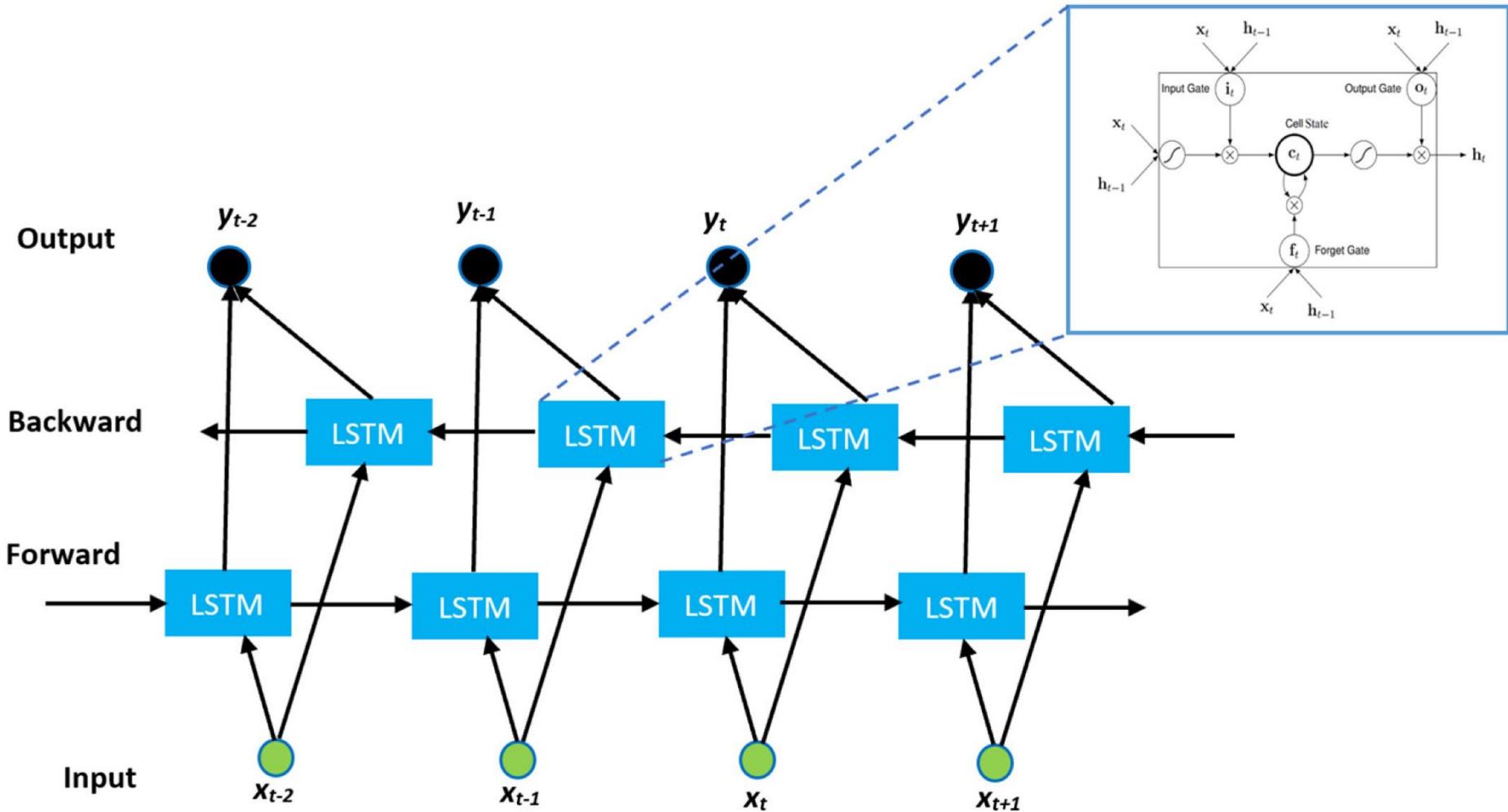
```
:
```

```
model.add(tf.keras.layers.GRU(units, activation='tanh',
    bias_initializer='zeros', kernel_regularizer=None, recurrent_regularizer=None,
    bias_regularizer=None, activity_regularizer=None, return_sequences=False,
    return_state=False, **kwargs) )
```



- **activation** : fonction d'activation appliquée à la sortie de chaque cellule GRU
- **recurrent\_activation** : fonction d'activation appliquée aux portes de la cellule GRU  
(**update gate** et **reset gate**)

# Autre Bidirectionnel LSTM



# Quiz

1. Quel modèle n'est pas un réseau de neurones cyclique? (une seule réponse)

- GBDT (Gradient Boosted Decision Tree)
- RNN
- LSTM
- GRU (Gated recurrent unit)

2. Lequel des types de RNN suivants peut être utilisé pour la traduction automatique?  
(une seule réponse)

- Many to One
- One to Many
- Many to Many
- One to One

3. Lequel des éléments suivants est utilisé comme fonction de porte dans LSTM?  
(une seule réponse)

- Relu function
- Softplus function
- Sigmoid function
- Tanh function

## Évolution...

RNN → LSTM → Attention

# BIBLIOGRAPHIE

- [1] S. Koki. Deep Learning from the Basics: Python and Deep Learning: Theory and Implementation. Packt Publishing Ltd, 2021
- [2] G. Aurélien. Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media, 2019
- [3] J. Brownlee. Deep Learning with Python: develop deep learning models on Theano and TensorFlow using Keras. Machine Learning Mastery, 2016
- [4] G. Lan, Y. Bengio and A. Courville. Deep Learning. MIT press, 2016

The background of the image features a complex network graph composed of numerous small, semi-transparent white dots connected by thin white lines. Interspersed among these dots are vertical columns of binary digits (0s and 1s), rendered in a larger, lighter blue font. The text "TO BE CONTINUED..." is overlaid on this background.

**TO BE CONTINUED...**