IIT
INSTITUT INTERNATIONAL TECHNOLOGIE
Université Nord Américaine privée

MIR@CL
Informatique & multimédia

[Techno IT V1.0

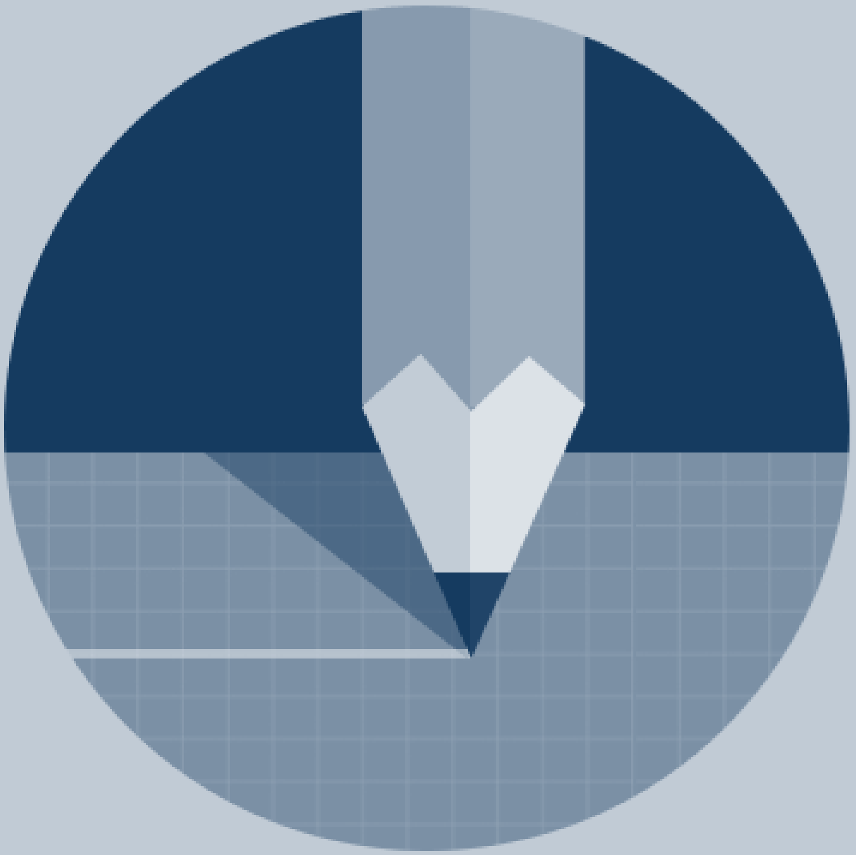
Deep Learning

(Apprentissage profond)

Sportissimo
Sfax

07-08
Avril

2020-2021



Taoufik BEN ABDALLAH

التوفيق بن عبد الله

Docteur en Informatique

Enseignant permanent à IIT-Sfax

Instructeur Huawei en Intelligence Artificielle

taoufik.benabdallah@iit.ens.tn

+216 26 445 012

Objectifs de la formation

- Présenter le domaine de l'apprentissage profond
- Maitriser 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
- Maîtriser des principes **pratiques** de l'apprentissage profond pour la classification

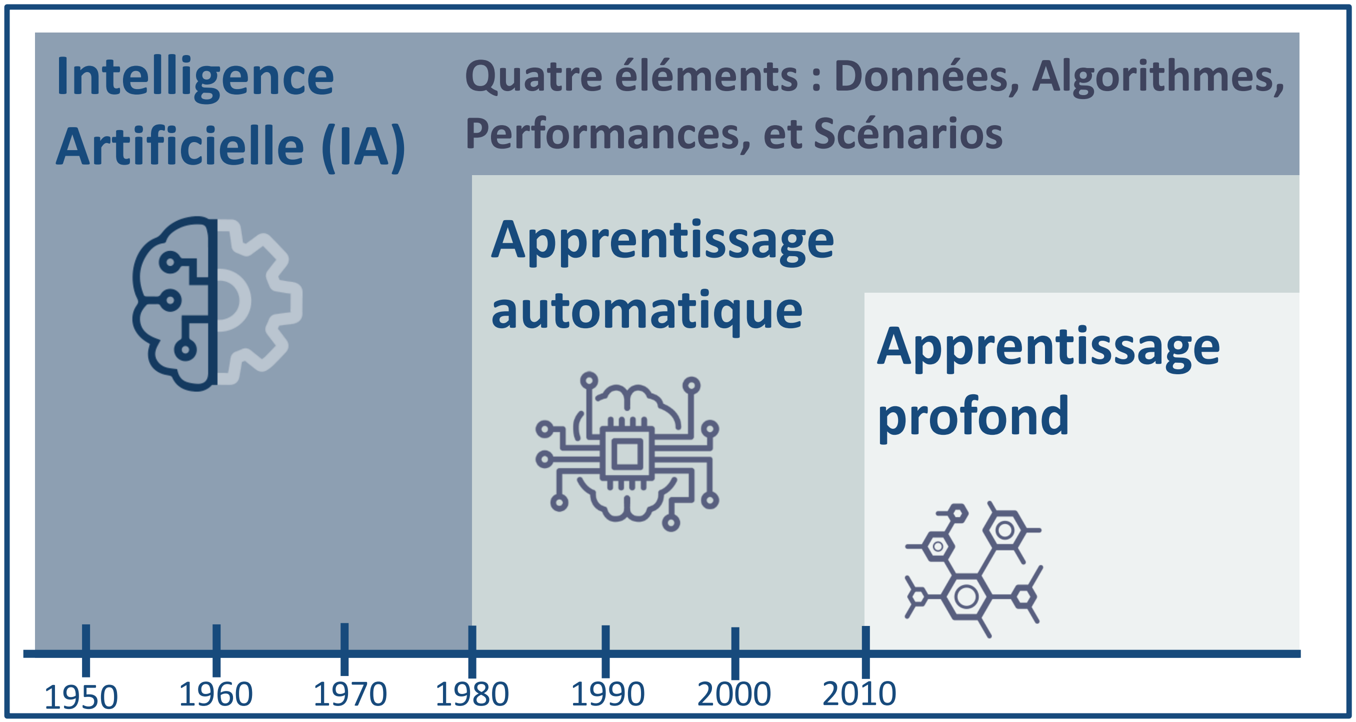
Plan de la Formation

Plan de la formation

Partie 1

Introduction générale

IA, Apprentissage automatique vs. Apprentissage profond



Intelligence Artificielle→ Artificial Intelligence (**AI**)
Apprentissage automatique→ Machine Learning (**ML**)
Apprentissage profond→ Deep Learning (**DL**)

Types d’apprentissage automatique ♣

Apprentissage supervisé

Apprentissage non-supervisé

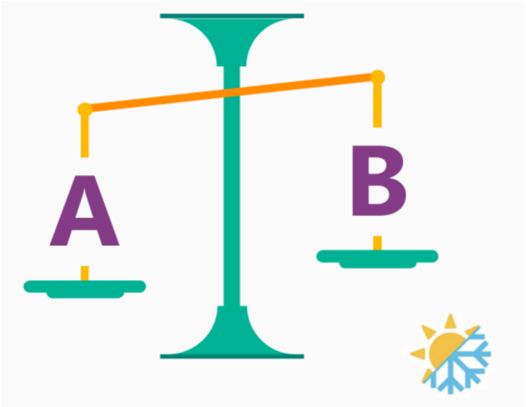
Apprentissage semi-supervisé

Apprentissage par renforcement

Rappel/ Apprentissage supervisé

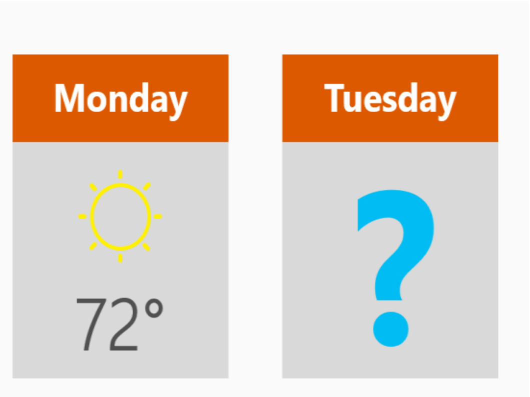
Classification : Prédire des valeurs discrets

- Sera-t-il froid ou chaud demain? **Froid (A)/ chaud (B)**



Régression : Prédire des valeurs continues

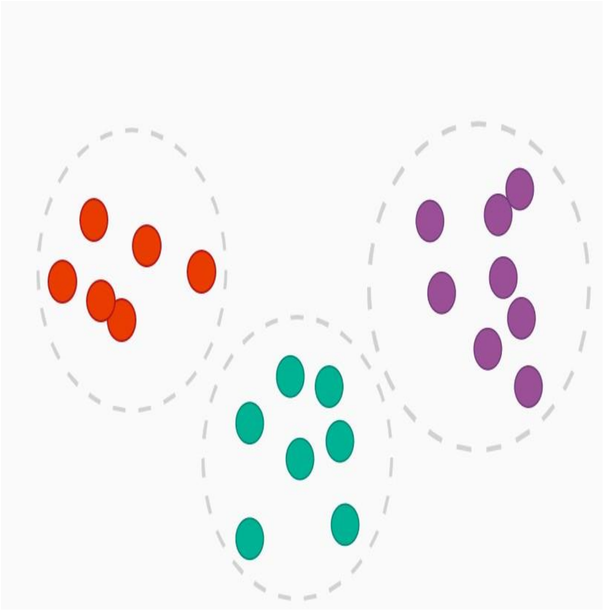
- Quelle est la température demain?



Rappel/ Apprentissage non-supervisé

Regroupement (Clustering)

- Organisation des données en groupes (clusters)



Réduction de dimensionnalité

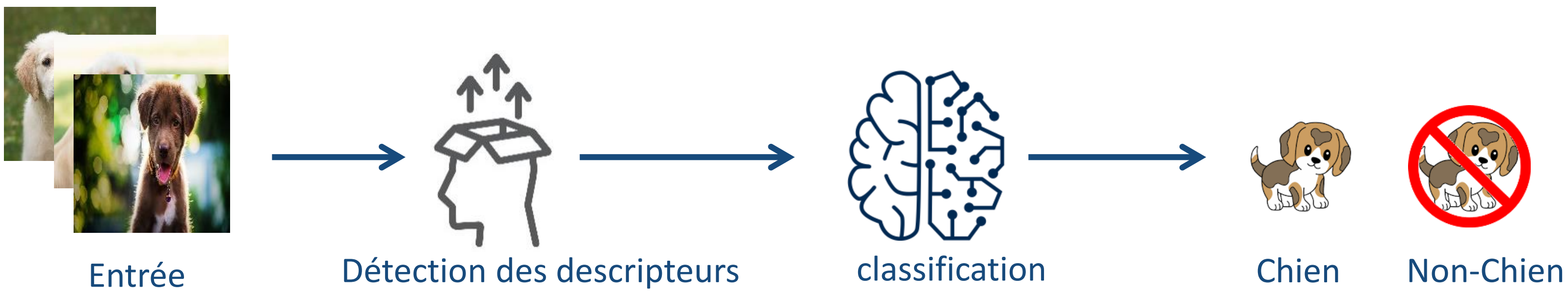
- Sélection des descripteurs les plus discriminants
- Transformation les descripteurs dans un autre espace

Détection d'anomalies (Outliers detection)

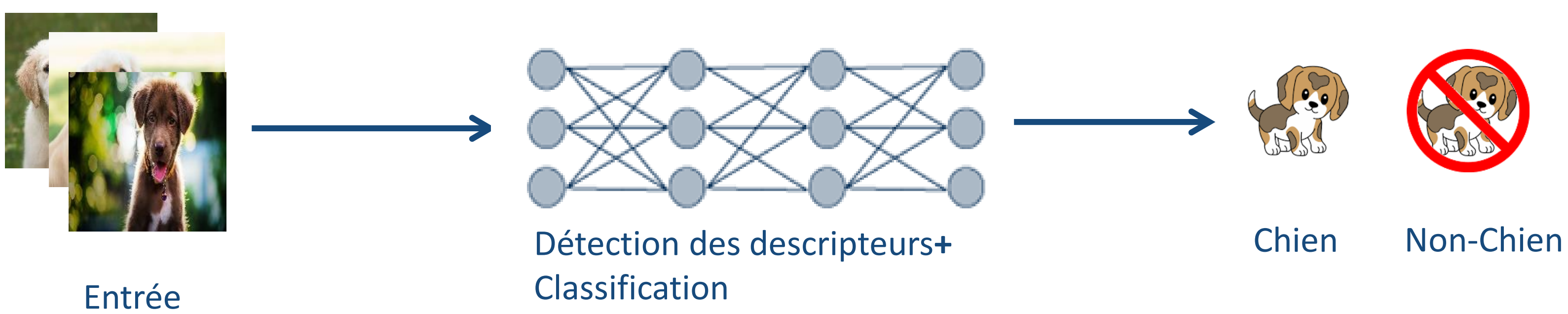
- Identification d'observations qui soulèvent des suspicions en différant

Apprentissage automatique vs. Apprentissage profond 1/2

Apprentissage automatique



Apprentissage profond



Apprentissage automatique vs. Apprentissage profond 2/2

Apprentissage automatique	Apprentissage profond
Faible besoin en matériel informatique : **CPU** (**C**entral **P**rocessing **U**nit)	Exigences matérielles plus élevées sur l'ordinateur : **GPU** (**G**raphics **P**rocessing **U**nit)
Apprentissage sur une petite quantité de données	Apprentissage sur une quantité de données massives
Détection manuelle des descripteurs (**Handcrafted features**) Faciles à expliquer	Détection automatique des descripteurs (**High level features**) Difficiles à expliquer
Analyse des problèmes **niveau par niveau**	Apprentissage de **bout en bout** (end-to-end)

Les performances ne peuvent pas être améliorées lorsque la quantité de données augmente!

Quelques applications*

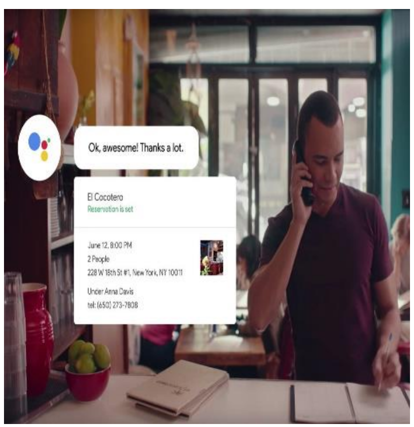
Reconnaissance et classification à partir des images/vidéos	✓
Reconnaissance d'actions	✓
Ré identification des personnes	✓
Détection d'objets en mouvement	✓
Traduction automatique	✓
Reconnaissance des entités nommés	✓
Génération automatique de texte	✓
Génération automatique de l'écriture manuscrite	✓

Succès de DL!

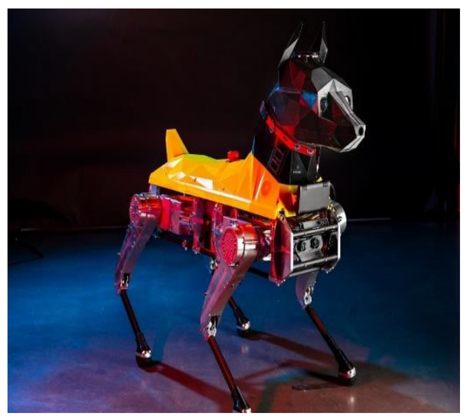
DeepFace 2014

Skype Translator 2015

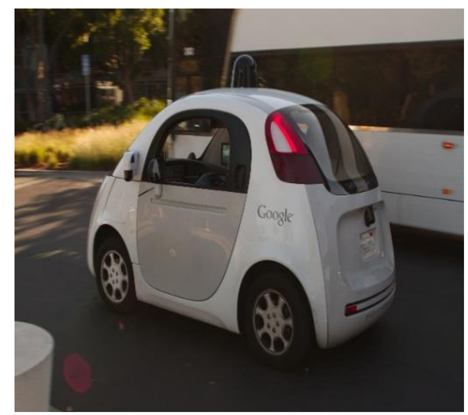
AlphaGo Zero 2017



Google Duplex 2018



Astro robot dog 2019



Google car 2020

Frameworks de DL

TensorFlow

PyTorch

MindSpore

Keras

Caffe

mxnet

*TensorFlow : Framework le plus populaire pour la vision par ordinateur et l'IA basées sur le DL

TensorFlow2.x

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 (une seule réponse)
   - ☑ True
   - ☐ False

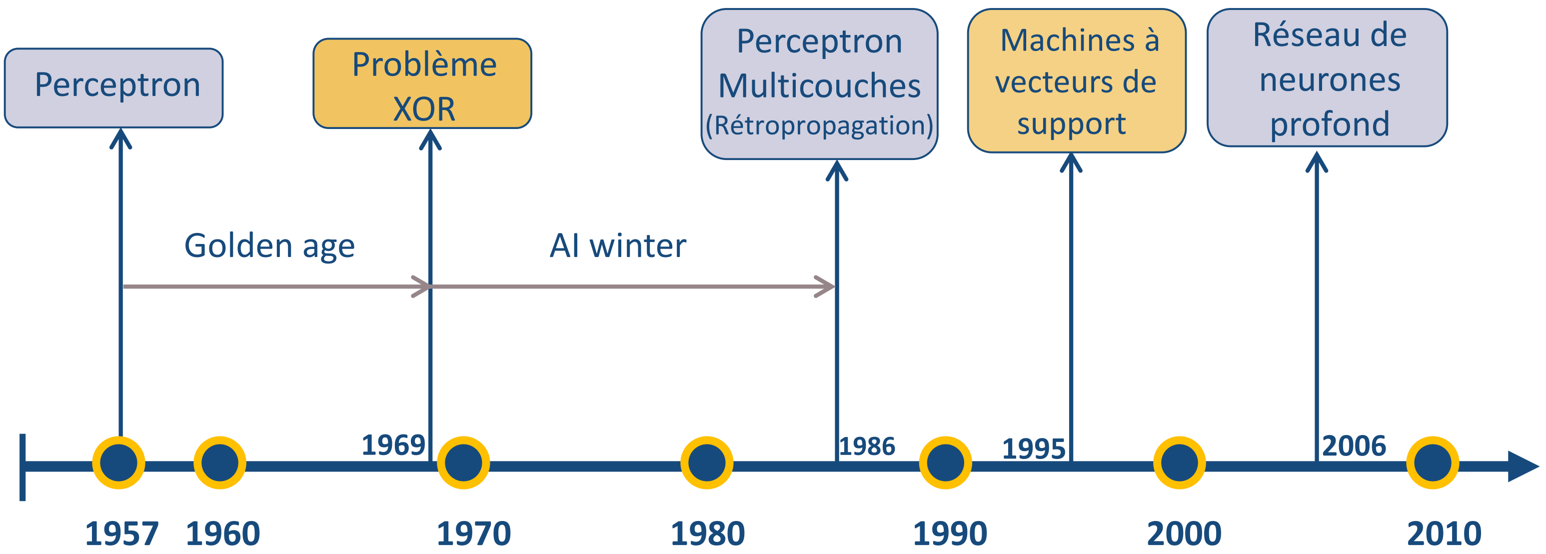
2. TensorFlow 2.0 (choix multiples)
   - ☑ Intègre Keras, ce qui améliore considérablement la convivialité
   - ☑ Est un Framework Open Source
   - ☑ Est un Framework flexible
   - ☐ Est inventé par Facebook

3. Parmi les éléments suivants, lequel ne fait pas partie du Framework de développement de l'apprentissage profond (une seule réponse)
   - ☐ MindSpore
   - ☐ Keras
   - ☑ sklit-learn
   - ☐ MXNet

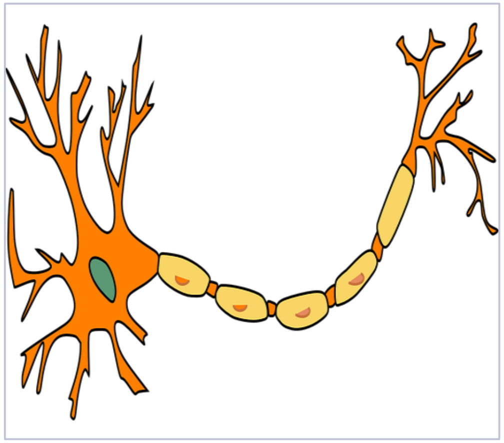
Partie 2

Apprentissage de Réseaux de neurones profond

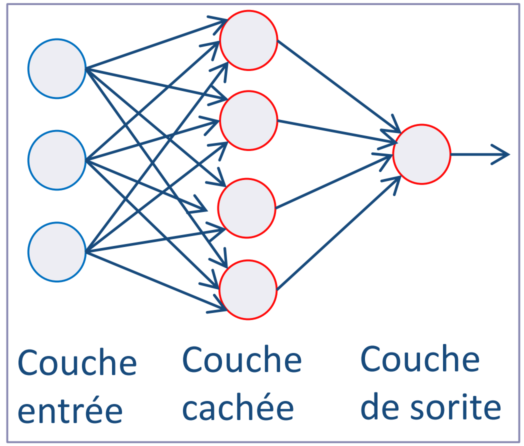
Historique du développement des réseaux de neurones



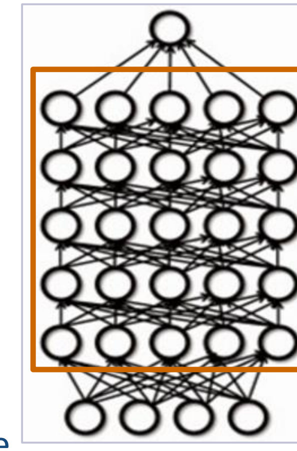
Réseau de neurones



Neurone
biologique



Perceptron
multicouches



Réseau de neurones
profond

Le terme "profond" dans "apprentissage profond" fait référence au nombre de couches caché du réseau de neurones (généralement $\geq$5)

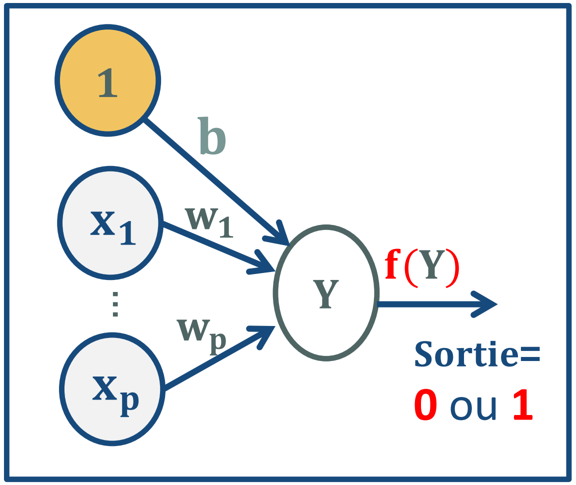
Perceptron simple

Perceptron simple

- Comporte seulement deux couches : entrée & sortie
- Couche d'entrées→ Input vector $\mathbf{x_1}$, $\mathbf{x_2}$,…, $\mathbf{x_p}$+ unité **Bias** (généralement =$\mathbf{1}$)
- Poids $\mathbf{w_1}, \mathbf{w_2}, \dots, \mathbf{w_p}, \mathbf{b}$
- Couche de sortie $\mathbf{Y} = \sum_{\mathbf{i=1}}^{\mathbf{p}} \mathbf{w_i x_i} + \mathbf{b}$
- Fonction d'activation **Heaviside** $\boldsymbol{f}(\mathbf{Y})$

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

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



- ☑ Classification **binaire**
- ☑ Problème linéairement séparable!

Perceptron simple Loi d'apprentissage

Loi de Widrow-Hoff (Delta rule)

$\Delta \mathbf{w_i} = \mathbf{R}(\mathbf{d} - \mathbf{sortie})\mathbf{x_i}$

avec $\mathbf{i} = \mathbf{1..p}$

$\mathbf{d}$: Classe désirée
$\mathbf{R}$: force d'apprentissage (learning rate) $\in]\mathbf{0..1}]$: contrôle la rapidité avec laquelle le modèle est adapté au problème

Si $\mathbf{sortie=d}$, $\mathbf{w_i}$ sont inchangeables
Si $\mathbf{sortie<d}$, $\mathbf{w_i}$ va diminuer
Si $\mathbf{sortie>d}$, $\mathbf{w_i}$ va augmenter

$\mathbf{w_i} = \mathbf{w_i} + \Delta \mathbf{w_i}$

Exemple introductif (1/4)

Étant donné le **jeu de donnée d'apprentissage** or_logique

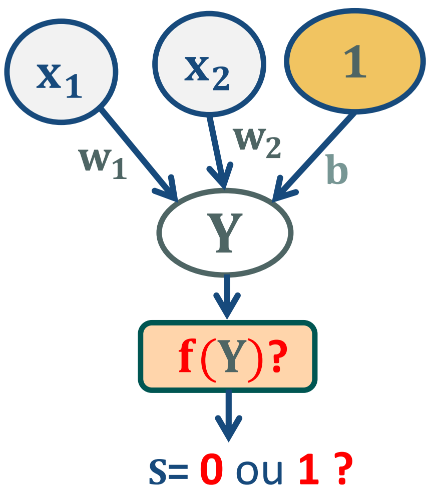
or_logique

	x1	x2	d
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

Variable à prédire (cible) : **d**→ classification binaire **(0/ 1)**

Variables explicatives : $\mathbf{x_1}$, et $\mathbf{x_2}$

* 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 R =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

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

$f(Y)$=1 si $Y \geq 0$; $f(Y)$=0 sinon$\Longrightarrow f(Y)$=1

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

$w_2 = w_2 + R(d - f(Y))x_2$=-1+1(0-1)0=-1

$b = b + R(d - f(Y))1$=0+1(0-1)1=-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 R =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 R =1

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

⋮

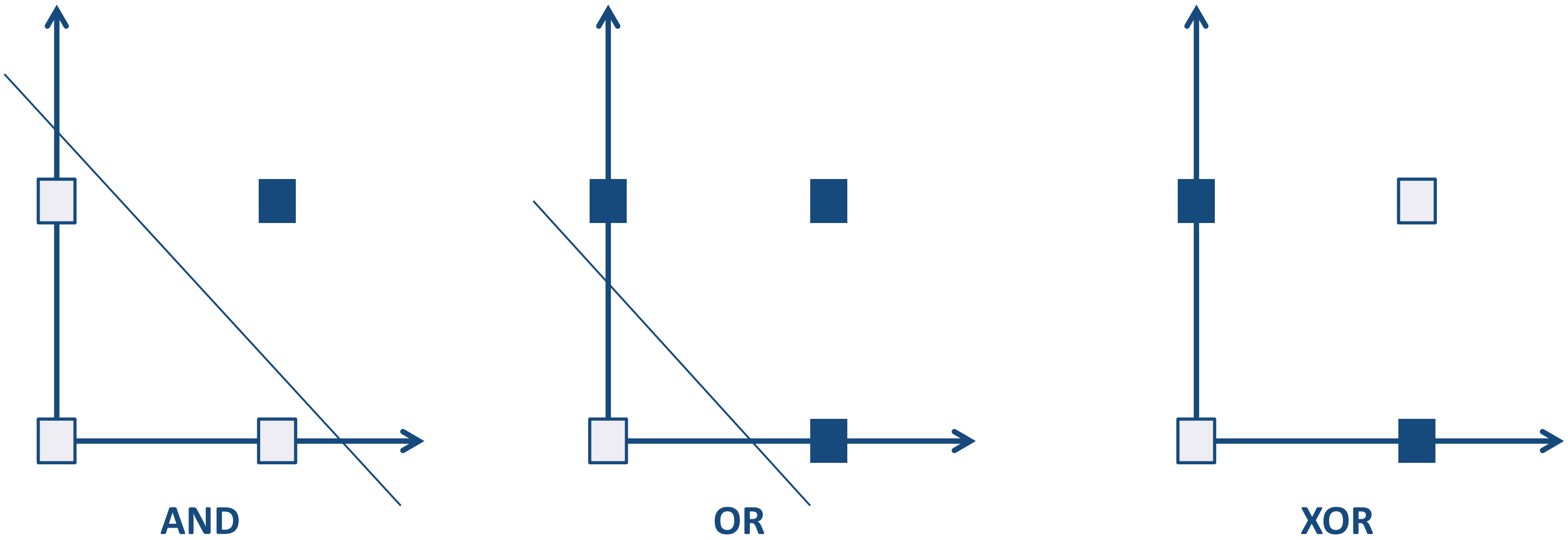
Exemple introductif (4/4)

⋮

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

Problème XOR (ou exclusif)

☹ Le perceptron est incapable de classer les données non séparables linéairement (processus ne converge pas!)

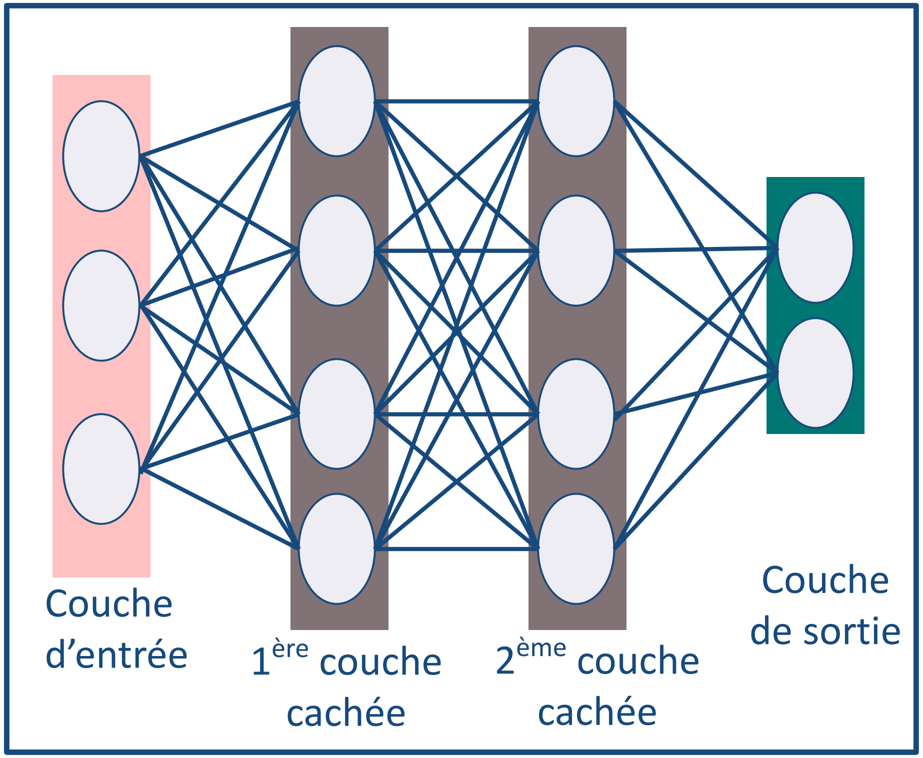


Perceptron multicouches &

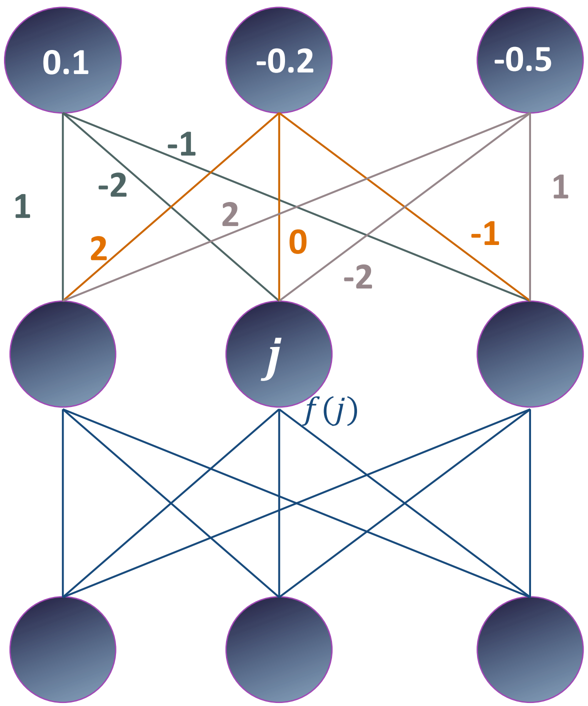
Reseau de neurones profond

Perceptron multicouches (MultiLayers Perceptron, MLP)

- Comporte une couche d'entrée, une ou plusieurs couche(s) cachée(s), et une couche de sortie
- Chaque neurone n'est connecté qu'aux neurones de la couche précédente
- Chaque couche comprend plusieurs neurones, et les neurones d'une même couche ne sont pas connectés entre eux
- Fonctions d'activation des neurones des couches cachées et de sorties sont **non linéaire**



Formulation mathématique d'un MLP



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

$\boldsymbol{p}$ Nombre de neurones dans la couche d'entrée

Multiplication matricielle

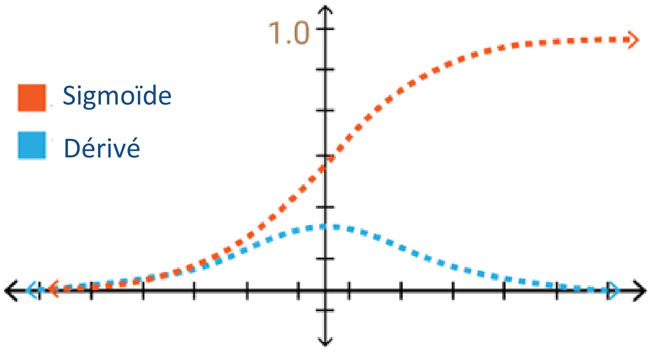
$$\underset{(3,3)}{\begin{bmatrix} 1 & -2 & -1 \\ 2 & 0 & -1 \\ 2 & -2 & 1 \end{bmatrix}} \times \underset{(3,1)}{\begin{bmatrix} 0.1 \\ -0.2 \\ -0.5 \end{bmatrix}} = f\left(\underset{(3,1)}{\begin{bmatrix} ? \\ ? \\ -? \end{bmatrix}} \right)$$

$f(\mathrm{Y_j})$?

Fonctions d'activation 1/4

Étant donné $\mathbf{p}$ neurones $(x_1, \dots, x_i, x_p)$ dans la couche $\mathbf{n-1}$ et $\mathbf{k}$ neurones $(Y_1, \dots, Y_j, Y_k)$ dans la couche $\mathbf{n}$

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

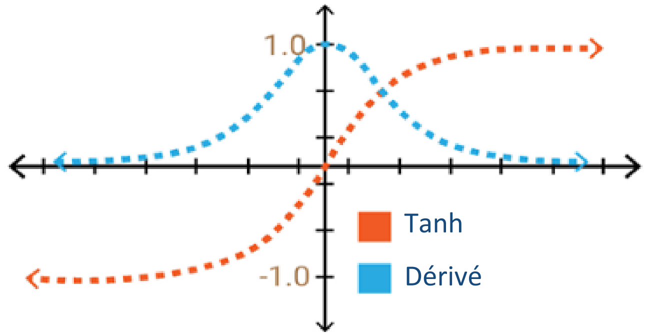


Sigmoïde $f(Y_j) = \frac{1}{1+e^{-Y_j}}$; $]0,1[$

Sa dérivé $f'(Y_j) = f(Y_j)(1 - f(Y_j))$; $]0,0.25[$

Tangente hyperbolique (Tanh) $f(Y_j) = \frac{e^{2Y_j}-1}{e^{2Y_j}+1}$; $]-1,1[$

Sa dérivé $f'(Y_j) = 1 - f(Y_j)^{2}$; $]0,1]$



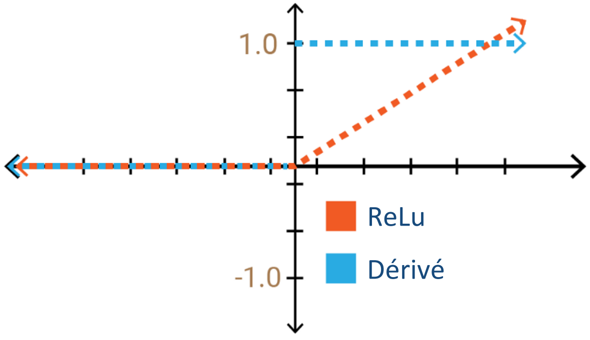
Étant donné $\mathbf{p}$ neurones $(x_1, \dots, x_i, x_p)$ dans la couche $\mathbf{n-1}$ et $\mathbf{k}$ neurones $(Y_1, \dots, Y_j, Y_k)$ dans la couche $\mathbf{n}$

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

ReLu (Rectified Linear Unit)

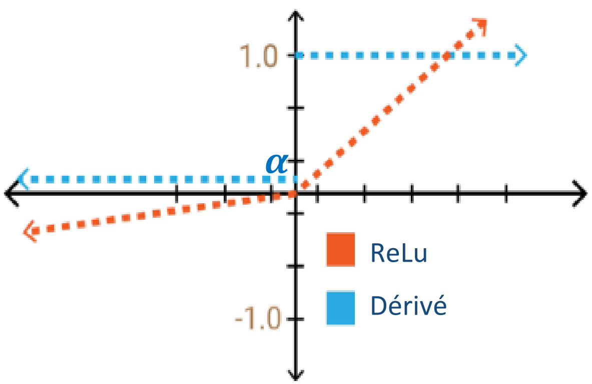
$f(\mathbf{Y_j})$=**max**(0,$\mathbf{Y_j}$) = $\begin{cases} 0 & \text{si } \mathbf{Y_j} < 0 \\ \mathbf{Y_j} & \text{si } \mathbf{Y_j} \geq 0 \end{cases}$ $[0, +\infty[$

Sa dérivé $f'(\mathbf{Y_j})$= $\begin{cases} 0 & \text{si } \mathbf{Y_j} < 0 \\ \mathbf{1} & \text{si } \mathbf{Y_j} \geq 0 \end{cases}$ 0 ou 1



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

Sa dérivé $f'(\mathbf{Y_j})$= $\begin{cases} \boldsymbol{\alpha} & \text{si } \mathbf{Y_j} < 0 \\ \mathbf{1} & \text{si } \mathbf{Y_j} \geq 0 \end{cases}$ $\boldsymbol{\alpha}$ ou 1



Fonctions d'activation 3/4

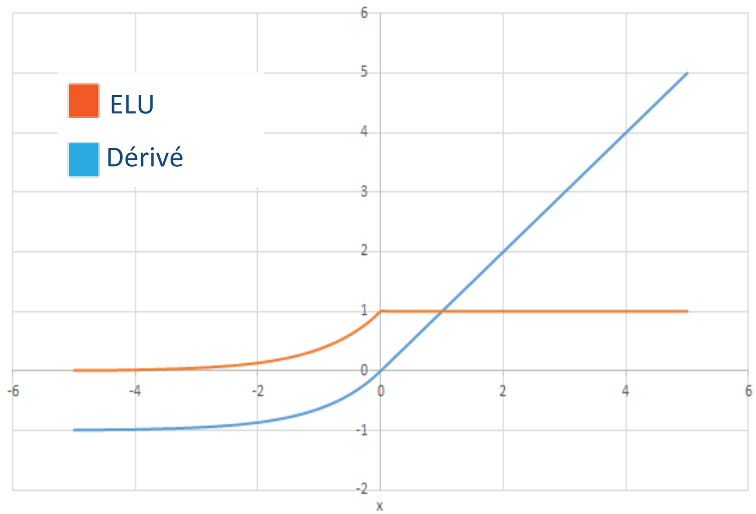
Étant donné $\mathbf{p}$ neurones $(x_1, \ldots, x_i, x_p)$ dans la couche $\mathbf{n-1}$ et $\mathbf{k}$ neurones $(Y_1, \ldots, Y_j, Y_k)$ dans la couche $\mathbf{n}$

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

Elu (Exponential Linear Units)

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

Sa dérivé $f'(\mathbf{Y_j}) = \begin{cases} f(\mathbf{Y_j}) + \boldsymbol{\alpha} & \text{si } \mathbf{Y_j} < 0 \\ \mathbf{1} & \text{si } \mathbf{Y_j} \geq 0 \end{cases} \qquad]-\infty, 1]$

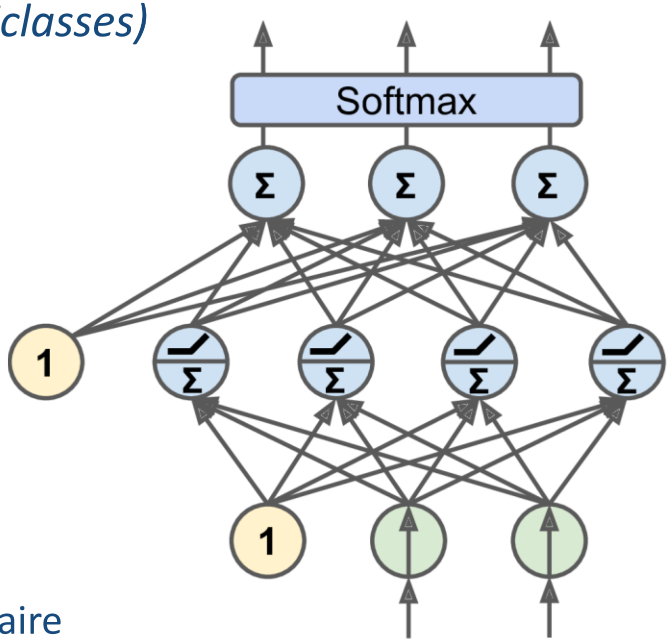


ELU > leaky ReLU > ReLU > tanh > Sigmoïde

Fonctions d'activation 4/4

Softmax *(souvent utilisée pour activer les neurones de la couche de sortie dans un cas de classification muticlasses)*

Étant donné $\mathbf{p}$ neurones $(x_1, \ldots, x_i, x_p)$ dans **la dernière couche cachée** et $\mathbf{k}$ neurones $(Y_1, \ldots, Y_j, Y_k)$ dans **la couche de sortie**



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

$i \in [1..p]$

$j \in [1..k]$

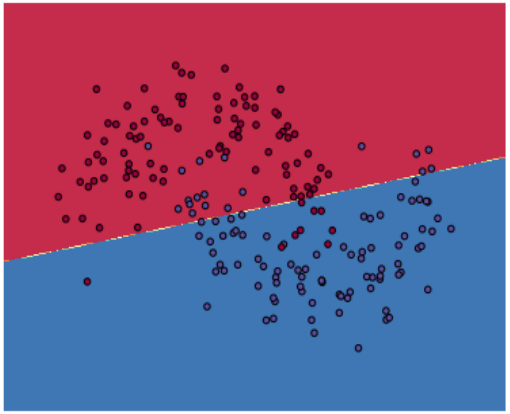
$$f'(Z_i) = \begin{cases} f(Z_i)(1 - f(Z_j)) & \text{si } i = j \\ -f(Z_i)f(Z_j) & \text{si } i \neq j \end{cases}$$

Pour $\mathbf{i}$ de 1 à p faire

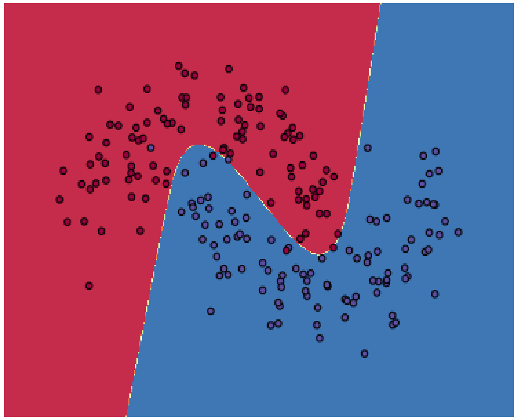
    Pour $\mathbf{j}$ de 1 à k faire

        calculer $f'(Z_i)$?

Impacts des couches cachées!



0 Couche cachée



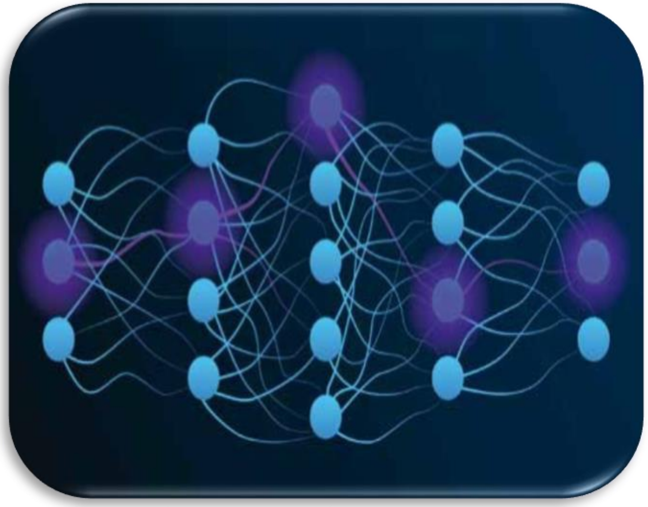
3 Couches cachées

20 Couches cachées

Apprentissage d'un MLP Objectifs

- Calculer l'erreur entre les classes prédites & celles désirées de l'ensemble d'apprentissage
- Déterminer la manière dont chaque neurone des couches cachées contribue à l'erreur

Modifier les poids du réseau de neurones pour ***minimiser l'erreur***



- La minimisation des erreurs est obtenue par la méthode de la **descente de gradient**
- L'erreur sur chaque neurone est calculée à l'aide de la **rétropropagation**

Descente de gradient

1- Initialiser les **poids** aléatoirement

2- Répéter

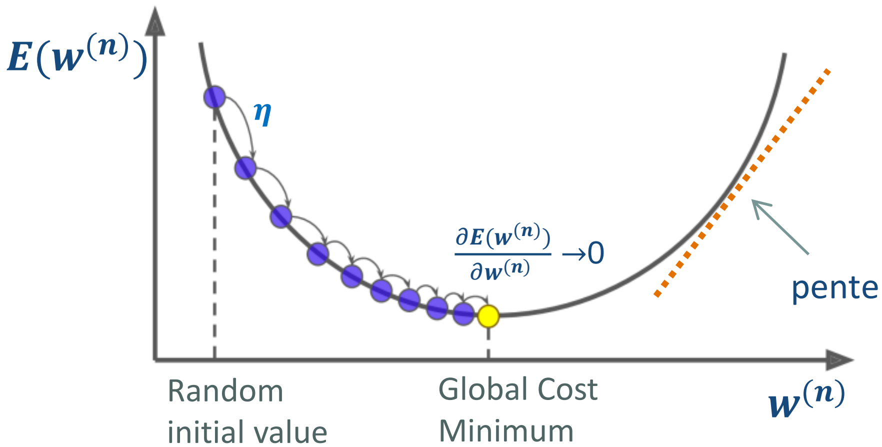
a. calculer le gradient (pente)

$$\frac{\partial E(w^{(n)})}{\partial w^{(n)}}$$

b. Mettre à jour les poids

$$w^{(n)} = w^{(n)} - \eta \frac{\partial E(w^{(n)})}{\partial w^{(n)}}$$

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



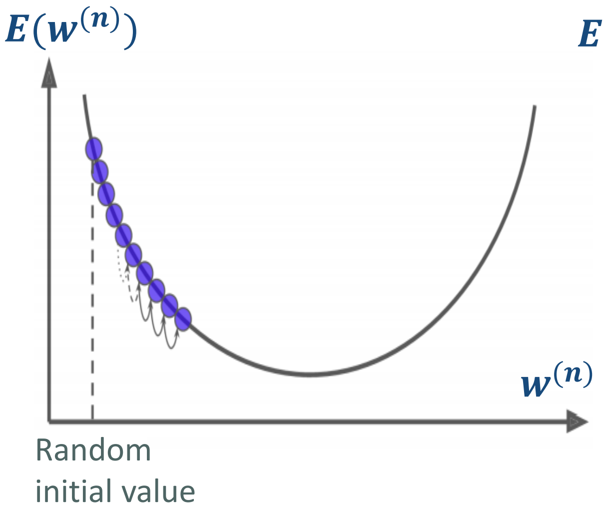
(Extrema)

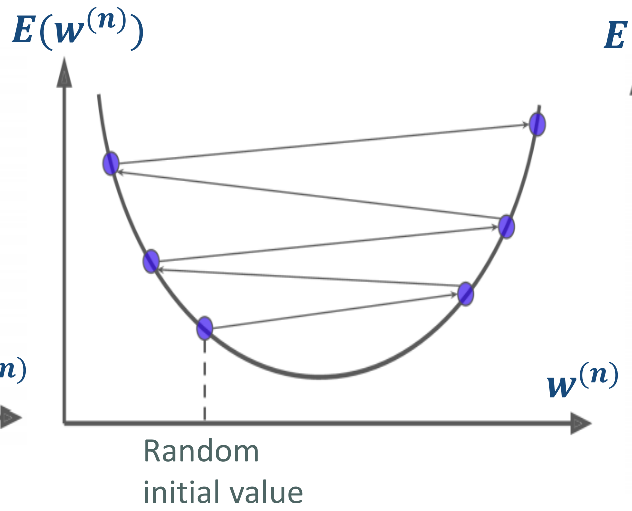
η *pas* d'apprentissage (learning rate)
$w^{(n)}$ poids reliant les neurones de couches n et $n+1$
$E(w^{(n)})$ Fonction du perte (Loss)

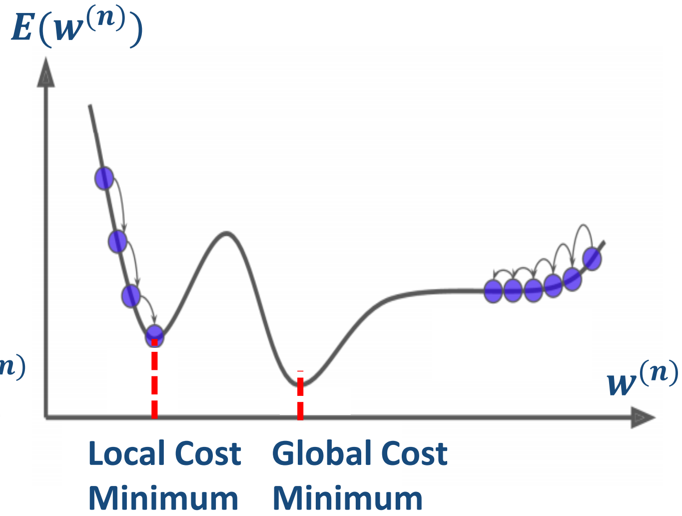
$\frac{\partial E(w^{(n)})}{\partial w^{(n)}} < 0 \rightarrow$ pente descend vers la droite

$\frac{\partial E(w^{(n)})}{\partial w^{(n)}} > 0 \rightarrow$ pente monte vers la droite

Problèmes de descente de gradient*







η EST TROP PETIT

☹ Nombreuses itérations pour converger

☹ Temps important de calcul

η EST TROP ÉLEVÉ

☹ Sauter d'un côté à un autre

☹ Diverger l'algorithme

TOMBER DANS LE MINIMUM LOCAL

☹ Local Cost Minimum>Global Cost Minimum

Fonction du perte $E(w^{(n)})$

- Classification multiclasses : **Cross Entropy**

$$E(w^{(n)}) = -\frac{1}{N}\sum_{j=1}^{N}\sum_{d=1}^{M}[\hat{y}_d^{(j)}\log o_d^{(j)} + \left(1-\hat{y}_d^{(j)}\right)\log(1-o_d^{(j)})]$$

- Classification binaire : **Binary Cross Entropy**

$$E(w^{(n)}) = -\frac{1}{N}\sum_{j=1}^{N}[\hat{y}^{(j)}\log o^{(j)} + \left(1-\hat{y}^{(j)}\right)\log(1-o^{(j)})]$$

- Régression : **Quadratic Cost Function**

$$E(w^{(n)}) = \frac{1}{2}\sum_{d=1}^{M}(\hat{y}_d - o_d)^2$$

$D < X, t >$ Ensemble d'apprentissage (Training set) de N observations
X Matrice de descripteurs (Input vector)
o vecteur des classes désirées (Actual output)
$\hat{y}$ vecteur des classes prédites (Target output)
M Nombre de neurones dans la couche de sortie (*Nombre de classes*)

Epoch, size_batch & Iteration*

ÉPOQUE (EPOCH)

Passage sur l'ensemble de **toutes** les $\mathbf{n}$ 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

*Batch $\neq$ Size_batch!

Variantes de descente de gradient

Descente de gradient classique

Batch Gradient Descent (BGD)

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

☺ 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)

☺ Mise à jour rapide des poids

☹ Instable

☹ Difficile de converger vers l'extrema

Descente de gradient Mini-batch

Mini-Batch Gradient Descent (MBGD)

Mettre à jour par blocs d'observations (taille d'un batch

☺ Rapide, stable et efficace

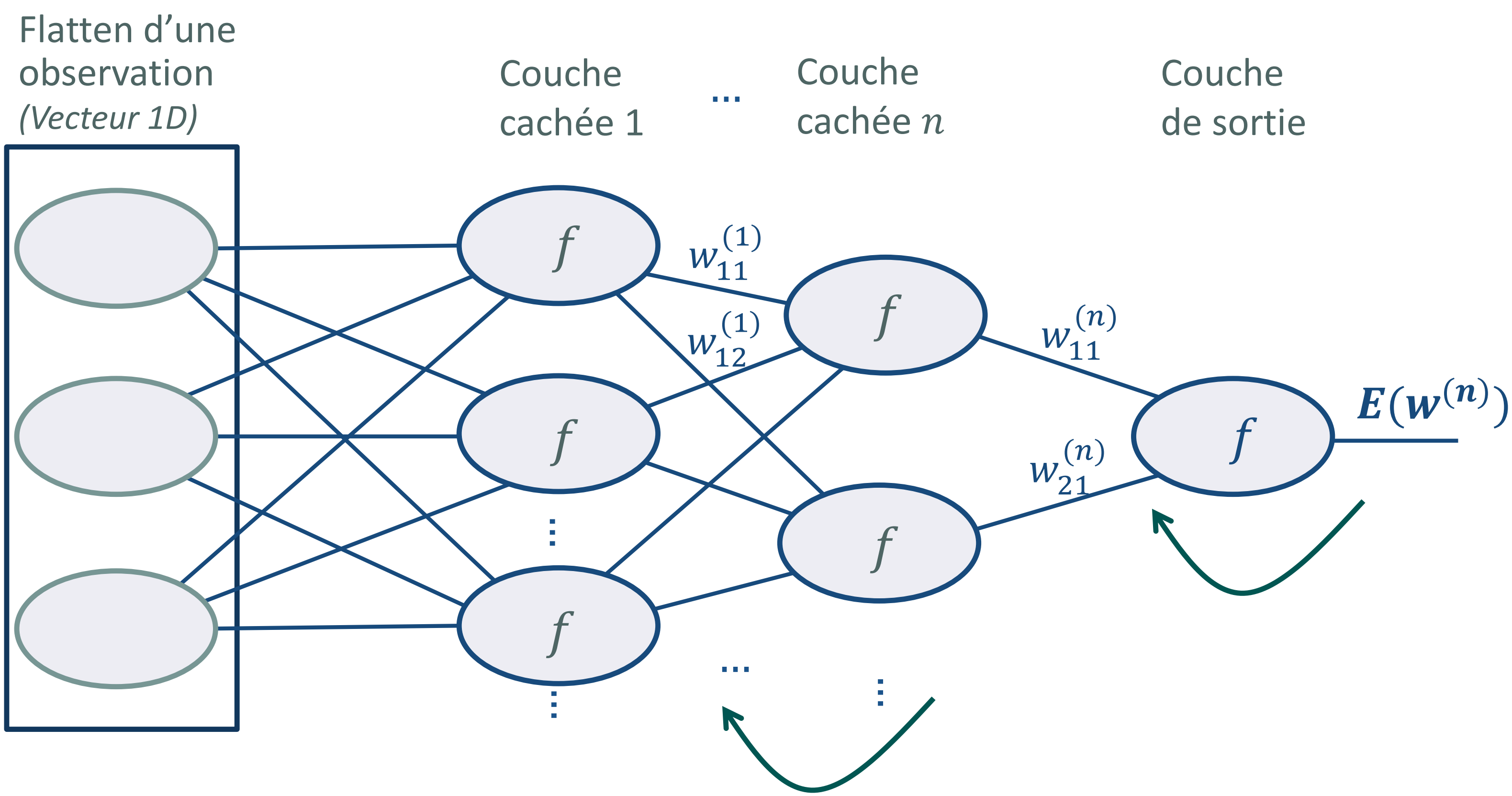
☺ Facile de dépasser le minimum local

NB. batch-size varie en fonction des problèmes spécifiques (généralement **batch-size** =128)

Quiz

1. Étant donné [1,2,4,2,1], l'entrée d'une fonction Softmax, laquelle des solutions suivantes peut être la sortie ? (une seule réponse)
   - [ ] [0.04, 0.20, 0.75, 0.20, 0.04]
   - [x] [0.04, 0.10, 0.72, 0.10, 0.04]
   - [ ] [10]
   - [ ] [3]
2. Quelles sont les fonctions fréquemment utilisées pour l'activation des neurones de couches cachées ? (choix multiples)
   - [x] Relu
   - [x] Tanh
   - [x] Sigmoid
   - [ ] Softmax
3. L'algorithme de descente de gradient est rapide et stable (une seule réponse)
   - [ ] Vrai
   - [x] Faux

Rétropropagation du gradient 1/2



Rétropropager l'erreur $E(w^{(n)})$ jusqu'aux entrées & corriger les poids

$w^{(j)}$ poids reliant les neurones de couches j et $j+1$ ($j = 1..n$)
$E(w^{(n)})$ Fonction du perte (Loss)

Rétropropagation du gradient 2/2

Étant donné $\boldsymbol{E}(\boldsymbol{w}^{(n)})$ l'erreur de sortie

1- Calculer l'erreur rétropropagé $\boldsymbol{\delta}$ sur les neurones de la couche de sortie

$$\boxed{\delta_i = E(w^{(n)}) f'(Y_i)}$$

2- Calculer l'erreur rétropropagé $\delta_a^{(j)}$ sur les neurones de couches cachées de $\boldsymbol{n}$ à $\boldsymbol{1}$ ($\boldsymbol{j} = \boldsymbol{n}..\boldsymbol{1}$)

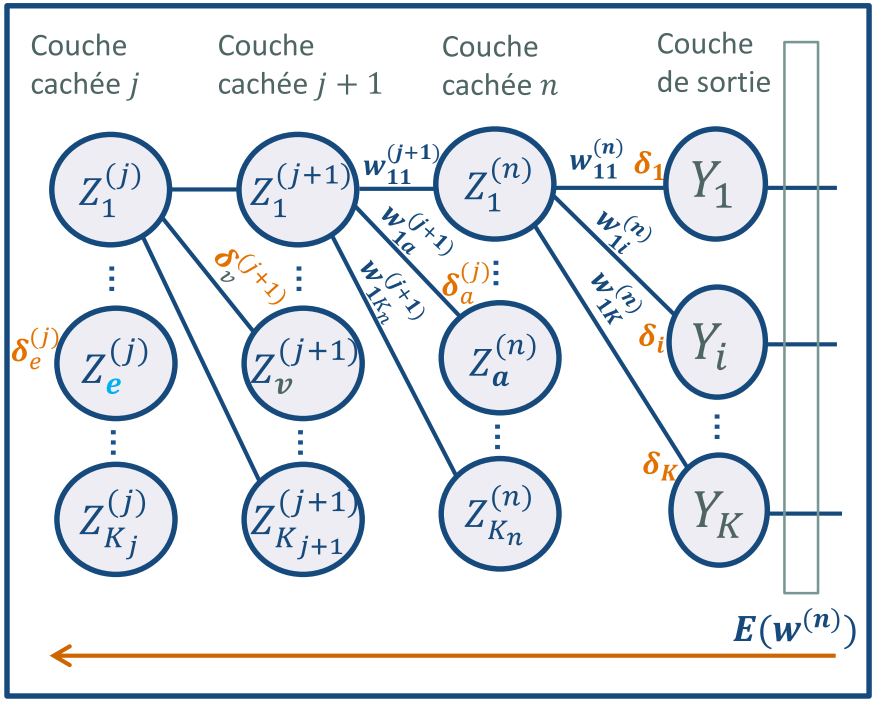
$$\boxed{\delta_a^{(n)} = \left(\sum_{i=1}^{K} \delta_i \, w_{ai}^{(n)}\right) \times f'(Z_a^{(n)})}$$

3- Mettre à jour les poids (bias) des couches cachées $\boldsymbol{w}_{ev}^{(j)}$ ($\boldsymbol{b}_e^{(j)}$) et de la couche de sortie $\boldsymbol{w}_{ai}^{(n)}$ ($\boldsymbol{b}_i$)

$$w_{ai}^{(n)} = w_{ai}^{(n)} - \eta \times \delta_i \times f(Z_a^{(n)})$$

$$w_{ev}^{(j)} = w_{ev}^{(j)} - \eta \times \delta_v^{(j+1)} \times f(Z_e^{(j)})$$

$$b_e^{(j)} = b_e^{(j)} - \eta \times \delta_e^{(j)} \times 1 \quad ; \; b_i = b_i - \eta \times \delta_i \times 1$$



$\boldsymbol{i} = \boldsymbol{1}..\boldsymbol{K}$
$\boldsymbol{a} = \boldsymbol{1}..K_n$; $\boldsymbol{e} = \boldsymbol{1}..K_j$; $\boldsymbol{v} = \boldsymbol{1}..K_{j+1}$
$\boldsymbol{K}$ est le nombre de neurones dans la couche de sortie
K_j est le nombre de neurones dans la couche cachée j

Optimizeur* 1/2

Définir la *manière* d'ajustement de poids

- Accélérer la convergence toute en minimisant la fonction de perte
- Éviter ou dépasser les minimums locaux : atteindre le minimum global
- Simplifier le choix de la force d'apprentissage (learning rate η)

Optimiseurs les plus courants

SGD	Momentum	NAG	Adagrad	Adadelta	RMSprop	Adam

*Stochastic Gradient Descent (SGD)
*Nesterov Accelerated Gradient (NAG)
*Adaptive Gradient Algorithm (Adagrad & Adadelta)
*Root Mean Square Propagation (RMSprop)
*Adaptive Moment Estimation (Adam)

Optimizeur* 2/2

Momentum

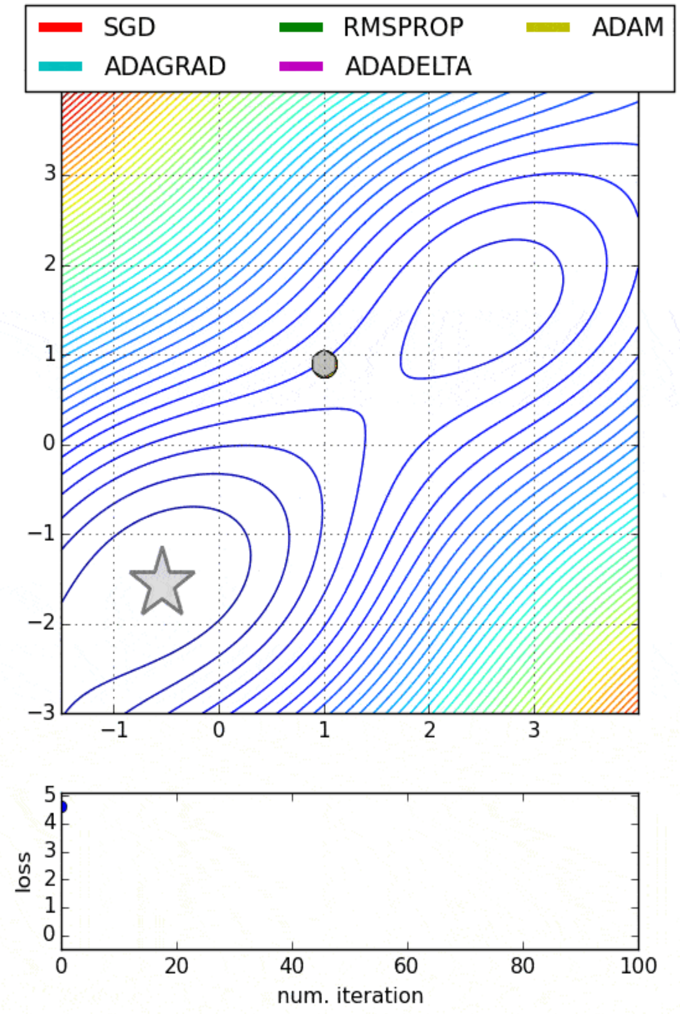
- ☺ Convergence plus rapide que SGD
- ☹ Nécessité de la définition d'un nouveau hyper paramètre $0 < \mathbf{momentum} < 1$

Adagrad & Adadelta

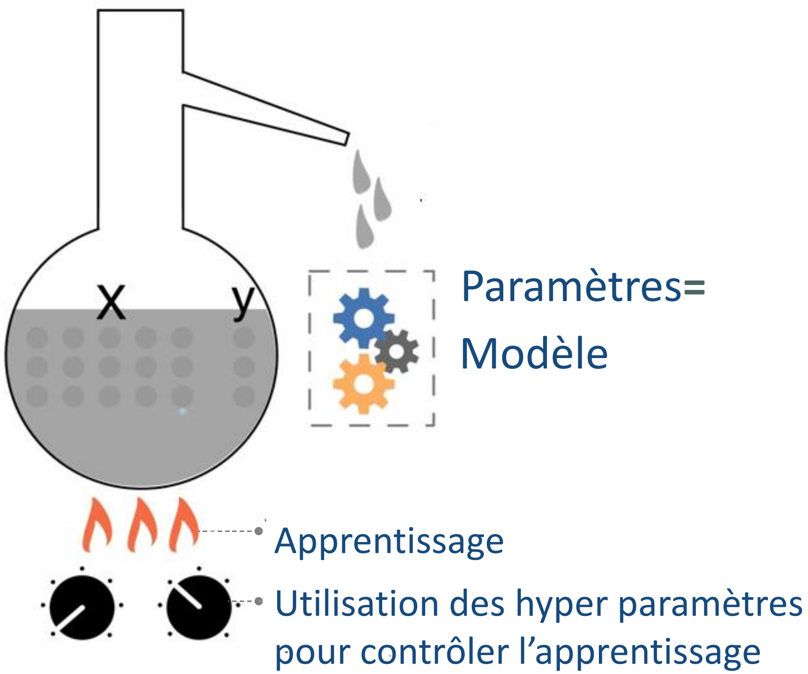
- ☺ Learning rate η modifiable
- ☺ Possible de s'entraîner sur des données peu nombreuses
- ☹ Coûteux en terme de calcul
- ☹ Learning rate η est toujours en diminution → Apprentissage lent

Adam

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



Paramètres & Hyper paramètres



Les **paramètres** sont automatiquement appris par les modèles

- Poids
- Biais

Les **hyper paramètres** sont configurés manuellement

- Learning rate η
- Nombre de couches cachées
- Nombre de neurones dans chaque couche cachée
- Nombre d'époques
- Taille d'un Batch
- Choix d'Optimizeur

Méthodes de recherche des hyper paramètres!

Grid
search

Random
search

Heuristics
intelligent search

Bayesian
search

Quiz

1. Parmi les éléments suivants, lequel n’est pas considéré comme hyper-paramètre dans un réseau de neurones profond (une seule réponse)
   - [ ] Learning rate
   - [x] Les poids
   - [ ] La taille d’un batch
   - [ ] Le nombre d’époques
2. Parmi les algorithmes suivants qui sont considérés comme une méthode d’optimisation dans l’apprentissage profond ? (Choix multiples)
   - [x] Momentum
   - [x] Adam
   - [x] Descente de gradient stochastique
   - [x] Rétropropagation du gradient
3. Lors de l’apprentissage d’un réseau de neurones, l’erreur ne diminue pas lors des premières époques, quelles sont les raisons possibles ? (Choix multiples)
   - [ ] La valeur de learning rate est faible
   - [x] Blocage dans minimum local
   - [x] La valeur de learning rate est élevée

Estimation de performances

Méthode de validation

Objectif : Estimation correcte de l'erreur de classification

→ Utiliser un ensemble d'observations qui n'ont pas servi pour l'apprentissage dans le test

Split validation

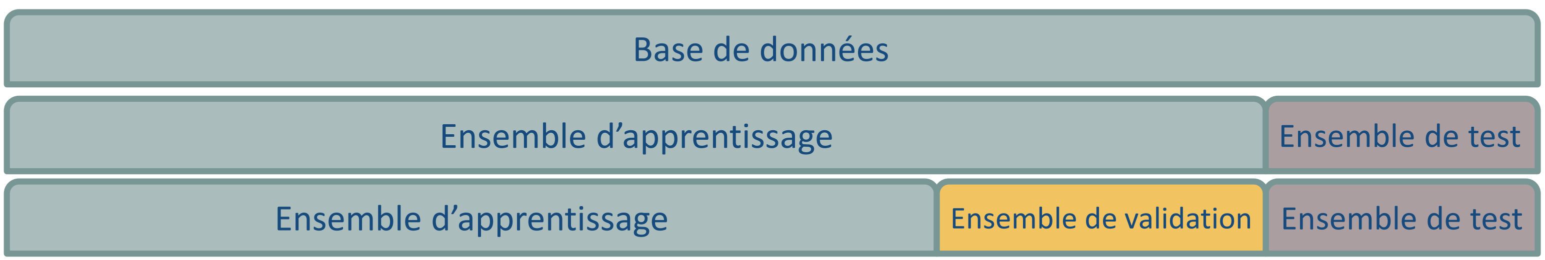
Processus d'apprentissage et de test n'est effectué qu'une seule fois

Cross-validation

Faire plusieurs tests sur différents ensembles d'apprentissage et de test

Échantillonnage*

- Diviser le jeu de données en **deux groupes**
  - ❑ **Ensemble d'apprentissage** : utilisé pour **former le classifieur**
  - ❑ **Ensemble de validation :** utilisé pour déterminer les valeurs des hyper paramètres du modèle
  - ❑ **Ensemble de test** : utilisé pour **estimer la performance** du classifieur formé



→ Processus de validation qui s'exécute qu'une seule fois

Evaluation d'un réseau de neurones

CM		Classe prédite			
		C_1	…	C_i	C_N
Classe désirée	C_1	CM(1,1)		CM(1,i)	
	⋮		⋮		
	C_i			CM(i,i)	
	C_N				CM(N,N)

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

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

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

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

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

True Positive

False Negative

False Positive

True Negative

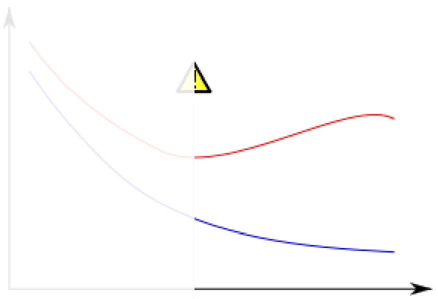
Cas de 2 classes

recall = True Positive rate

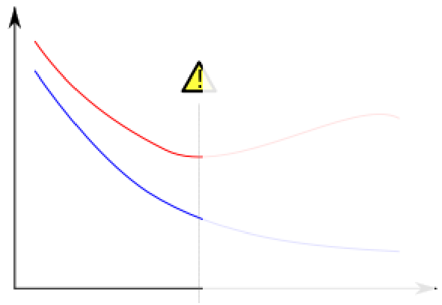
= Sensitivity = $\frac{TP}{TP+FN}$

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

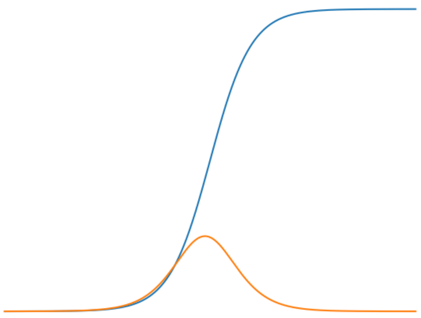
Problèmes dans l'apprentissage profond!



Sur apprentissage (Overfitting)

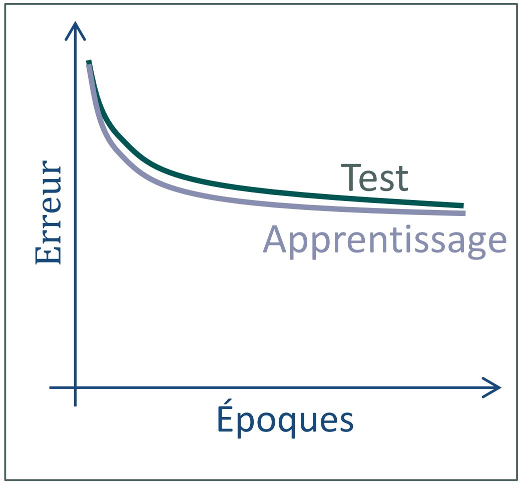


Sous apprentissage (underfiting)

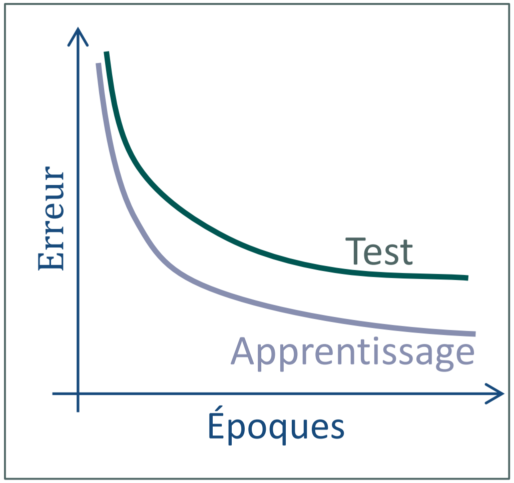


Vanishing Gradient

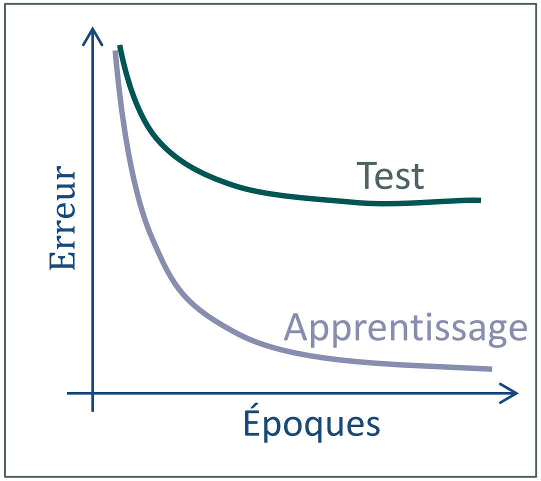
1- Sous-apprentissage vs. Sur-apprentissage (1/2)



Sous-apprentissage
(Underfitting)



Parfait
(perfect)



Sur-apprentissage
(Overfitting)

1- Sous-apprentissage vs. Sur-apprentissage (2/2)

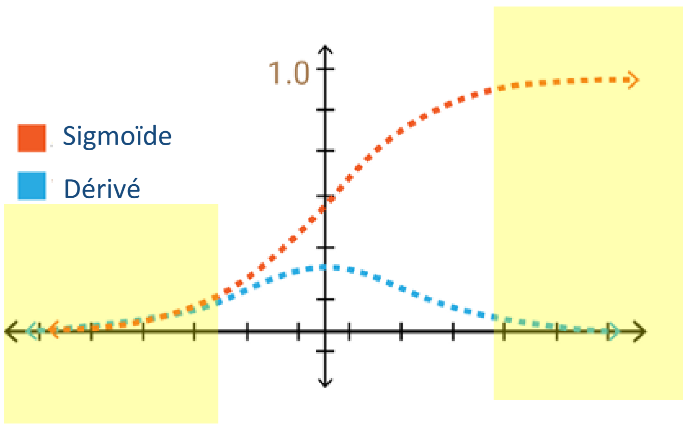
♣ Solutions pour éviter le sur-apprentissage

- Ajouter de contraintes aux paramètres (L_1 and L_2 norms)
- Étendre l'ensemble d'apprentissage
- Dropout
- Early stopping

2- Vanishing Gradient

- Dérivé de la fonction de perte s'approchent de zéro
  - Rendre le réseau difficile à apprendre
  - L'erreur rétropropagé diminue d'une couche à une autre ($\frac{\partial \mathbf{E}}{\partial \mathbf{w}_{\mathbf{pk}}^{(\mathbf{n})}}$ ↘) tend vers 0
  - Les poids et les biais des couches initiales ne seront pas mis à jour
  - Incapable d'apprendre (Sous-apprentissage)

- Fonction sigmoïde !

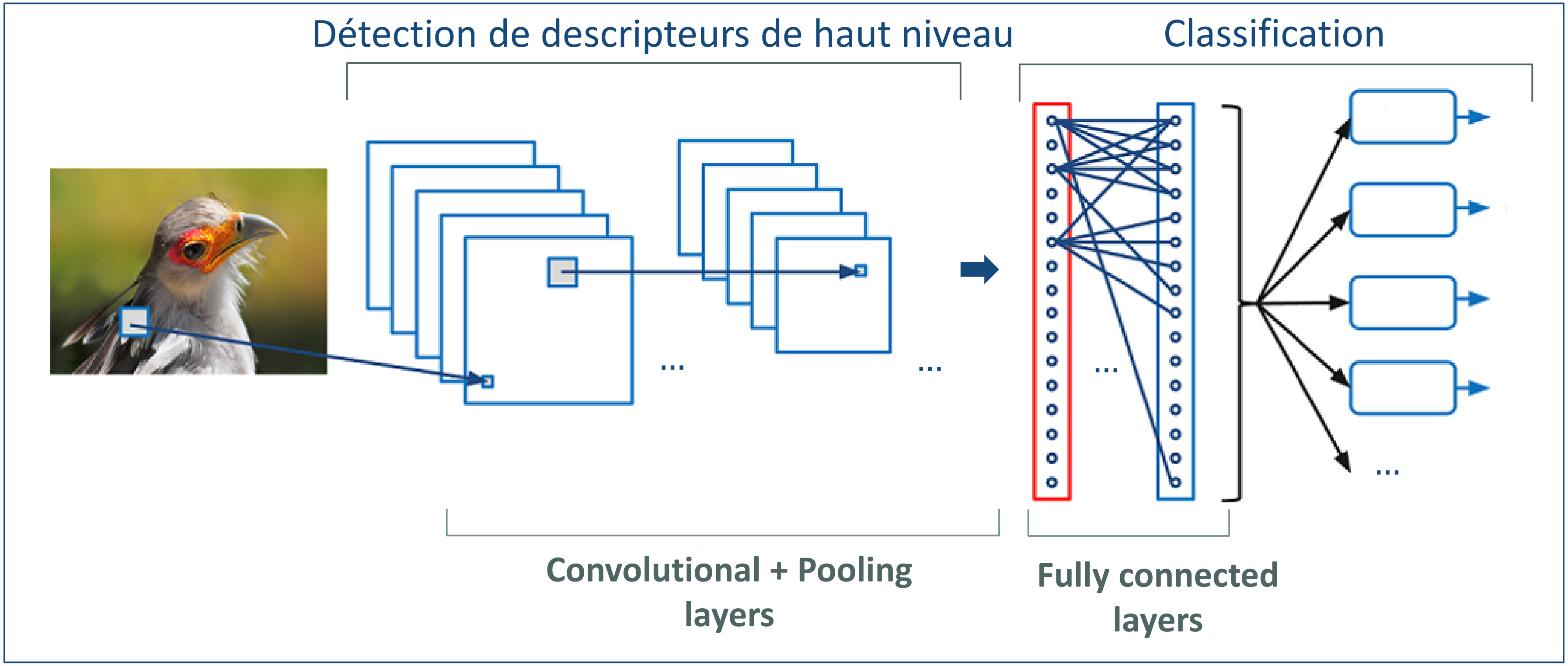


Partie 3

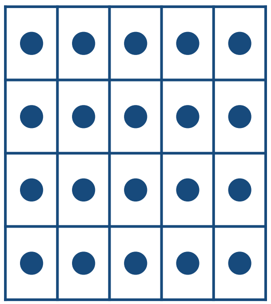
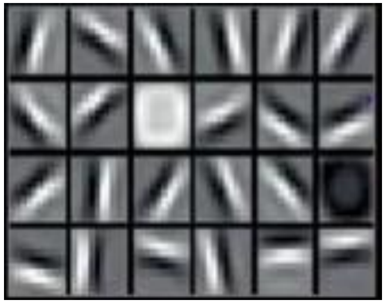
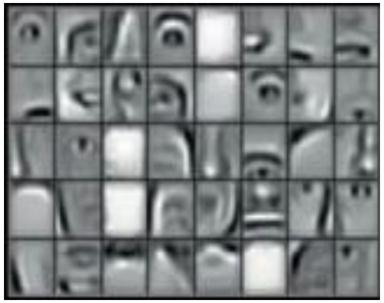
Réseaux de neurones convolutifs

Réseaux de neurones convolutifs

- Convolutional Neural Network (**ConvNet/CNN**)



Descripteurs haut niveau!

	Descripteurs bas niveau (Low level features)	Descripteurs moyen niveau (Middle level features)	Descripteurs haut niveau (High level features)
			
Pixels	Contours (edges)	Parties d'objet (Object parts)	Objet (Object models)

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

Couches de CNN

CONVOLUTION

التفافي

Appliquer des ***filtres*** pour extraire des descripteurs

POOLING

تجميع الميزات

Conserver les descripteurs importantes (réduction)

FLATTENING

جعله مسطح

Convertir les matrices de descripteurs en tableau 1D

FULLY CONNECTED

اتصال كلي

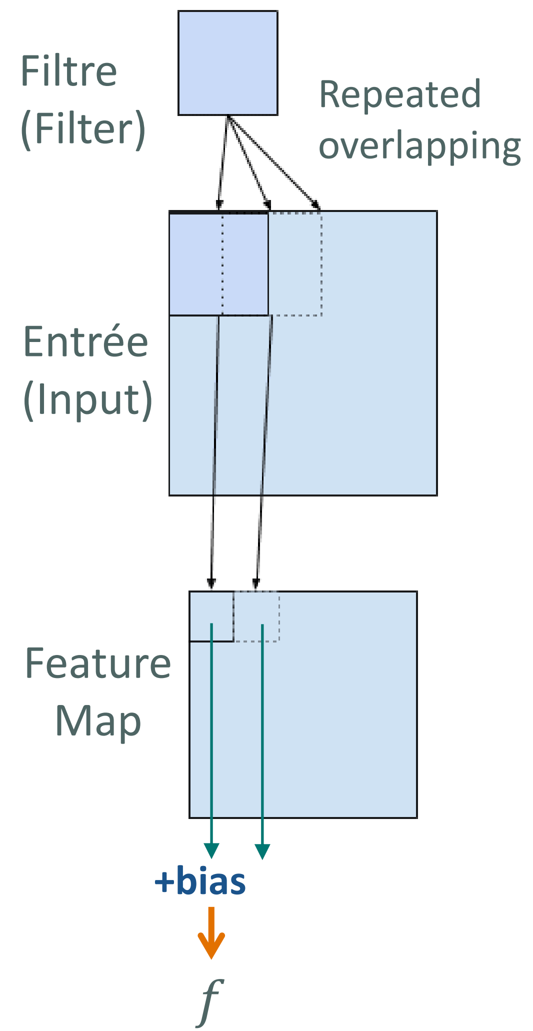
Construire le modèle

1- 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

Une fois la convolution terminée,

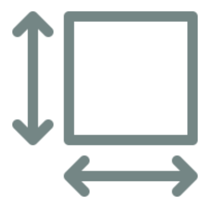
- ☑ Le résultat doit être **biaisé** et **activé** par une **fonction d'activation** f
- ☑ Généralement **Relu** ou l'une de sa famille (Elu, Leaky Relu, etc.)



1- Convolutional layers*

Depth (Volume)

Nombre de filtres



kernel_size

Taille d’un filtre ou un noyau

Stride

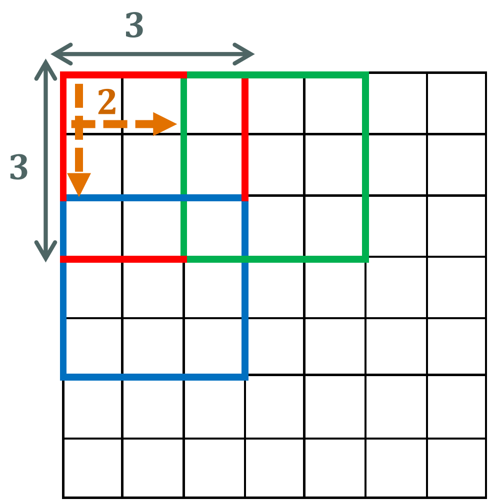
Détermine la façon dont le filtre est glissé sur l'entrée

Padding

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

Convolution simple depth $= 1$

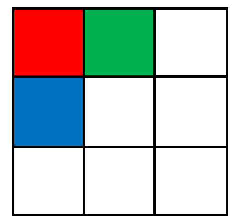
Depth $= 1 \rightarrow$ **single-Filter**



Entrée 7×7

kernel_size= 3×3

Stride= **2**



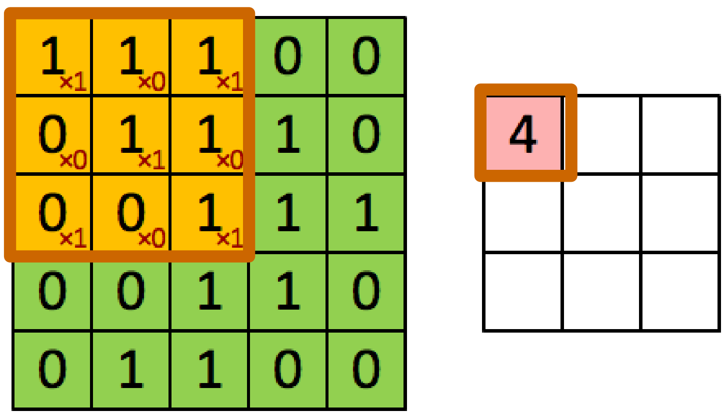
Sortie 3×3 **?**

Entrée

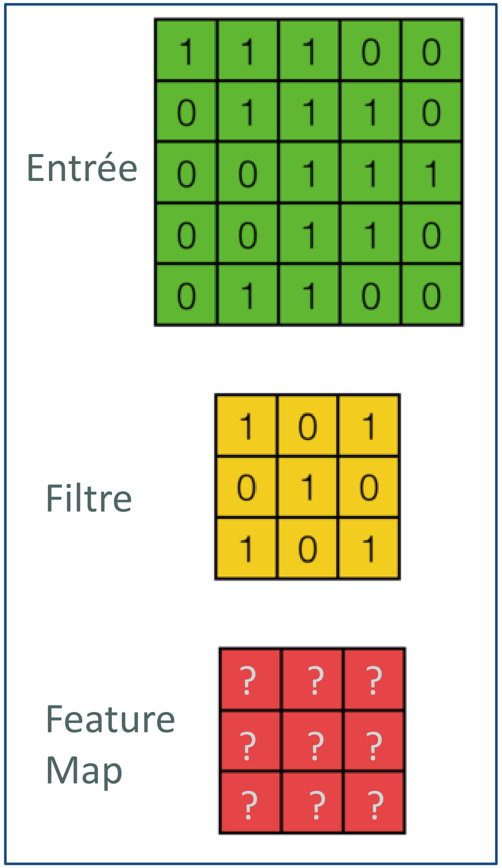
Convolution simple depth = 1

Exemple de calcul

Étant donné Entrée= **(5,13)**; Taille du filtre=**(3,3)**; et Stride=1



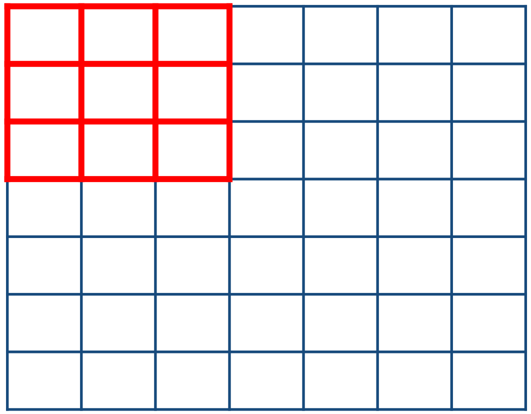
$1 \times 1 + 1 \times 0 + 1 \times 1 + 0 \times 0 + 1 \times 1 + 1 \times 0 + 0 \times 1 + 0 \times 0 + 1 \times 1$**=4**



Padding

- padding=**"VALID"** : pas de remplissage par des zéros→**Le taille sera réduit** : Ignorer certaines lignes et colonnes de la matrice d'entrée, en fonction de Stride

$$\mathbf{P}=\boldsymbol{P_H}=\boldsymbol{P_L}=\mathbf{0}$$



- padding=**"SAME"** : des zéros sont ajoutés régulièrement **pour s'assurer que la sortie a la même taille que l'entrée**

$$\mathbf{P} =(\boldsymbol{P_H}= \frac{\mathbf{H_f}-\mathbf{1}}{\mathbf{2}} \,;\, \boldsymbol{P_L}= \frac{\mathbf{H_L}-\mathbf{1}}{\mathbf{2}})$$

$\mathbf{H_f}, \mathbf{L_f}$ sont généralement **impairs**
Avec $\mathbf{H_f} \times \mathbf{L_f}$ = taille de filtre

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

Convolution en volume Formule générale

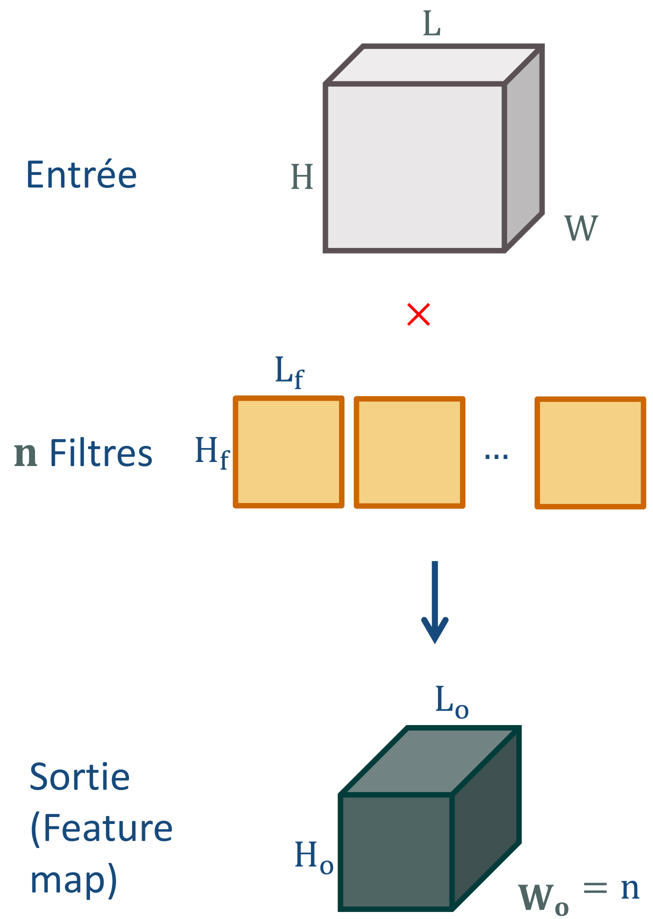
Étant donné,

L'entrée (Input) est de taille $\mathbf{H} \times \mathbf{L} \times \mathbf{W}$
$\mathbf{H}$=height= nombre de lignes
$\mathbf{L}$ =length= nombre de colonnes
$\mathbf{W}$= width= *nombre de channels*

$\mathbf{n}$ **Filtres** de taille $\mathbf{H_f} \times \mathbf{L_f}$
$\mathbf{H_f}, \mathbf{L_f}$ sont généralement impairs
avec **Stride** $\boldsymbol{S}(\boldsymbol{s1}, \boldsymbol{s2})$
Padding $\mathbf{P}=\boldsymbol{P_H}=\boldsymbol{P_L}$

Sortie (output) =$\mathbf{H_o} \times \mathbf{L_o} \times \mathbf{n}$

$$\mathbf{H_o} = \frac{\mathbf{H} + \boldsymbol{2P} - \mathbf{H_f}}{\boldsymbol{s1}} + 1 \; ; \; \mathbf{L_o} = \frac{\mathbf{L} + \boldsymbol{2P} - \mathbf{L_f}}{\boldsymbol{s2}} + 1$$



Exemple avec depth = 3

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

Input Channel 1

×

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

Kernel 1 Channel 1

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

Input Channel 2

×

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

Kernel 1 Channel 2

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

Input Channel 3

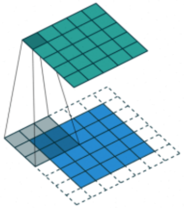
×

0	1	1
0	1	0
1	-1	1

Kernel 1 Channel 3

308 + -489 + 164 +1=-25

Biais



2- Pooling layers 1/4

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

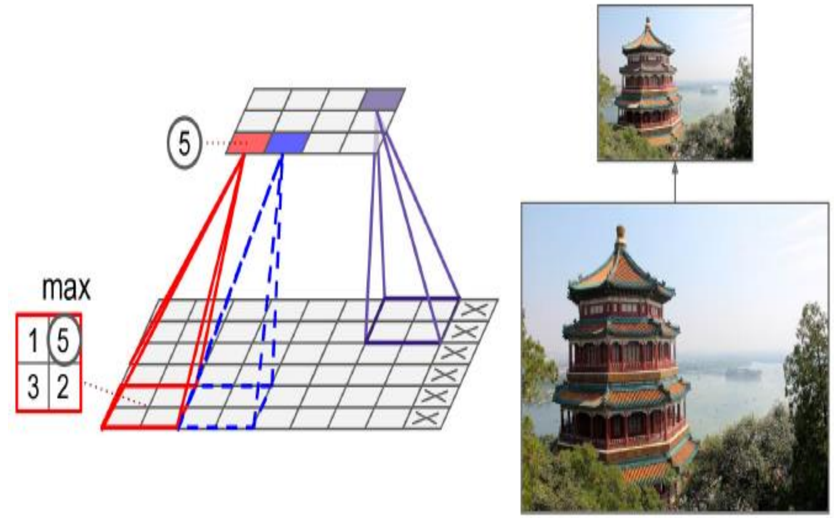
Max pooling & Average pooling

- En général, la *taille du noyau* dans Pooling **pool_size** est **(2,2)**
- Le **Max pooling** est le plus utilisé

☺ Prévenir le sur apprentissage

☺ Obtenir des données de longueur fixe

☺ Invariance



pool_size=**(2,2)=2**
Max pooling
stride**= 2**
Pas de padding

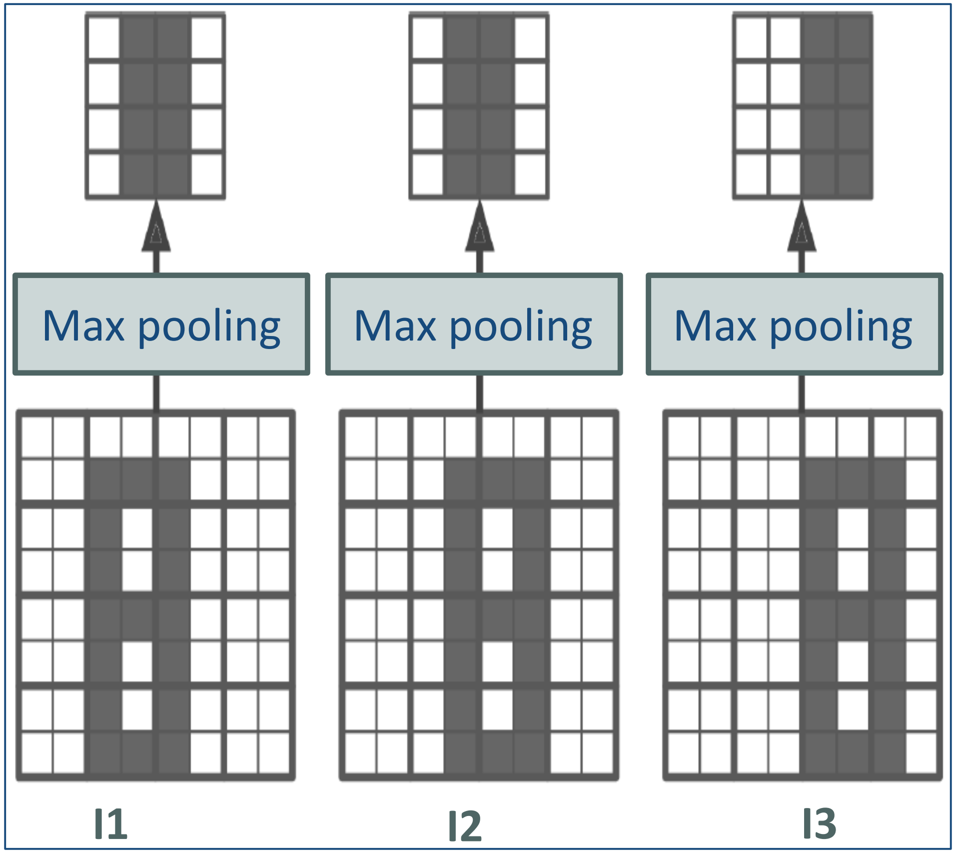
Étant donné **I1,I2** et **I3** trois images
I1, **I2** et **I3** représentent les mêmes images mais chacune est décalée de 2 pixels vers la droite

Valeur de pixel □ < valeur de pixel ■

Noyau de Pooling=(2,2) ; Stride=2

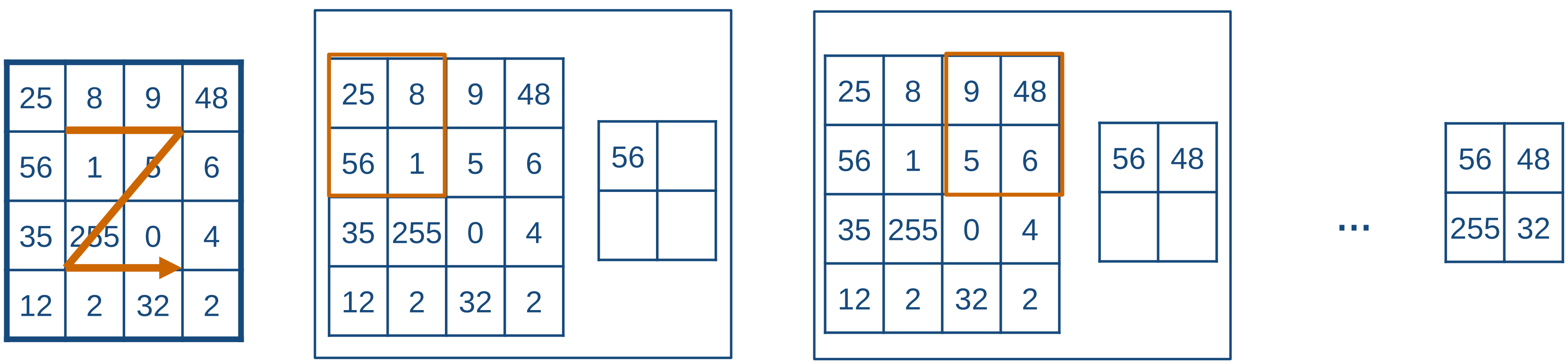
→ En appliquant **Max pooling** : **I1=I2**

→ Le **Max pooling** garantit **l'invariance** dans une certaine marge

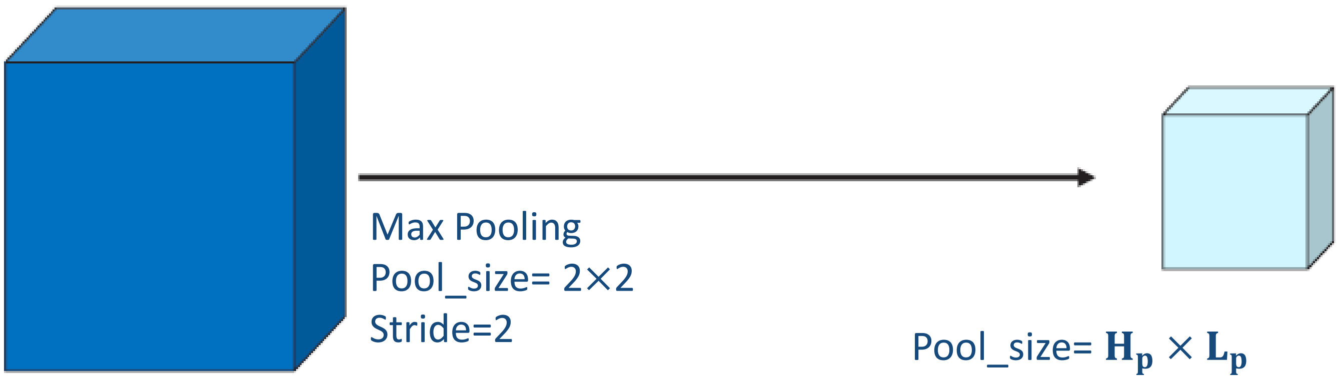


♣ Exemple

Étant donné pool_size=**(2,2)=2**; Max pooling; stride**= 2**; et Pas de padding



2- Pooling layers 4/4



Pool_size= $\mathbf{H_p} \times \mathbf{L_p}$

padding=**"VALID"** $\mathbf{H_o} = \frac{\mathbf{H}-\mathbf{H_p}}{s1} + 1 \; ; \; \mathbf{L_o} = \frac{\mathbf{L}-\mathbf{L_p}}{s2} + 1$

padding=**"SAME"** $\mathbf{H_o} = \frac{\mathbf{H}}{s1} \; ; \; \mathbf{L_o} = \frac{\mathbf{L}}{s2}$

Quiz

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

- [x] 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 longueur variable
- [ ] Un vecteur 1D de longueur variable
- [ ] Un vecteur 2D de longueur fixe
- [x] Un vecteur 1D de longueur fixe

3. Les réseaux de neurones convolutifs permettent d'extraire automatiquement des descripteurs de haut niveaux à partir des images (une seule réponse)

- [x] Vrai
- [ ] Faux

MERCI

POUR

VOTRE ATTENTION

شكرًا على المتابعة*