

# Modèles Mathématiques des Réseaux de Neurones

Adnène ARBI

ENSTAB-EPT

Université de Carthage

# Contexte: Classification et régression

Données:

• Les réponses:

❖ Régression:  $y_i \in \mathbb{R}$ ,

❖ Classification:  $y_i \in C, C = \{1, 2, \dots, k\}$

Problème:

❖ Expliquez Y en fonction de X.

Principe:

❖ Echantillon supposé iid:  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$

❖ Modèle estimé:  $Y \sqsubset \tilde{f}(X)$

# Inférence

L'inférence (ou l'entraînement) d'un réseau de neurones consiste à estimer ses paramètres (ie les matrices de poids  $W_k$  pour chaque couche  $k$ ) sachant un ensemble d'apprentissage  $\{(x_1, y_1), \dots, (x_n, y_n)\}$ .

## Régression

Si la sortie  $y$  est une variable quantitative, on utilise en général la fonction de perte des moindres carrés :

$$\hat{f} = \arg \min_{f \in \mathcal{F}} \sum_{i=1}^n (y_i - f(x_i))^2$$

où on note  $\mathcal{F}$  l'ensemble des fonctions  $f$  obtenues en faisant varier les matrices de poids (pour une architecture fixée).

## Classification

Si la sortie  $y$  est une variable binaire,  $f(x)$  modélise la probabilité que  $y$  vaille 1 et on utilise la vraisemblance<sup>1</sup>

$$\hat{f} = \arg \min_{f \in \mathcal{F}} - \sum_{i=1}^n y_i \log f(x_i) + (1 - y_i) \log(1 - f(x_i))$$

# Minimisation de la fonction perte

La méthode usuelle pour optimiser les paramètres d'un réseau de neurones repose sur un algorithme de descente de gradient.

En notant  $\theta$  le vecteur des paramètres et

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n J_i(\theta)$$

la fonction de perte, on répète jusqu'à convergence :

$$\theta_r = \theta_{r-1} + \alpha \frac{1}{n} \sum_{i=1}^n \nabla_{\theta} J_i(\theta_r) \quad (1)$$

En pratique, on utilise la forme particulière du réseau pour obtenir l'algorithme efficace de **rétropropagation du gradient**.

## Algorithme du gradient stochastique

La théorie de l'approximation stochastique enseigne que (1) est inutilement onéreux et qu'il vaut mieux, en termes de coût calcul, réaliser un algorithme de la forme

$$\theta_r = \theta_{r-1} + \alpha_r \nabla_{\theta} J_i(\theta_r)$$

en bouclant éventuellement sur l'ensemble d'apprentissage.

En effet, intuitivement, on se doute que dans les premières étapes de la descente de gradient, il ne sert à rien d'introduire à tous les échantillons, puisque cette étape est forcément approximative ; et en fait le plus efficace est de les rentrer un à un.

Le taux d'apprentissage théorique est de l'ordre de  $1/r$ , mais choisir le meilleur n'est pas évident.

La parallélisation fait qu'on a en fait intérêt à faire entrer à chaque étape des **mini-batchs** dont la taille est de l'ordre de grandeur de la parallélisation

$$\theta_r = \theta_{r-1} + \alpha_r \frac{1}{B} \sum_{i=rB+1}^{(r+1)B} \nabla_{\theta} J_i(\theta_r)$$

## Divers algorithmes stochastiques

En deep learning, plusieurs variantes de l'algorithme de gradient stochastique (SGD) ont été proposées.

### ► Momentum

Cette méthode consiste à remplacer le gradient par une moyenne pondérée de tous les gradients "historiques". On utilise un poids avec oubli exponentiel ce qui conduit à la forme connue pour la moyenne mobile.

$$\begin{aligned}m_r &= \alpha m_{t-1} + (1 - \alpha) \nabla_{\theta} J(\theta; \mathbf{x}_{i:i+k}, y_{i:i+k}) \\ \theta_{r+1} &= \theta_r - \gamma m_r\end{aligned}$$

$\alpha$  est en général choisi proche de 0.9.

Cette technique est très utilisée en pratique.

### ► RMSprop

Dans cette variante, on joue sur le taux d'apprentissage.

$$\begin{aligned}v_r &= \alpha v_{r-1} + (1 - \alpha) (\nabla_{\theta} J(\theta; \mathbf{x}_{i:i+k}, y_{i:i+k}))^2 \\ \theta_{r+1} &= \theta_r - \frac{\gamma}{\sqrt{v_r + \epsilon}} \nabla_{\theta} J(\theta; \mathbf{x}_{i:i+k}, y_{i:i+k})\end{aligned}$$

L'intuition ici est qu'on peut faire des pas plus grands quand le gradient est faible (on est sur une zone plate de la fonction objectif) et inversement.

### ► Adam

Dans l'algorithme Adam, on combine les idées de Momentum et RMSprop.

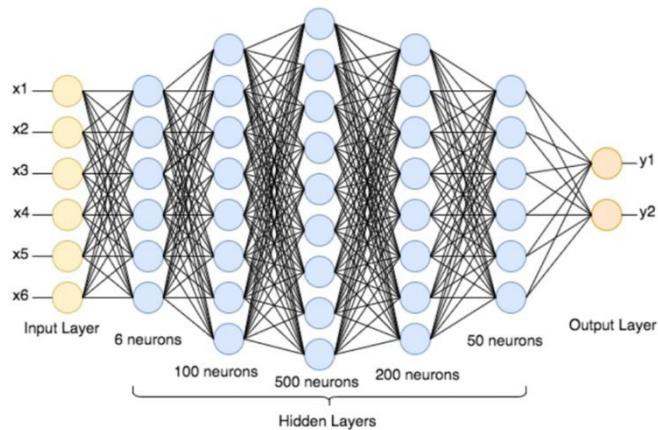
# Questions

- Qu'est ce que l'intelligence artificielle?
- Pourquoi le machine Learning?
- Qu'est ce que le machine Learning ?
- Comment fonctionne le machine learning?
- Pourquoi le deeplearning?
- C'est quoi le deeplearning?

# Introduction

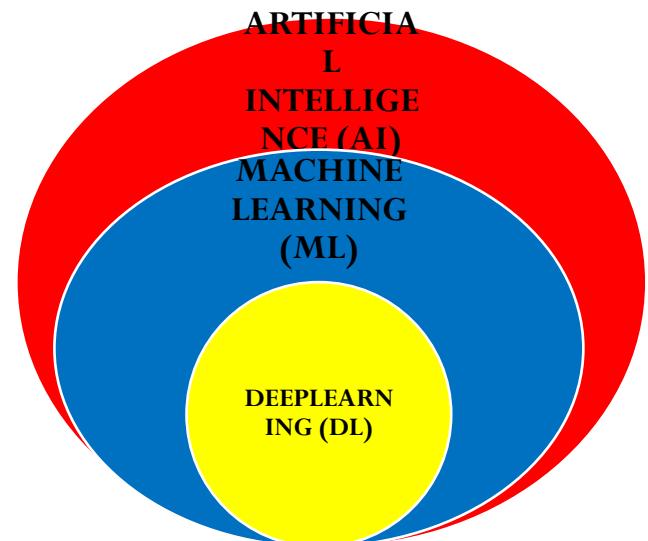
## Qu'est ce que l'Intelligence Artificielle?

Ensemble des théories, des méthodes, des techniques et des algorithmes mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence humaine



## Qu'est ce que ML et DL?

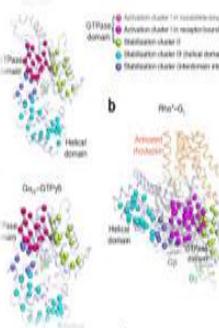
un type d'intelligence artificielle dérivé du machine learning (apprentissage automatique) où la machine est capable **d'apprendre** par **elle-même** qui s'appuie sur un réseau de neurone artificiel s'inspirant du cerveau humain composé de plusieurs neurones



# Domaines d'applications



Sécurité



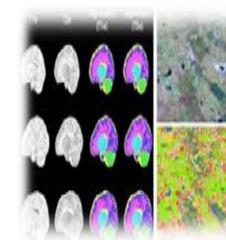
Biologie



Finance



Text Mining

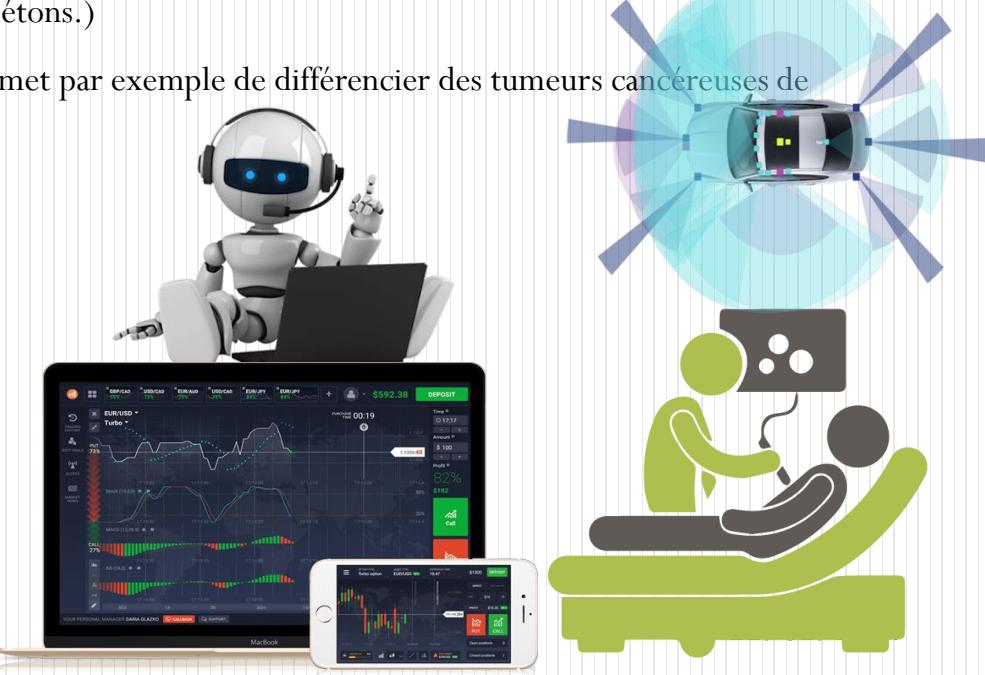


Segmentation d'image



Reconnaissance de signature

- ❖ La sécurité d'identité ( la reconnaissance faciale, la reconnaissance des empreintes digitales )
- ❖ Le Natural Language Processing ( Extraire le sens des mots , traduction automatique)
- ❖ Voiture autonome,(combine plusieurs algorithmes d'apprentissage profond, comme ceux reconnaissant les panneaux de signalisation ou ceux localisant les piétons.)
- ❖ Diagnostic médical,(L'intelligence artificielle permet par exemple de différencier des tumeurs cancéreuses de celles qui ne le sont pas)
- ❖ Modération automatique des réseaux sociaux,
- ❖ Prédiction financière et trading automatisé,
- ❖ Identification de pièces défectueuses,
- ❖ Détection de malwares ou de fraudes,
- ❖ Chatbots ,
- ❖ Exploration spatiale,
- ❖ Robots intelligents.



# Classification automatique des données multidimensionnelles

Croissance des données nécessite des méthodes d'analyse à des fin:

- ✓ **Exploratoire** : rechercher des régularités dans les données
- ✓ **Confirmatoire** : répondre à des questions précises
- ✓ **Descriptive** : caractériser les données observées
- ✓ **Décisionnelle** : aller au-delà des données observées



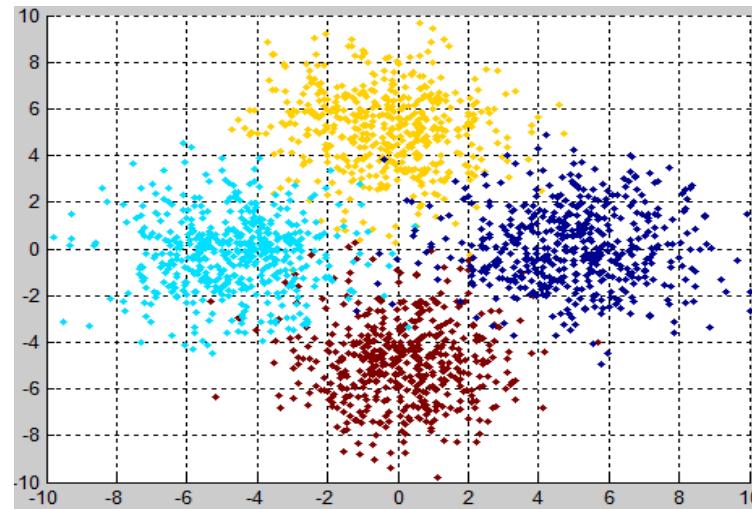
Ce qui nécessite des méthodes puissantes de classification  
des données

La *classification automatique (clustering)* est l'opération d'arranger, selon certains critères, une distribution composée d'un ensemble d'observations dans des groupes contenant des observations homogènes, appelés classes.

# Représentation des données

Soit un ensemble E d'observations représentées par des points dans l'espace Euclidien  $R^N$  :

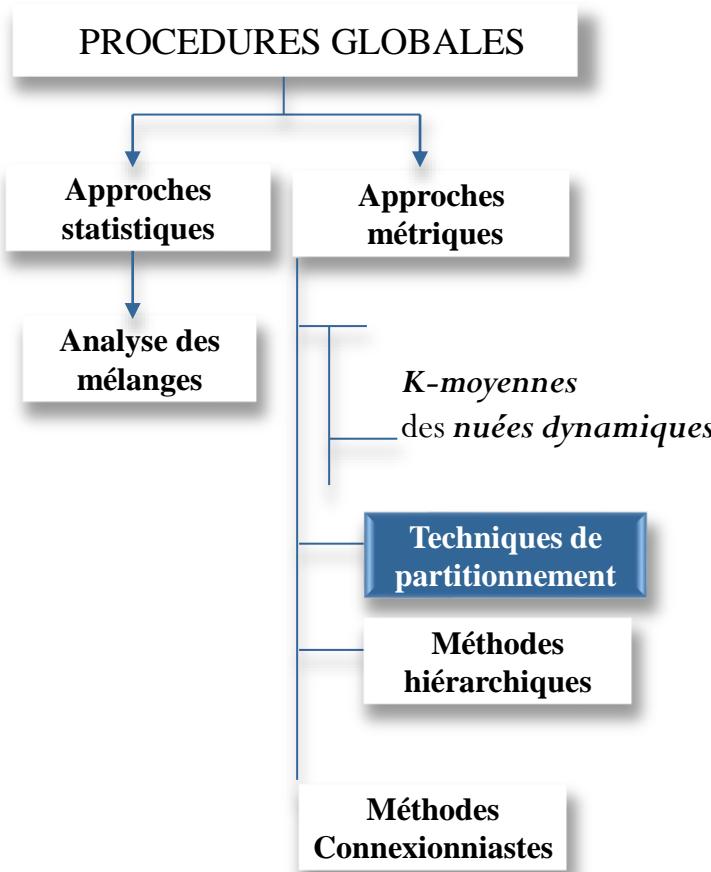
$$E = \{x_1, x_2, \dots, x_q, \dots, x_Q\} \quad \text{Un ensemble de } Q \text{ observations tel que:} \quad X_q = |x_{q,1}, x_{q,2}, \dots, x_{q,n}, \dots, x_{Q,N}|$$



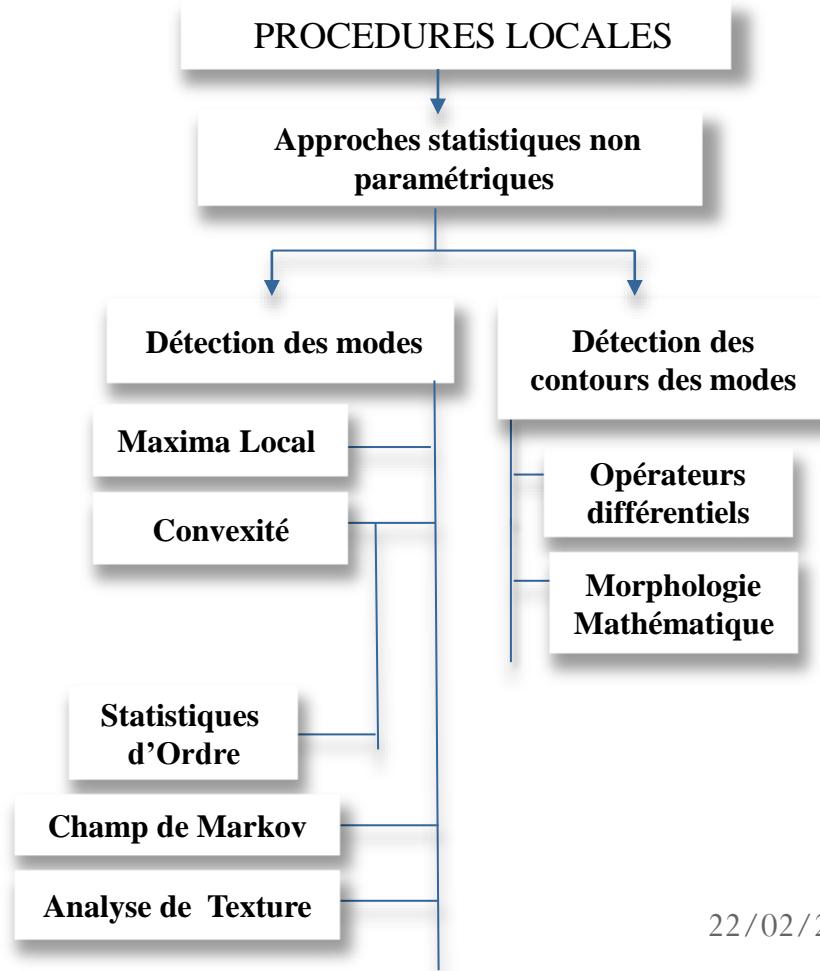
Data set

# Taxonomie des méthodes classiques de classification automatique des données multidimensionnelles

## Analyser globale



## Analyse ponctuelle



# Pourquoi le machine Learning?

**Problèmes des techniques classiques ( algorithmes de segmentation des images, algorithmes de classification automatique des données, les algorithmes de reconnaissance des formes, ...)**

- ⇒ Le traitement séquentiel des données ,
- ⇒ Performances dépendent de plusieurs facteurs (seuillage, paramètres difficiles à fixer).
- ⇒ Forme de data, classes Enchevêtrées, très complexes
- ⇒ De dimension très élevée

**Solution : Des techniques qui**

- Offrent aux machines plus de vitesse, Plus d'efficacité, Et plus d'automaticité

Le Passage du traitement séquentiel au traitement parallèle des données



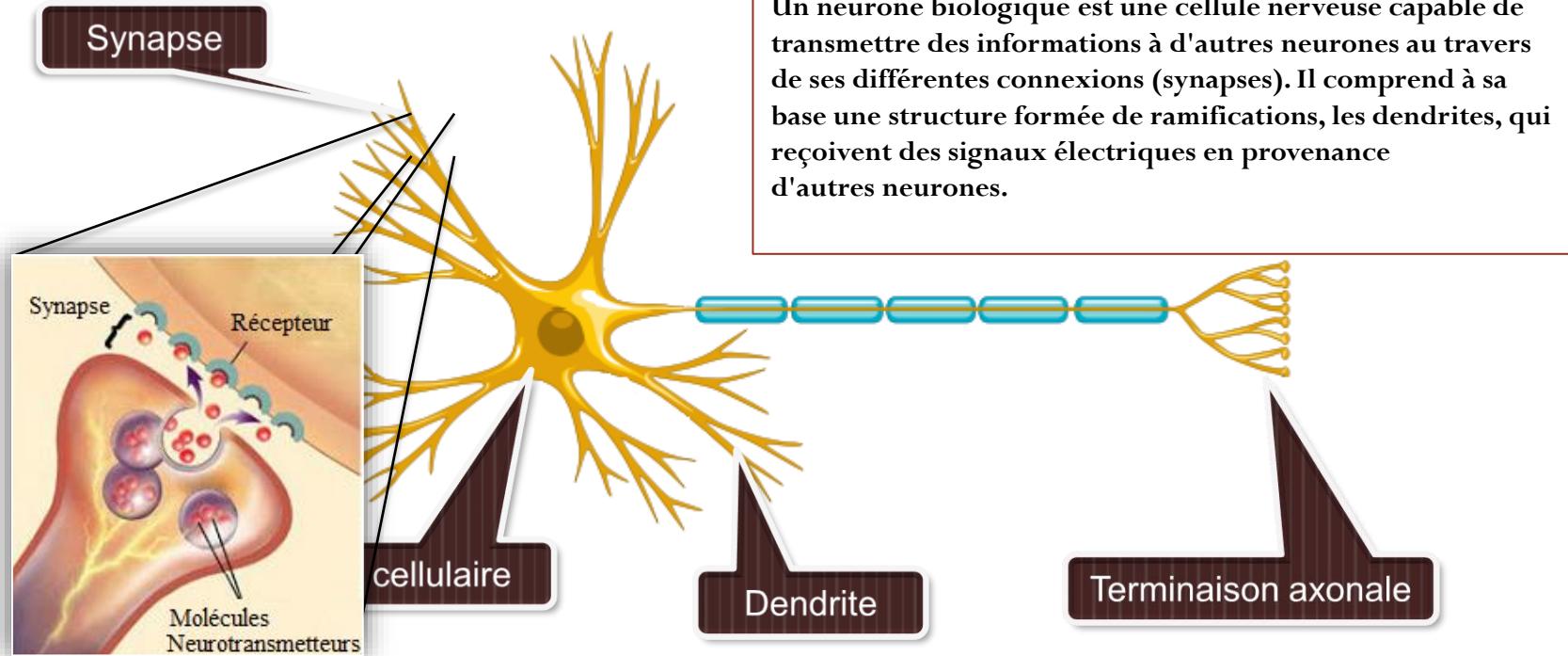
Réaliser **une modélisation** et une simulation de l'intelligence humaine

idée des réseaux neuronaux artificiels inspirés du fonctionnement du réseaux biologique

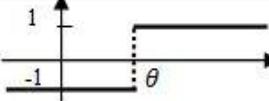
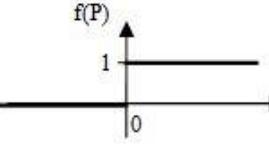
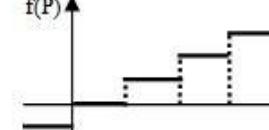
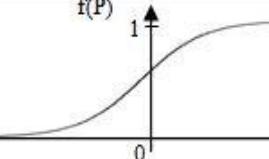
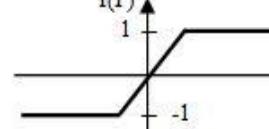
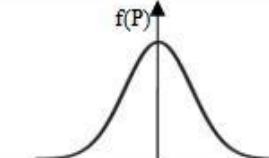
# Machine Learning

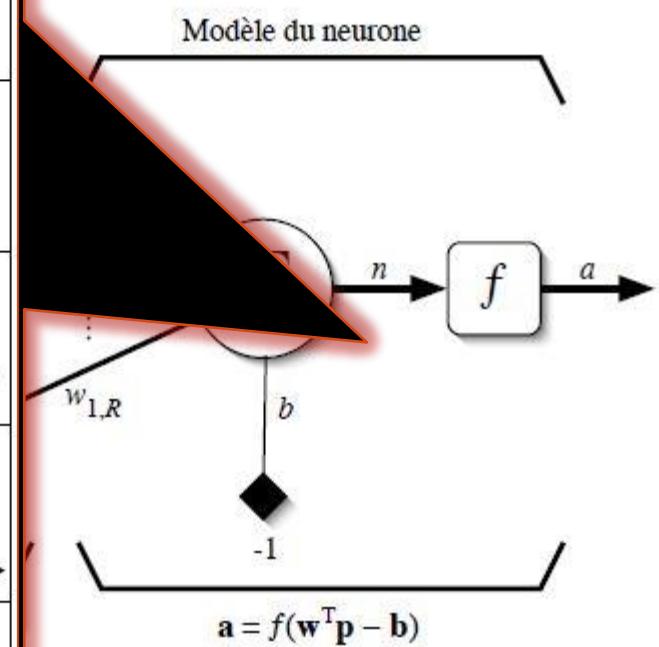
## Du neurone biologique au neurone artificiel

⇒ Le système nerveux du cerveau humain est constitué d'environ  $10^{11}$  neurones



## Neurone formel

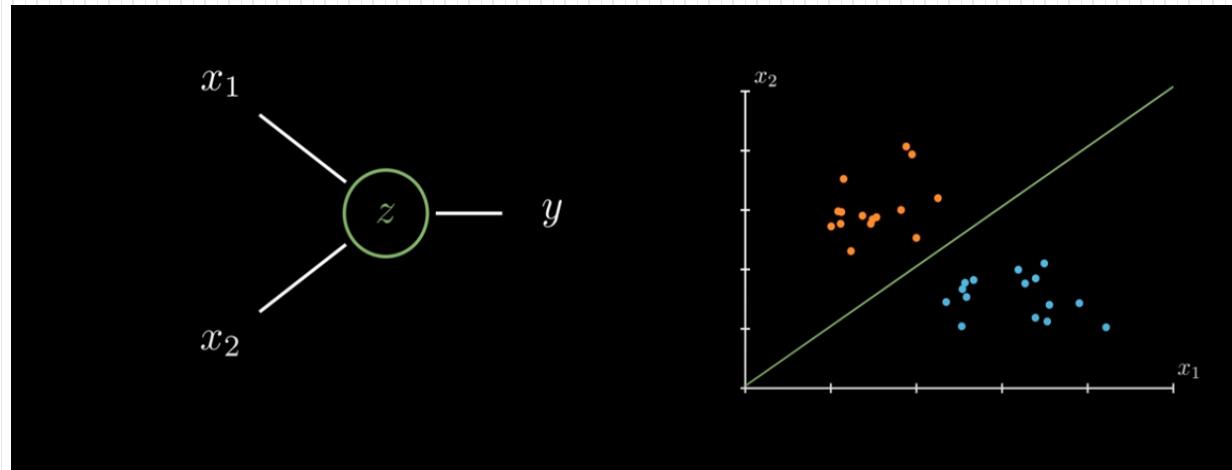
Etat de neurone	Fonction d'activation	Equation Mathématique	Présentation Graphique
Neurone binaire	Fonction Binaire à seuil	Si $P > \theta$ alors $f(P) = 1$ ; Sinon ; $f(P) = -1$ ;	
	Fonction Heaviside (tout ou rien)	Si $P > 0$ alors $f(P) = 1$ ; Sinon ; $f(P) = 0$ ;	
Neurone discret	Fonction plancher	$f(P) = [P]$	
Neurone continu	Fonctions sigmoïdes	$f(P) = \frac{1}{1 + e^{-\lambda P}}, ]0,1[$	
	Fonction Linéaire à seuils	$f(P) = \min\{1, \max(-1, P)\}, ]-1,1[$	
	Fonction Gaussienne Arbi	$f(P) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(P-\mu)^2}{2\sigma^2}}$	



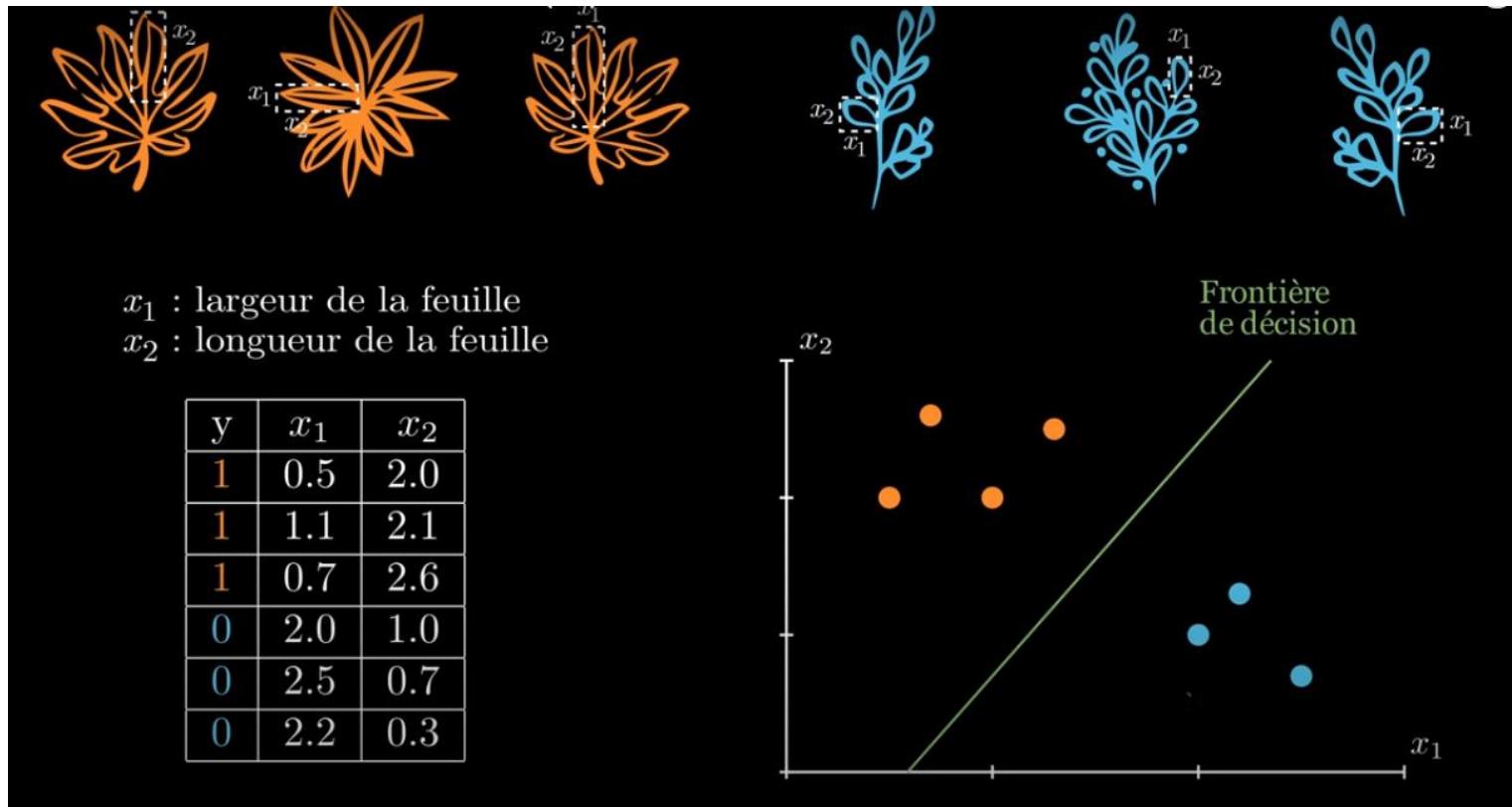
Modèle du neurone artificiel

# PERCEPTRON

Le **perceptron** est l'unité de base des réseaux de neurones. Il a été inventé en 1957 par [Frank Rosenblatt<sup>1</sup>](#) au laboratoire d'aéronautique de l'université Cornell. Il s'agit d'un neurone formel capable de séparer linéairement 2 classes de points

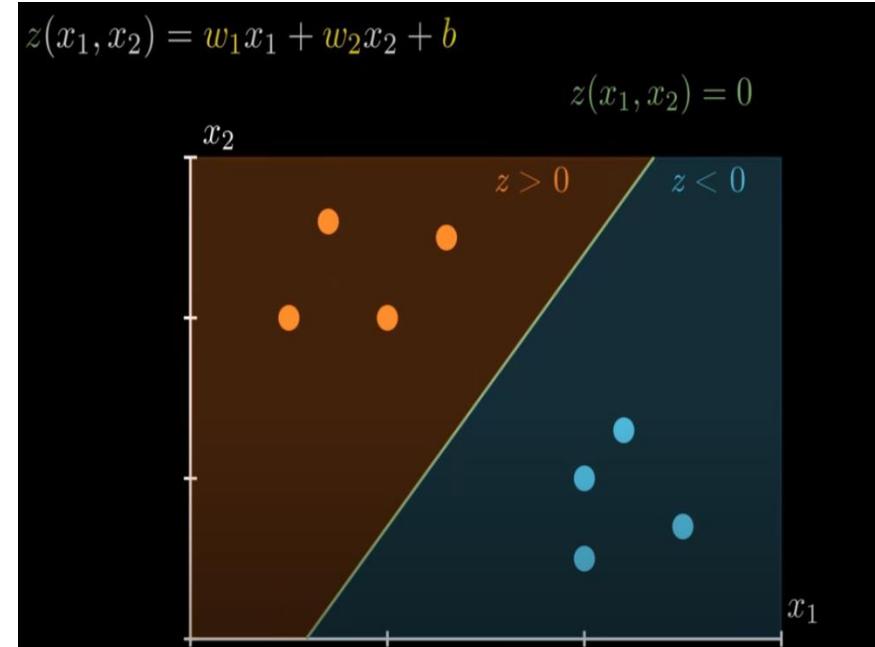
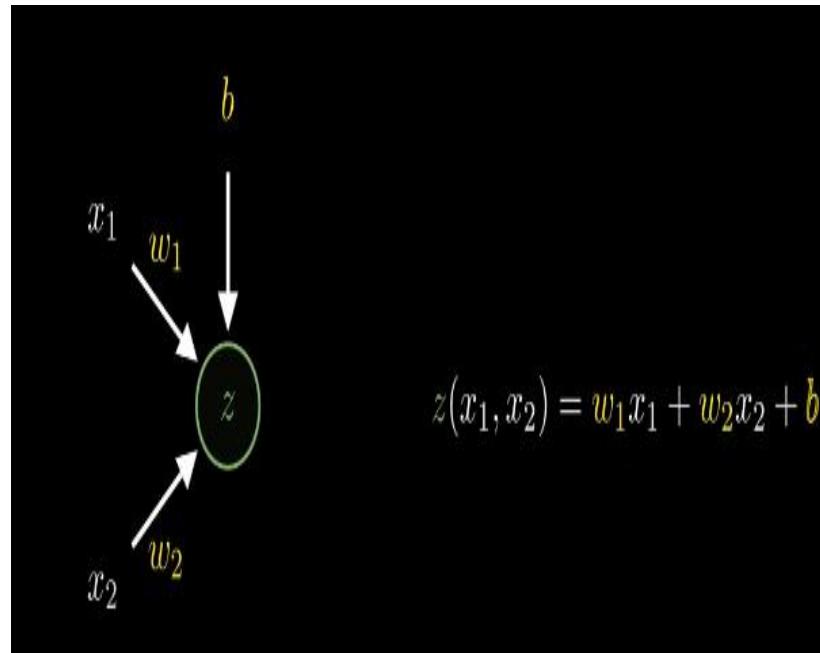


# Illustration



Comment trouver l'équation d'une droite?

# Modèle linéaire



# Fonction sigmoïde ou logistique

Fonction logistique

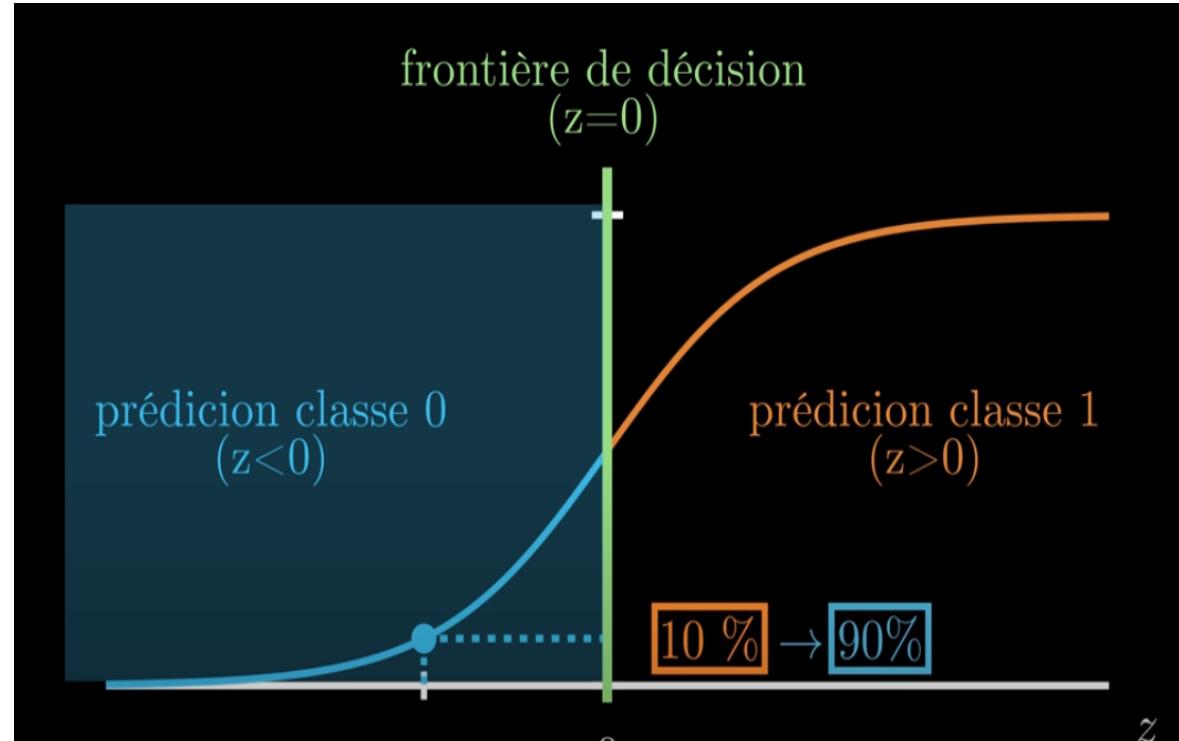
$$a(z) = \frac{1}{1+e^{-z}}$$

Ex

Si  $z=1.4$

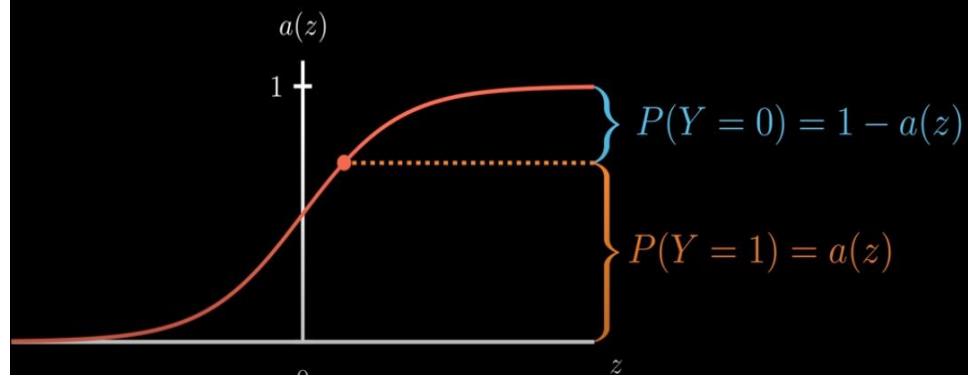
$$a = 0.8$$

$$\begin{cases} z > 0 & \text{classe 1} \\ z < 0 & \text{classe 0} \end{cases}$$



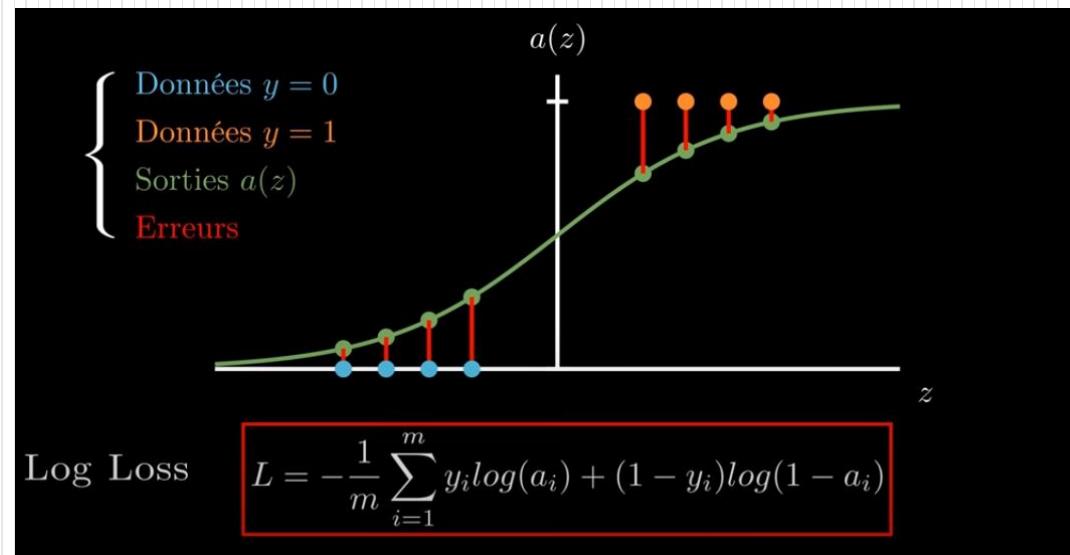
# Loi de Bernoulli

Loi de Bernoulli :  $P(Y = y) = a(z)^y \times (1 - a(z))^{1-y}$



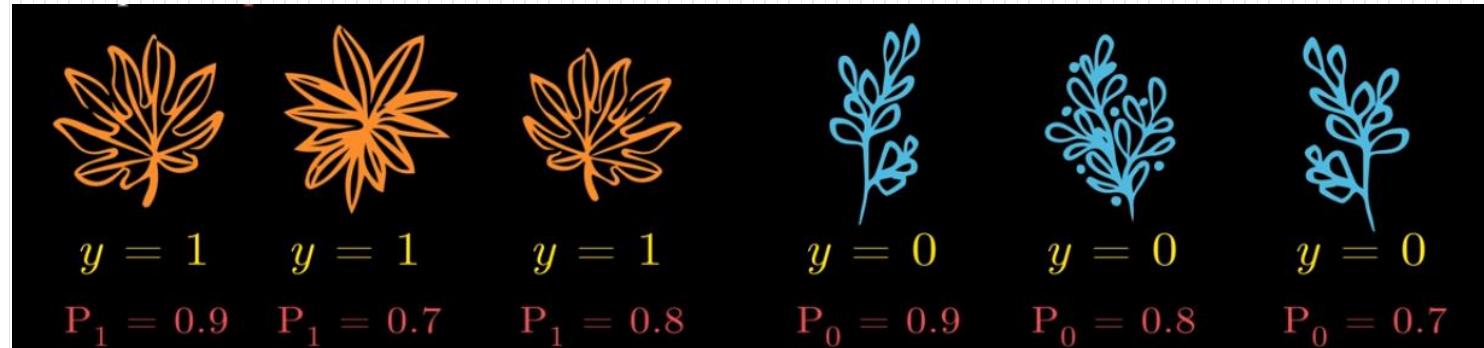
# Loss function

En machine learning, loss function est une fonction qui permet de quantifier les erreurs effectués par le modèle



# Quel est l'origine de la fonction loss function? La vraisemblance

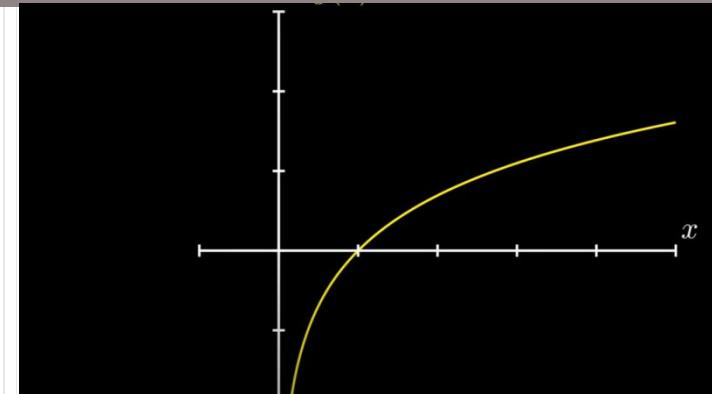
En statistique, la Vraissamblance indique la plausibilité du modèle vis-à-vis de vraies données



$$L = \prod_{i=1}^m a_i^{y_i} \times (1 - a_i)^{1-y_i}$$

# Quel est l'origine de cette fonction?

$$L = \prod_{i=1}^m a_i^{y_i} \times (1 - a_i)^{1-y_i}$$



Calculons  $\log(L)$

# Quel est l'origine de cette fonction? La vraisemblance

$$\text{Log(L)} = \sum_{i=1}^m y_i \log(a_i) + (1-y_i) \log(1 - a_i)$$

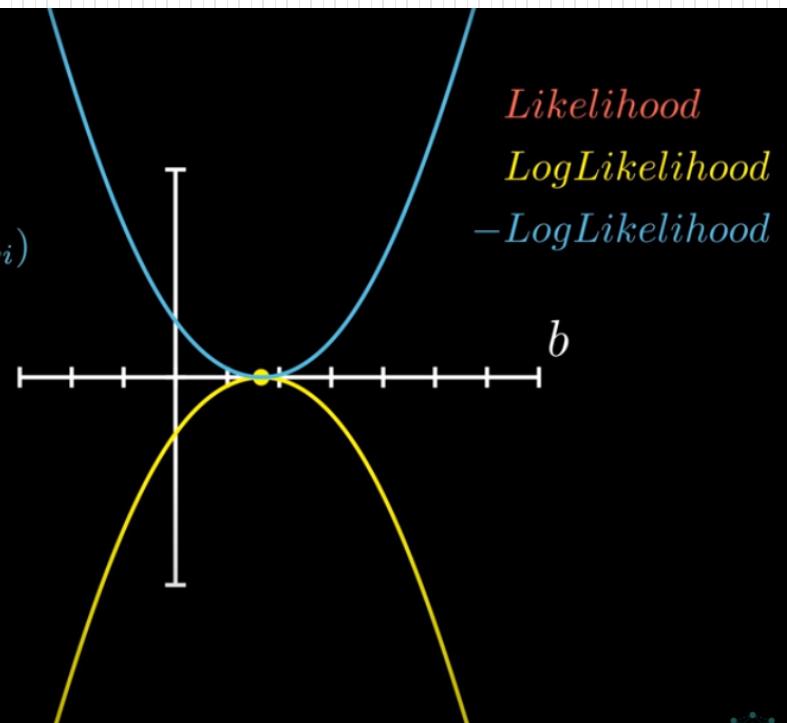
*Log Loss*

$$\mathcal{L} = -\frac{1}{m} \sum_{i=1}^m y_i \log(a_i) + (1 - y_i) \log(1 - a_i)$$

# Quel est l'origine de cette fonction? La vraisemblance

Maximiser  $f(x) =$  Minimiser  $-f(x)$

$$\mathcal{L} = -\frac{1}{m} \sum_{i=1}^m y_i \log(a_i) + (1 - y_i) \log(1 - a_i)$$



# Méthode de descente de gradient

**Objectif** Ajuster les paramètres du modèle  $W_1, W_2, b$  de façon à minimiser l'erreur entre la sortie prédite ( $a(z)$ ) et la sortie réelle  $c$  à  $d$  minimiser la fonction coût (log loss)

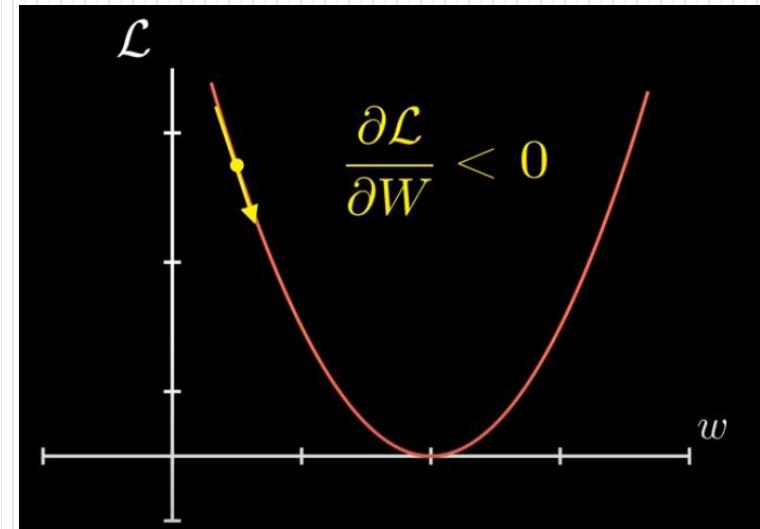
- Calculer le gradient ou la dérivée de la fonction coût par rapport aux différents paramètres

$$\text{Si } \frac{\partial \mathcal{L}}{\partial w} <$$

*0 la fonction cout est décroissante  
donc il faut augmenter W*

*Sinon*

*La fonction est croissante donc il  
faut diminuer W*



# Méthode de descente de gradient

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{w}}$$

Tel que  $\mathbf{W}(t+1)$  est le paramètre  $\mathbf{W}$  à l'instant  $t+1$

$\mathbf{W}(t)$  paramètre  $\mathbf{W}$  à l'instant  $t$

*$\alpha$  le pas d'apprentissage*

$\frac{\partial \mathcal{L}}{\partial \mathbf{w}}$  gradient à l'instant  $t$  **comment calculer ce gradient**

Nb la fonction coût doit être convexe car elle accepte un seul minimum

# Exercice Calculer les gradients de la fonction coût

$$z = wbX_1 + W_2X_2 + b$$

$$a(Z) = \frac{1}{1+e^{-z}}$$

Log Loss

$$\mathcal{L} = -\frac{1}{m} \sum_{i=1}^m y_i \log(a_i) + (1 - y_i) \log(1 - a_i)$$

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \alpha \frac{\partial \mathcal{L}}{\partial w}$$

La règle de chaîne

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w_1}$$

# Algorithme du gradient stochastique

➤ La théorie de l'approximation stochastique enseigne que l'équation d'itération est inutilement onéreux et qu'il vaut mieux, en termes de coût calcul, réaliser un algorithme de la forme.

# Réseaux de Neurones Artificiels

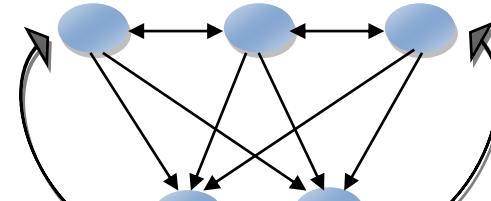
## Définition

Un réseau de neurones artificiel est un processseur massivement distribué en parallèle

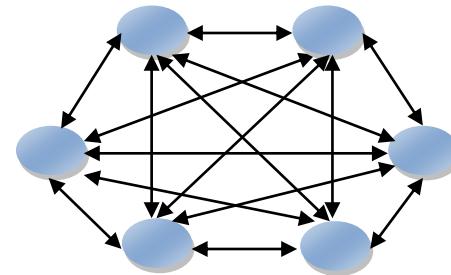
et la

age,

### Réseaux récurrents

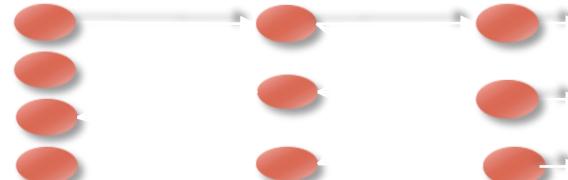


Réseau à connexions partielles

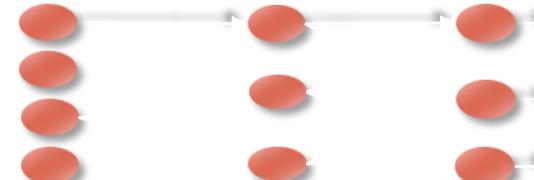


Réseau à connexions complètes

### Réseaux non bouclés



réseau multicouche



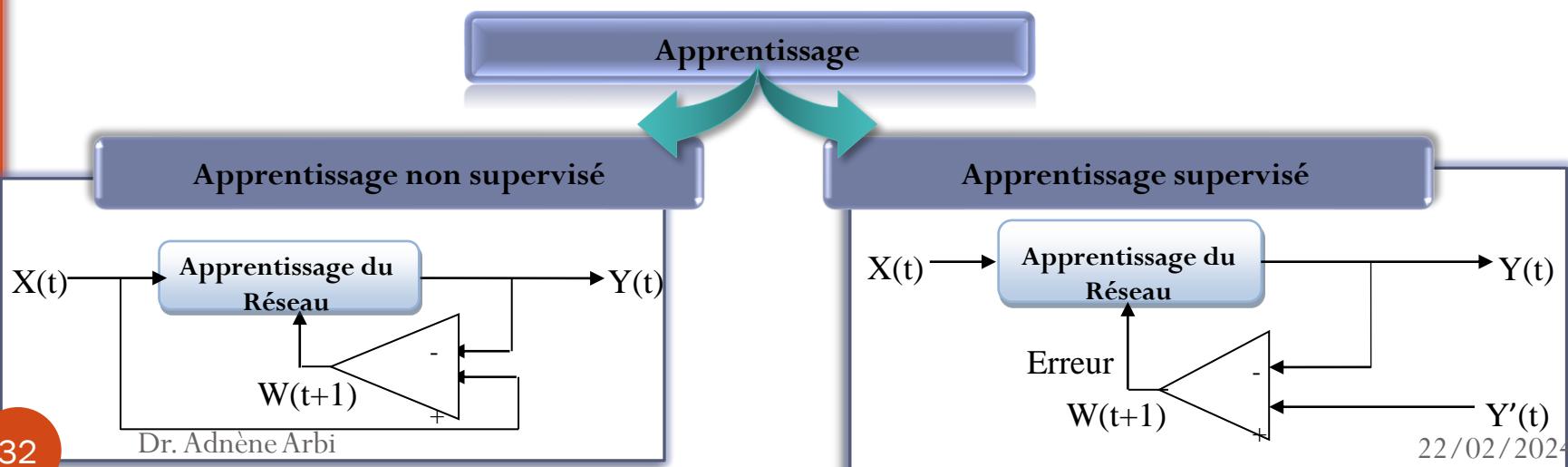
réseaux à connexions locales

# Réseaux de Neurones Artificiels

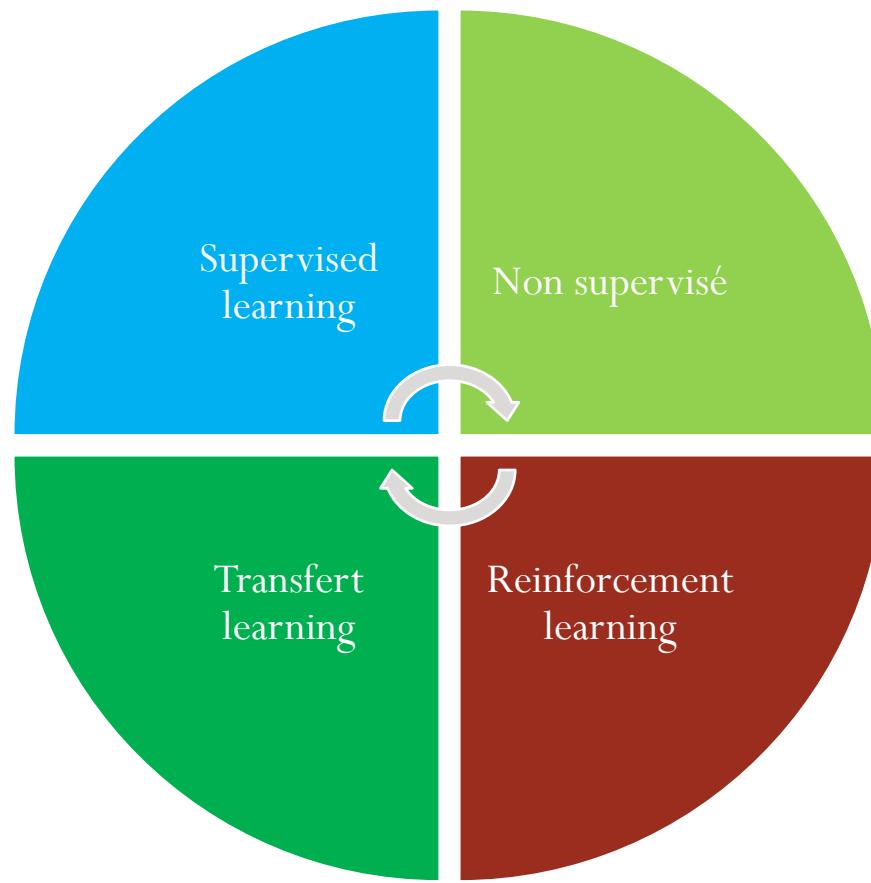
## Loi d'apprentissage

### Définition

L'apprentissage est un processus itératif de développement améliorant la performance d'un réseau de neurones, durant lequel les poids sont modifiés dans le but de permettre à la sortie d'être aussi proche que possible du comportement désiré.



# Machine Learning : Algorithmes d'apprentissages



## **Classification automatique des données multidimensionnelles dans un contexte supervisé par réseaux de neurones multicouches**

**Phase d'extraction** des caractéristiques des données (**feachers**):

**Phase d'apprentissage** permet d'ajuster les paramètres de réseau en minimisant l'erreur entre la sortie désirée (effectif) et la sortie calculée prédite ( méthode de descente du gradient)

**Phase de généralisation (test)** c'est la phase de classification

**Batch** : petit échantillon d'observations (tirées au hasard) et utilisées pour approcher le gradient lors d'une étape de l'algorithme de gradient stochastique.

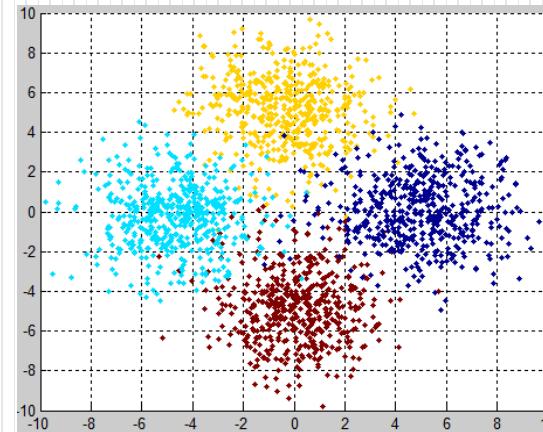
Typiquement on fait des batchs de taille  $b=128$  ou  $216$  ou  $512$  (etc).

En pratique, l'ensemble d'apprentissage est permué aléatoirement et le 1er batch est constitué des  $b$  premières observation, etc.

**Epoque (Epoch)** : cycle de descentes de gradient pendant lequel on fait autant d'itérations que nécessaire pour évaluer tous les batchs.

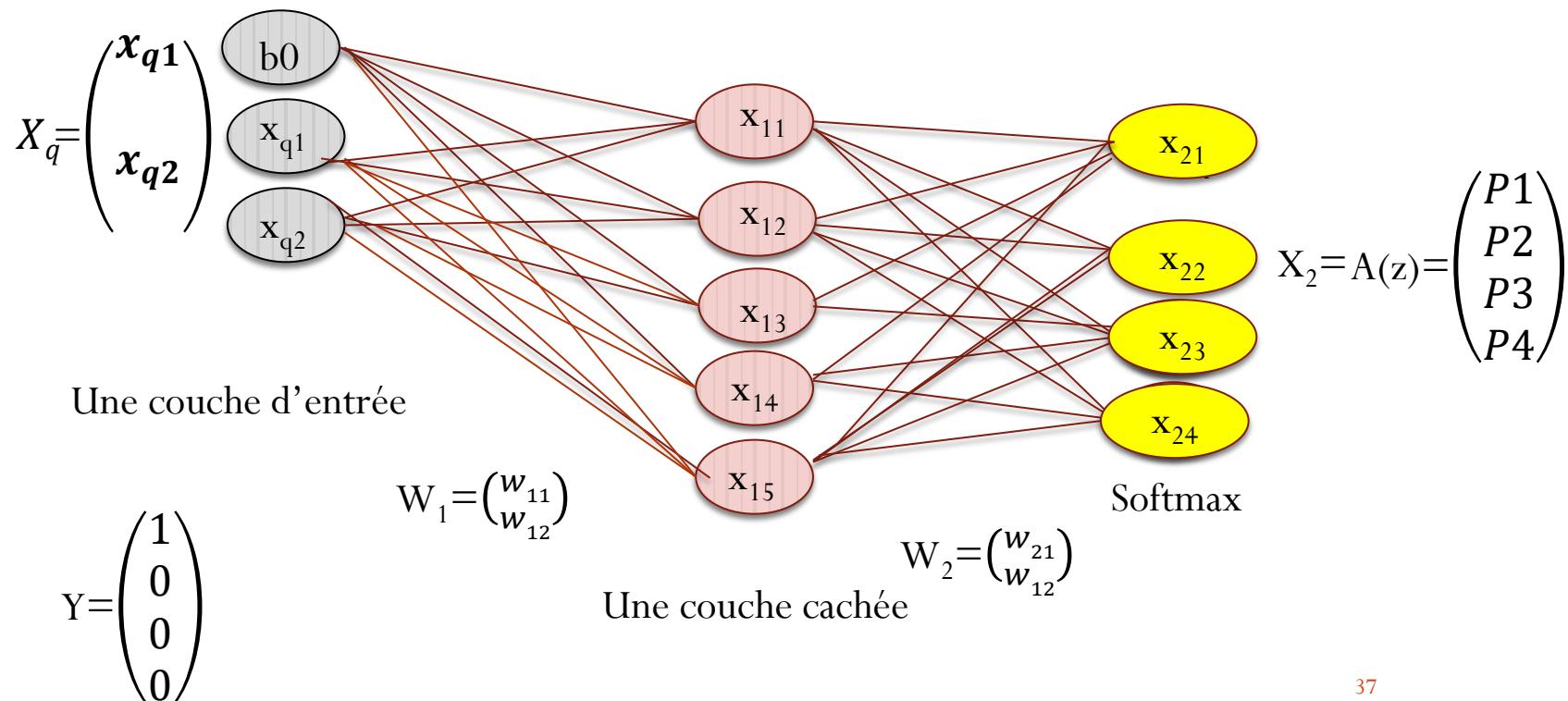
# Étude de cas: Classification automatique des données par réseaux de neurones multicouches

Soit un ensemble de  $Q$  observations est représenté par  $Q$  points dans un espace multidimensionnel  $\mathbf{R}^N$ , en associant à chacune de ces observations un vecteur  $\mathbf{X}_q$ , tel que  $\mathbf{X}_q = \{\mathbf{x}_{q,1}, \mathbf{x}_{q,2}, \dots, \mathbf{x}_{q,N}\}$ , avec  $q=1, \dots, Q$ .



# Classification automatique des données par réseaux de neurones multicouches

Soit un échantillon  $E = \{X_0, X_1, \dots, X_q, \dots X_m\}$ ,



# Phase d'apprentissage

$$z = wbX_1 + W_2X_2 + b$$

$$a(Z) = \frac{1}{1+e^{-z}}$$

*Log Loss*

$$\mathcal{L} = -\frac{1}{m} \sum_{i=1}^m y_i \log(a_i) + (1 - y_i) \log(1 - a_i)$$

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \alpha \frac{\partial \mathcal{L}}{\partial w}$$

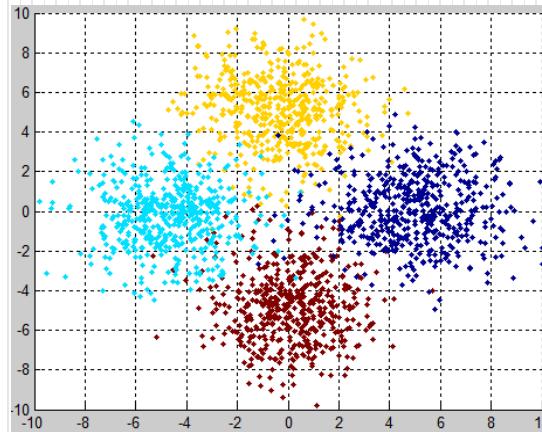
1. la propagation des calculs dans le réseau de neurones.

2. La méthode de rétropropagation:

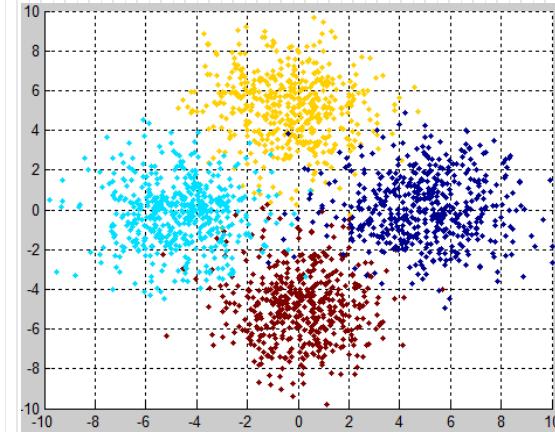
- Minimiser la fonction erreur (ex l'erreur quadratique moyenne) entre la sortie calculée à partir des paramètres des réseaux et la sortie désirée
- Propager l'erreur vers l'arrière jusqu'à la couche d'entrée.
- Ajuster les poids des connexions au niveau de chaque couche selon l'algorithme de descente du gradient

# Phase de généralisation (classification)

Principe tester le comportement du réseau face à d'autres objets



Data set



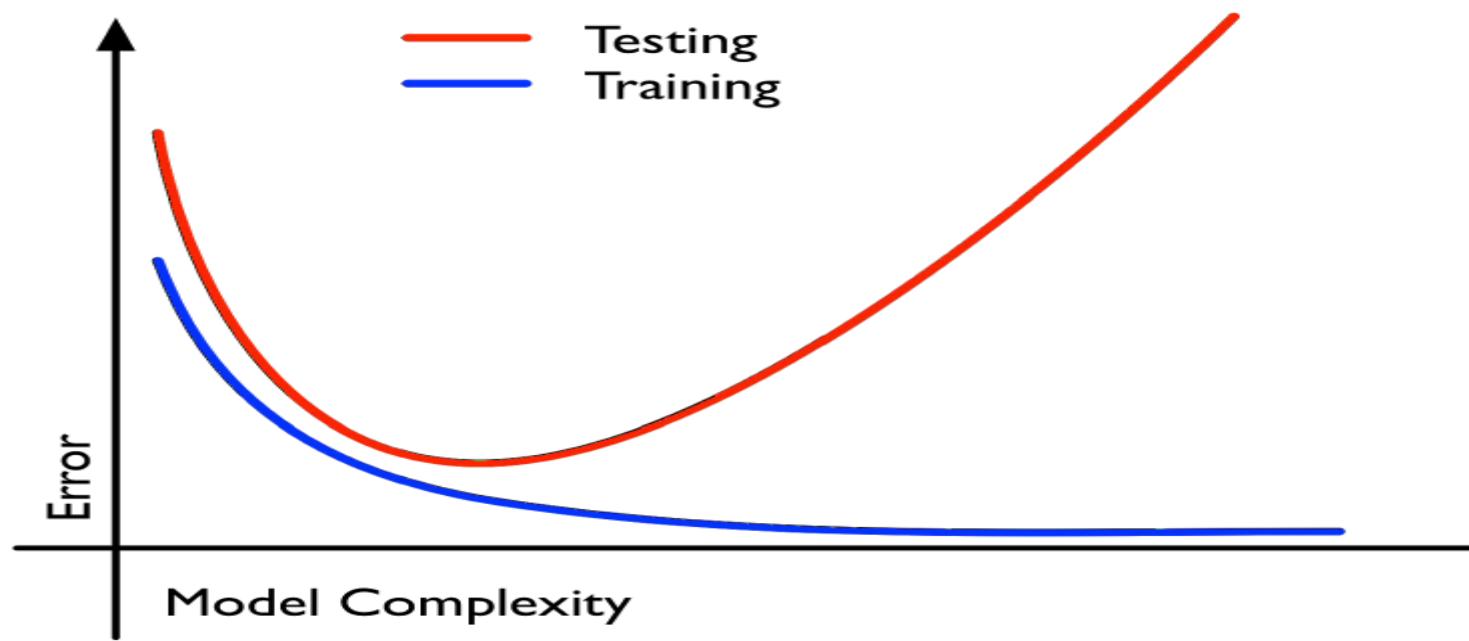
Data classified

# Évaluation du modèle

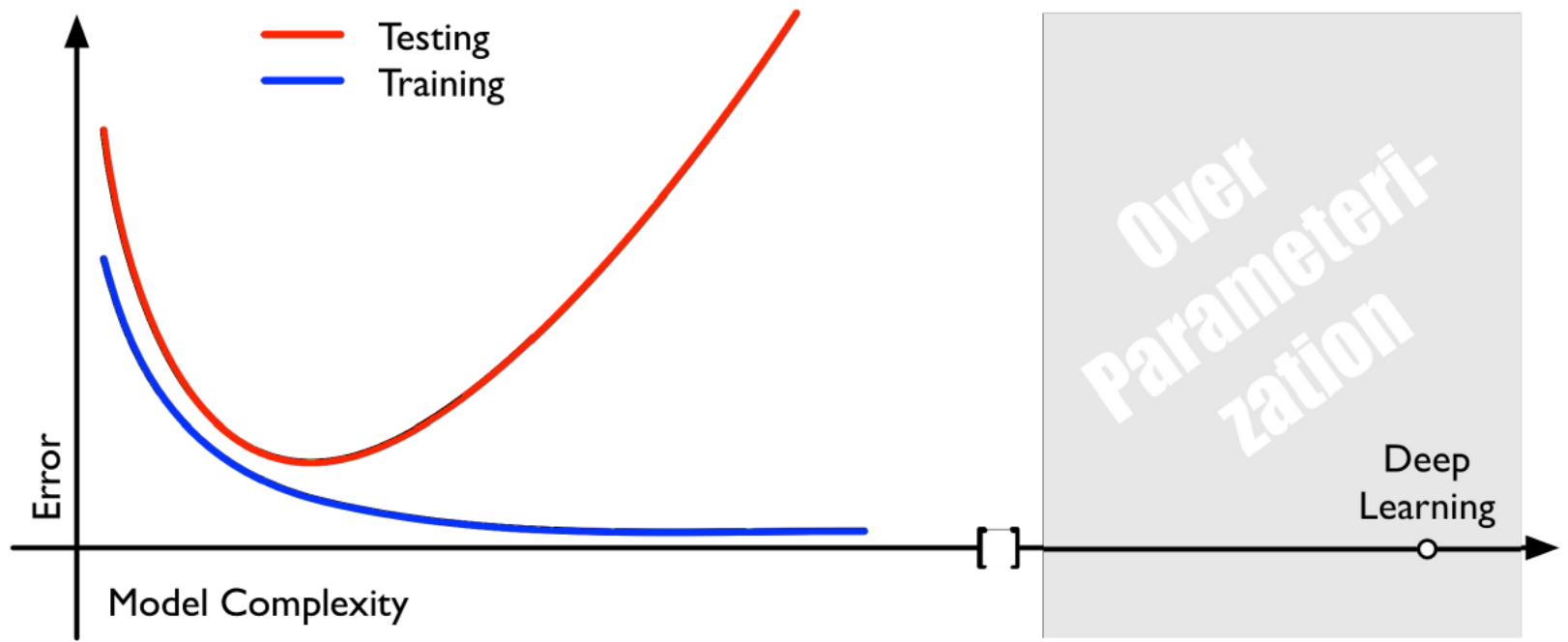
---

- **Taux d'erreur de classification (le pourcentage des observations mal classées),**
- **Matrice de confusion(La matrice de confusion est en quelque sorte un résumé des résultats de prédiction pour un problème particulier de classification)**
- **Accuracy ( the number of correctly predicted data points out of all the data points)**

# Detection de surapprentissage



# Cause



# Régularisation, pour éviter le sur-apprentissage

Dans les réseaux de neurones profonds (voire très profonds), le nombre de paramètres

est élevé et le risque de sur-apprentissage est important.

Différentes méthodes existent pour régulariser, ie diminuer la complexité du modèle lors

de la phase d'entraînement.

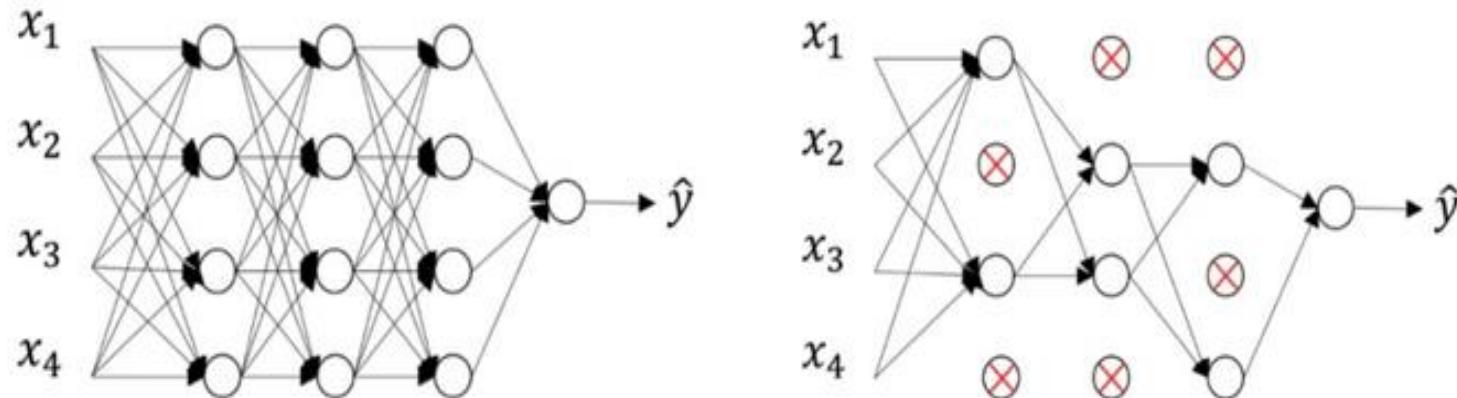
\* Pénalisation

Comme dans la régression Ridge ou Lasso, on peut ajouter à la fonction de perte une pénalité en norme L1 ou L2.

\* Dropout

A chaque étape de l'algorithme d'optimisation, on tire au hasard et, en probabilité P, des poids qui seront fixés 0 (le temps de cette étape).

# Dropout



Graph 3. Neural Network with dropout (right) and without (left). Source: Journal of Machine Learning Research 15 (2014)

# Classification des images par réseaux de neurones multicouches

**Une image** n'est autre qu'une représentation analogique d'un être ou d'une chose qu'on cherche souvent à traiter pour plusieurs raisons, entre autre:

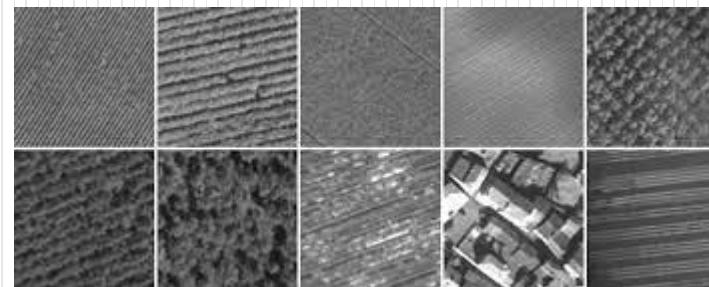
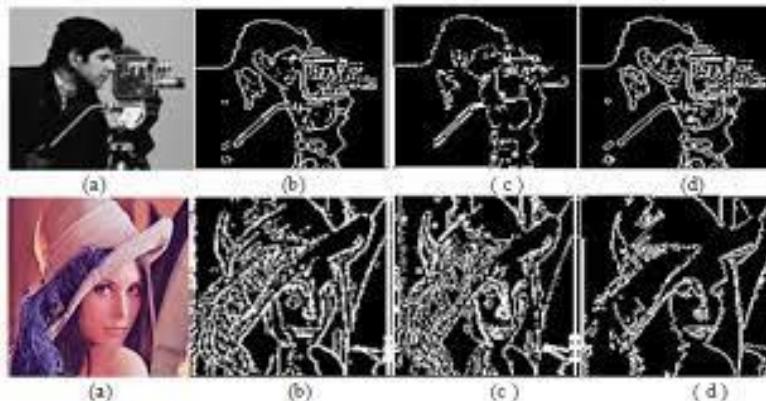
- **pour améliorer la qualité de l'image**
- **Pour détecter quelques formes, certains contours ou certaines textures de modèle connu**
- **Pour réduire les informations contenues dans une image**

Aussi, le traitement d'image concerne **l'analyse de l'image** qui:

Cherche à extraire des informations contenues dans les divers objets

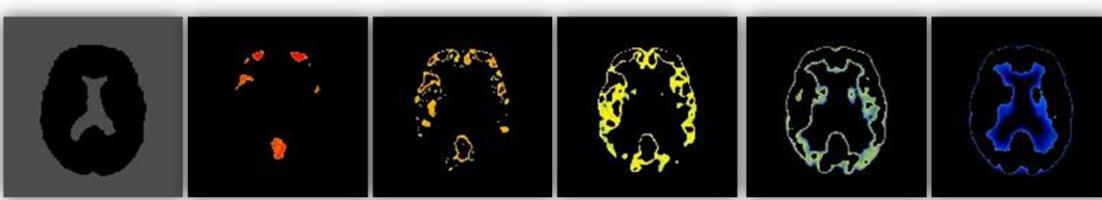
Les techniques de base utilisées sont essentiellement **l'extraction d'attributs et la segmentation de l'image en zone homogène**

# Traitement d'image



Détection de texture

# Segmentation d'images



Six régions séparées de l'image segmentée

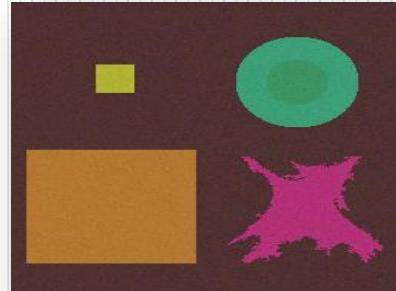


Image couleur

47

Dr. Adnène Arbi



six régions de l'image segmentée séparées

22/02/2024

# Représentation numérique d'une image et types d'images

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

Image binaire

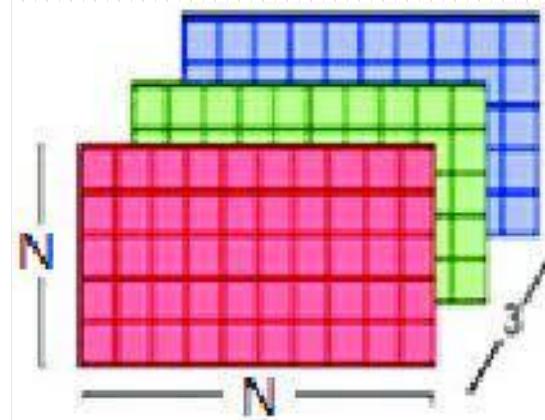
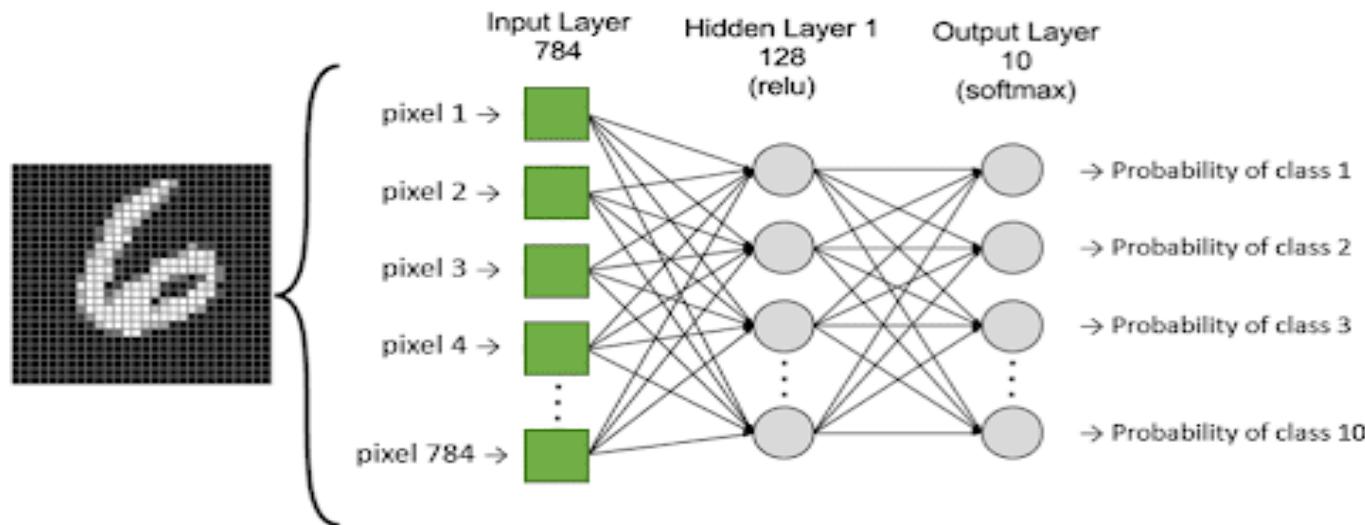


Image RGB

107	112	104	118	132	179	190	195
114	105	107	132	186	193	196	195
112	106	117	174	186	192	194	195
108	107	128	183	180	183	185	184
104	114	120	176	183	185	184	186
103	120	108	146	186	186	184	181
103	119	109	113	144	185	192	186
112	106	111	104	116	121	145	156

Image niveau de gris

# Application Segmentation d'images par DNN



# Limites des Réseaux de neurones

---

1. **Le résultat de la classification dépend de la qualité des Teachers présentés à l'entrée du réseau est un problème fondamental en vision par ordinateur**

# Limites des réseaux de neurones



64x64x3

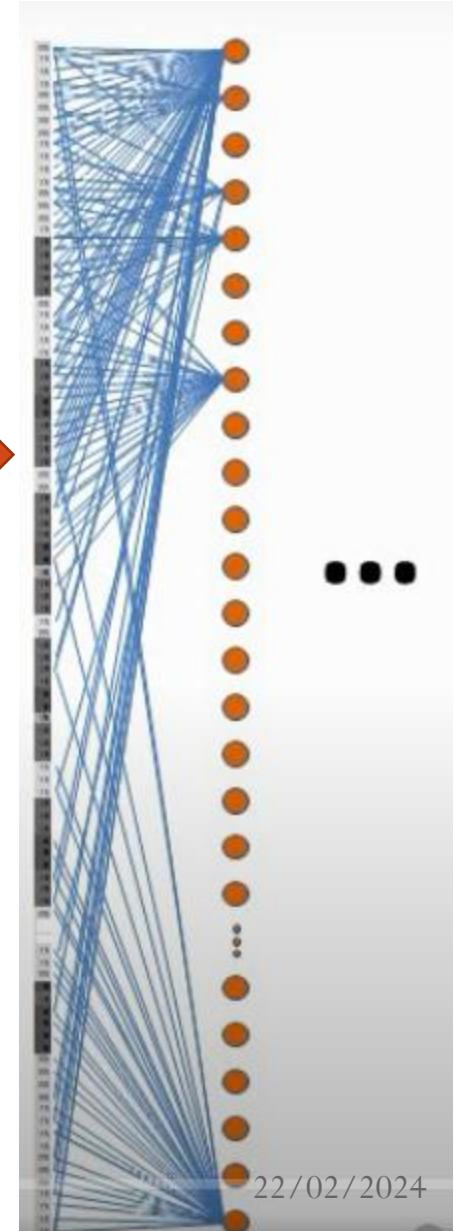


1000x1000x3

12288 neurones  
d'entrée



1000x1000x3=3 millions de neurones



- Fléau de la dimensionnalité ( overfitting )
- Perte de la sémantique de l'image lors de sa transformation en vecteur (



Pour la vision par notre système nerveux, est ce qu'on a besoin de capter l'information sur chaque pixel pour reconnaître une image? Ou bien notre œil se focalise surtout sur les caractéristiques les plus pertinents pour identifier et reconnaître l'image ?

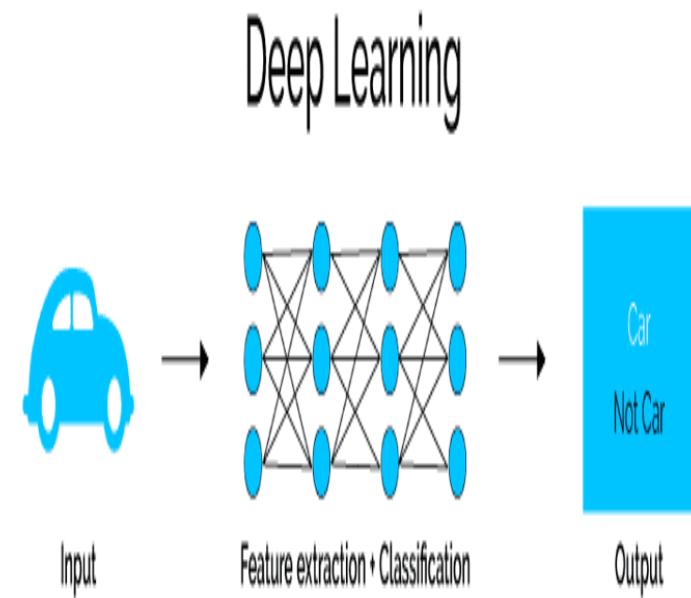
Et c'est l'idée du DEEPLARNING et plus particulièrement du réseaux de neurones convolutifs basé essentiellement sur un filtre (images de caractéristiques)

# Deeplearning les Réseaux de Neurone Convolutif

En 2012, une révolution se produit en *Deep Learning*. Il s'agit d'un réseau de neurones convolutif appelé AlexNet.

Un RN qui consiste à appliquer des filtres (noyau ou kernel) sur l'image à analyser et qui permet à la fois:

- D'extraire les feachers (contours, les arrêtes verticales, ...)
- De classer l'image



\* Les réseaux de neurones sont des régresseurs ou classificateurs qui ont une structure particulière.

\* Leur structure permet, à la limite, d'approcher toutes sortes de fonctions.

\* Cette structure particulière permet aussi d'utiliser des techniques efficaces numériquement pour calculer les gradients.

\* Les réseaux de neurones profonds sont des réseaux de neurones dans lesquels on a combiné un très grand nombre de perceptrons (en série et en parallèle).

- Ils sont utilisés dans un grand nombres d'applications.
- Le nombre de paramètres des ces réseaux est en général très grand devant le nombre d'observations ; pourtant les performances de ces modèles sont souvent excellentes.
- On sort alors du cadre classique de l'estimation statistique et des résultats (très) récents montrent que ces réseaux profonds appartiennent à un monde qui va au-delà de l'interpolation.  
Affaire à suivre...

# Concepts de base: La convolution

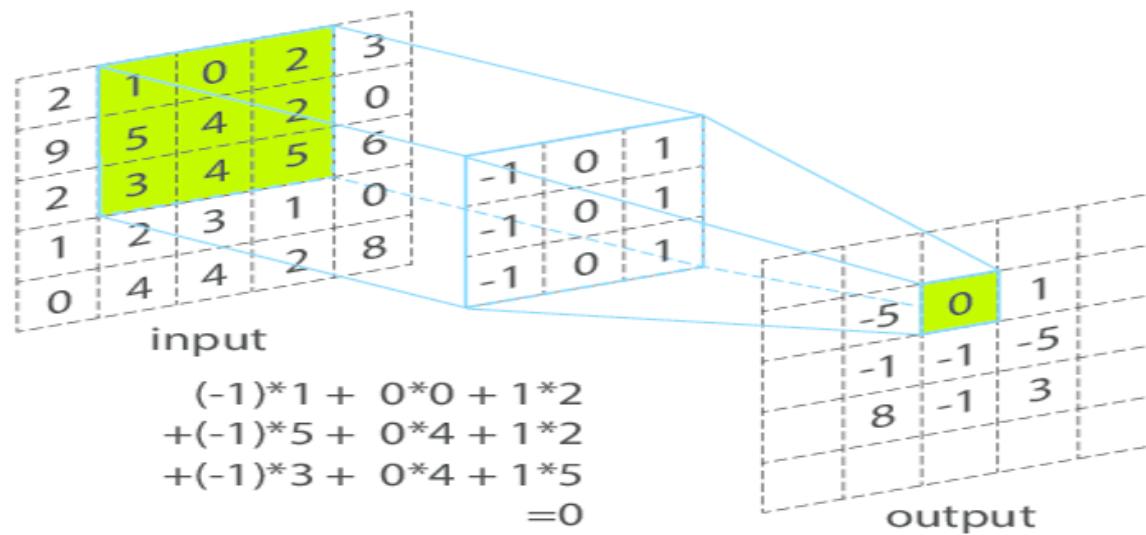
Le filtrage d'une image numérique permet de modifier son spectre spatial:

- chercher à atténuer les hautes fréquences pour la rendre moins nette,
- Chercher à réduire le bruit,
- Chercher à accentuer les hautes fréquences pour accentuer la netteté.

# Concepts de base: La convolution

## Exemple de détection d'un contour vertical

La convolution, ou produit de convolution, est une généralisation du filtre moyenneur où l'on considère cette fois une moyenne pondérée. La fenêtre glissante est alors elle même une image qui contient les coefficients de pondération. On l'appelle généralement noyau de convolution ou masque de convolution (kernel ou mask en anglais) :



# Concepts de base: La convolution

## Exemple de détection d'un contour vertical

Formellement, le produit de convolution est une opération entre deux images en niveau de gris  $f$  et  $g$ , notée  $f*g$  défini par:

$$\forall (x, y) \in \mathbb{Z}^2, (f * g)(x, y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i, j)g(x - i, y - j)$$

- $g(x-i, y-j)$  :comporte une opération de translation: il s'agit de déplacer l'image  $g$  à la position  $(x, y)$  ( $g$  joue donc le rôle de fenêtre glissante).,
- La multiplication point à point de  $f$  par la translatée de  $g$  : c'est l'opération de pondération,
- la sommation du tout : c'est l'opération de moyennage.

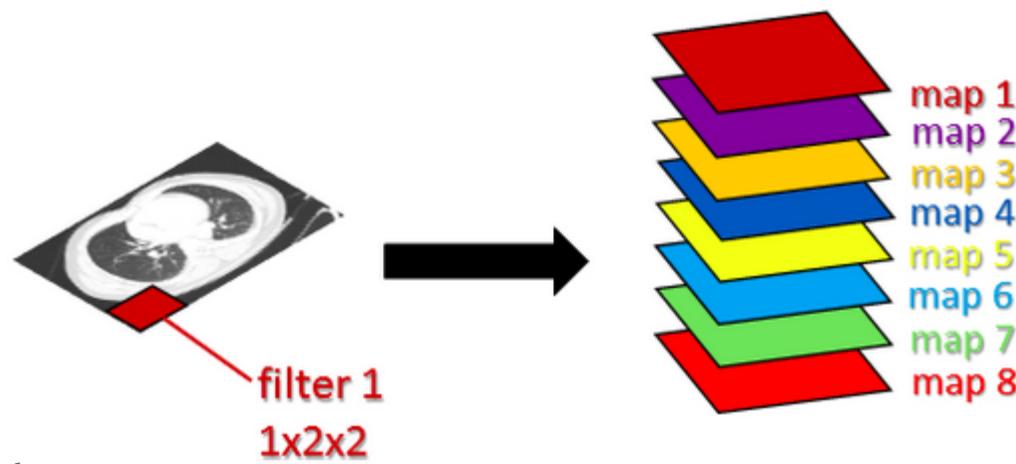
# Choix du noyau de convolution

Filtre moyenneur (lissage)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Filtre gaussien $3 \times 3$ (lissage)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Filtre gaussien $5 \times 5$ (lissage)	$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Filtre réhausseur (renforce les contours)	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Filtre Laplacien (détecteur de contours)	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	

# CNN avec 8 filtres

En pratique, à chaque couche du CNN, on applique plusieurs filtres de même taille en parallèle. La poids de chacun des filtres sont différents. On explore ainsi différentes caractéristiques de l'image.

first CNN layer with 8 filters



# Les hyperparamètres Padding, Striede

## Padding

Soit,  $n \times n$  la taille de l'image,  $f \times f$  la taille du filtre et  $p$  le padding

Avec la convolution, l'image devient plus petite et les pixels des bords et des coins sont beaucoup moins utilisés

→ la solution tamponner l'image (remplir par des zero)

Deux types de padding:

- Valid convolution no padding (  $p=0$ )
- Same convolution the input and the output are the same size

La taille de l'image de sortie est calculée à partir de la formule suivante :

$$n + 2p - f + 1 \times n + 2p - f + 1$$

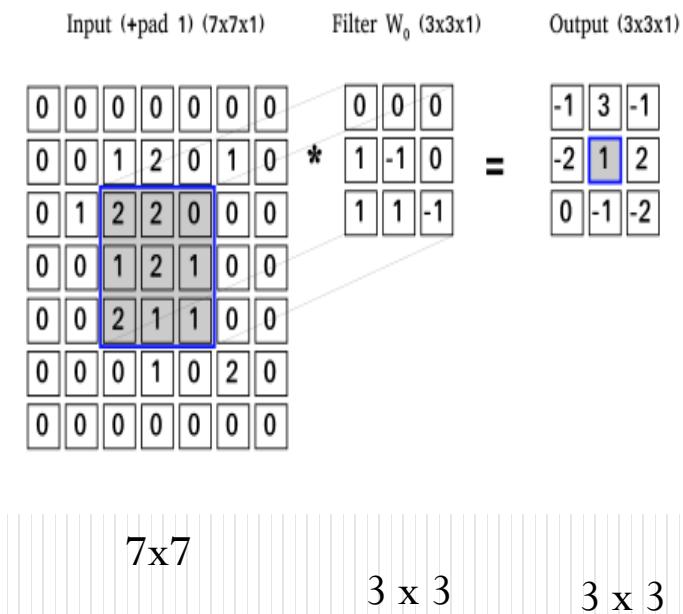
# Padding Exemples

- $P=1$
- Pour same convolution  $n + 2p - f + 1 = n$

donc

$$p = \frac{f - 1}{2}$$

Utiliser des filtres de nombre impaire



# Les hyperparamètres : Padding, Striede

## Striede convolution

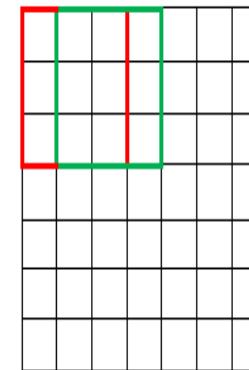
Striede dénoté  $S$  est le saut (le pas) de convolution

La taille de l'image de sortie est :

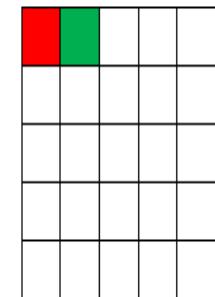
$$\frac{n + 2P - f}{S} + 1 \times \frac{n + 2P - f}{S} + 1$$

$n=8$   $p=2$   $s=2$  quelle sera la taille de sortie?

7 x 7 Input Volume



5 x 5 Output Volume



# Le Réseau de Neurones Convolutif ( CNN)

- Les CNN (s) sont des réseaux de neurones multicouches très utilisés dans le traitement de l'image ou de la vidéo par exemple la reconnaissance faciale , la classification d'image.
- Deux avantages importants inhérents aux réseaux convolutifs :  
**le réseau peut apprendre par étape** à reconnaître les éléments caractéristiques d'une image. Pour reconnaître un visage par exemple: il apprendra à reconnaître d'abord des paupières, des pupilles, pour arriver à identifier des yeux; une fois un élément appris à un endroit de l'image **le réseau sera capable de le reconnaître n'importe où d'autre dans l'image**

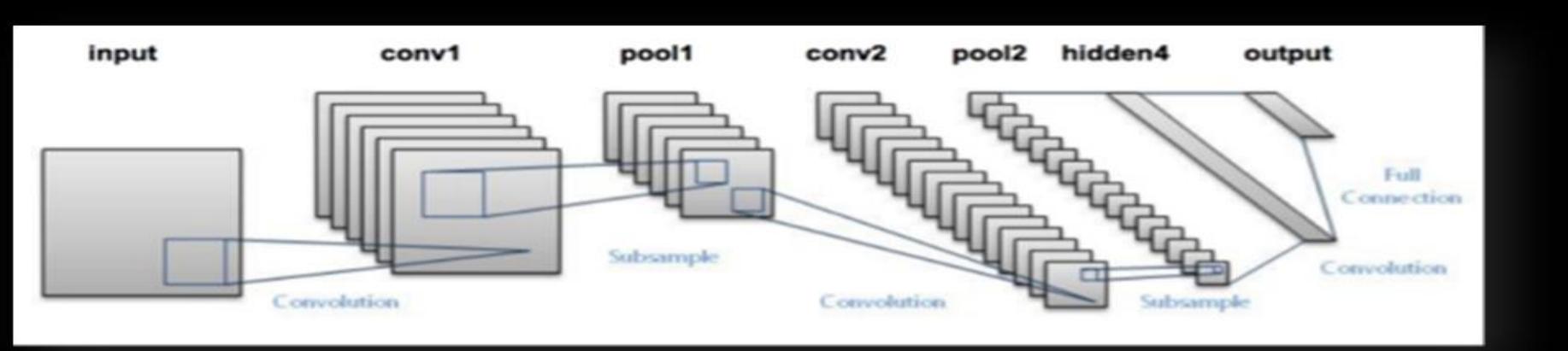
# Architecture du réseau de neurone Convolutif

Il existe plusieurs architectures dans le domaine des réseaux convolutifs  
**LeNet, AlexNet, ZF Net, GoogLeNet, ResNet, VGGN, VGG16, VGG19...**

**Ce sont des exemples d'architecture utilisées pour augmenter la résolution d'images.**

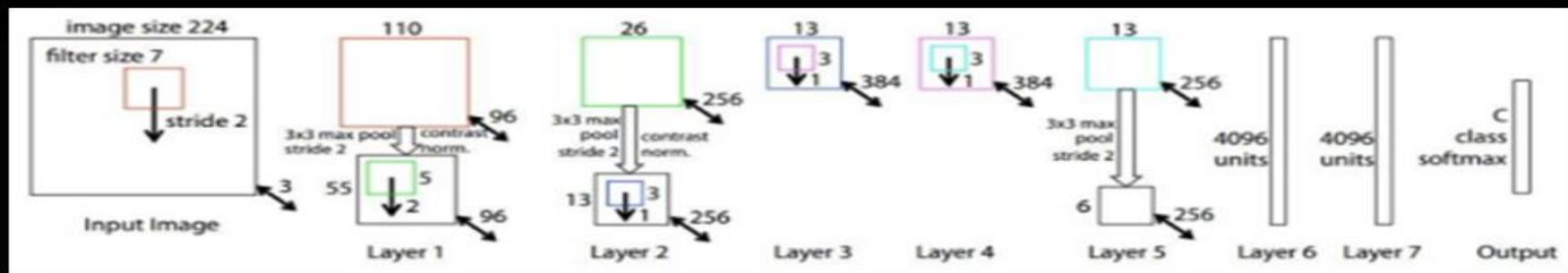
# Architecture du réseau de neurone Convolutif

LeNet. développé par Yann LeCun dans les années 1990.  
l'architecture LeNet utilisée pour lire les codes postaux, les chiffres, etc.



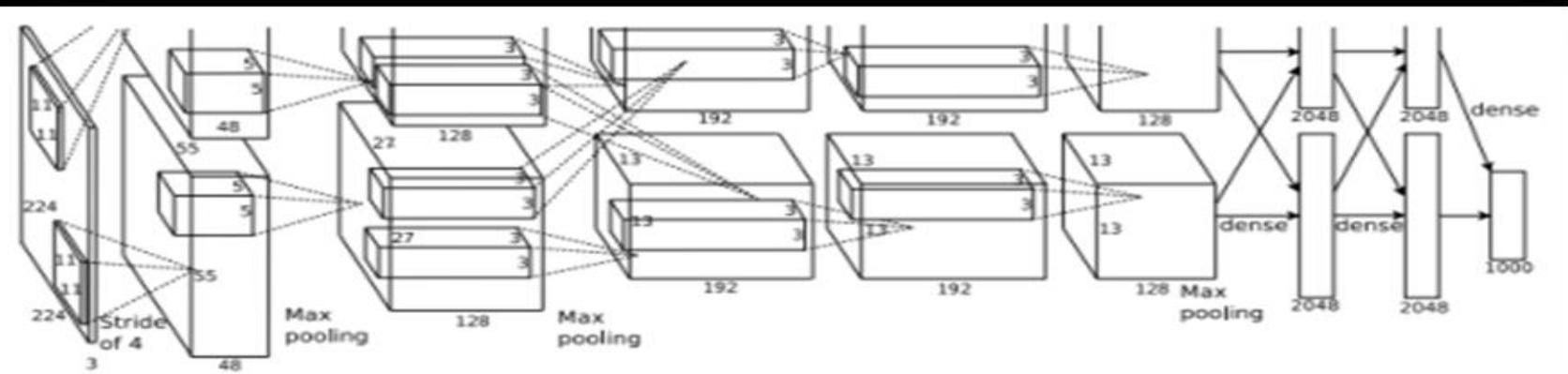
# Architecture du réseau de neurone Convolutif

**ZF Net.** un réseau convolutionnel de Matthew Zeiler et Rob Fergus. C'était une amélioration de AlexNet en modifiant les hyperparamètres de l'architecture, en particulier en augmentant la taille des couches de convolution moyennes et en réduisant la taille des pas et des filtres sur la premier couche.

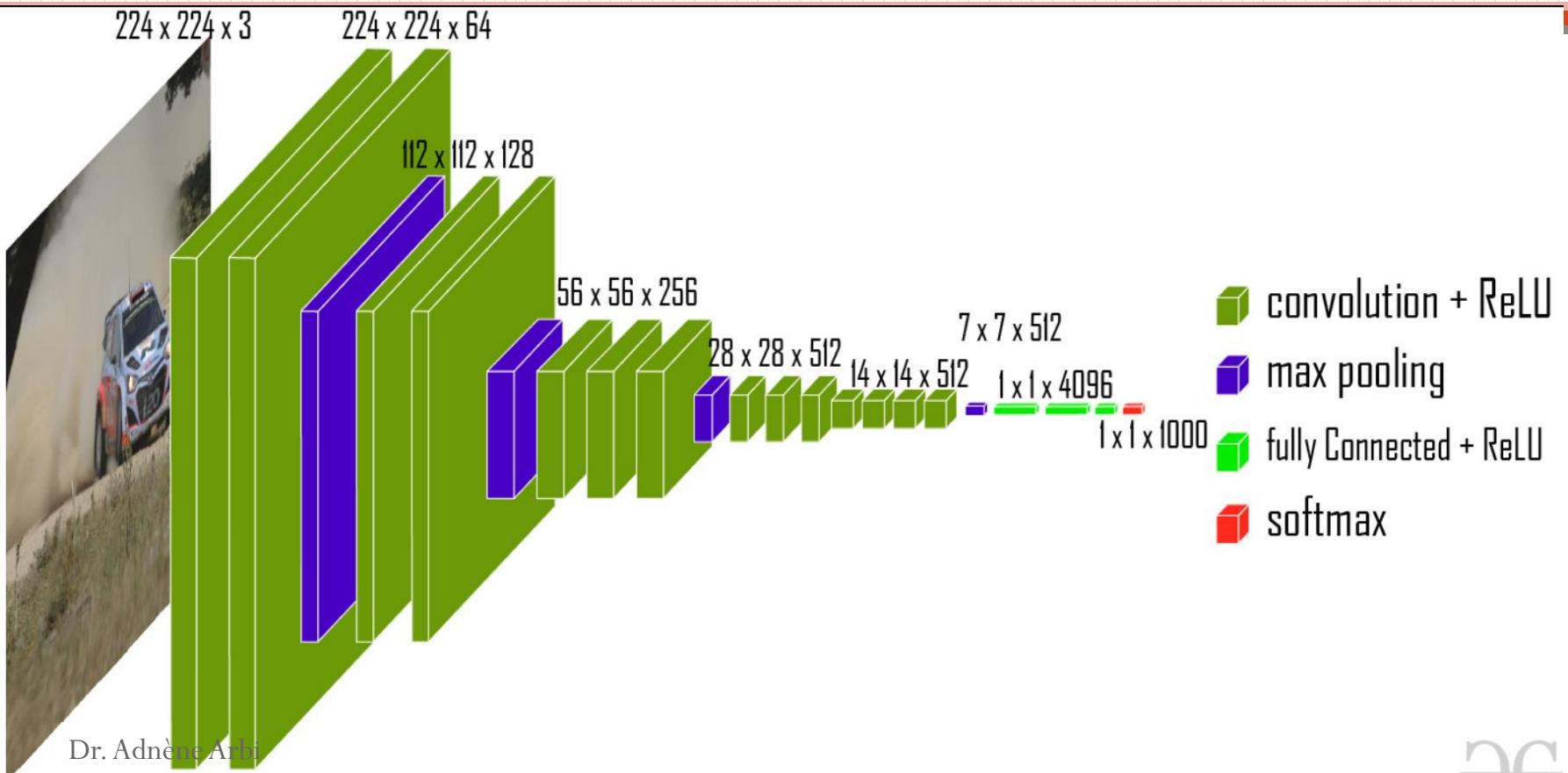


# Architecture du réseau de neurone Convolutif - AlexNet

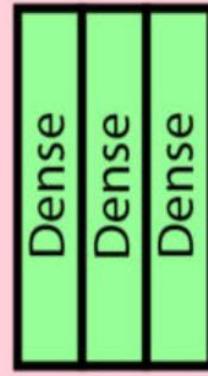
AlexNet développé par Alex Krizhevsky, Ilya Sutskever et Geoff Hinton. en 2012 Le réseau avait une architecture très similaire à celle de LeNet, mais il était plus profond, plus grand et comportait des couches convolutives empilées les unes sur les autres



# VGG16



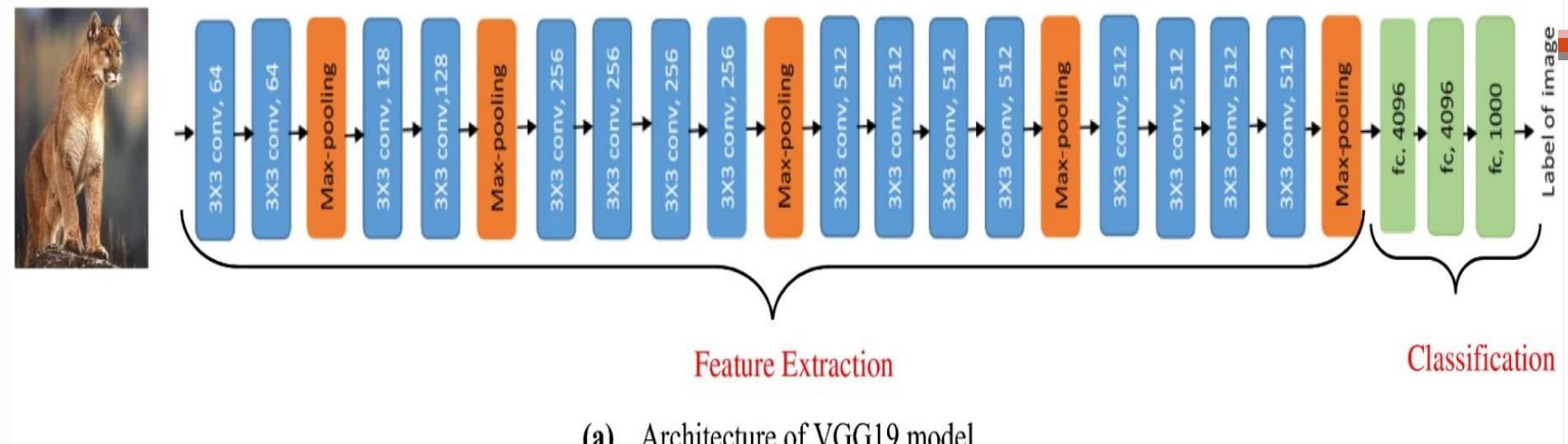
INPUT →



## VGG - 16

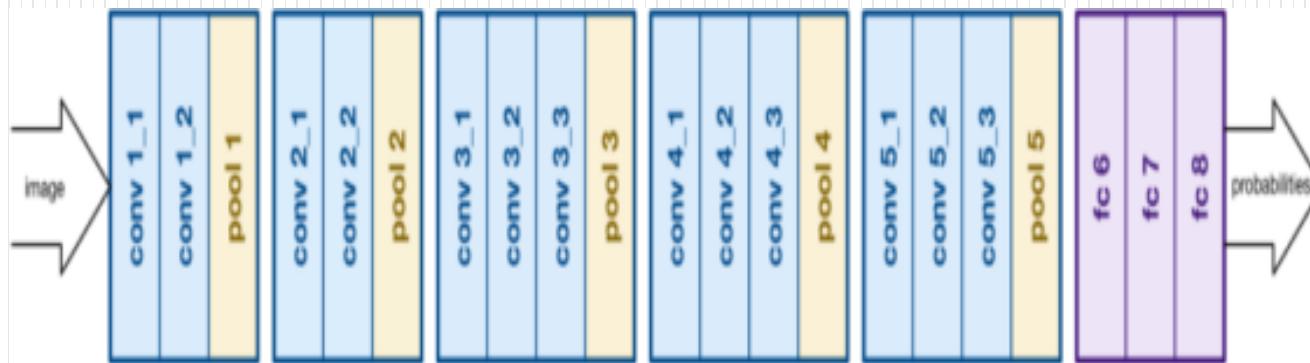
OUTPUT →

From: [Transfer learning for image classification using VGG19: Caltech-101 image data set](#)



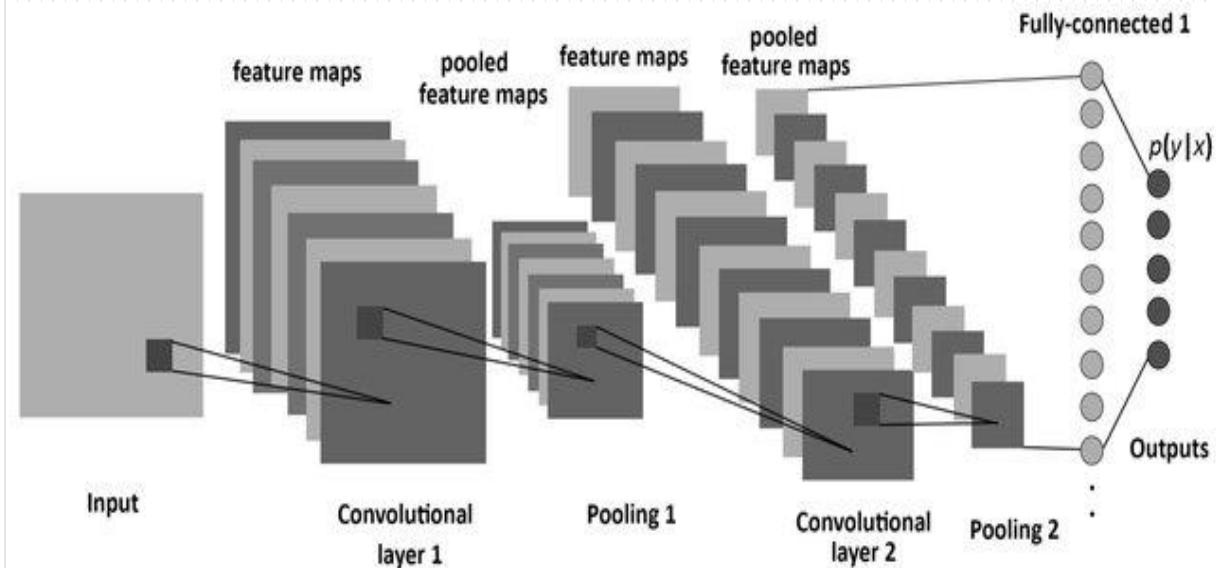
# Architecture du réseau CNN

**VGGNet.** Développé par Karen Simonyan et Andrew Zisserman, Sa principale contribution a été de montrer que la profondeur du réseau est un composant essentiel pour de bonnes performances. Leur meilleur réseau final contient 16 couches CONV / FC et, de manière attrayante, présente une architecture extrêmement homogène qui ne réalise que  $3 \times 3$  convolutions et un regroupement  $2 \times 2$  du début à la fin



# Architecture du réseau CNN

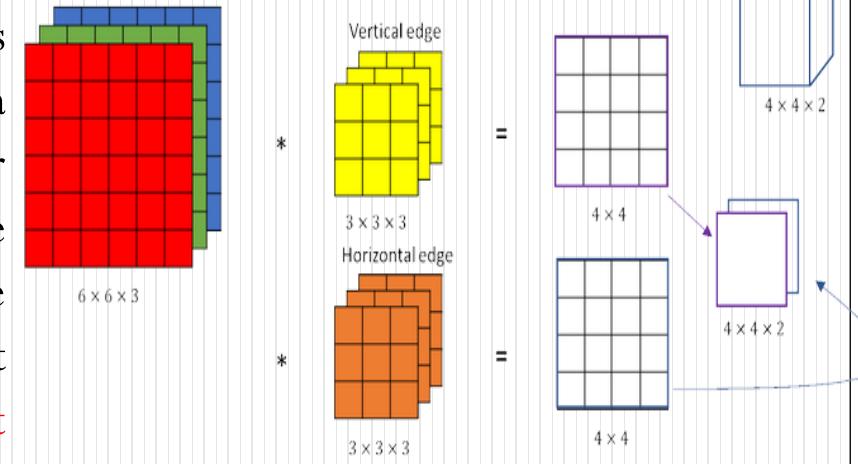
- **Convolutional layers** ou couches de convolution (CONV)
- **Pooling layers** ou couches de Pooling (POOL)
- **ReLU layers** ou couches d'activation ReLU (Rectified Linear Units)
- **Fully Connected layers** ou couches Fully Connected (FC)



# Architecture du réseau CNN

## La couche de convolution(

**CONV)** La couche de convolution est la composante clée des réseaux de neurones convolutifs. Son but est de **repérer** la présence d'un ensemble de **features** sur l'image reçue en entrée. Pour cela, on réalise **un filtrage par convolution**: le principe est de faire "glisser" une fenêtre représentant la **feature** sur l'image, et de calculer **le produit de convolution** entre **la feature** et chaque portion de l'image balayée.



<https://www.coursera.org/learn/convolutional-neural-networks-fr/home/week/1>

# Architecture du réseau CNN

**La couche de pooling:** est souvent placée entre deux couches de convolution

Elle reçoit en entrée plusieurs *feature maps*, et applique à chacune d'entre elles l'opération de *pooling*.

L'opération de *pooling* consiste à réduire la taille des images, tout en préservant leurs caractéristiques importantes. On obtient en sortie le même nombre de *feature maps* qu'en entrée, mais celles-ci sont bien plus petites. **En pratique, on utilise souvent une fenêtre de 2 ou 3 pixels de côté et une valeur de 2 pixels pour ce qui est de la valeur d'un pas.**

Pooling layer: Max pooling

1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

4 x 4

9	2
6	3

2 x 2

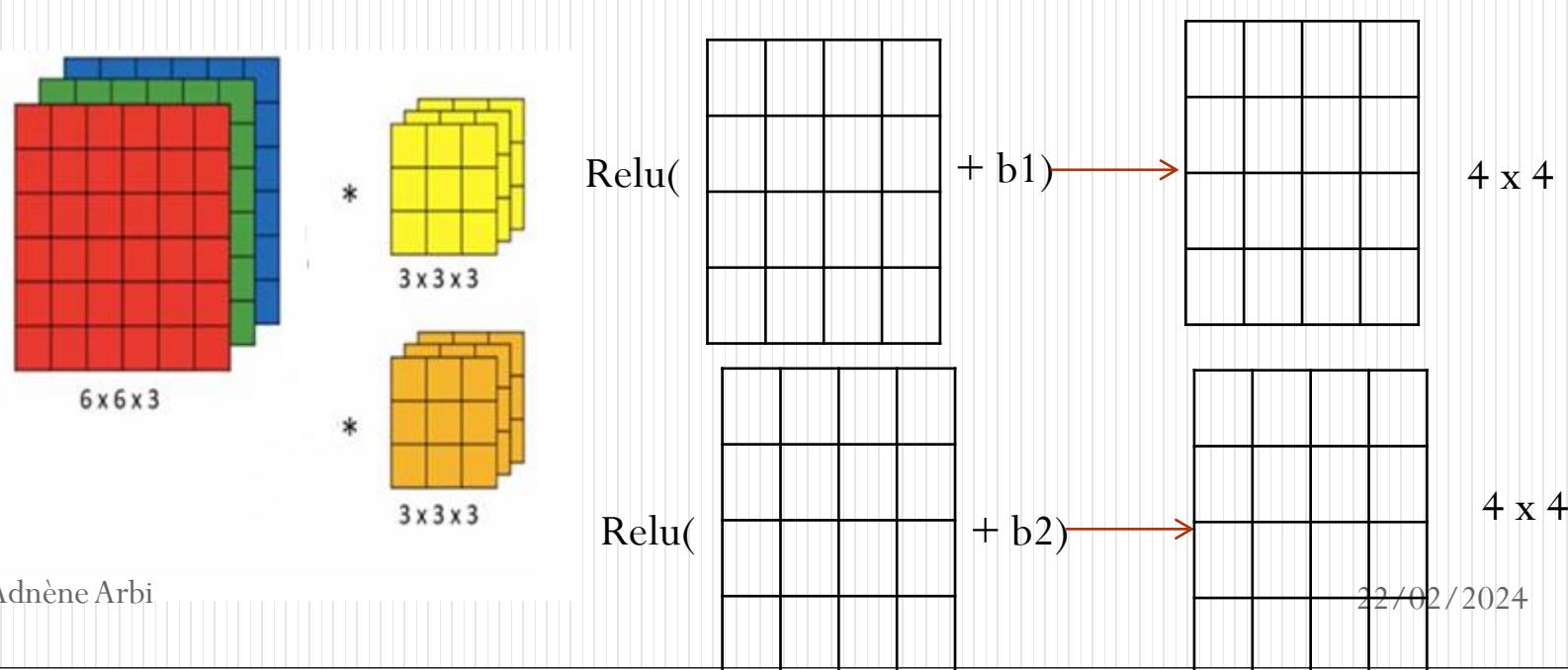
Hyperparamètres

f=2, s=2

# Architecture du réseau CNN

**Une couche de correction ReLU:** ReLU (*Rectified Linear Units*) désigne la fonction réelle non-linéaire définie par  $\text{ReLU}(x) = \max(0, x)$ .

La couche de correction ReLU remplace donc toutes les valeurs négatives reçues en entrées par des zéros. Elle joue le rôle de fonction d'activation.



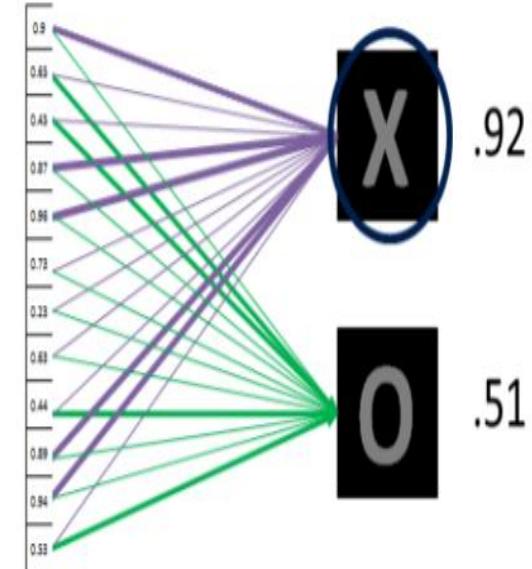
# Architecture du réseau de neurone Convolutif

**La couche fully-connected:** constitue toujours la dernière couche d'un réseau de neurones convolutif

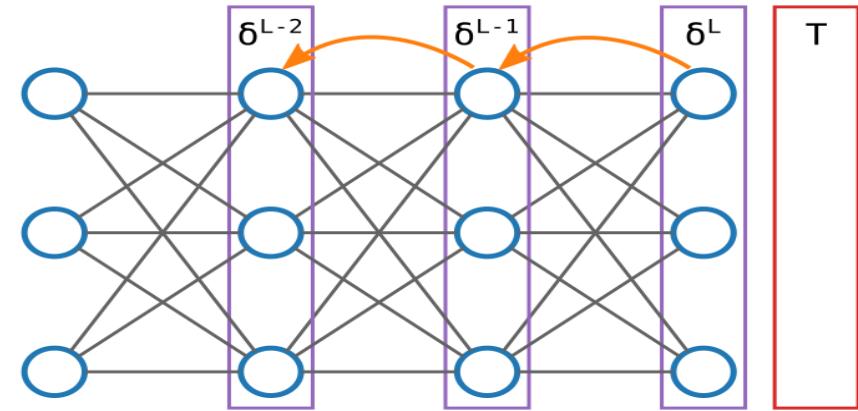
Ce type de couche reçoit un vecteur en entrée et produit un nouveau vecteur en sortie

*fully-connected* permet de classifier l'image en entrée du réseau

Elle renvoie un vecteur de taille  $n$ , où  $n$  est le nombre de classes dans notre problème de classification d'images. Chaque élément du vecteur indique la probabilité d'appartenance d'une image en entrée à une classe.



# Backpropagation



D'où viennent les caractéristiques? Et comment définit-on les poids des couches entièrement connectées?

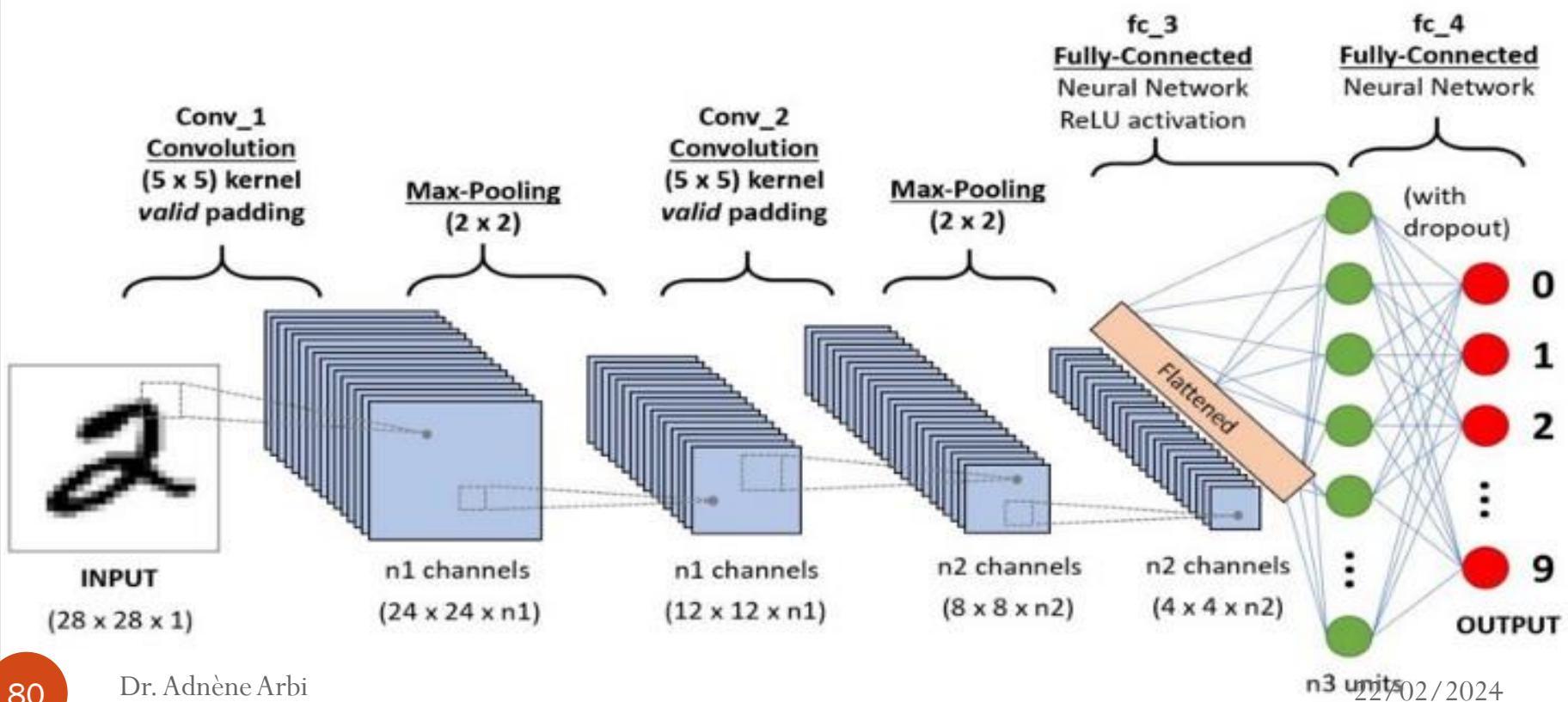
1. la propagation des calculs dans le réseau de neurones.
2. La méthode de rétropropagation:
  - **Minimiser l'erreur entre la sortie calculée à partir des paramètres des réseaux et la sortie désirée**
  - **Propager l'erreur vers l'arrière jusqu'à la couche précédente.**
  - **Ajuster les poids des connexions au niveau de chaque couche selon l'algorithme de descente du gradient**

# Synthèse

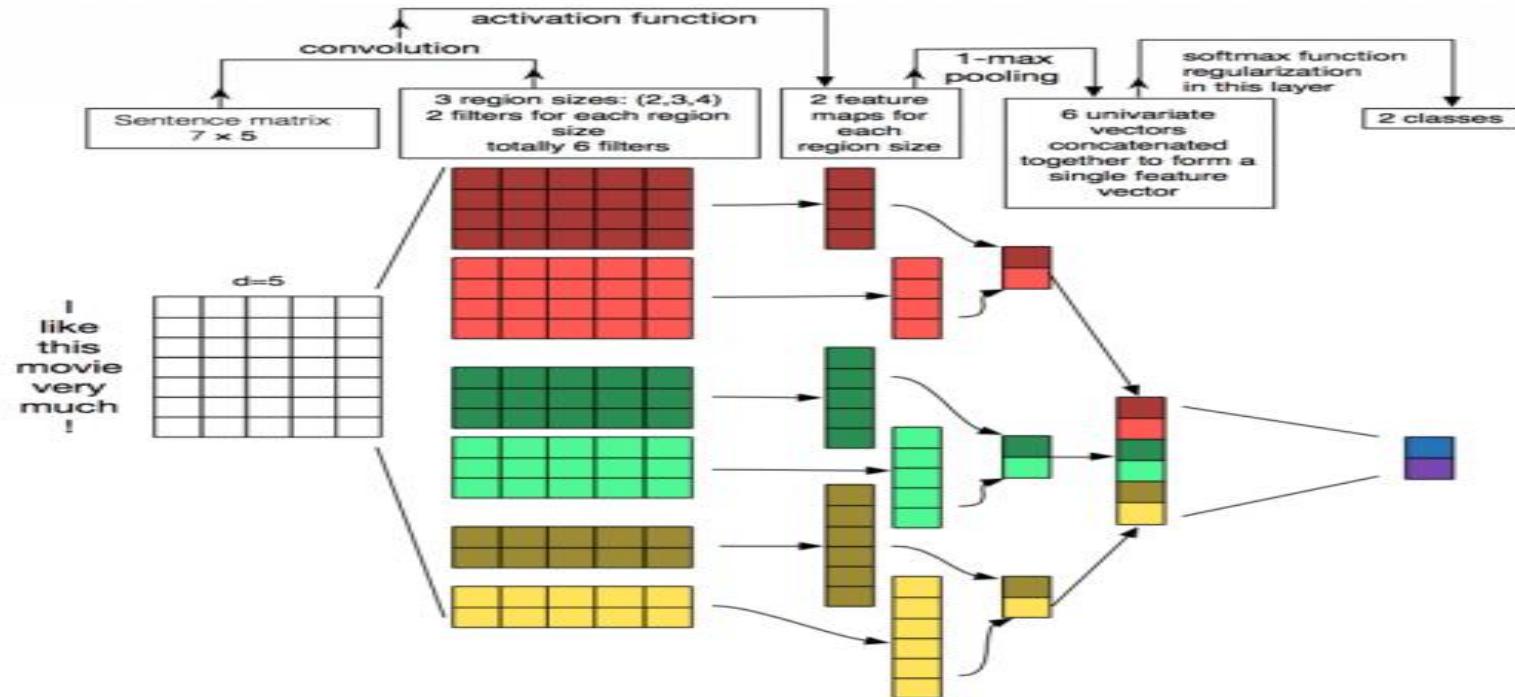
1. Les images brutes sont filtrées, **rectifiées et mises en commun** ,
2. pour créer un ensemble d'images **rétrécies** et filtrées par caractéristiques visibles dans chaque image.
3. Celles-ci peuvent ainsi être filtrées et rétrécies encore et encore.

A chaque fois, les caractéristiques deviennent plus grandes et plus complexes, et les images deviennent plus compactes. Cela permet aux **couches inférieures de représenter des aspects simples de l'image, tels que les bords et points lumineux. Les couches supérieures quant à elles représentent des aspects beaucoup plus complexes de l'image, tels que des formes et des patterns.**

# Application des CNN pour la reconnaissance de caractères manuscrits



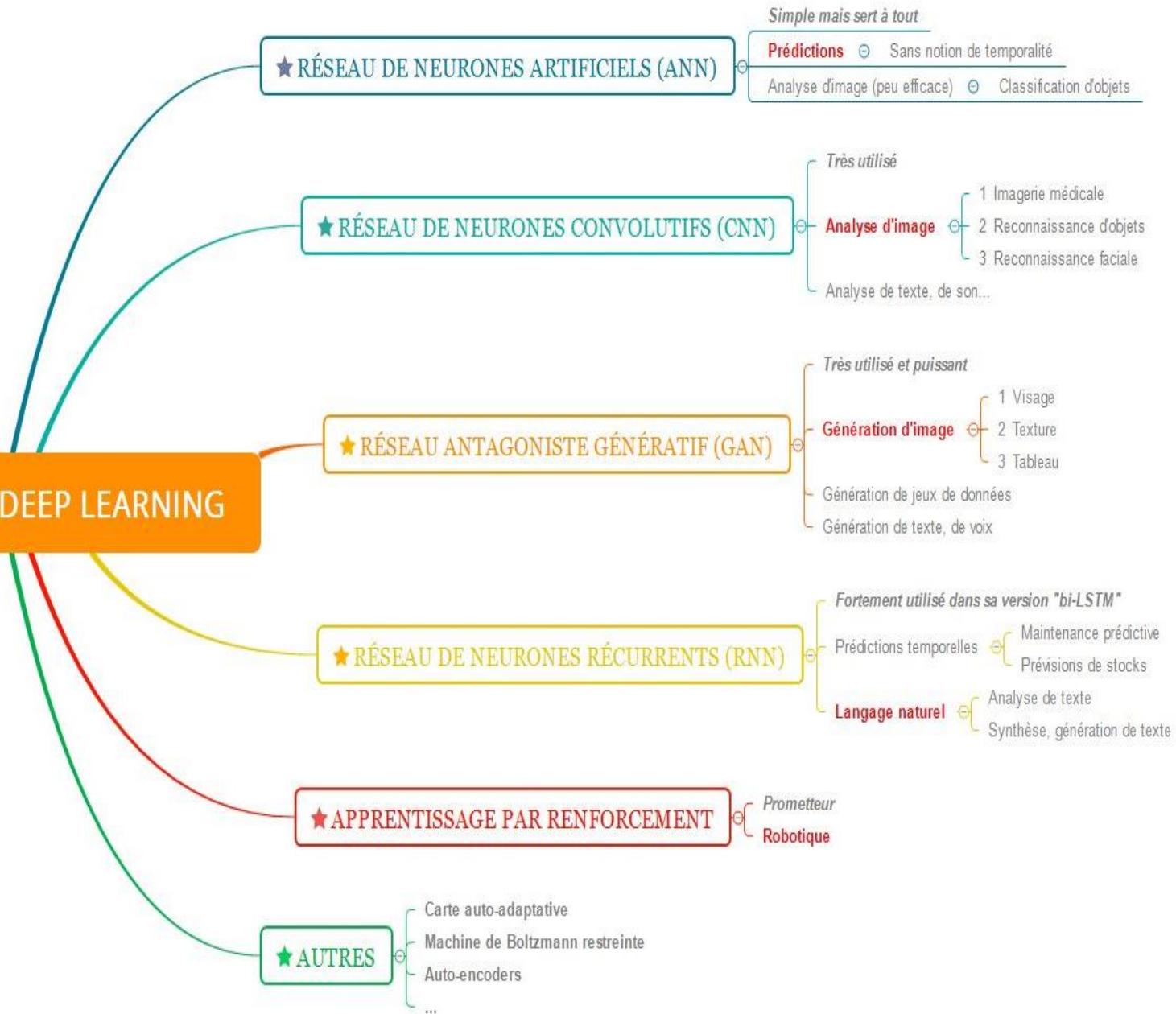
# Application pour le traitement automatique de la langue naturelle (NLP)



Zhang, Y., & Wallace, B. (2015). *A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification*. Zhang, Y., & Wallace, B. (2015). *A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification*.

# ALGORITHMES DE DEEP LEARNING

Source : [www.penseeartificielle.fr](http://www.penseeartificielle.fr)



# Conclusion

+ Les CNNs permettent à la fois l'extraction des feachers et la classification des données ce qui minimise le taux d'erreur de classification

+ Les CNNs conservent la sémantique des images entrées du réseau

+ Le problème de la dimensionnalité ne se pose pas

Les domaines d'application sont très vastes :La détection de localisation , La classification , La segmentation , La reconnaissance d'image, Le traitement du langage naturel...

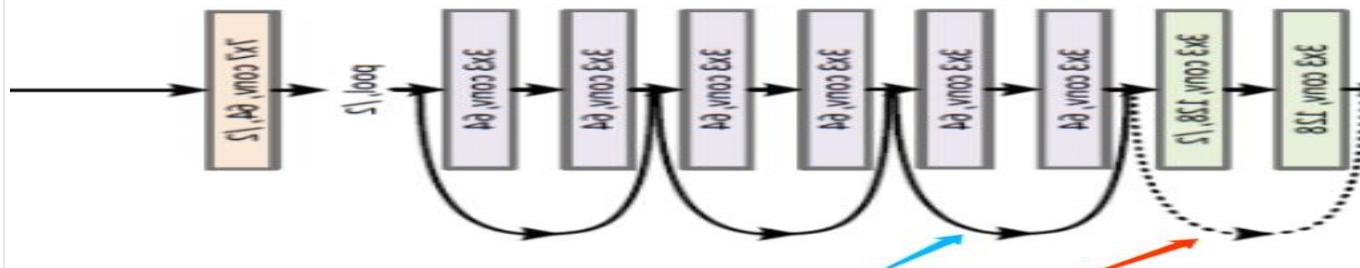
- Comme le gradient est rétro-propagé aux couches précédentes, une multiplication répétée peut rendre le gradient infiniment petit. En conséquence, à mesure que le réseau s'approfondit, ses performances se satureront ou commencent même à se dégrader rapidement.

# Conclusion

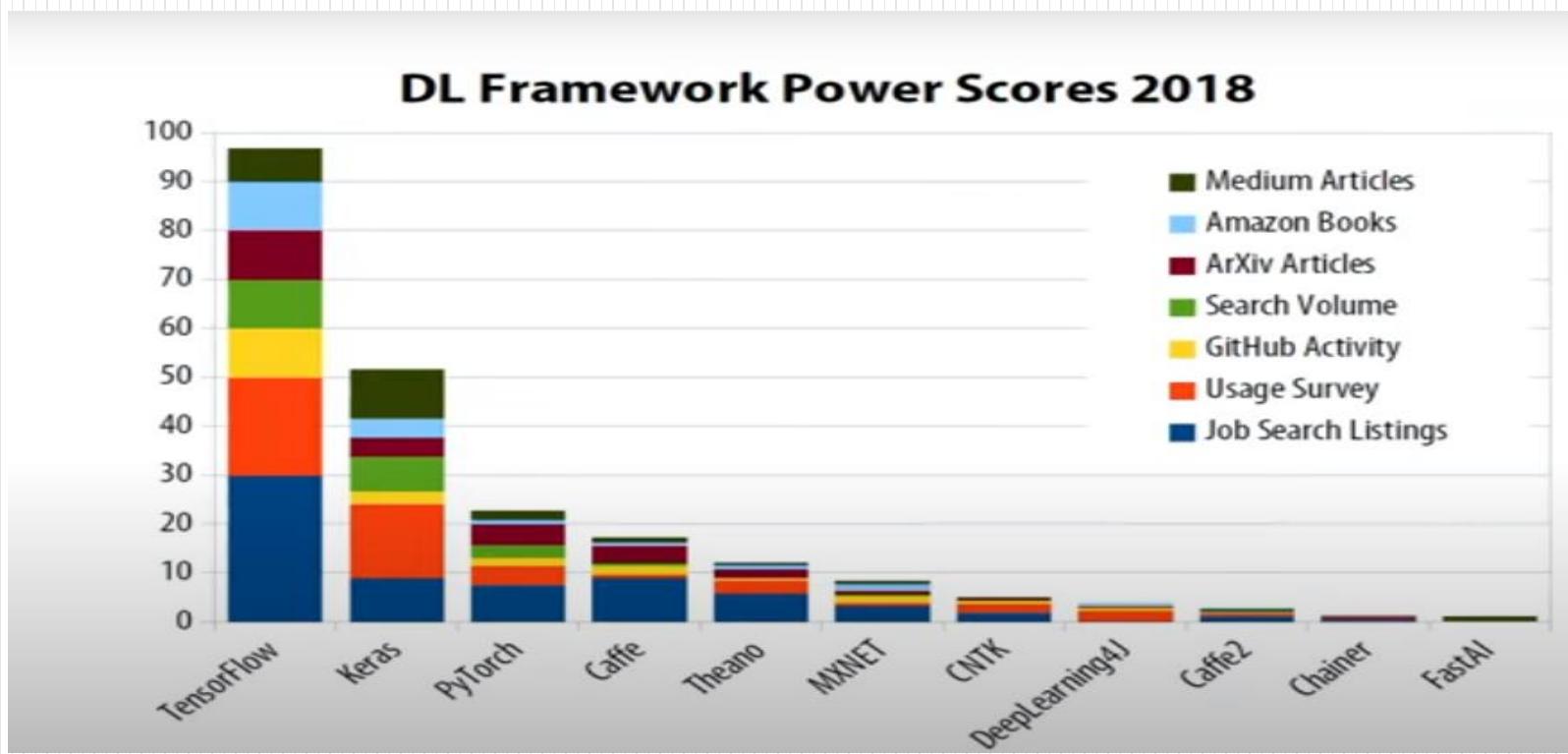
Cependant, l'augmentation de la profondeur du réseau ne fonctionne pas simplement en empilant des couches ensemble. Les réseaux profonds sont difficiles à entraîner en raison **du problème notoire du gradient de disparition**

- comme le gradient est rétro-propagé aux couches précédentes, une multiplication répétée peut rendre le gradient infiniment petit. En conséquence, à mesure que le réseau s'approfondit, ses performances se saturent ou commencent même à se dégrader rapidement.

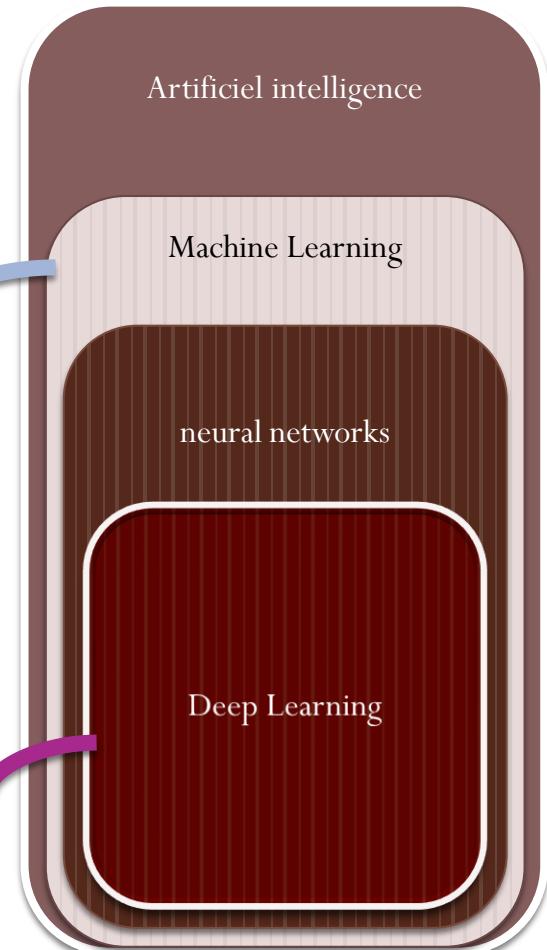
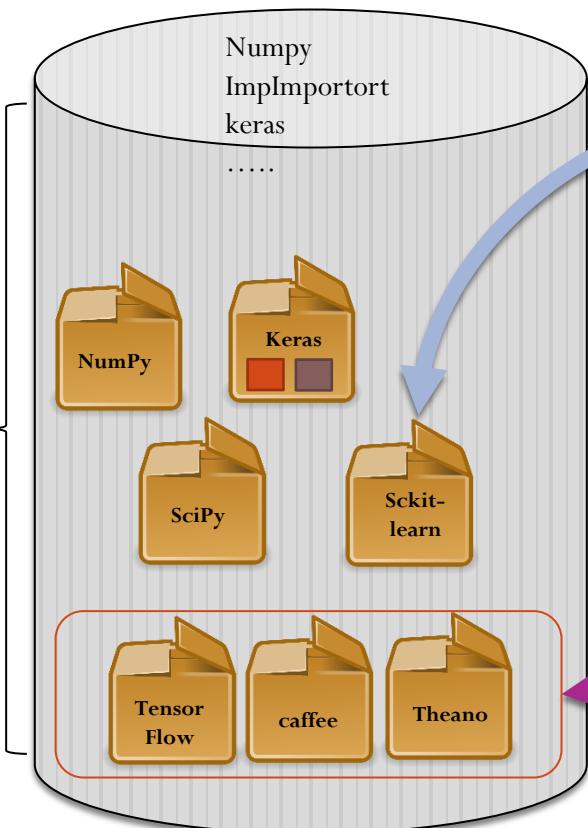
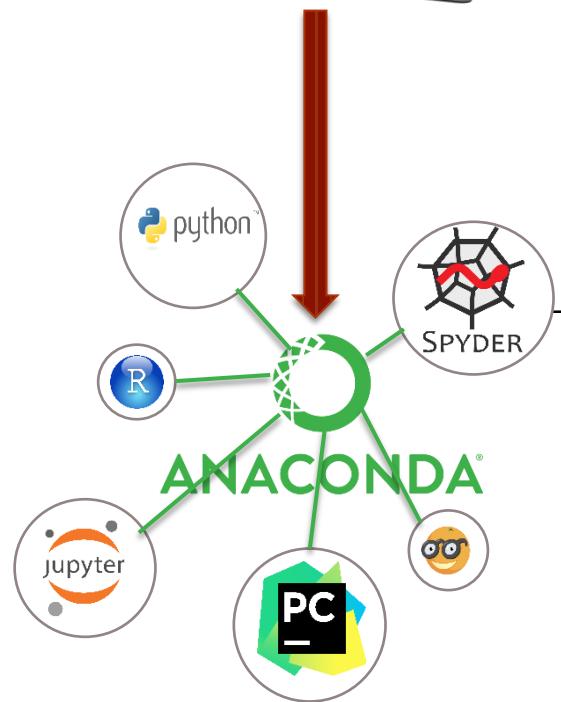
L'apparition du réseau **ResNet** (**residual neural network**)



# Environnement de travail

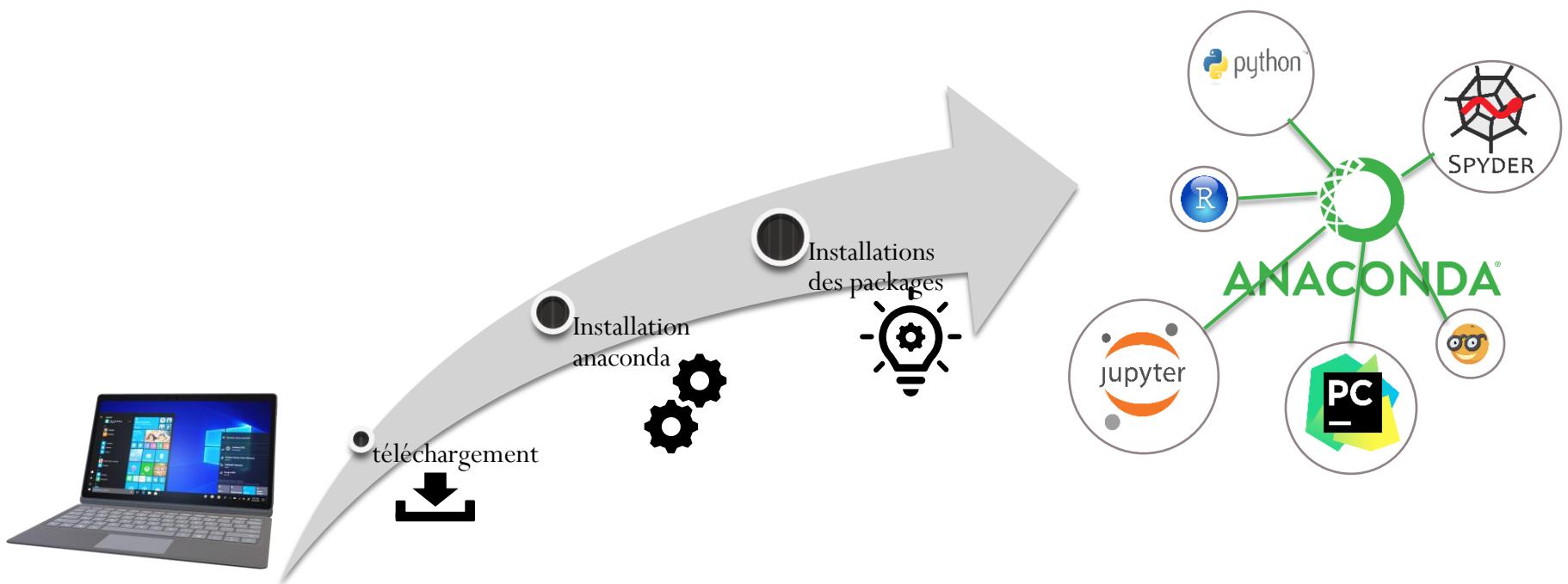


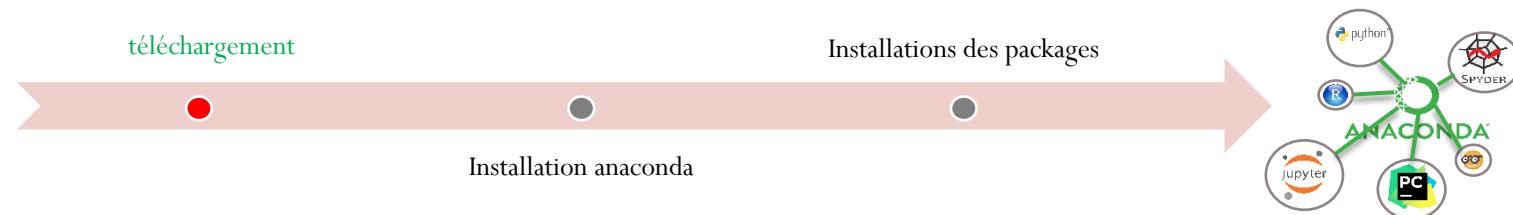
# ENVIRONNEMENT DE TRAVAIL



# Guide d'installation (ANACONDA PYTHON)

- Anaconda est une distribution scientifique de Python : c'est-à-dire qu'en installant Anaconda, vous installez Python, Jupyter Notebook, Spyder... et des dizaines de packages scientifiques, dont certains indispensables à l'analyse de données et d'autres pour faciliter la manipuler des réseaux de neurones.





**Individual Edition**

# Your data science toolkit

With over 25 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.

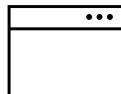
Anaconda Individual Edition

[Download](#)

For Windows  
Python 3.9 • 64-Bit Graphical Installer • 510 MB

Get Additional Installers

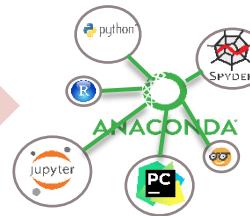
Windows | Mac | Linux



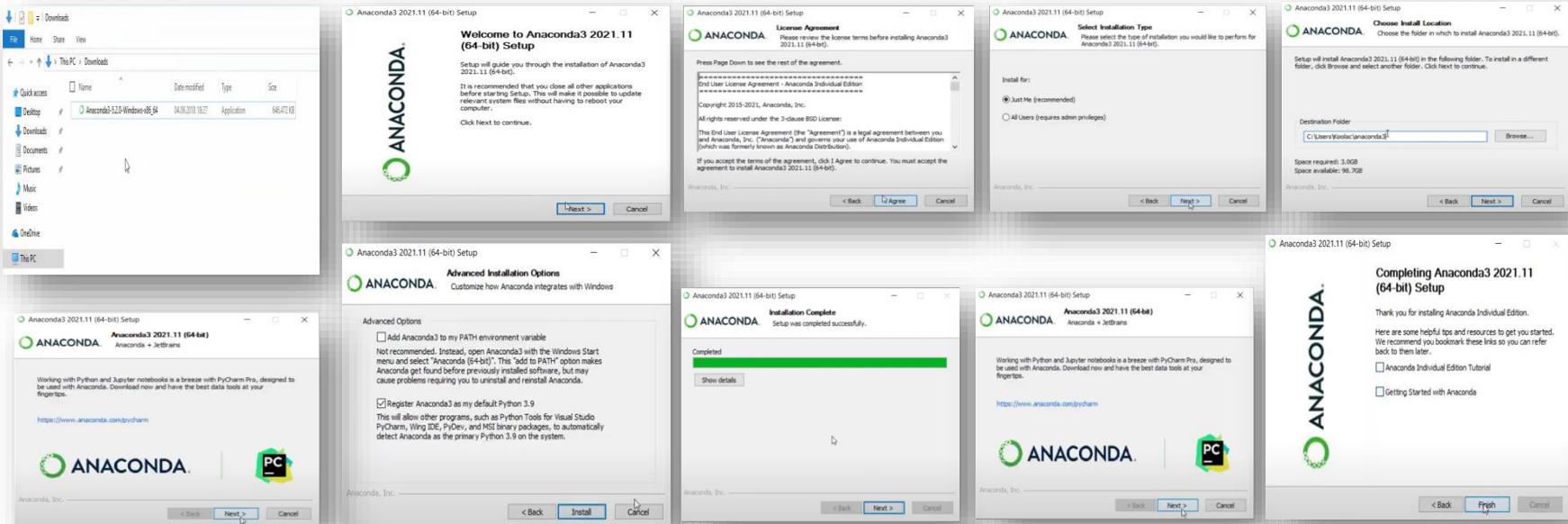
1. Téléchargez la distribution Anaconda correspondant à votre système d'exploitation, en Python version 3 : <https://www.anaconda.com/distribution/>

## téléchargement

## Installations des packages



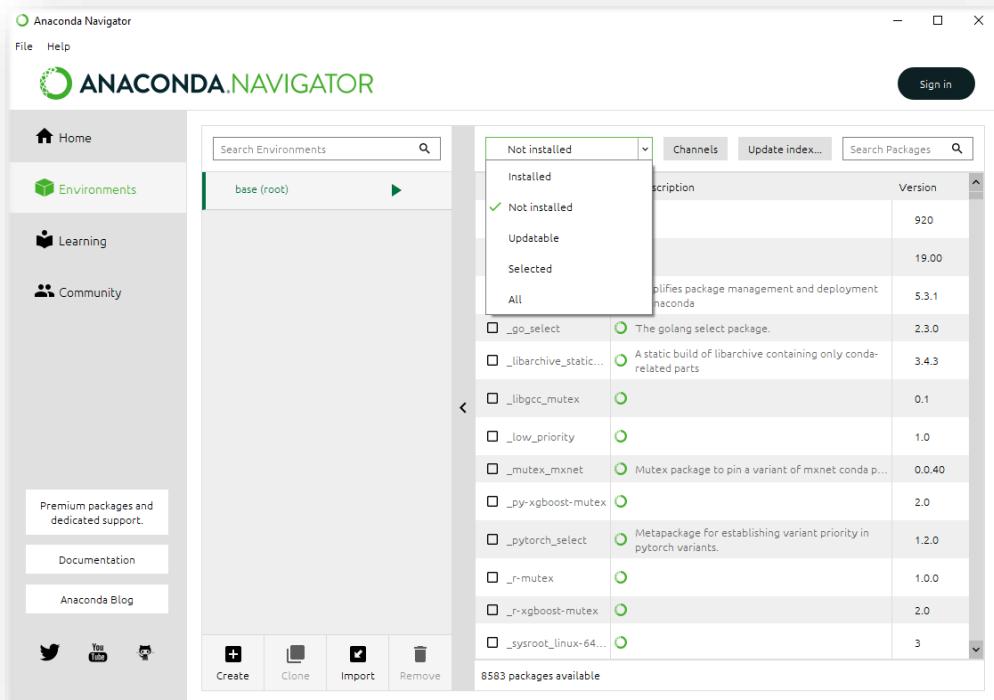
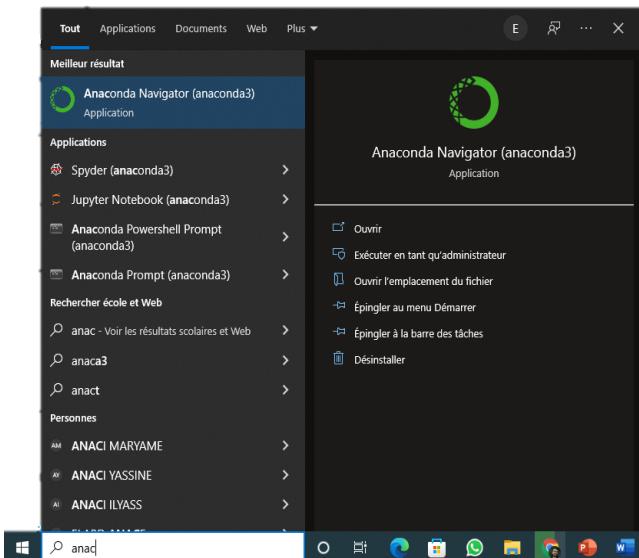
### Installation anaconda



téléchargement

Installations des packages

Installation anaconda



Après l'installation d'anaconda, vous lancez les éditeurs à partir de « Anaconda Navigator »

Avec l'onglet « Environnements » vous pouvez installer des autres Packages

# Un exemple : un réseau de neurones convolutifs avec jeu de données MNIST

Éditeur utilisé : SPYDER



La base de données utilisé : MNIST

0 0 0 0 0 0 0 0 0 0 0 0 0  
1 1 1 1 1 1 1 1 1 1 1 1 1  
2 2 2 2 2 2 2 2 2 2 2 2 2  
3 3 3 3 3 3 3 3 3 3 3 3 3  
4 4 4 4 4 4 4 4 4 4 4 4 4  
5 5 5 5 5 5 5 5 5 5 5 5 5  
6 6 6 6 6 6 6 6 6 6 6 6 6  
7 7 7 7 7 7 7 7 7 7 7 7 7  
8 8 8 8 8 8 8 8 8 8 8 8 8  
9 9 9 9 9 9 9 9 9 9 9 9 9

Ce que nous allons faire :



## Préparation des données

### Importation

importer les module de keras et les couches de convolutions



```
from keras.models import Sequential  
from keras.layers import Dense,Dropout,Flatten  
from keras.layers import Convolution2D,MaxPooling2D  
from keras.utils import np_utils as npu  
from tensorflow.keras import layers  
from tensorflow.keras import activations  
import matplotlib as plt
```

importer les données:  
utiliser le célèbre jeu de données MNIST



```
from keras.datasets import mnist  
(x_train,y_train),(x_test,y_test)=mnist.load_data(path="mnist.npz")
```

unifier les données:



```
x_train=x_train.reshape(x_train.shape[0],28,28,1)  
x_test=x_test.reshape(x_test.shape[0],28,28,1)
```

changement de type :  
Pour éviter les problèmes de calculs



```
x_train= x_train.astype('float32')  
x_test= x_test.astype('float32')
```

Normalisation :  
Changer échelle [0,255] vers  $[0,1]$   
Dr. Adnène Arbi



```
x_train/=255  
x_test/=255
```

## Créer un modèle

définir le modèle:

(Architecture)

Couche de convolution2D

MaxPooling2D

Dropout

Flatten

....

```
model=Sequential()  
model.add(Convolution2D(64, 3,3, activation='relu',input_shape=(28,28,1)))  
model.add(MaxPooling2D(pool_size=(2,2)))  
model.add(Dropout(0.25))  
model.add(Flatten())  
model.add(Dense(128,activation="relu"))  
model.add(Dropout(0.5))  
model.add(Dense(10,activation='softmax'))
```

Compiler le modèle



```
model.compile(loss='categorical_crossentropy',optimizer='adam' ,metrics=[ 'accuracy'])
```

Entraîner le modèle :

Ajuster les hyperparamètres  
du model



```
model.fit(x_train,Y_train,batch_size=16,epochs=5)
```

Evaluer le modèle



```
model.evaluate(x_test,Y_test,verbose=0)
```

# Interpréter les résultats

Les résultats



```
Epoch 1/5  
3750/3750 [=====] - 22s 6ms/step - loss: 0.4535 - accuracy: 0.8563  
Epoch 2/5  
3750/3750 [=====] - 21s 6ms/step - loss: 0.2456 - accuracy: 0.9232  
Epoch 3/5  
3750/3750 [=====] - 21s 6ms/step - loss: 0.2051 - accuracy: 0.9360  
Epoch 4/5  
3750/3750 [=====] - 21s 6ms/step - loss: 0.1844 - accuracy: 0.9426  
Epoch 5/5  
3750/3750 [=====] - 21s 6ms/step - loss: 0.1721 - accuracy: 0.9460
```

## Remarque :

- accuracy : 0,9460       $\Leftrightarrow$       la précision du modèle 94 %
- Loss : 0,1721       $\Leftrightarrow$       1,7% de classification incorrect
- Le changement d'architecture permet d'augmenter ou diminuer la précision (accuracy)