



# Réseaux de neurones - DL

---

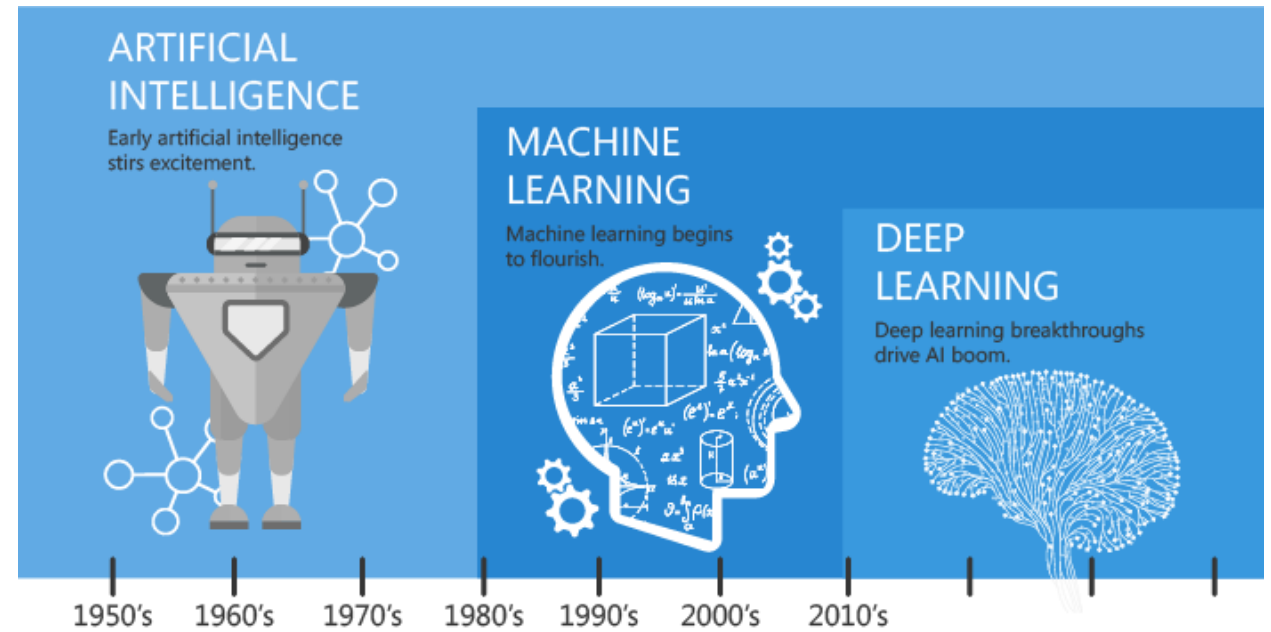
MME. KHAOULA ELBEDOUI

# Sommaire

---

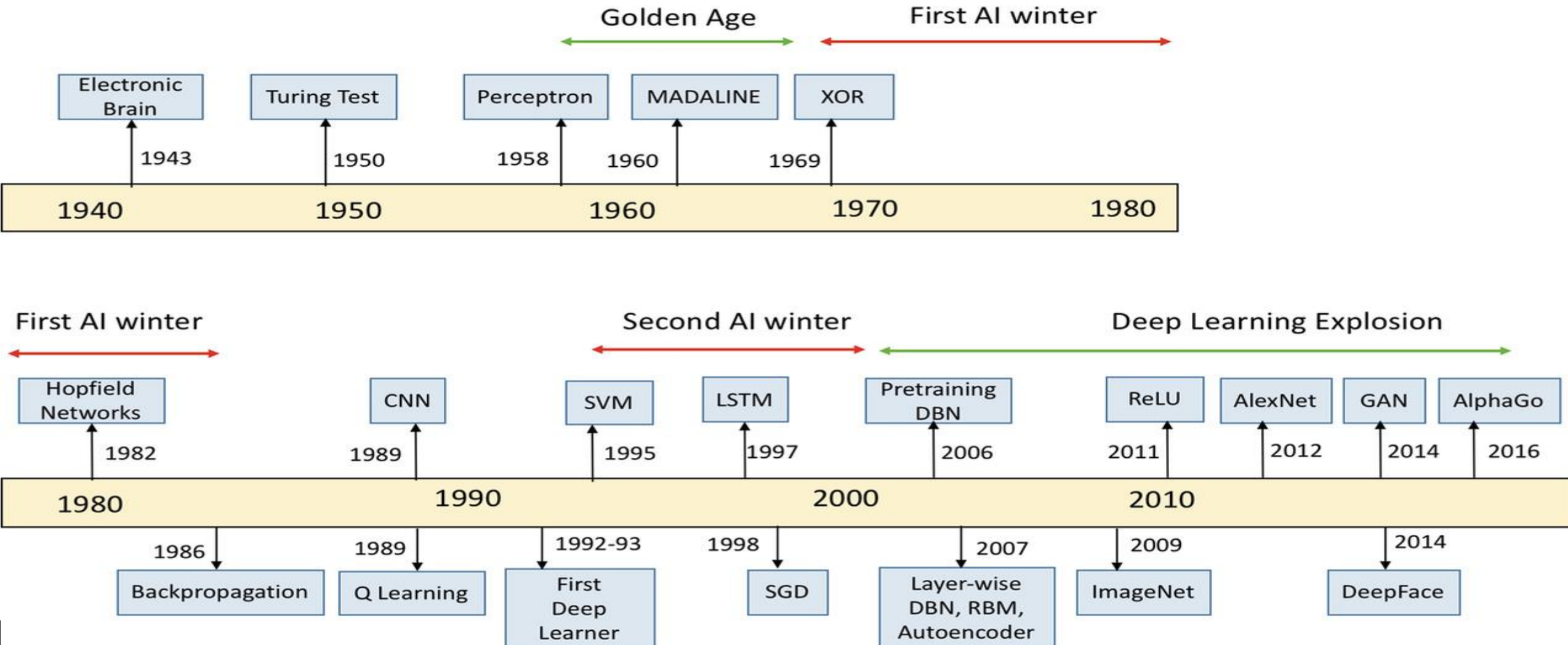
1. Préambule
2. Aperçu général
3. Perceptron
4. Premier RN
5. Architectures des RN
6. Gestion du RN
7. Etude des cas
  - a) CNN
  - b) RNN

# Préambule



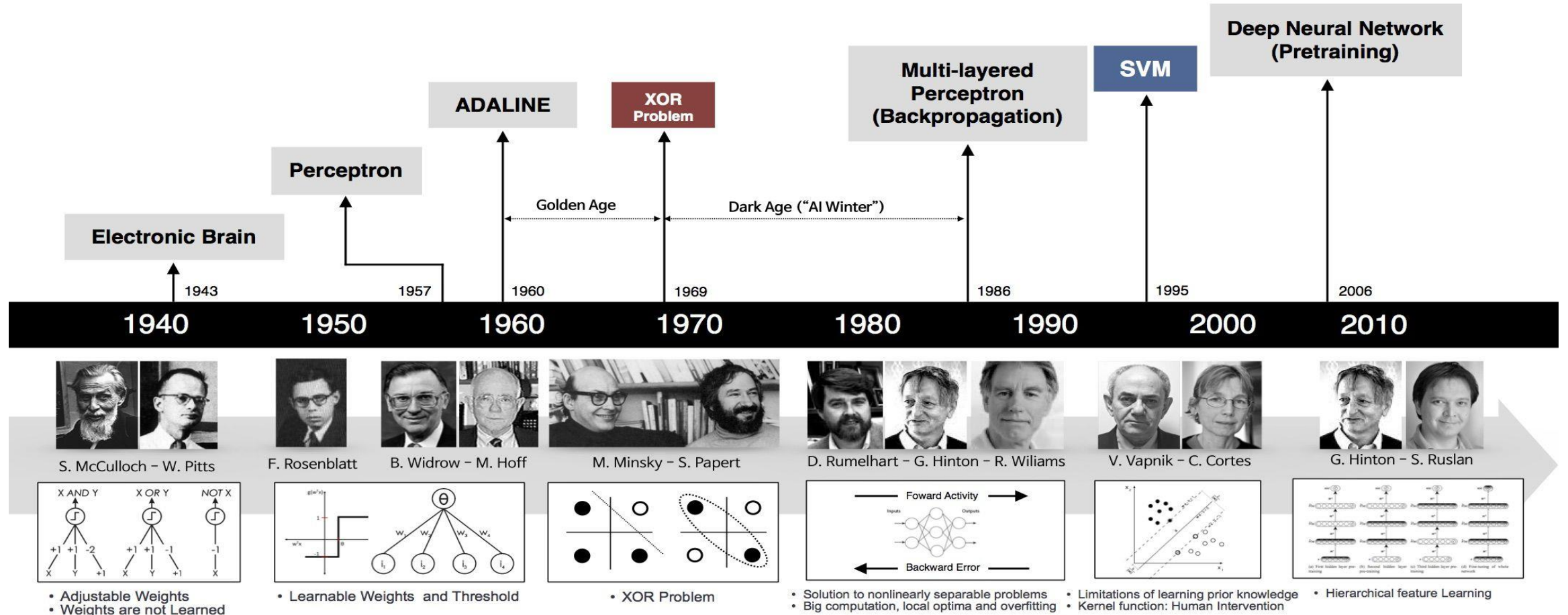
# Préambule

## 1. Historique



# Préambule

## 1. Historique



# Préambule

---

## 2. Ressources

### Aperçu général et histoire

- <https://www.youtube.com/watch?v=XUFLq6dKQok>
- <https://www.youtube.com/watch?v=trWrEWfhTVg>

### Pionniers

- [https://www.ted.com/talks/fei\\_fei\\_li\\_how\\_we\\_re\\_teaching\\_computers\\_to\\_understand\\_pictures](https://www.ted.com/talks/fei_fei_li_how_we_re_teaching_computers_to_understand_pictures)
- <https://www.youtube.com/watch?v=t4kyRyKyOpo>

### DL et traitement avancé d'image

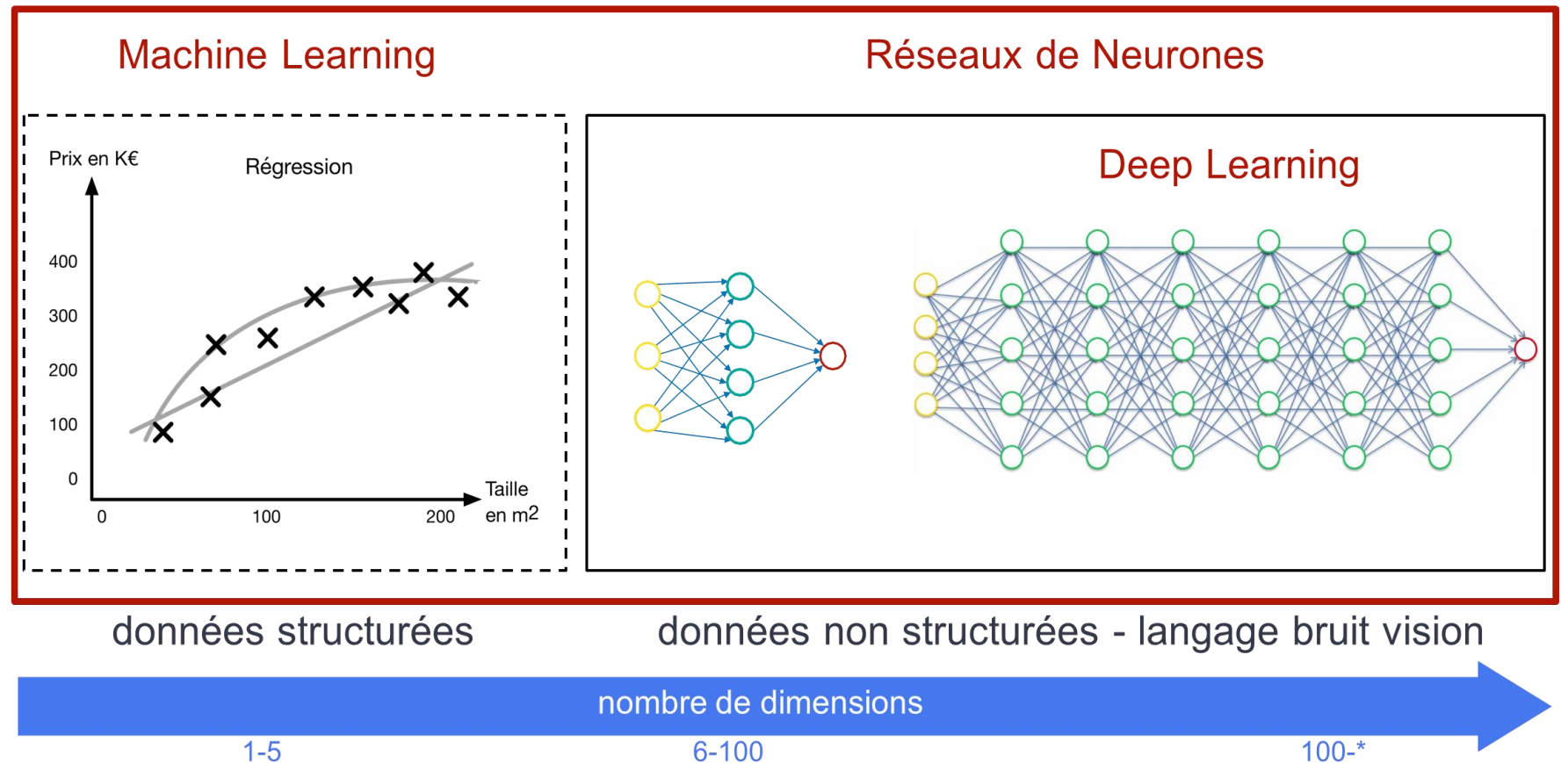
- <https://www.youtube.com/watch?v=7uE6hypji0o&list=PLHae9ggVvqPgyRQQOtENr6hK0m1UquGaG>



# Préambule

## 3. Naissance

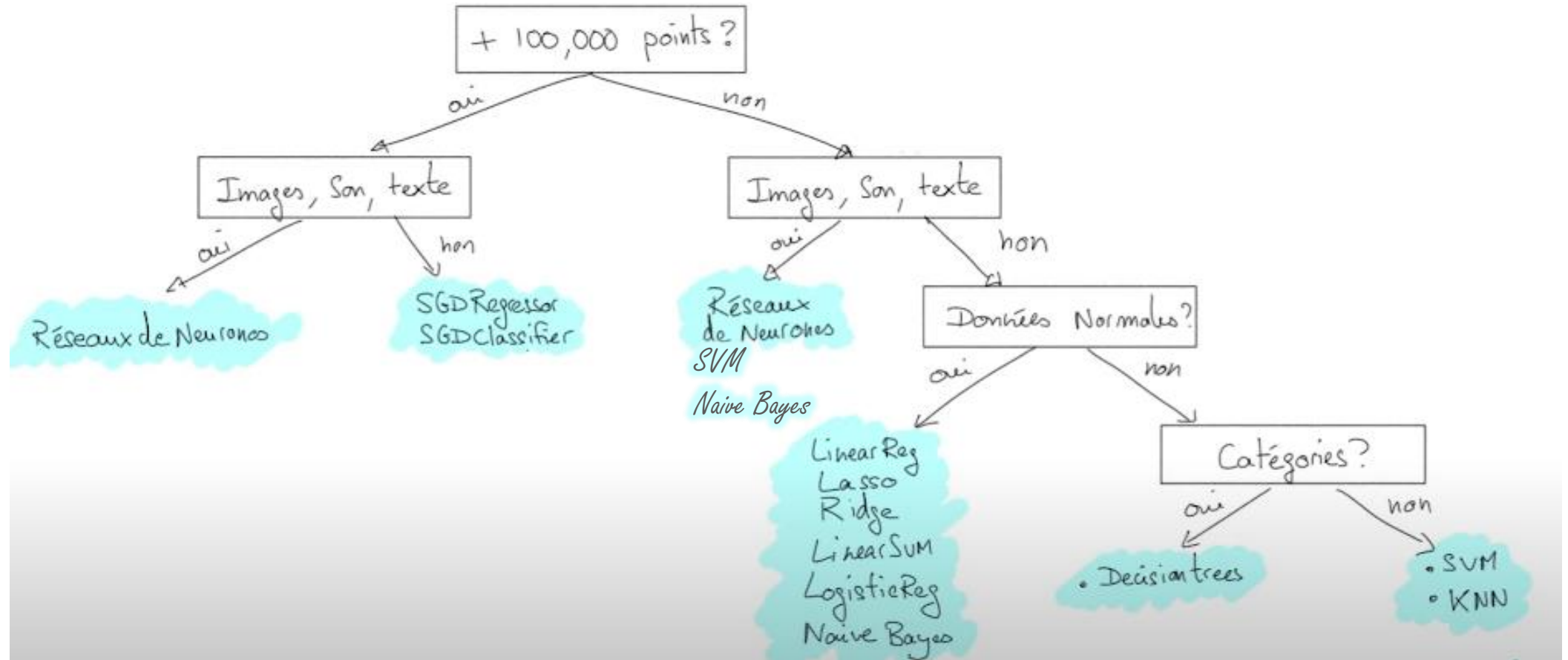
- Capacité de calcul (Loi de Moore)
- Quantité de données



# Préambule

## 4. Choix du modèle

- <https://www.youtube.com/watch?v=4mqKmTbAnHY>

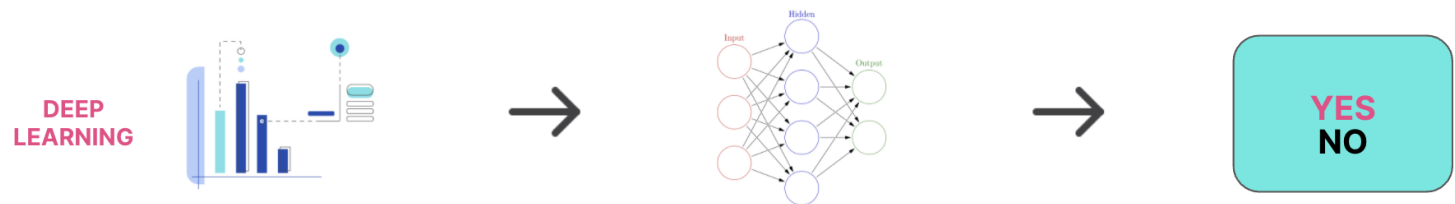
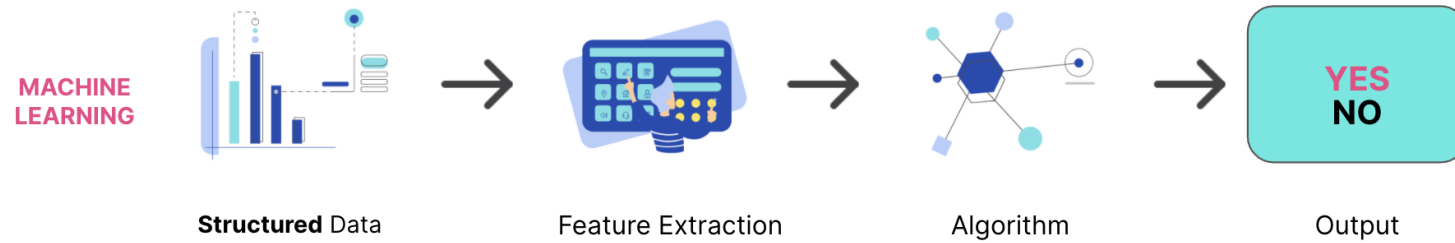




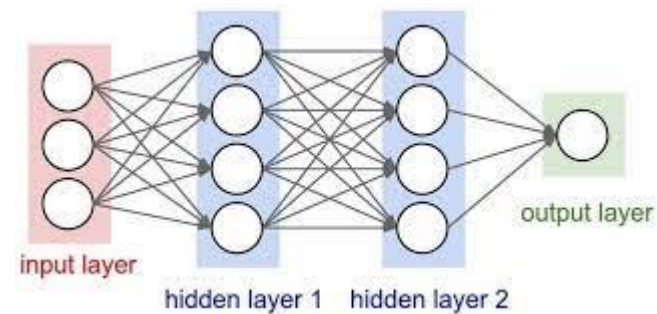
# Préambule

---

## 5. ML vs DL



# Aperçu général sur le réseau de neurones



# Aperçu général

## 1. Fonctionnement théorique

Données : images des chiffres (MNIST)



7



Traitements



Objectif :

Déterminer la classe de l'image c.à.d elle représente quel chiffre ?

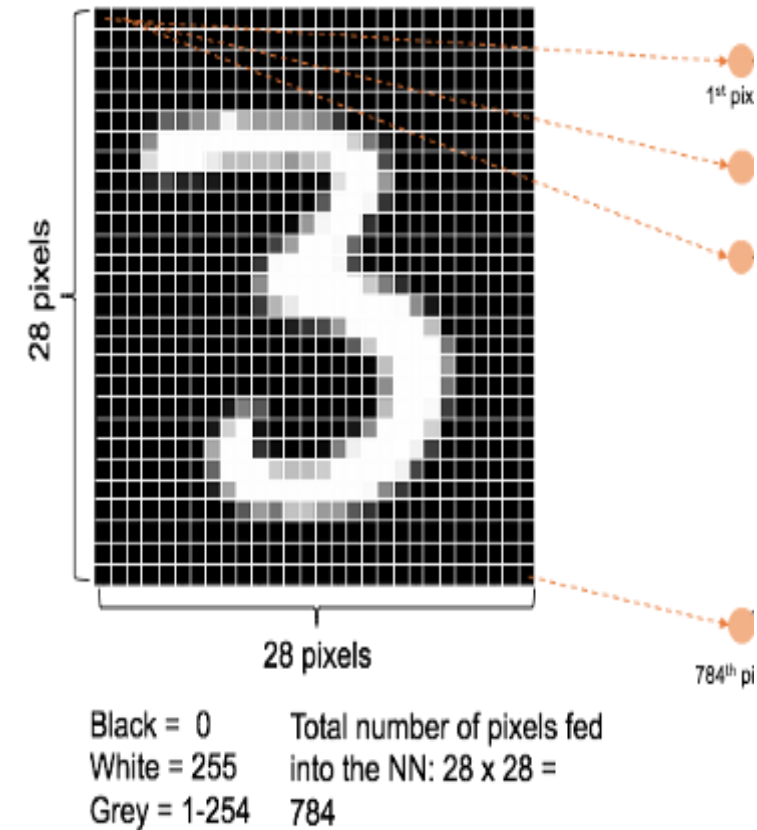
# Aperçu général

## 1. Fonctionnement théorique

Données : images des chiffres (MNIST)

Chaque image est de taille 28x28

L'image est représentée sur 784 pixels (0 :Noir, 255:Blanc)

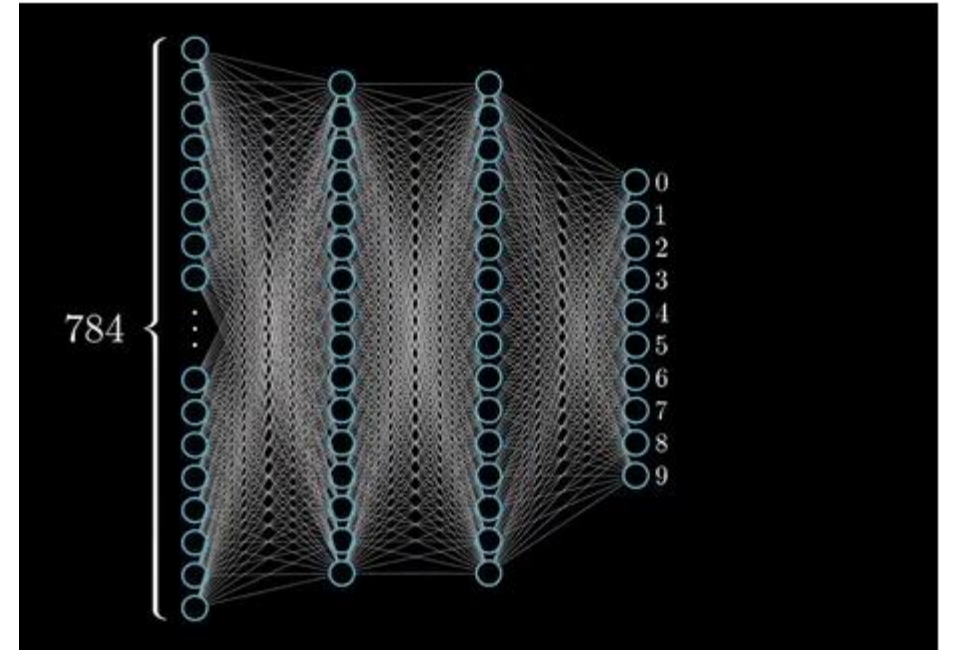


# Aperçu général

## 1. Fonctionnement théorique

Données : images des chiffres (MNIST)

On peut utiliser donc un réseau de neurones totalement connecté (FCNN) pour déterminer chaque image correspond à quel chiffre



# Aperçu général

## 1. Fonctionnement théorique

Données : images des chiffres (MNIST)

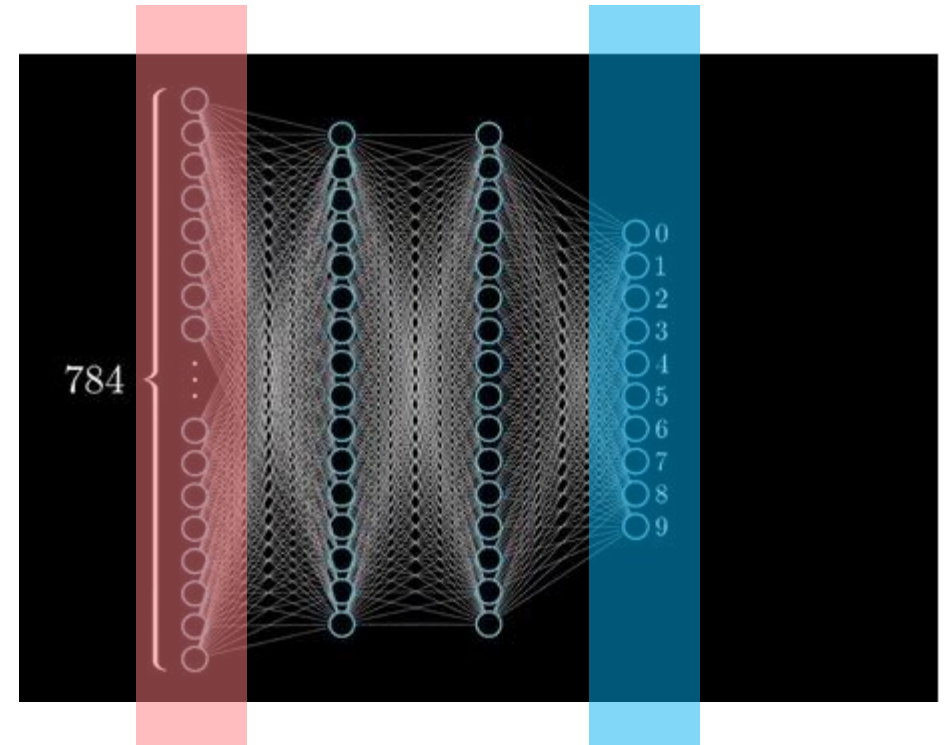
On peut utiliser donc un réseau de neurones totalement connecté (FCNN) pour déterminer chaque image correspond à quel chiffre

### Input Layer :

contient **784** entrées chacune correspond à la valeur d'un pixel

### Output Layer :

contient 10 sorties chacune correspond à la classe de l'image (chiffre 0, chiffre 1, ..., chiffre 9)





# Aperçu général

## 1. Fonctionnement théorique

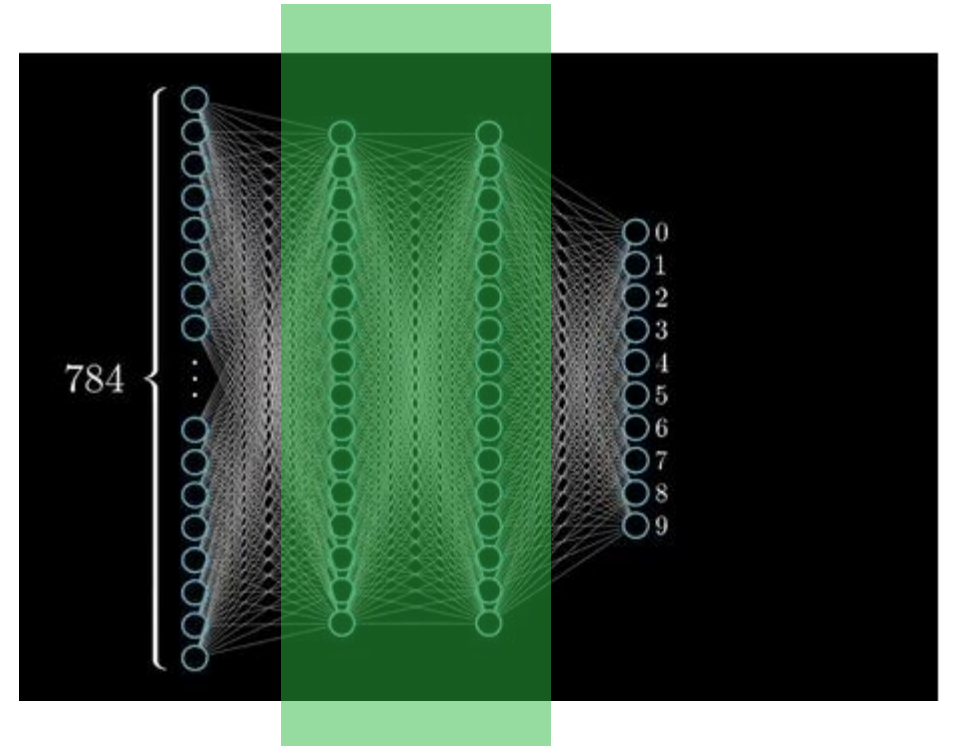
Données : images des chiffres (MNIST)

On peut utiliser donc un réseau de neurones totalement connecté (FCNN) pour déterminer chaque image correspond à quel chiffre

### Hidden Layers :

Le nombre de ces couches cachées dépend du problème  
Il faut au moins 2 couches cachées pour dire DL

Le nombre de couches en total représente la **profondeur** du réseau



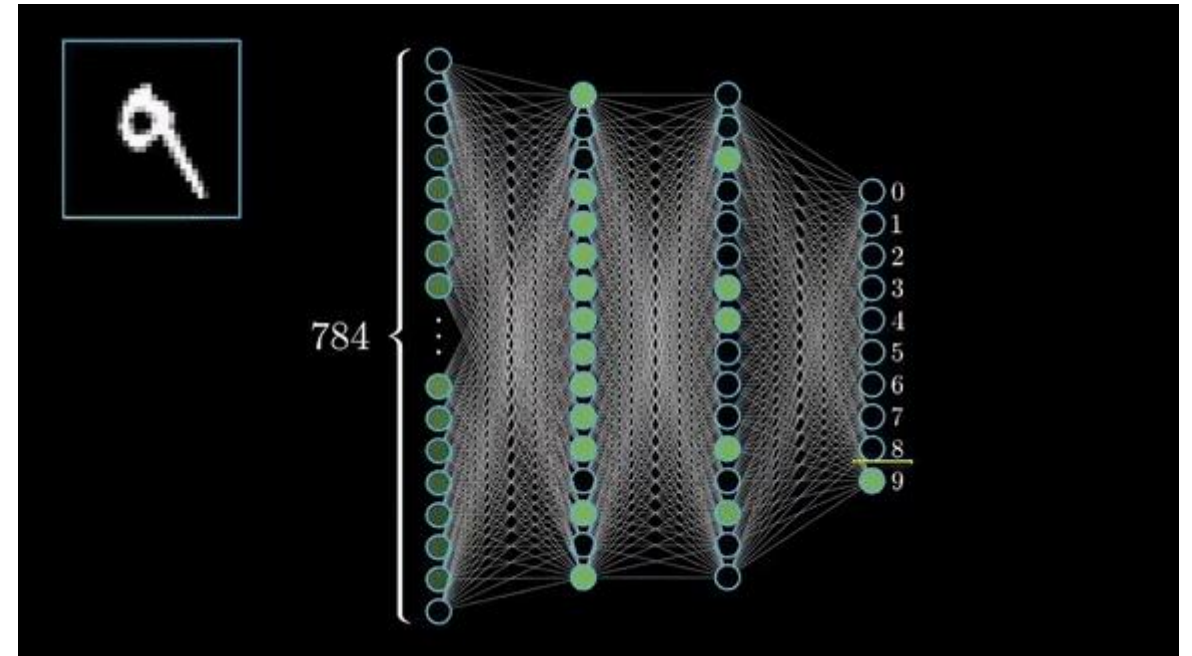
# Aperçu général

## 1. Fonctionnement théorique

Données : images des chiffres (MNIST)

Le fonctionnement du réseau de neurones fait :

**1- Propagation (Forward-propagation)** d'une couche à l'autre (suivant) en fonction des poids (initialisés aléatoirement)



# Aperçu général

## 1. Fonctionnement théorique

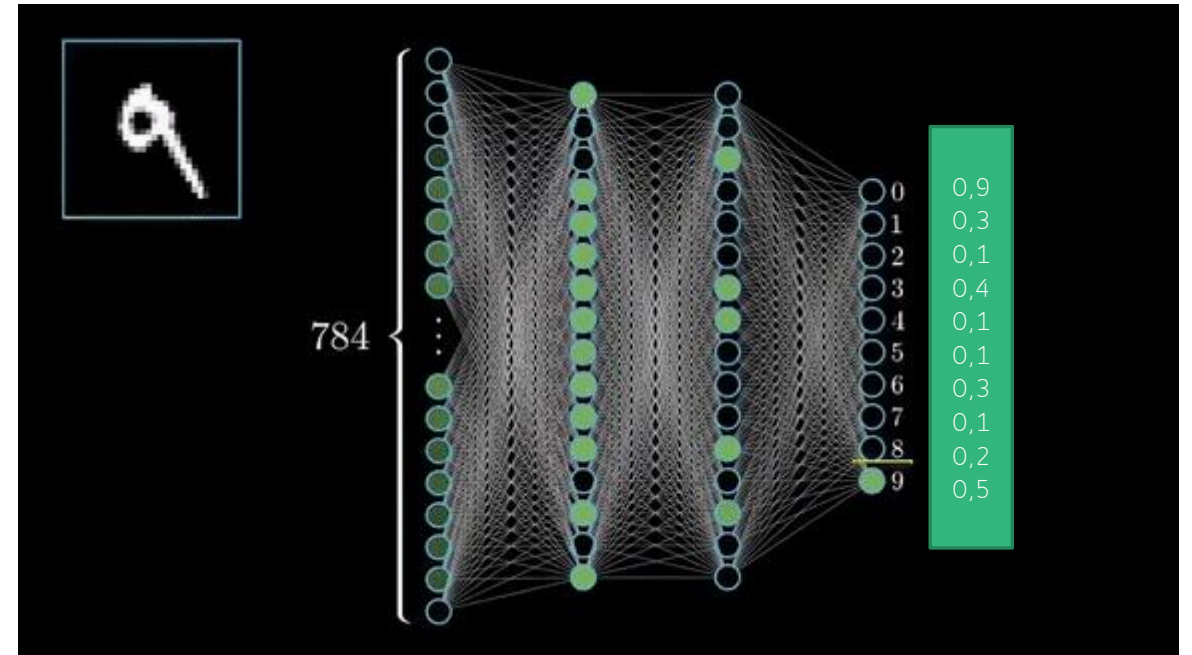
Données : images des chiffres (MNIST)

Le fonctionnement du réseau de neurones fait :

**1- Propagation (Forward-propagation)** d'une couche à l'autre (suivant) en fonction des poids (initialisés aléatoirement)

**En sortie :**

On aura le pourcentage d'appartenance de l'image à la classe d'un chiffre (% confidence/probabilité)



# Aperçu général

## 1. Fonctionnement théorique

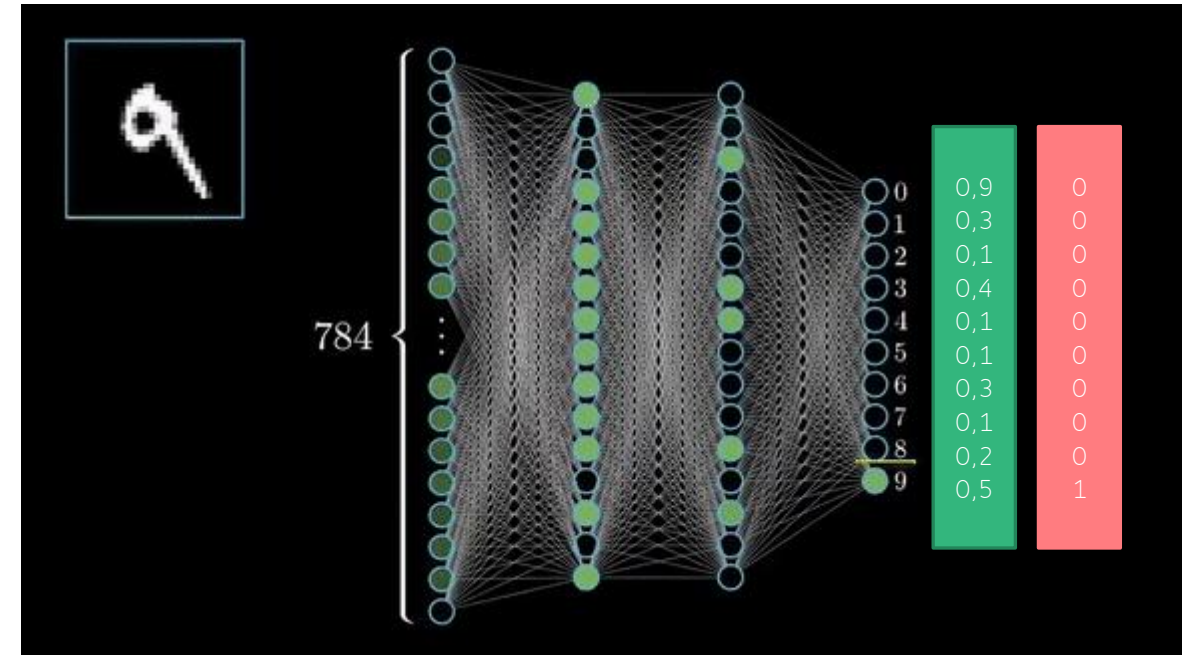
Données : images des chiffres (MNIST)

Le fonctionnement du réseau de neurones fait :

**1- Propagation (Forward-propagation)** d'une couche à l'autre (suivant) en fonction des poids (initialisés aléatoirement)

**Evaluation :**

Le coût est la différence entre le % d'appartenance prédit et sa valeur réelle (d'autres formes sont possibles)



# Aperçu général

## 1. Fonctionnement théorique

Données : images des chiffres (MNIST)

Le fonctionnement du réseau de neurones fait :

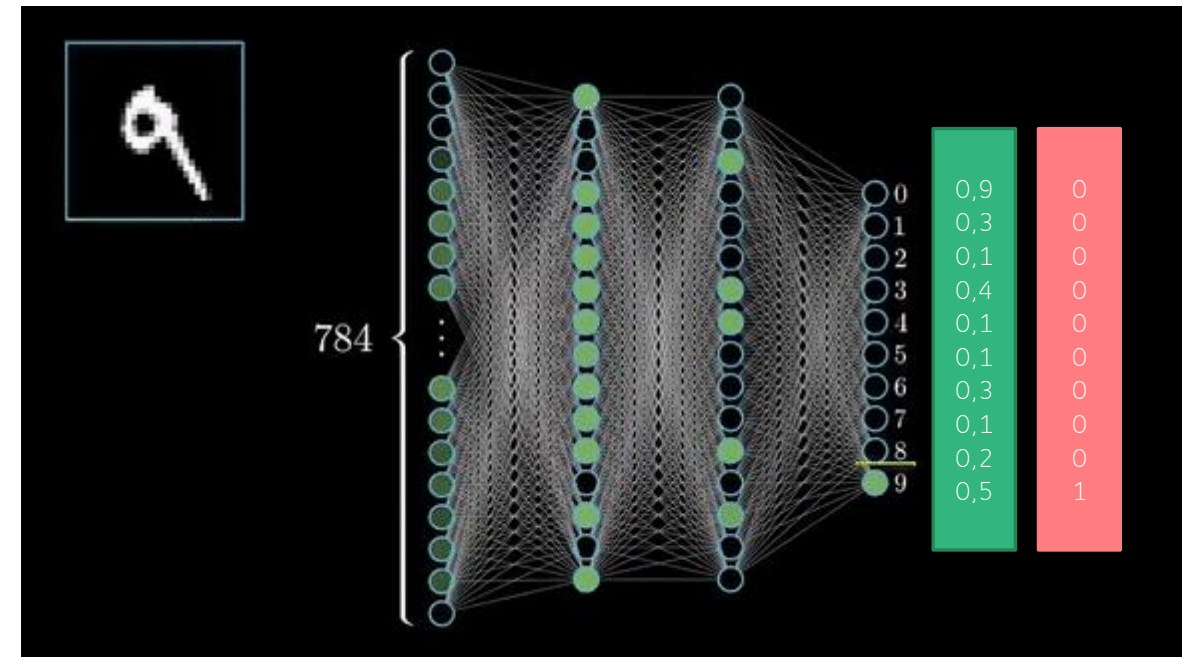
**1- Propagation (Forward-propagation)** d'une couche à l'autre (suivant) en fonction des poids (initialisés aléatoirement)

**Evaluation :**

Le coût est la différence entre le % d'appartenance prédit et sa valeur réelle (d'autres formes sont possibles)

**Solution :**

Réajuster les poids pour savoir lesquels sont **prépondérants** pour avoir le meilleur résultat (coût minimal)



# Aperçu général

## 1. Fonctionnement théorique

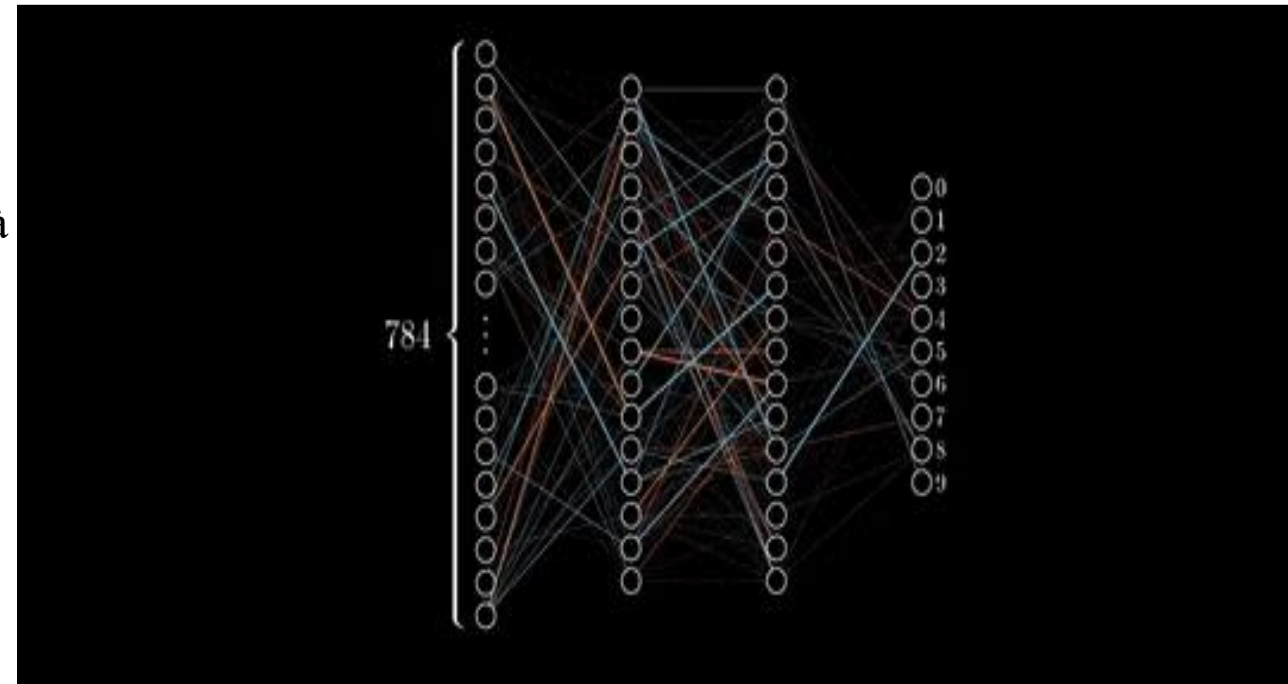
Données : images des chiffres (MNIST)

Le fonctionnement du réseau de neurones fait :

**1- Propagation (Forward-propagation)** d'une couche à l'autre (suivant) en fonction des poids

**2- Rétro Propagation (Back-propagation)** pour réajuster les poids

Forward-propagation



Back-propagation





# Aperçu général

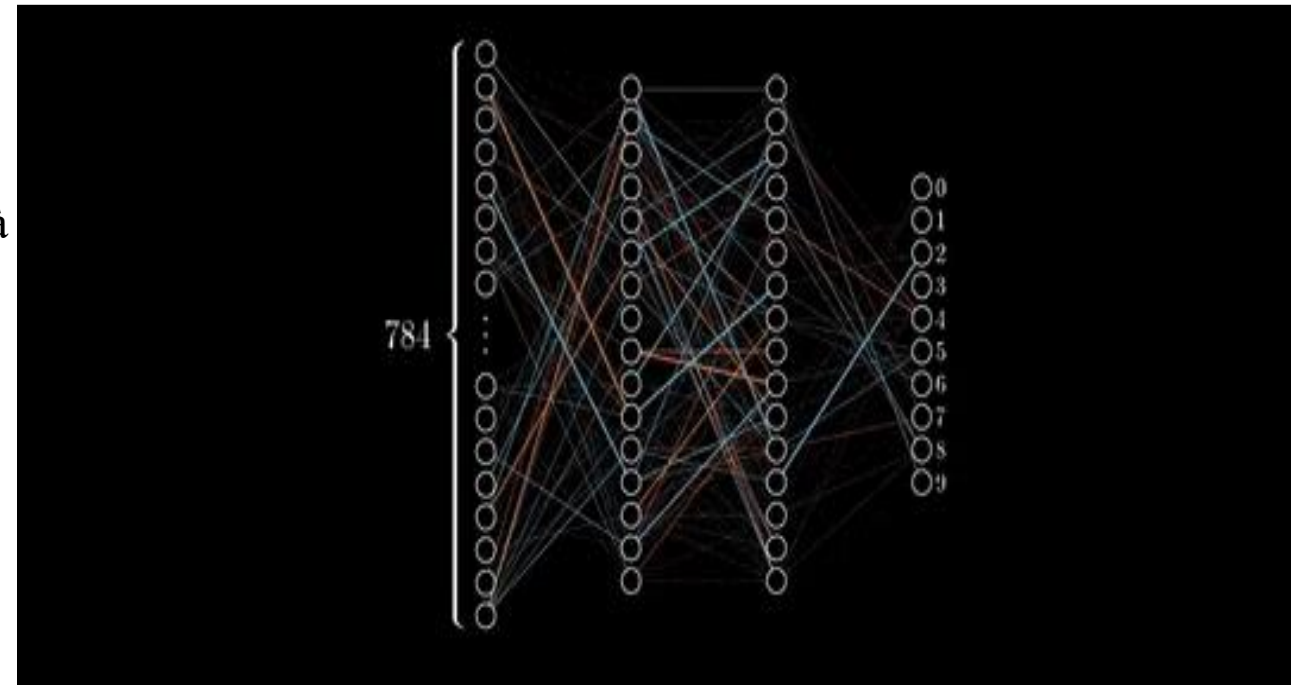
## 1. Fonctionnement théorique

Données : images des chiffres (MNIST)

Le fonctionnement du réseau de neurones fait :

**1- Propagation (Forward-propagation)** d'une couche à l'autre (suivant) en fonction des poids

**2- Rétro Propagation (Back-propagation)** pour réajuster les poids



Ce processus est répété pour plusieurs échantillons d'images jusqu'à ce que le coût soit minimal = **Phase d'Apprentissage (training)**  
Le nombre de répétitions est dit époque (**Epoch**)

# Aperçu général

---

## 2. Rétropropagation (back propagation)

### Poids

Pour déterminer la valeur optimale d'un poids qui minimise le coût, il est inutile de parcourir toutes les valeurs possibles du poids

Il faut :

- Initialiser les poids par des valeurs aléatoires
- Calculer la dérivée de la fonction coût en fonction du poids
  - Si dérivée **positive** alors il faut **diminuer** la valeur du poids
  - Si dérivée **négative** alors il faut **augmenter** la valeur du poids
- Répéter ce processus jusqu'à la convergence  $\|\nabla w\| \approx 0$  = **Gradient (Gradient descent)**

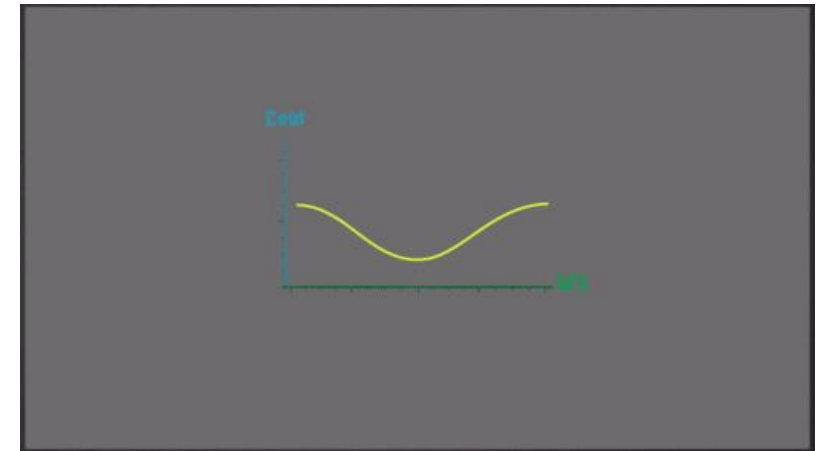
# Aperçu général

---

## 2. Rétropropagation (back propagation)

### Poids

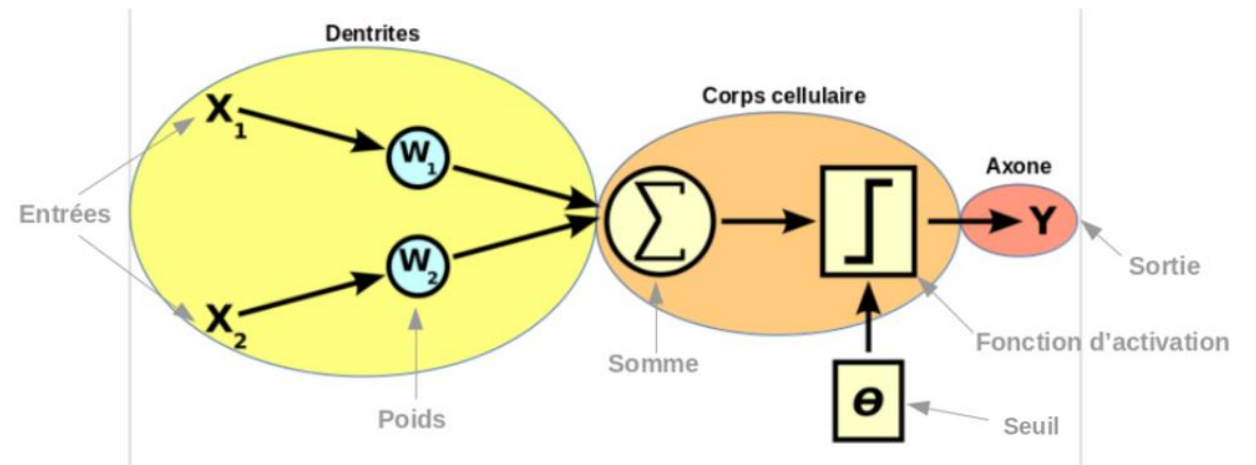
Pour déterminer les valeurs optimales des poids => **Gradient (Gradient descent)**  
dans un espace à plusieurs dimensions  $w^k_i$  et coût



Lien pour plus de détail :

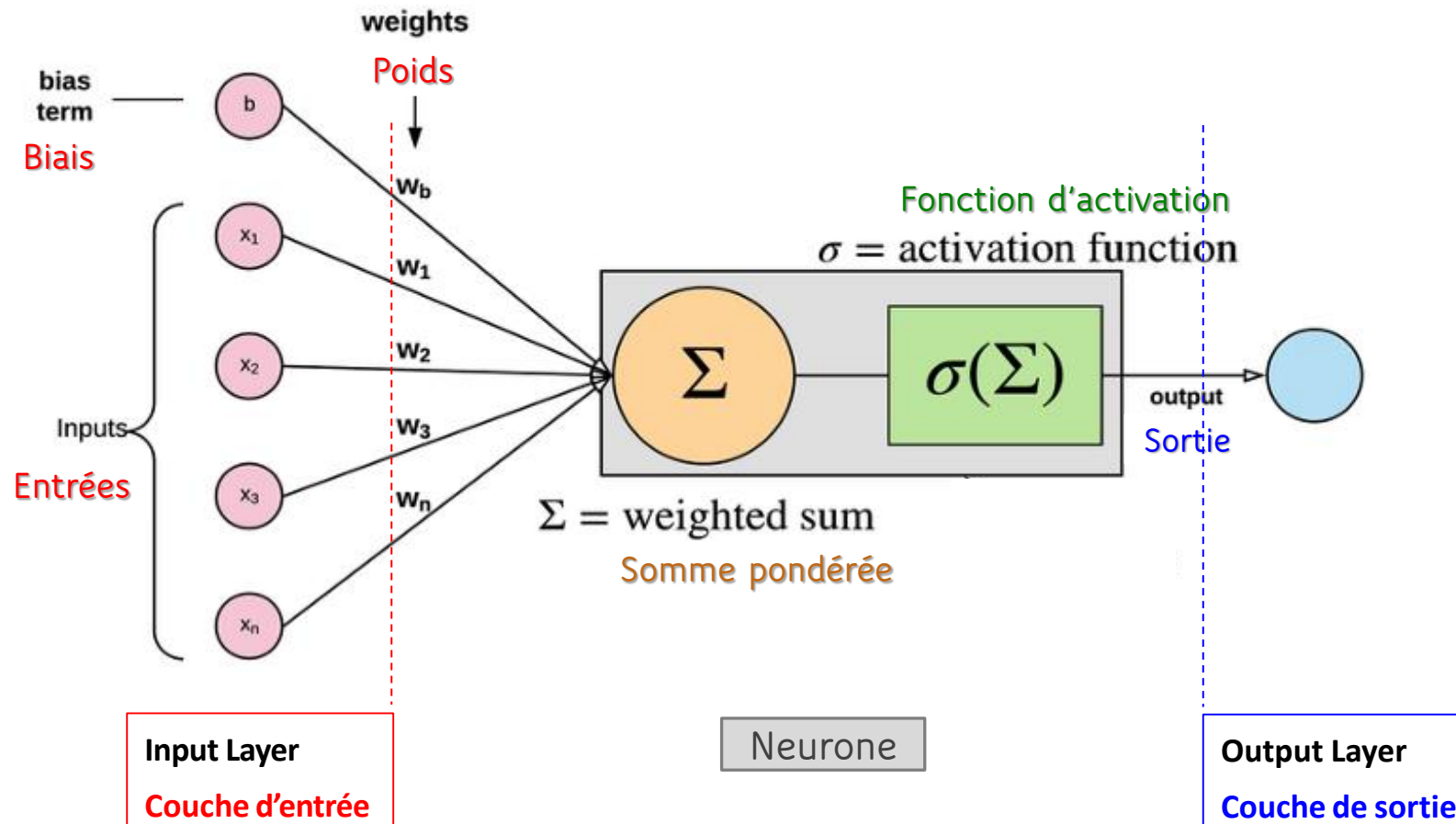
[https://www.youtube.com/watch?v=JtmzCiVKAR4&list=PLO\\_fdPEVlfKoanjvTJbIbd9V5d9Pzp8Rw&index=3](https://www.youtube.com/watch?v=JtmzCiVKAR4&list=PLO_fdPEVlfKoanjvTJbIbd9V5d9Pzp8Rw&index=3)

# Perceptron



# Perceptron simple

## 1. Modèle



# Perceptron simple

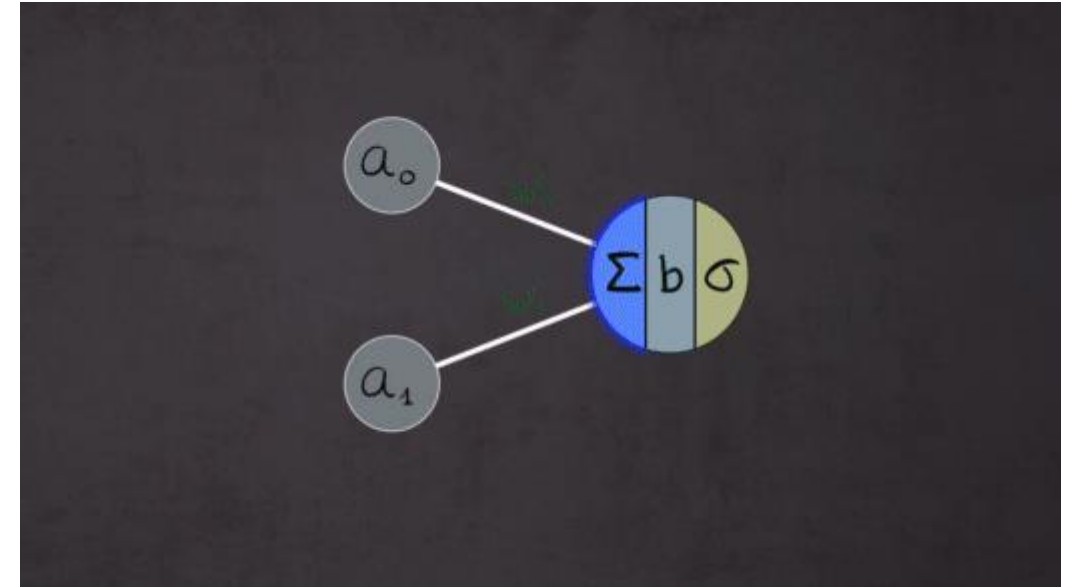
## 2. Fonctionnement

Chaque neurone travaille en trois étapes :

Étape 1: Somme pondérée (agrégation)

Étape 2: Ajout du biais (seuil)

Étape 3: Fonction d'activation





# Perceptron simple

---

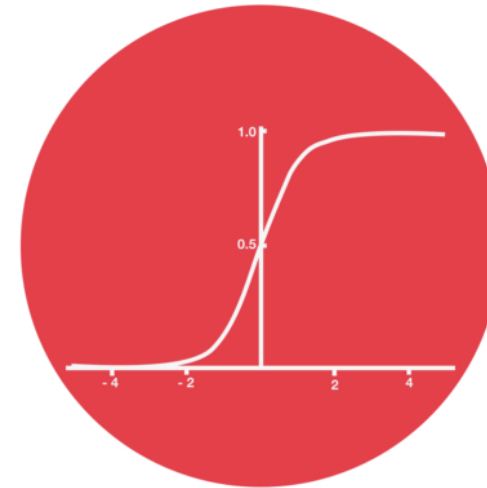
## 2. Fonctionnement

### Fonction d'activation :

Le résultat de la fonction d'activation indique si **le neurone est actif** ou **inactif**

La fonction d'activation est pour introduire l'aspect **non-linéaire** au modèle

5
0.1
-0.5

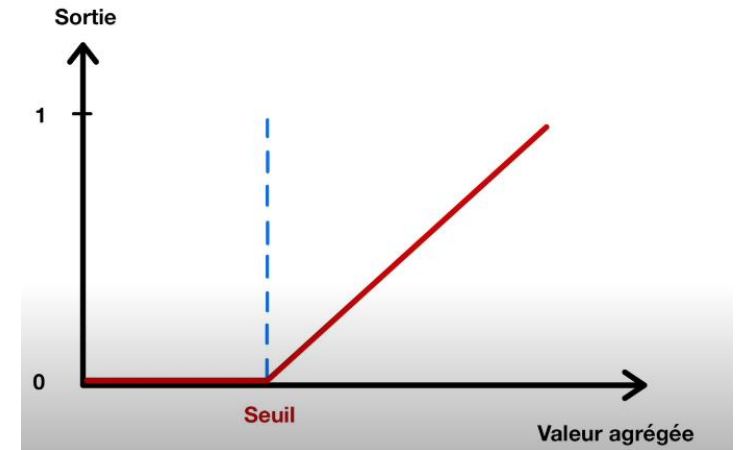
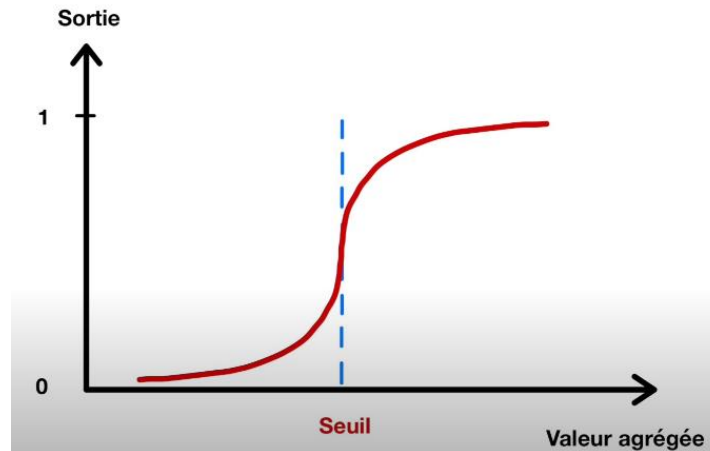
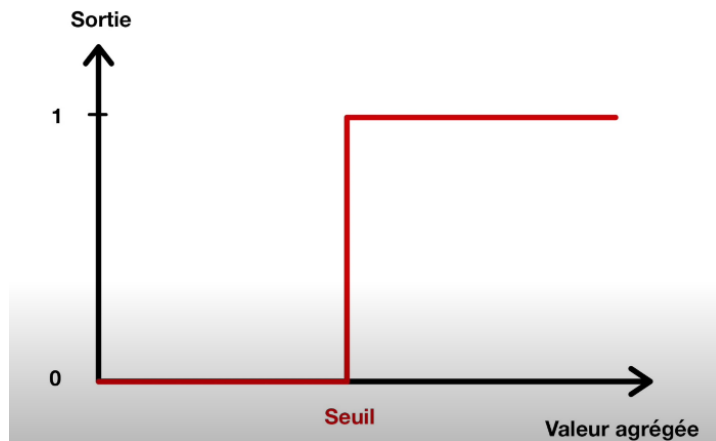


# Perceptron simple

## 2. Fonctionnement

### Fonction d'activation :

Le **biais** permet de décaler la fonction d'activation et représente le **seuil** à partir duquel le neurone sera considéré comme actif

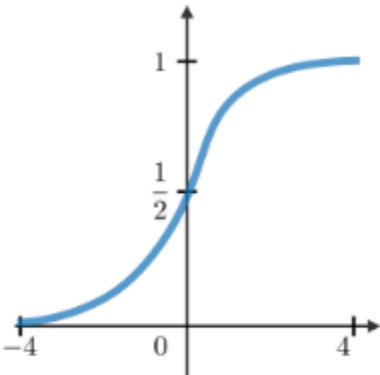
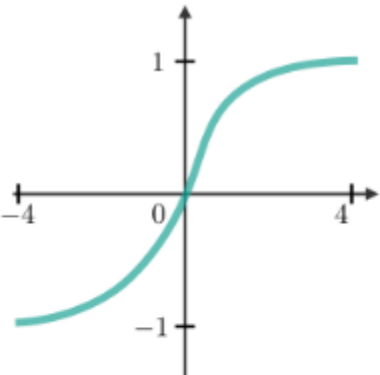
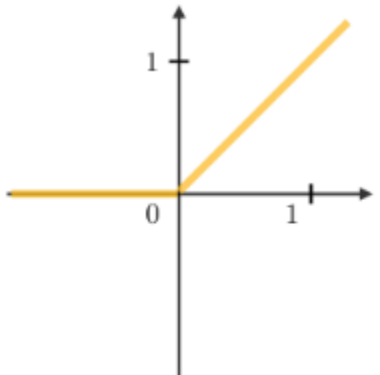
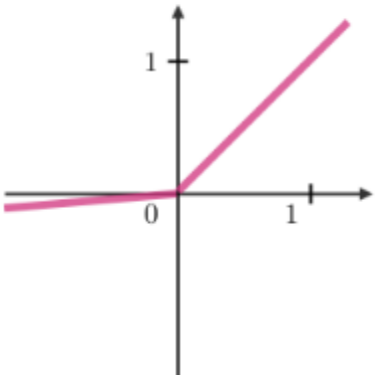


# Perceptron simple

## 2. Fonctionnement

### Fonction d'activation :

Voici les plus fréquentes

Sigmoïde	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

# Perceptron simple

---

## 2. Fonctionnement

### Fonction d'activation : Choix ???

#### 1- Output : en fonction du problème à résoudre

- Sigmoid si classification non-exclusive : par exemple 0 ou 1 avec plusieurs classes pouvant contenir 1
- Softmax si classification exclusive
- ReLU si nombre positif
- ...

#### 2- Pour les couches intermédiaires :

Conseil : privilégier ReLU, apprentissage plus rapide (calculs plus simples), moins de problème d'évanescence ou d'explosion du gradient

# Perceptron simple

---

## 2. Fonctionnement

L'ensemble de  $(W, b)$  : les poids et biais à apprendre (ou non, certains peuvent être fixes) sont appelés les **paramètres**

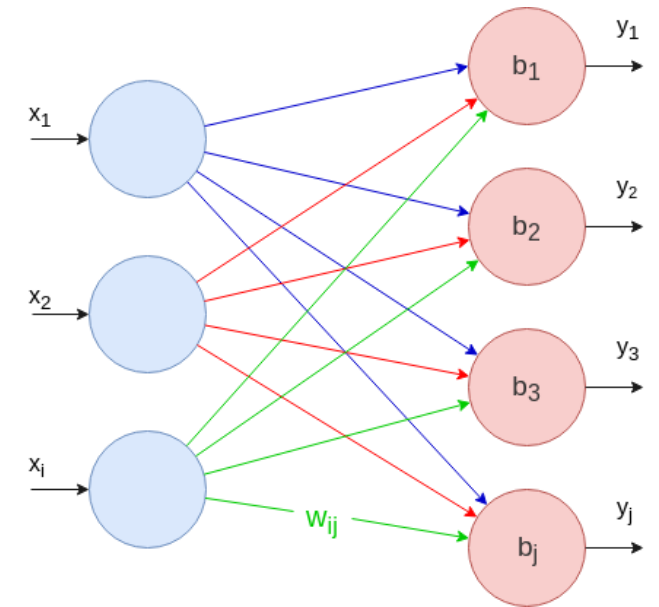
Tout le reste qui est fixé par l'utilisateur est appelé les **hyperparamètres** :

- Le nombre de couches
- Le nombre de neurones par couche
- Le type de couche
- Les fonctions d'activation
- Le taux d'apprentissage

# Couche de perceptron

---

## 1. Illustration graphique





# Couche de perceptron

## 2. Formulation

On a  $y_j = b_j + \sum_i x_i w_{ij}$

Avec le recours aux notations matricielle et vectorielle, on aura :

- Une formulation plus simple
- Code plus réduit

Soit alors :

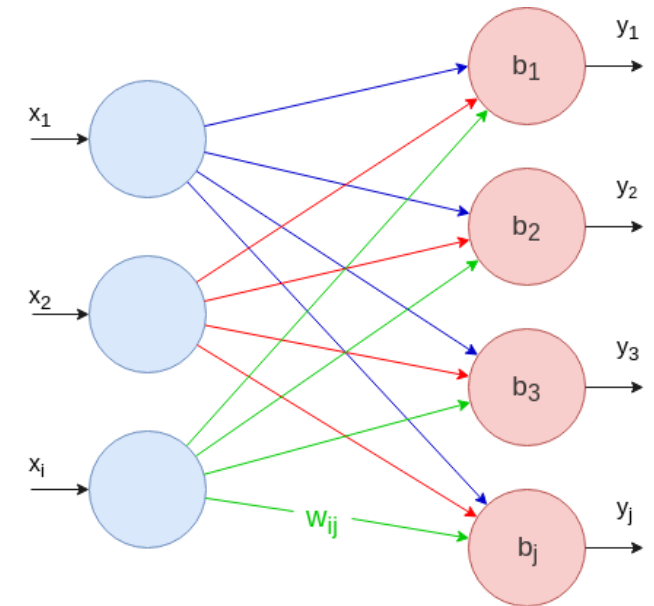
$$X = \begin{bmatrix} x_1 & \dots & x_i \end{bmatrix} \quad W = \begin{bmatrix} w_{11} & \dots & w_{1j} \\ \vdots & \ddots & \vdots \\ w_{i1} & \dots & w_{ij} \end{bmatrix} \quad B = \begin{bmatrix} b_1 & \dots & b_j \end{bmatrix}$$

Ainsi, on aura :

$$Y = XW + B$$

Mais quelle est l'utilité de la fonction d'activation réellement ?

Voyons de près l'origine :



# Perceptron multicouches

---

## 1. Problème de XOR

- La fonction XOR est l'opération booléenne qui correspond au non exclusif :

$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	0

- On cherche les poids  $w_1$  et  $w_2$  tels que l'équation :

$$w_1x_1 + w_2x_2 = y$$

soit vérifiée pour tous les exemples.

- On ne s'intéresse pas à la généralisation, on souhaite juste une erreur nulle sur l'ensemble d'apprentissage.

# Perceptron multicouches

---

## 1. Problème de XOR

- Cela revient à trouver une solution au système d'équation suivant :

$$w_1 \times 0 + w_2 \times 0 = 0$$

$$w_1 \times 0 + w_2 \times 1 = 1$$

$$w_1 \times 1 + w_2 \times 0 = 1$$

$$w_1 \times 1 + w_2 \times 1 = 0$$

- qui n'en admet pas !

$$w_2 = 1$$

$$w_1 = 1$$

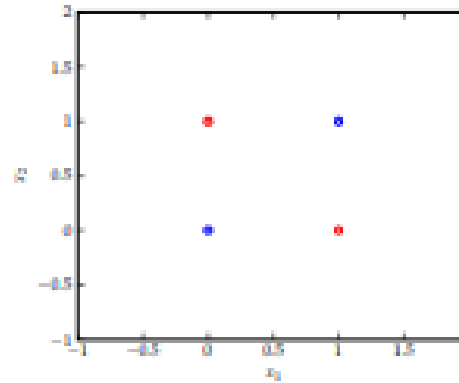
$$w_1 + w_2 = 0$$

# Perceptron multicouches

---

## 1. Problème de XOR

### Interprétation géométrique

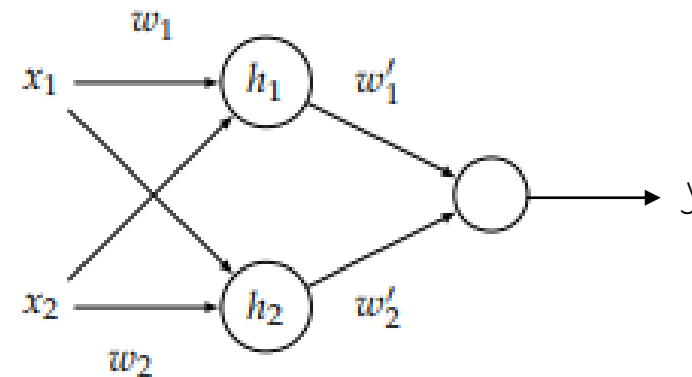


- On ne peut séparer les deux classes à l'aide d'une droite.
- Mais on pourrait appliquer une **transformation**  $\phi$  aux données de manière à rendre les données linéairement séparables.
- L'application de la transformation permet d'aboutir à une nouvelle **représentation** des données.
- $\phi$  ne peut pas être linéaire, cela ne permettrait pas de résoudre le problème.

# Perceptron multicouches

---

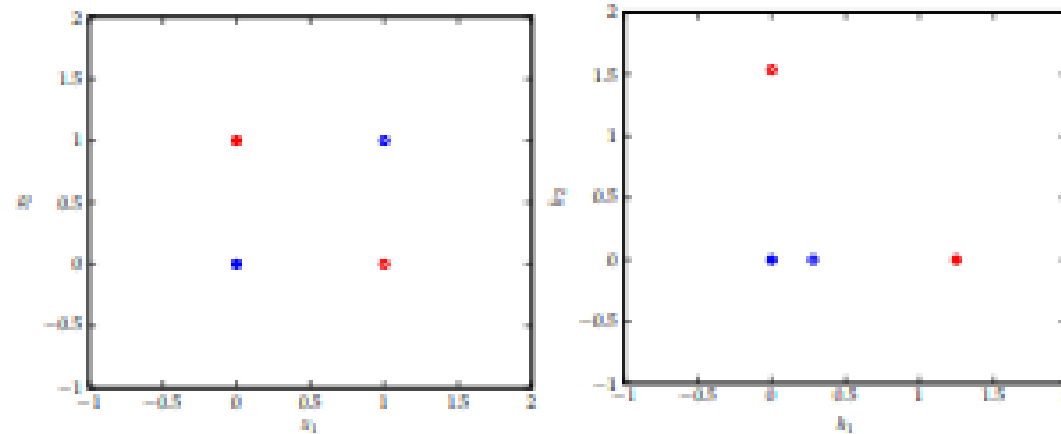
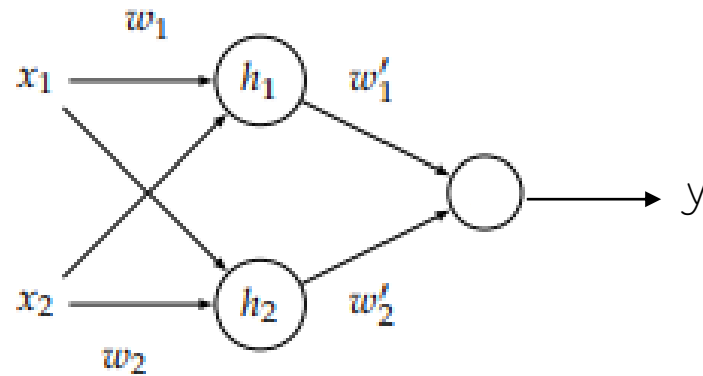
## 1. Problème de XOR



- Il est possible de résoudre le problème du XOR à l'aide d'un réseau simple
- La couche cachée ( $h_1, h_2$ ) permet de trouver une transformation des données
- La couche de sortie ( $y$ ) effectue une combinaison linéaire

# Perceptron multicouches

## 1. Problème de XOR



$$h_1 = \text{ReLU}(1.2484406x_1 - 1.5363349x_2 - 0.00033336)$$

$$h_2 = \text{ReLU}(-1.247532x_1 + 1.5365471x_2 - 0.00017454)$$

$x_1$	$x_2$	$h_1$	$h_2$	$y$
0	0	0	0	0.299
0	1	0	1,53	0.717
1	0	1.24	0	0.709
1	1	0.28	0	0.299

# Perceptron multicouches

## 1. Problème de XOR

### Implémentation

```
import numpy as np
from keras.models import Sequential
from keras.layers.core import Activation, Dense

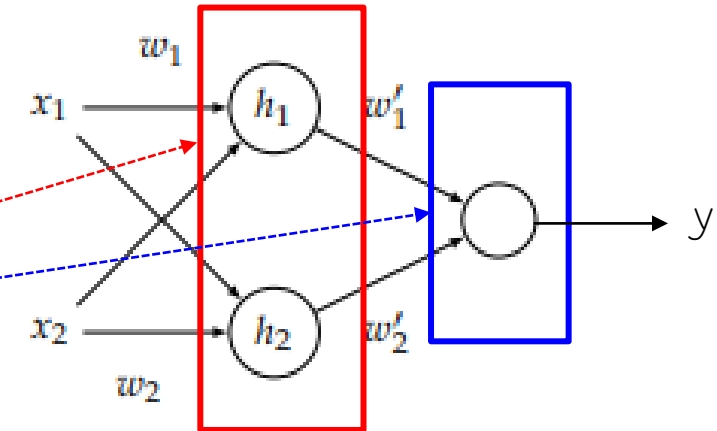
training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
target_data = np.array([[0],[1],[1],[0]], "float32")

model = Sequential()
model.add(Dense(2, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['binary_accuracy'])

model.fit(training_data, target_data, epochs=1000, verbose=2)

print(model.predict(training_data))
```

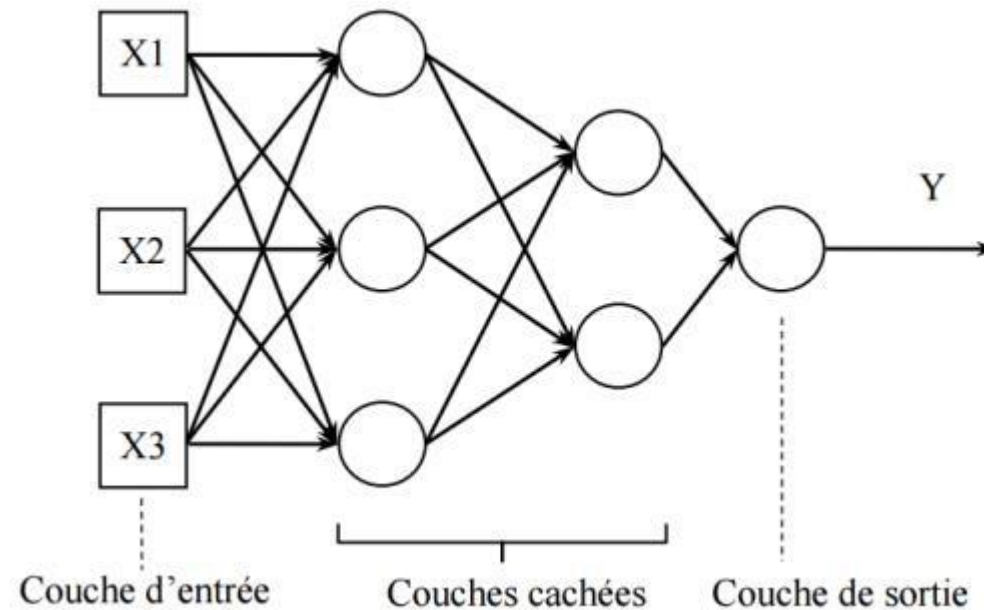


# Perceptron multicouches

---

## 2. Architecture d'un perceptron multicouches

### Multi Layer Perceptron (MLP)





**Premier RN**

# Construire son RN

## 1. Lab

<https://colab.research.google.com/drive/1VcgSBqlXOTG30xMwn7YfYcJBkvKPW1zz>

Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

# Construire son RN

## 2. Code

```
# Matplotlib
import matplotlib.pyplot as plt
# Tensorflow
import tensorflow as tf
# Numpy and Pandas
import numpy as np
import pandas as pd
# Other imports
import sys
```

```
from sklearn.preprocessing import StandardScaler

# Fashion MNIST
fashion_mnist = tf.keras.datasets.fashion_mnist
(images, targets), (images_test, targets_test) = fashion_mnist.load_data()

# Get only a subpart of the dataset
images = images[:10000]
targets = targets[:10000]

# Reshape the dataset and convert to float
images = images.reshape(-1, 784)
images = images.astype(float)
images_test = images_test.reshape(-1, 784)
images_test = images_test.astype(float)

scaler = StandardScaler()
images = scaler.fit_transform(images)
images_test = scaler.transform(images_test)

print(images.shape)
print(targets.shape)
```

```
targets_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal",
                 "Shirt", "Sneaker", "Bag", "Ankle boot"]
]
```

# Construire son RN

## 2. Code

```
# Flatten
model = tf.keras.models.Sequential()

# Add the layers
model.add(tf.keras.layers.Dense(256, activation="relu"))
model.add(tf.keras.layers.Dense(128, activation="relu"))
model.add(tf.keras.layers.Dense(10, activation="softmax"))

model_output = model.predict(images[0:1])
print(model_output, targets[0:1])
```

```
model.summary()
```



Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	200960
dense_1 (Dense)	(None, 128)	32896
dense_2 (Dense)	(None, 10)	1290

=====  
Total params: 235,146  
Trainable params: 235,146  
Non-trainable params: 0  
=====

Calcul ?

# Construire son RN

## 2. Code

```
# Compile the model
model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer="sgd",
    metrics=["accuracy"]
)
```

```
history = model.fit(images, targets, epochs=10, validation_split=0.2)
```

```
loss_curve = history.history["loss"]
acc_curve = history.history["accuracy"]
```

```
loss_val_curve = history.history["val_loss"]
acc_val_curve = history.history["val_accuracy"]
```

```
plt.plot(loss_curve, label="Train")
plt.plot(loss_val_curve, label="Val")
plt.legend(loc='upper left')
plt.title("Loss")
plt.show()
```

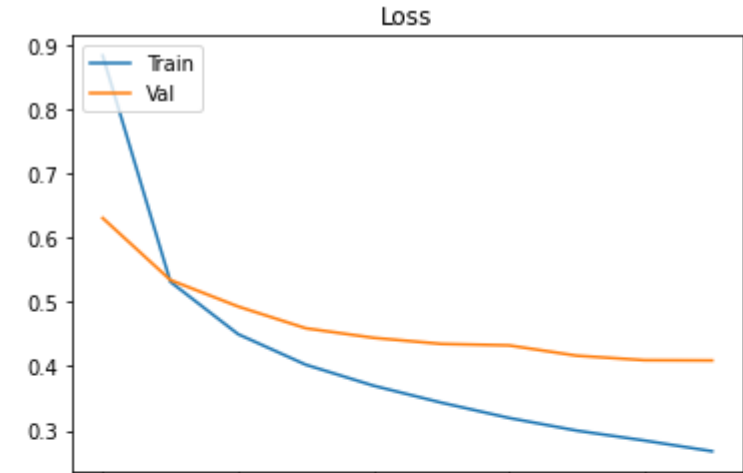
```
plt.plot(acc_curve, label="Train")
plt.plot(acc_val_curve, label="Val")
plt.legend(loc='upper left')
plt.title("Accuracy")
plt.show()
```

Données

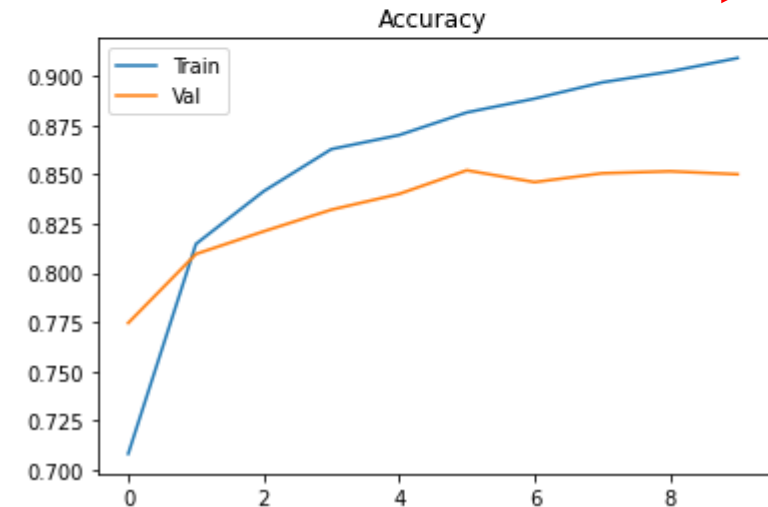
20%

Train

Validation



10 epochs



# Architectures

# Architectures des RN

## A mostly complete chart of Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

-  Backfed Input Cell
-  Input Cell
-  Noisy Input Cell
-  Hidden Cell
-  Probablistic Hidden Cell
-  Spiking Hidden Cell
-  Output Cell
-  Match Input Output Cell
-  Recurrent Cell
-  Memory Cell
-  Different Memory Cell
-  Kernel
-  Convolution or Pool

Perceptron (P)



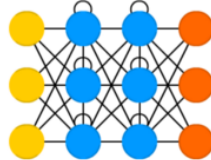
Feed Forward (FF)



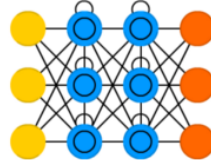
Radial Basis Network (RBF)



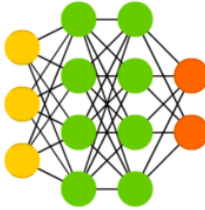
Recurrent Neural Network (RNN)



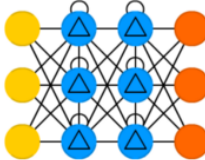
Long / Short Term Memory (LSTM)



Deep Feed Forward (DFF)



Gated Recurrent Unit (GRU)



Auto Encoder (AE)



Variational AE (VAE)



Denoising AE (DAE)



Sparse AE (SAE)



Markov Chain (MC)



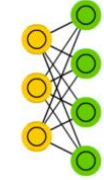
Hopfield Network (HN)



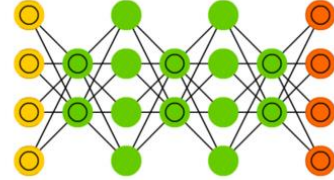
Boltzmann Machine (BM)



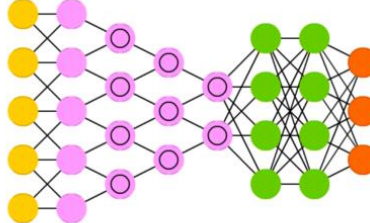
Restricted BM (RBM)



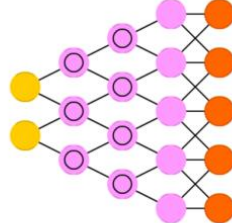
Deep Belief Network (DBN)



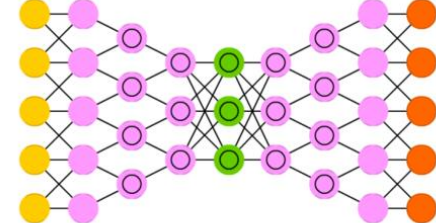
Deep Convolutional Network (DCN)



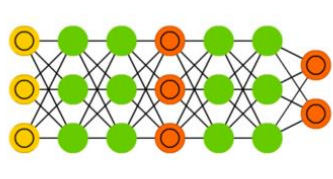
Deconvolutional Network (DN)



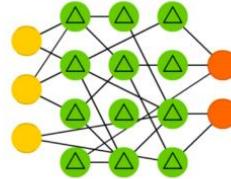
Deep Convolutional Inverse Graphics Network (DCIGN)



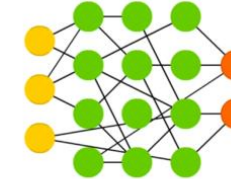
Generative Adversarial Network (GAN)



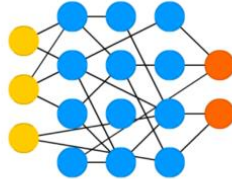
Liquid State Machine (LSM)



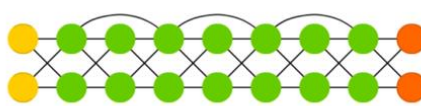
Extreme Learning Machine (ELM)



Echo State Network (ESN)



Deep Residual Network (DRN)



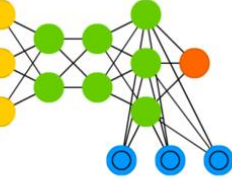
Kohonen Network (KN)



Support Vector Machine (SVM)



Neural Turing Machine (NTM)



# Gestion du RN



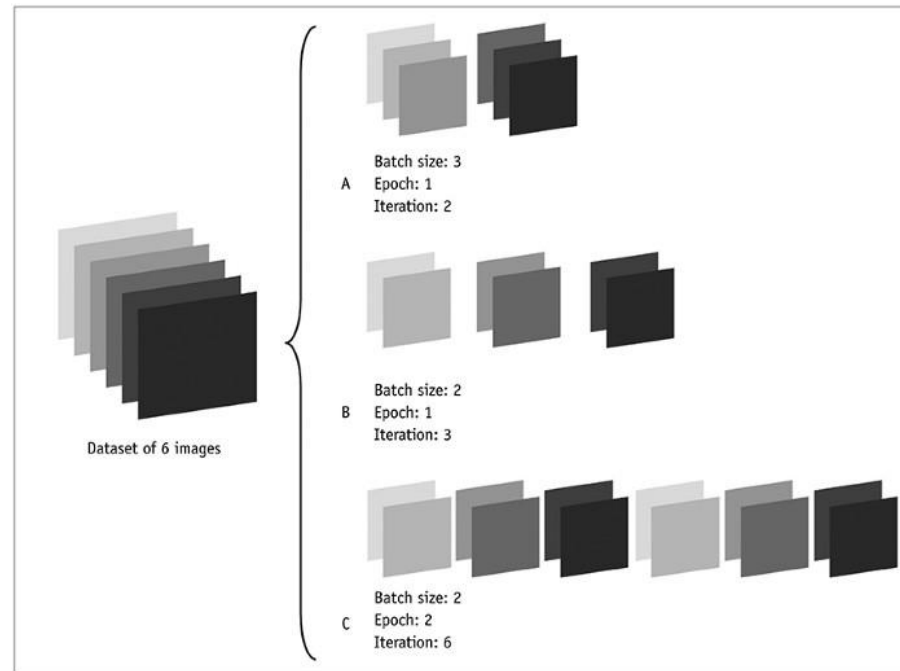
# Hyperparamètres

## 1. Exemple de base

Supposons que nous avons 2000 données pour l'apprentissage (Training)

Cet ensemble de données peut être divisé, par exemple, en lots (batches) de 500 éléments

Ceci dit qu'il faut 4 itérations pour terminer un jet d'apprentissage (epoch).



# Hyperparamètres

---

## 2. Choix

Donc pour l'apprentissage, on a les notions suivantes :

- Epoch : c'est lorsque l'ensemble de données ENTIER est transmis une seule fois en avant et en arrière via le RN
- Iteration : le nombre de lots nécessaires pour terminer une époque
- Batch : ensemble d'exemples d'apprentissage

# Hyperparamètres

---

## 2. Epoch, iteration & batch

Donc pour l'apprentissage, on a les hyperparamètres suivants :

- Epochs number : le nombre de répétitions/époques à faire pour diminuer le coût
- Iteration number : le nombre de lots nécessaires pour terminer une époque  
= nombre d'échantillons d'apprentissage en total / taille d'un lot
- Batch size : nombre total d'exemples d'apprentissage présents dans un seul lot

# Hyperparamètres

---

## 2. Epoch, iteration & batch

Comment choisir les valeurs de ces hyperparamètres ?

- Epochs number
- Iteration number
- Batch size

Lien pour le lab : voir l'impact de changement de la taille de (epoch & Batch) sur le temps et les performances du modèle

A tester pour différentes valeurs : epoch = 10, 100 et 1000, batch = 100, 200 et 300

[https://colab.research.google.com/drive/1ADY\\_JKuxFlDMLppkIEELJ1nfgOER5X0o#scrollTo=lRYHH\\_A3eeQB](https://colab.research.google.com/drive/1ADY_JKuxFlDMLppkIEELJ1nfgOER5X0o#scrollTo=lRYHH_A3eeQB)

# Hyperparamètres

---

## 2. Epoch, iteration & batch

Comment choisir les valeurs de ces hyperparamètres ?

- Epochs number
- Iteration number
- Batch size

Comment s'y prendre pour choisir les meilleures synergies ?

- Le principe serait de tester diverses combinaisons ; lent, lourd et il ne faut pas oublier !!!
- La solution simple, rapide et efficace est la GridSearchCV de Scikit-learn !

# Hyperparamètres

---

## 2. Epoch, iteration & batch

Comment choisir les valeurs de ces hyperparamètres ?

- Epochs number
- Iteration number      Tributaire du nombre d'échantillons et de la taille du lot (batch size) c,à.d. pas de choix
- Batch size

# Hyperparamètres

---

## 2. Epoch, iteration & batch

Comment choisir les valeurs de ces hyperparamètres ?

- Epoch number

# Hyperparamètres

---

## 2. Epoch, iteration & batch

Comment choisir les valeurs de ces hyperparamètres ?

- **Batch size**

**Petite taille** : ce choix peut être fait tenant compte des caractéristiques matérielles du système (RAM+GPU)

⇒ Chaque étape du GD peut être moins précise et donc l'algorithme de GD va prendre plus de temps pour converger

⇒ Plus de temps mais converge bien

**Grande taille** : ce choix peut être fait vu les caractéristiques matérielles du système (RAM+GPU)

⇒ Dégradation significative de la qualité du modèle en terme de pouvoir de généralisation

⇒ Très rapide mais converge mal



# Hyperparamètres

---

## 3. Fonction d'activation

Pourquoi ?

Le but des fonctions d'activation est d'ajouter des non-linéarités dans le RN

La non linéarité permet d'approximer des fonctions complexes (ce qui est souvent le cas dans le monde réel)

# Optimisation

---

## Modèles ML – Algorithme d'optimisation

Modèles linéaires – Descente de gradient

Arbre de décision – CART

SVM – Marge Maximale

# Optimisation

---

## Modèles ML – Algorithme d'optimisation

Modèles linéaires – Descente de gradient

Arbre de décision – CART

SVM – Marge Maximale

## DL = RN

**SGD** (Stochastic Gradient Descent)

**Momentum** : utilisée avec SGD. Au lieu d'utiliser uniquement le gradient de l'étape actuelle pour guider la recherche, Momentum accumule également le gradient des étapes précédentes pour déterminer la direction à suivre.

**Adam** (Adaptive Moment Estimation) probablement l'optimiseur le plus utilisé de nos jours. Elle combine les meilleures propriétés des techniques précédentes.

# Evaluation

---

## Fonction de perte/coût (Loss Function)

Comment choisir la fonction coût ?

### ▪ Régression

MSE (Mean Squared Error)

MAE (Mean Absolute Error)

MBE (Mean Bias Error)

### ▪ Classification

CCE (Categorical Cross Entropy) : La classe (Label) doit être encodée avec le one-hotencoding

BCE (Binary Cross Entropy) : si le nombre de classes = 2

- Cross-Entropy = 0.00: Perfect Match.
- Cross-Entropy < 0.02: Great match.
- Cross-Entropy < 0.05: Good.
- Cross-Entropy < 0.20: Acceptable.
- Cross-Entropy > 0.30: Not good.
- Cross-Entropy > 1.00: Bad.
- Cross-Entropy > 2.00 Horrible.

# Evaluation

---

## Fonction de perte/coût (Loss Function)

Comment choisir la fonction coût ?

- **Classification**

BCE (Binary Cross Entropy)

Fonction la plus connue, définissant comme mesure de la différence entre deux probabilité de distribution pour une variable prise aléatoirement.

### Quand l'utiliser ?

Utilisée pour de la classification binaire, ou classification multi-classes où l'on souhaite plusieurs label en sortie.

Utilisable aussi pour de la segmentation sémantique. Fonction basée sur la distribution de Bernoulli.

Référence de base à utiliser comme première fonction pour tout nouveau réseau.

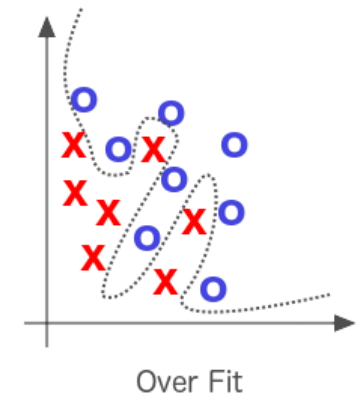
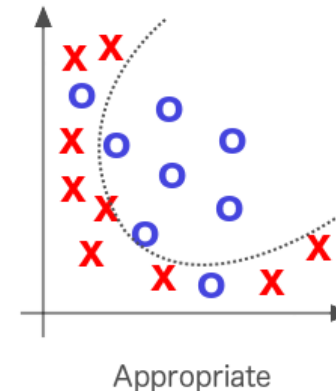
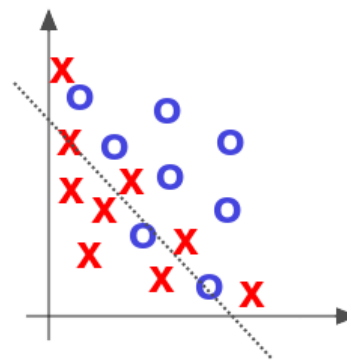
Non adaptée à des datasets biaisées/déséquilibrées.

# Régularisation du modèle

## 1. Problèmes d'apprentissage

Les problèmes d'entraînement sont principalement:

- **Surapprentissage (Overfitting)** : il se produit lorsqu'un modèle apprend les détails et le bruit dans les données d'apprentissage dans la mesure où cela a un impact négatif sur les performances du modèle sur de nouvelles données.
- **Sous-apprentissage (Underfitting)** : Il fait référence à un modèle qui ne peut ni modéliser les données d'entraînement ni être généralisé pour des nouvelles données.



# Régularisation du modèle

---

## 2. Solutions

Comme solution pour :

- **Sous-apprentissage (Underfitting)**
  - Augmenter le nombre d'échantillons
  - Augmenter le nombre d'epochs
  - ...

# Régularisation du modèle

## 2. Solutions

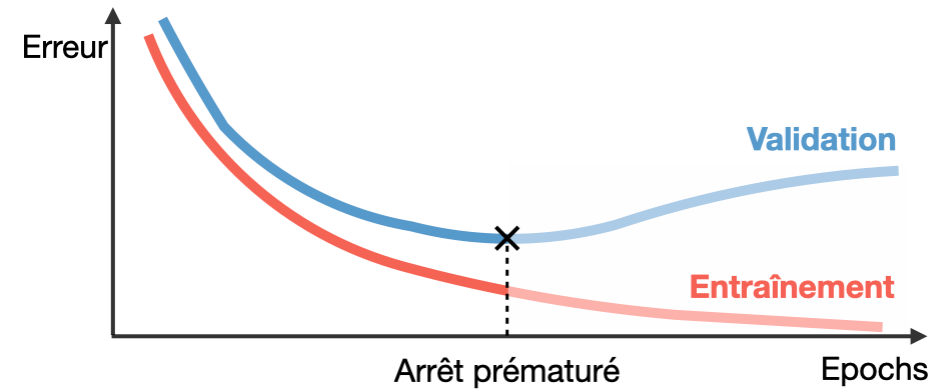
Comme solution pour :

- **Surapprentissage (Overfitting)**

### Early Stopping (Arrêt précoce):

Un trop grand nombre d'époques peut entraîner un surajustement de l'ensemble de données d'entraînement, alors que trop peu peut entraîner un sous-ajustement du modèle.

L'arrêt précoce est une méthode qui vous permet de spécifier un grand nombre arbitraire d'époques d'entraînement et d'arrêter l'entraînement une fois que les performances du modèle ne s'améliorent plus.





# Régularisation du modèle

---

## 2. Solutions

Comme solution pour :

- **Surapprentissage (Overfitting)**

**Early Stopping (Arrêt précoce):**

```
from keras.callbacks import EarlyStopping # Set callback functions to early stop training
mycallbacks = [EarlyStopping(monitor='val_loss', patience=2)]
history = model.fit(
    train_images, #inputs
    to_categorical(train_labels), #target vector
    epochs=5, # number of epochs
    batch_size=32,
    callbacks = mycallbacks, # early stopping
    validation_data=(test_images, to_categorical(test_labels)))
```

# Régularisation du modèle

---

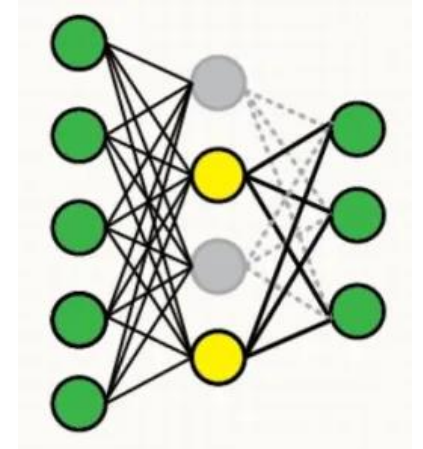
## 2. Solutions

Comme solution pour :

- **Surapprentissage (Overfitting)**

### Dropout (Abandon):

Il consiste à désactiver certains neurones du modèle, et ce de façon aléatoire d'une même couche, qui ne contribuera donc ni à la phase de propagation, ni à la phase de rétropropagation. D'un point de vue du réseau, cela revient à instancier la valeur en sortie d'une fonction d'activation à 0,



# Régularisation du modèle

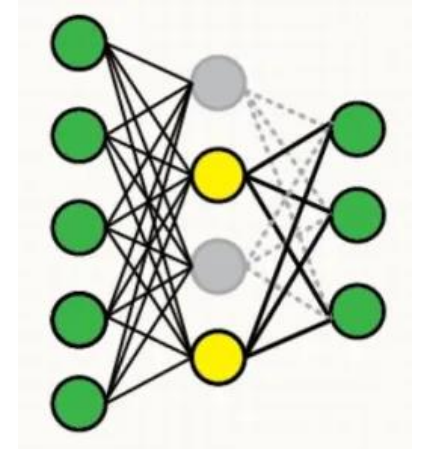
## 2. Solutions

Comme solution pour :

- **Surapprentissage (Overfitting)**

**Dropout (Abandon):**

```
#Build the new model. .  
model = Sequential([  
    Dense(64, activation='relu', input_shape=(784,)),  
    Dropout(0.5),  
    Dense(64, activation='relu'),  
    Dropout(0.5),  
    Dense(10, activation='softmax')])
```



# Régularisation du modèle

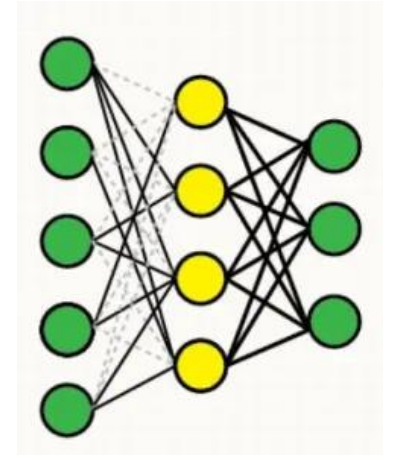
## 2. Solutions

Comme solution pour :

- **Surapprentissage (Overfitting)**

### DropoutConnect:

On va reprendre le même principe que précédemment. Mais au lieu de désactiver des neurones, on va simplement désactiver les connexions entrantes (toujours de façon aléatoire) sur une couche depuis la précédente. D'un point de vue du réseau, cela revient à instancier les valeurs des poids des connexions à 0.



# Régularisation du modèle

## 2. Solutions

- **Surapprentissage (Overfitting)**

D'autres techniques existent et qui agissent sur les 1

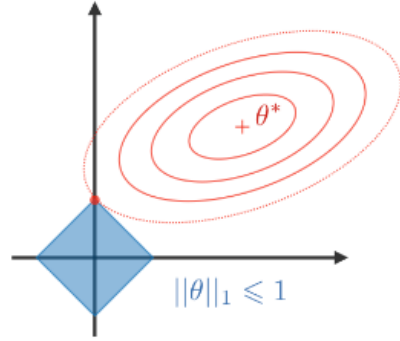
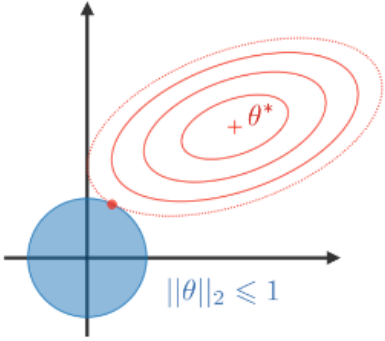
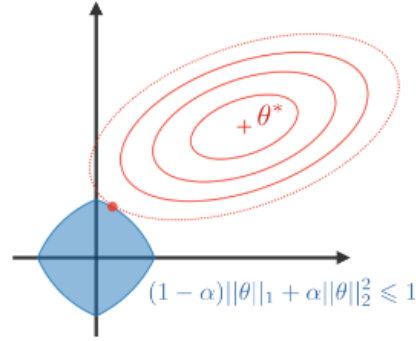
**LASSO regression**

**Ridge regression**

**Elastic Net**

**Max-Norm regularization**

...

LASSO	Ridge	Elastic Net
<ul style="list-style-type: none"><li>• Réduit les coefficients à 0</li><li>• Bon pour la sélection de variables</li></ul>	Rend les coefficients plus petits	Compromis entre la sélection de variables et la réduction de la taille des coefficients
		
$\dots + \lambda \ \theta\ _1$ $\lambda \in \mathbb{R}$	$\dots + \lambda \ \theta\ _2^2$ $\lambda \in \mathbb{R}$	$\dots + \lambda \left[ (1 - \alpha) \ \theta\ _1 + \alpha \ \theta\ _2^2 \right]$ $\lambda \in \mathbb{R}, \alpha \in [0, 1]$

# Etude des cas

## a) Cas de CNN

# Origine de CNN

---

**A voir pour les curieux !**



ScienceEtonnante ✓

1,15 M d'abonnés

<https://www.youtube.com/watch?v=trWrEWfhTVg>

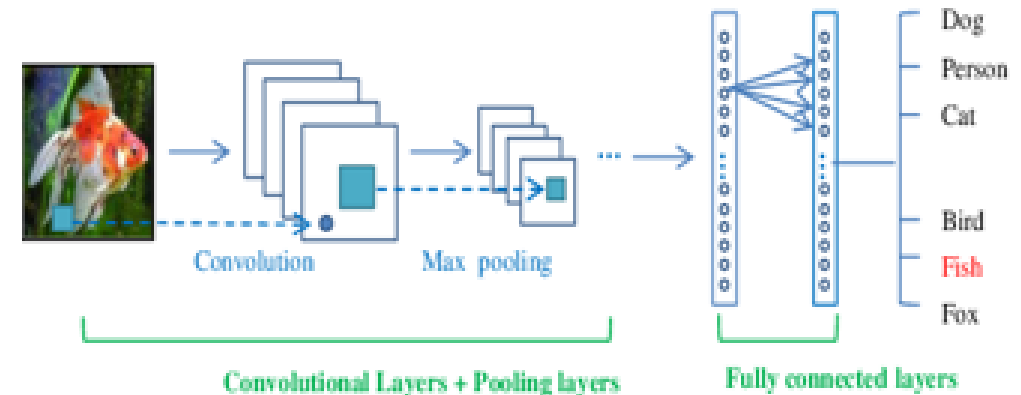


# Couches de CNN

## 1. Architecture de base

Le réseau de CNN (**Convolutional Neural Network**) repose sur les principales couches suivantes :

- **Convolution** (CONV) : extraction des caractéristiques et application des filtres
- **Pooling** (POOL) : réduction de la dimension et préserver les principales caractéristiques
- **Fully connected** (FC) : construction des connexions nécessaires



Design of Convolution Neural Network

# Couches de CNN

## 2. Convolution layer

**Objectif** : extraction des caractéristiques (features)  
et application des filtres

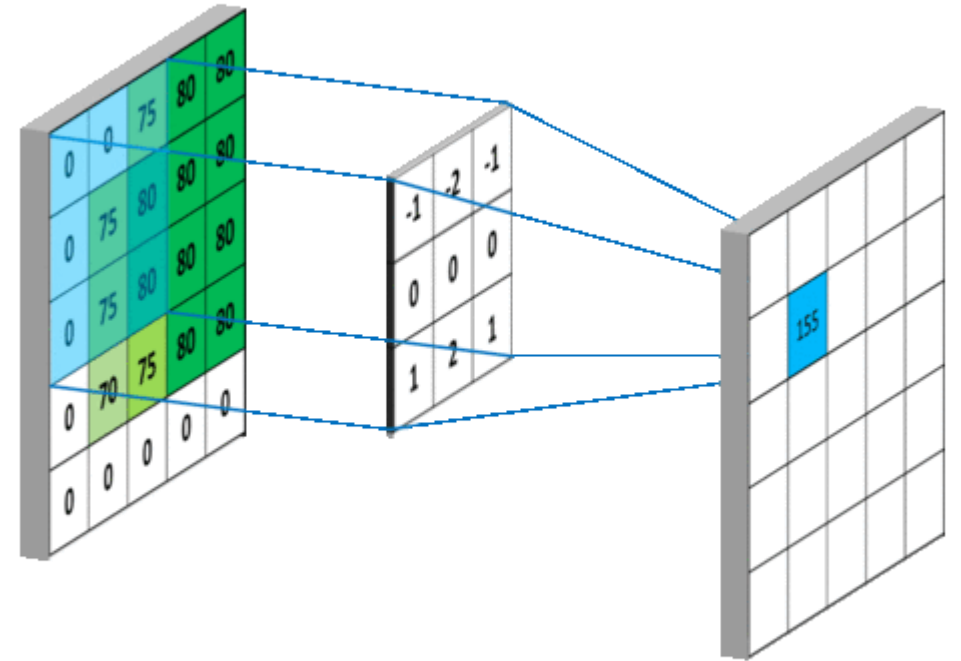
**Principe** : filtrage par convolution

=

faire "glisser" une fenêtre représentant le filtre sur l'image

+

calculer le produit de convolution entre le filtre et chaque portion  
de l'image balayée

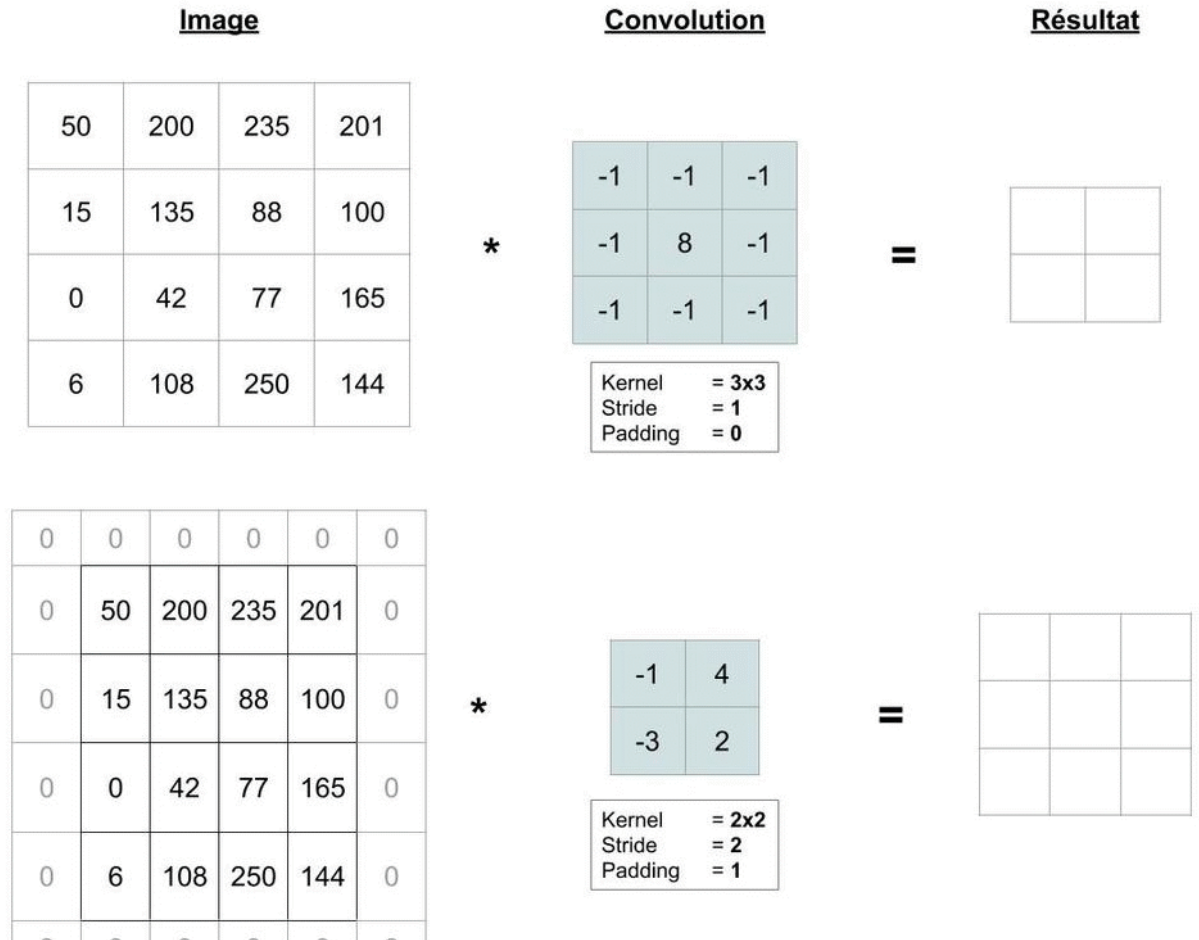


# Couches de CNN

## 2. Convolution layer

### Paramètres

- Kernel (F) : taille de la fenêtre du Filtre
- Stride (S) : pas de déplacement du filtre
- Padding (P) : inclusion ou non des frontières de l'image



# Couches de CNN

---

## 2. Convolution layer

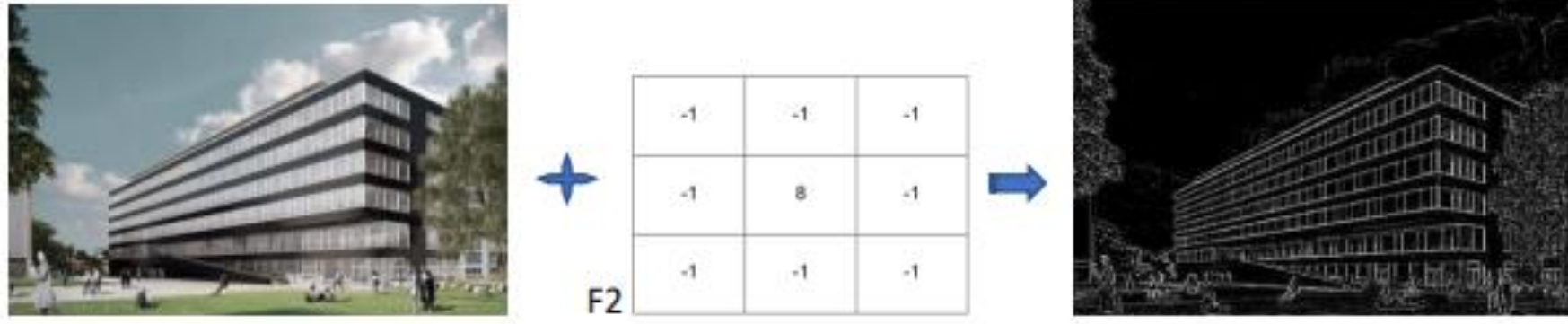
### Fonction d'activation

- Pour réduire le calcul
- Pour éviter le problème de linéarité : Si la fonction d'activation n'est pas appliquée, la fonction créée sera linéaire et le problème de XOR persiste.
- ReLu est très utilisée dans les réseaux de neurones à convolution car il s'agit d'une fonction rapide à calculer :  $f(y) = \max(0, y)$ . Sa performance est donc meilleure que d'autres fonctions où des opérations coûteuses doivent être effectuées.

# Couches de CNN

## 2. Convolution layer

Exemple de résultat



On obtient pour chaque paire (**image**, **filtre**) une carte d'activation (**feature map**), qui nous indique où se situent les features dans l'image (plus la valeur est élevée, plus l'endroit correspondant dans l'image ressemble à la feature)

# Couches de CNN

---

## 2. Convolution layer

### Choix des Features/Filtres

- Contrairement aux méthodes traditionnelles, les features ne sont pas pré-définies selon un formalisme particulier, mais **appries** par le réseau lors la phase **d'entraînement** !
- Les noyaux des filtres désignent les poids de la couche de convolution. Ils sont initialisés puis mis à jour par la méthode de rétropropagation du gradient.
- C'est là toute la force des réseaux de neurones convolutifs : ceux-ci sont capables de déterminer tout seul les éléments discriminants d'une image, en s'adaptant au problème posé.

# Couches de CNN

---

## 3. Pooling layer

### Principe

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

**Pooling** : réduire la taille des images, tout en préservant leurs caractéristiques importantes.

Ceci peut être fait avec :

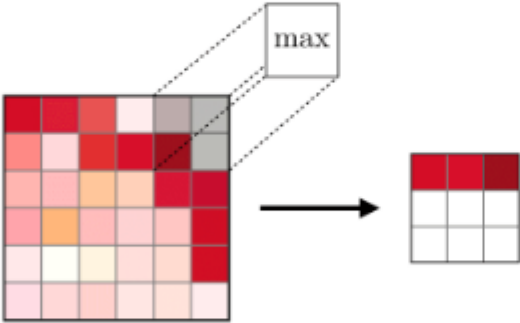
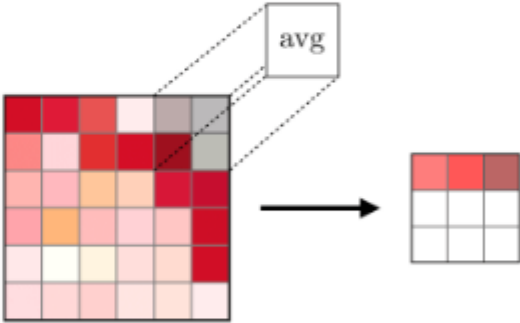
- Max pooling : la plus utilisée, elle prend le max des valeurs
- Average pooling : elle prend la moyenne des valeurs

**Remarque** : l'opération de pooling est généralement faite indépendamment sur chaque carte d'activation

# Couches de CNN

## 3. Pooling layer

### Pooling

Type	Max pooling	Average pooling
But	Chaque opération de pooling sélectionne la valeur maximale de la surface	Chaque opération de pooling sélectionne la valeur moyenne de la surface
Illustration		
Commentaires	<ul style="list-style-type: none"><li>• Garde les caractéristiques détectées</li><li>• Plus communément utilisé</li></ul>	<ul style="list-style-type: none"><li>• Sous-échantillonne la <i>feature map</i></li><li>• Utilisé dans LeNet</li></ul>



# Couches de CNN

## 3. Pooling layer

### Paramètres

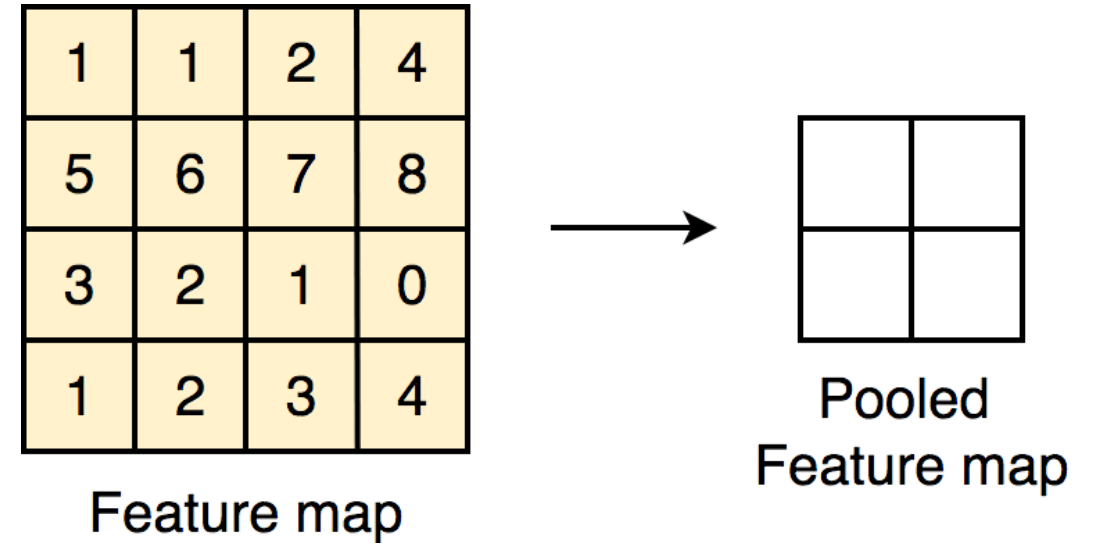
- F : la taille de la fenêtre
- S : pas de déplacement

### Max Pooling

Dans ce cas, on prend la valeur maximale dans une fenêtre de taille F de la feature map, et puis on passe de S pas pour regarder la nouvelle fenêtre

### Remarque

La taille de la fenêtre de pool est dans la majorité des cas (2x2) avec un pas de 2 chose qui permet de réduire la taille de la carte d'activation par 2 (la taille de pooled feature map sera alors la moitié de la taille de feature map)



# Couches de CNN

---

## 3. Pooling layer

### Avantages

- Réduire la **résolution** des cartes d'activation. On améliore ainsi l'efficacité du réseau et on évite le sur-apprentissage.
- Consommation de moins de **mémoire** et moins de temps de **calcul**.
- Augmente la **robustesse** du réseau à la rotation et aux changements d'échelle. En effet, les valeurs maximales sont repérées de manière moins exacte dans les feature maps obtenues après le pooling, ainsi, la couche de pooling rend le réseau moins sensible à la position des features : le fait qu'une feature se situe un peu plus en haut ou en bas, ou même qu'elle ait une orientation légèrement différente ne devrait pas provoquer un changement radical dans la classification de l'image.

# Couches de CNN

---

## 4. Fully connected layer

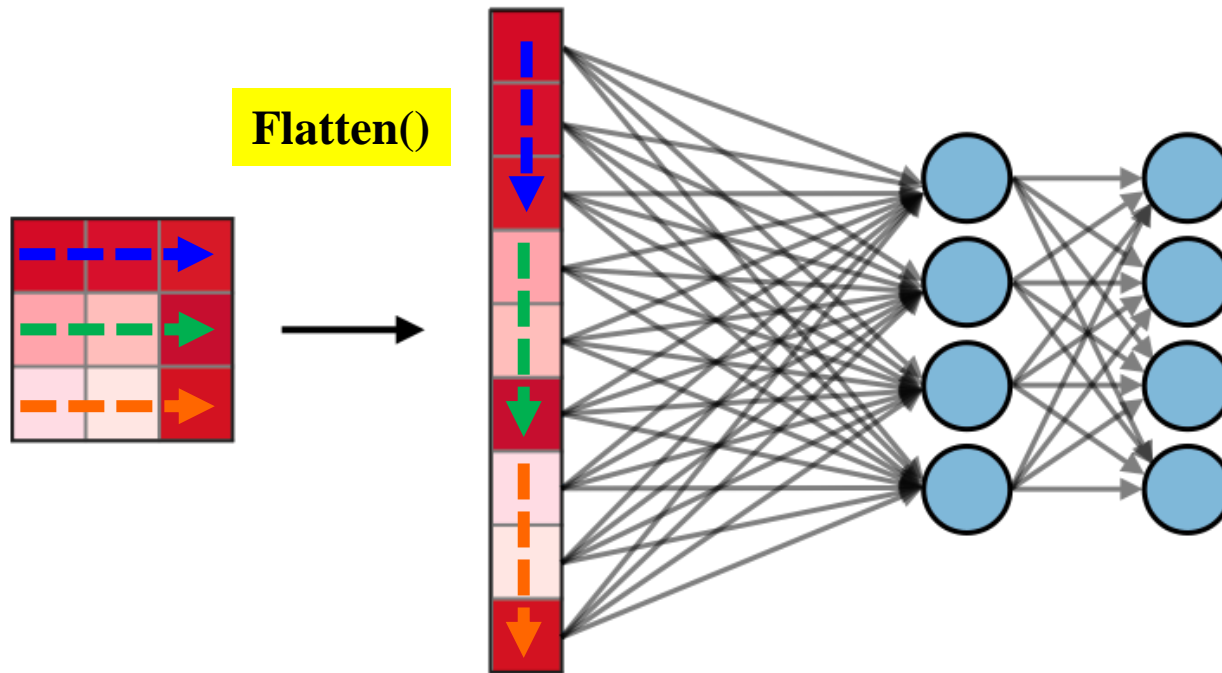
- Cette couche apparaît à la fin du RN (créé avec **Dense()**).
- Elle reçoit un vecteur en entrée et produit un nouveau vecteur en sortie (via la fonction **Flatten()**). Pour cela, elle applique une combinaison linéaire puis éventuellement une fonction d'activation aux valeurs reçues en entrée.
- Elle permet de classer l'image en entrée du réseau : elle renvoie un vecteur de taille N (N étant le nombre de classes).
- Chaque élément du vecteur indique la probabilité pour l'image en entrée d'appartenir à une classe.
- Pour calculer les probabilités d'appartenance à une classe, la couche fully-connected multiplie donc chaque élément en entrée par un **poids**, fait la **somme**, puis applique une fonction **d'activation** (logistique si N=2, sinon softmax).

### Remarque : Comment connaît-on les valeurs de ces poids ?

Le CNN apprend les valeurs des poids de la même manière qu'il apprend les filtres de la couche de convolution : lors de phase d'entraînement, par rétropropagation du gradient.

# Couches de CNN

## 4. Fully connected layer



# Couches de CNN

## 5. Image en couleur RGB

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

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

Kernel Channel #1



308

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

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

Kernel Channel #2



-498

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



164

+

+

+ 1 = -25

↑  
Bias = 1

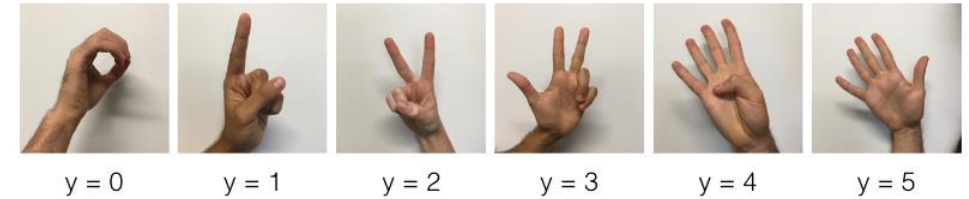
-25				...
				...
				...
				...
...	...	...	...	...

# Exemple

---

## 1. Lab

Classification des images main/chiffre



<https://colab.research.google.com/drive/1RlWnceQNfaWlzpWcjxf9RZ6sORs2Zmab>

# Exemple

---

## 2. Classification des images main/chiffre

```
import math
import numpy as np
import h5py
import matplotlib.pyplot as plt
import scipy
from PIL import Image
from scipy import ndimage
import tensorflow as tf
from tensorflow.python.framework import ops

import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
```

# Exemple

---

## 2. Classification des images main/chiffre

```
train_dataset = h5py.File('/content/gdrive/MyDrive/tp_1/data/train_signs.h5', "r")
train_set_x_orig = np.array(train_dataset["train_set_x"][:])
train_set_y_orig = np.array(train_dataset["train_set_y"][:])

test_dataset = h5py.File('/content/gdrive/MyDrive/tp_1/data/test_signs.h5', "r")
test_set_x_orig = np.array(test_dataset["test_set_x"][:])
test_set_y_orig = np.array(test_dataset["test_set_y"][:])

classes = np.array(test_dataset["list_classes"][:])

train_set_y_orig = train_set_y_orig.reshape((1, train_set_y_orig.shape[0]))
test_set_y_orig = test_set_y_orig.reshape((1, test_set_y_orig.shape[0]))

input_shape = train_set_x_orig[0].shape
```

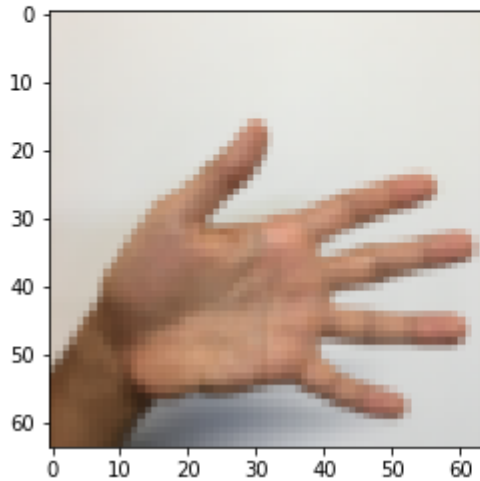


# Exemple

## 2. Classification des images main/chiffre

```
# Example of a picture
index = 0
plt.imshow(train_set_x_orig[index])
print ("y = " + str(np.squeeze(train_set_y_orig[:, index])))
```

y = 5



```
print(train_set_x_orig[0].shape)
```

(64, 64, 3)



=

Blue				
Green				
	123	94	83	2
Red	123	94	83	4
				30
123	94	83	2	92
				124
34	44	187	92	4
				142
34	76	232	124	4
67	83	194	202	

# Exemple

---

## 2. Classification des images main/chiffre

```
print('Valeur maximum de nos images : ', np.min(train_set_x_orig))
print('Valeur minimum de nos images : ', np.max(train_set_x_orig))

X_train = train_set_x_orig.astype('float32')/255
X_test = test_set_x_orig.astype('float32')/255

print('Valeur maximum de nos images normalisées : ', np.min(X_train))
print('Valeur minimum de nos images normalisées : ', np.max(X_train))

Valeur maximum de nos images :  4
Valeur minimum de nos images :  244
Valeur maximum de nos images normalisées :  0.015686275
Valeur minimum de nos images normalisées :  0.95686275

print(train_set_y_orig.shape)
print(test_set_y_orig.shape)

(1, 1080)
(1, 120)

Y_train = np.eye(6)[train_set_y_orig.reshape(-1)]
Y_test = np.eye(6)[test_set_y_orig.reshape(-1)]
print(Y_train[150, :])
print((train_set_y_orig[:, 150]))

[0. 0. 0. 1. 0. 0.]
[3]
```

# Exemple

## 2. Classification des images main/chiffre

```
model = Sequential()
```

← Architecture séquentielle

```
model.add(Conv2D(32,  
                kernel_size=(3, 3),  
                activation='relu',  
                input_shape=(64,64,3)))  
model.add(Conv2D(32,  
                kernel_size=(3, 3),  
                activation='relu',  
                input_shape=(64,64,3)))
```

← 2 Couches de convolution

```
model.add(MaxPooling2D(pool_size=(2, 2)))
```

← Couche Max pooling

```
model.add(Flatten())
```

← Transformation en vecteur

```
model.add(Dense(6, activation='softmax'))
```

← Couche Fully connected

```
model.summary()
```

# Exemple

## 2. Classification des images main/chiffre

```
model = Sequential()
```

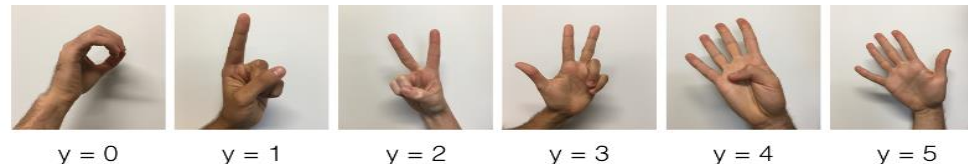
```
model.add(Conv2D(32, ← Nombre de neurones pour cette couche de convolution  
                 kernel_size=(3, 3), ← Forme du Filtre (3x3)  
                 activation='relu',  
                 input_shape=(64,64,3))) ← Image 64x64 pixels avec 3 niveaux de couleurs (RGB)  
model.add(Conv2D(32,  
                 kernel_size=(3, 3),  
                 activation='relu', ← Fonction d'activation à appliquer au niveau de cette couche de convolution  
                 input_shape=(64,64,3)))
```

```
model.add(MaxPooling2D(pool_size=(2, 2))) ← Fenêtre de pool de taille (2x2)
```

```
model.add(Flatten())
```

```
model.add(Dense(6, activation='softmax')) ← 6 neurones puisqu'on a 6 classes en sortie (chiffre 0, 1, ..., ou 5)
```

```
model.summary()
```



# Exemple

## 2. Classification des images main/chiffre

```
model = Sequential()

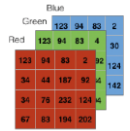
model.add(Conv2D(32,
                 kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(64,64,3)))
model.add(Conv2D(32,
                 kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(64,64,3)))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(6, activation='softmax'))

model.summary()
```



Model: "sequential"

Forme ?

Calcul ?

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 62, 62, 32)	896
conv2d_1 (Conv2D)	(None, 60, 60, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 30, 30, 32)	0
flatten (Flatten)	(None, 28800)	0
dense (Dense)	(None, 6)	172806
Total params: 182,950		
Trainable params: 182,950		
Non-trainable params: 0		

# Exemple

## 2. Classification des images main/chiffre

Entraînement de notre modèle

```
model.compile(loss="categorical_crossentropy", optimizer=SGD(lr=0.01), metrics=["accuracy"])
model.fit(X_train, Y_train,
        epochs = 100,
        validation_data=(X_test, Y_test))
```

Evaluation de la partie  
Train  
Loss Function | Accuracy

Evaluation de la partie  
Validation  
Loss Function | Accuracy

Epoch 1/100			
34/34 [=====]	- 31s 26ms/step	- loss: 1.8081 - accuracy: 0.1917	- val_loss: 1.7707 - val_accuracy: 0.1667
Epoch 2/100			
34/34 [=====]	- 1s 15ms/step	- loss: 1.7591 - accuracy: 0.2509	- val_loss: 1.7347 - val_accuracy: 0.3667
Epoch 3/100			
34/34 [=====]	- 1s 15ms/step	- loss: 1.7227 - accuracy: 0.2824	- val_loss: 1.7173 - val_accuracy: 0.1750
Epoch 4/100			
34/34 [=====]	- 1s 15ms/step	- loss: 1.6625 - accuracy: 0.3491	- val_loss: 1.5693 - val_accuracy: 0.5000
Epoch 5/100			
34/34 [=====]	- 0s 15ms/step	- loss: 1.5538 - accuracy: 0.4130	- val_loss: 1.4910 - val_accuracy: 0.3833
Epoch 6/100			
34/34 [=====]	- 1s 16ms/step	- loss: 1.4501 - accuracy: 0.4380	- val_loss: 1.3293 - val_accuracy: 0.5417
Epoch 7/100			
34/34 [=====]	- 1s 16ms/step	- loss: 1.3621 - accuracy: 0.4833	- val_loss: 1.3385 - val_accuracy: 0.4583
Epoch 8/100			
34/34 [=====]	- 1s 15ms/step	- loss: 1.1941 - accuracy: 0.5574	- val_loss: 1.1238 - val_accuracy: 0.6083
Epoch 9/100			
34/34 [=====]	- 1s 15ms/step	- loss: 1.0942 - accuracy: 0.6009	- val_loss: 1.0754 - val_accuracy: 0.5417
Epoch 10/100			
34/34 [=====]	- 1s 16ms/step	- loss: 0.9265 - accuracy: 0.6889	- val_loss: 1.1072 - val_accuracy: 0.6500
Epoch 11/100			
34/34 [=====]	- 1s 15ms/step	- loss: 0.9525 - accuracy: 0.6852	- val_loss: 0.9552 - val_accuracy: 0.5917
Epoch 12/100			
34/34 [=====]	- 1s 15ms/step	- loss: 0.7814 - accuracy: 0.7361	- val_loss: 1.2600 - val_accuracy: 0.5750
Epoch 13/100			
34/34 [=====]	- 1s 16ms/step	- loss: 0.7365 - accuracy: 0.7463	- val_loss: 0.7778 - val_accuracy: 0.7083
Epoch 14/100			
34/34 [=====]	- 1s 16ms/step	- loss: 0.7349 - accuracy: 0.7454	- val_loss: 0.8299 - val_accuracy: 0.6917
Epoch 15/100			
34/34 [=====]	- 1s 16ms/step	- loss: 0.6408 - accuracy: 0.7880	- val_loss: 0.7888 - val_accuracy: 0.7083
Epoch 16/100			
34/34 [=====]	- 1s 15ms/step	- loss: 0.5870 - accuracy: 0.8194	- val_loss: 0.6510 - val_accuracy: 0.8167

# Exemple

---

## 2. Classification des images main/chiffre

Pour finir ce notebook, nous allons sauvegarder les paramètres de notre modèle si nous voulons l'utiliser à nouveau plus tard.

```
model.save_weights("model.h5")
```

## **b) Cas de RNN**