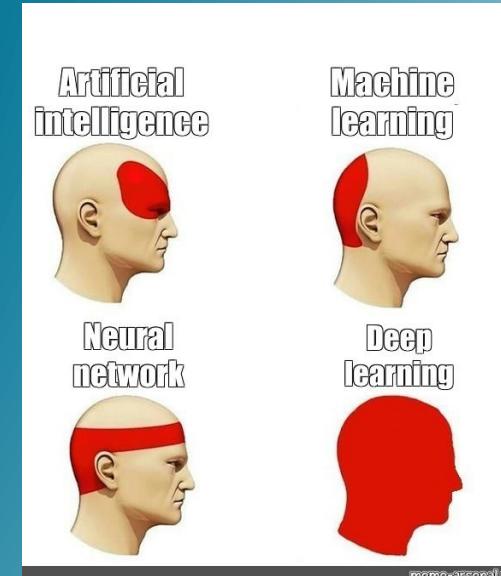


Méthodes Numériques et Modélisation

Fondements de l'intelligence artificielle et
applications aux sciences atmosphériques

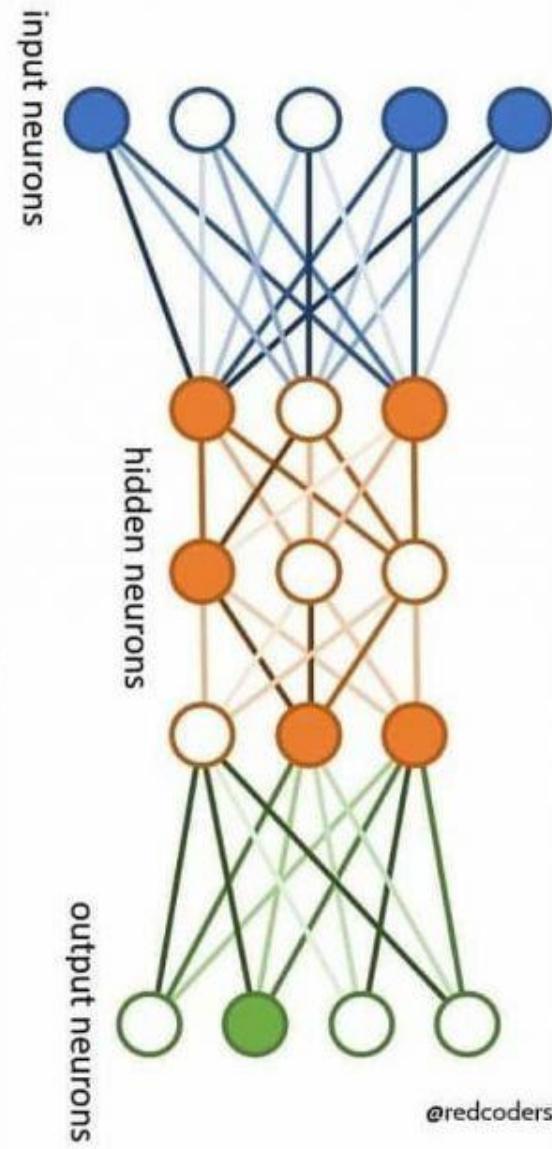


Les Réseaux de Neurones

**THIS IS A NEURAL
NETWORK.**

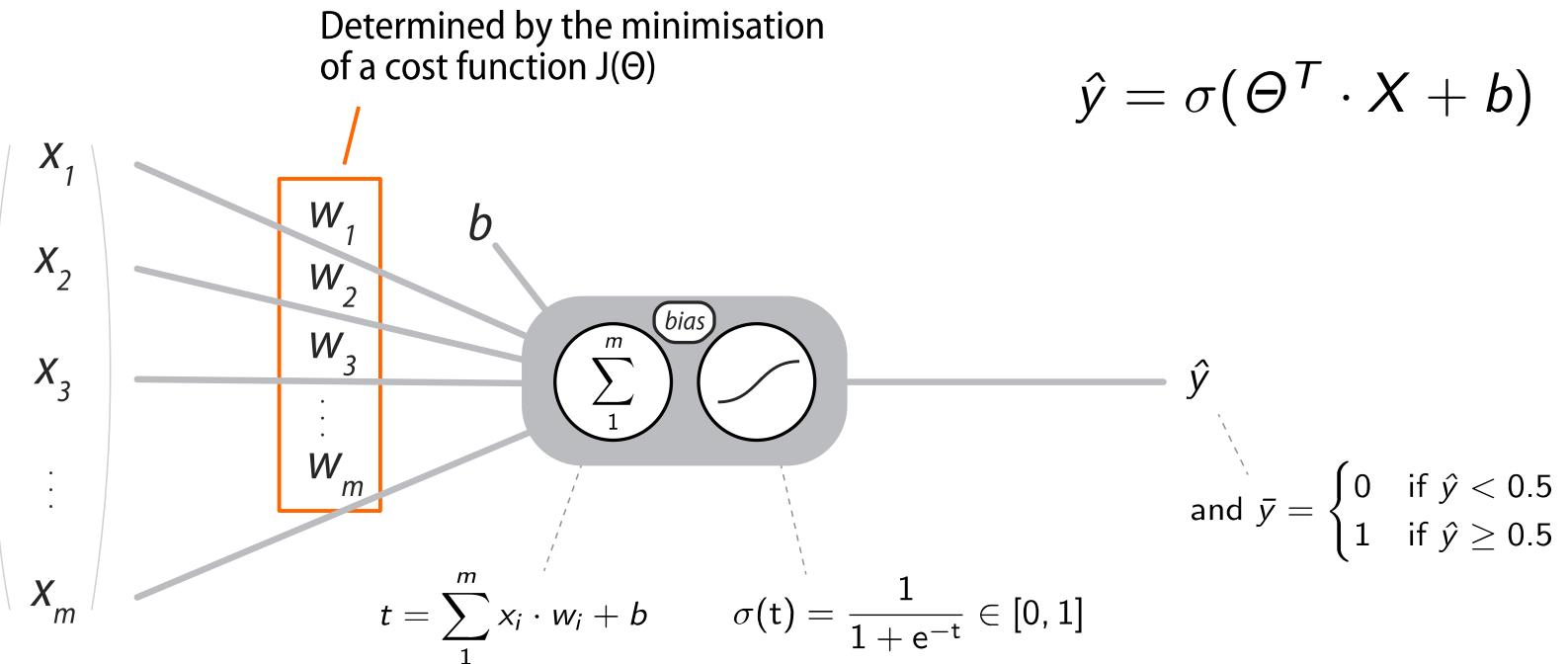
**IT MAKES MISTAKES.
IT LEARNS FROM THEM.**

**BE LIKE A NEURAL
NETWORK.**



@redcoders

SCHÉMA D'UNE RÉGRESSION LOGISTIQUE

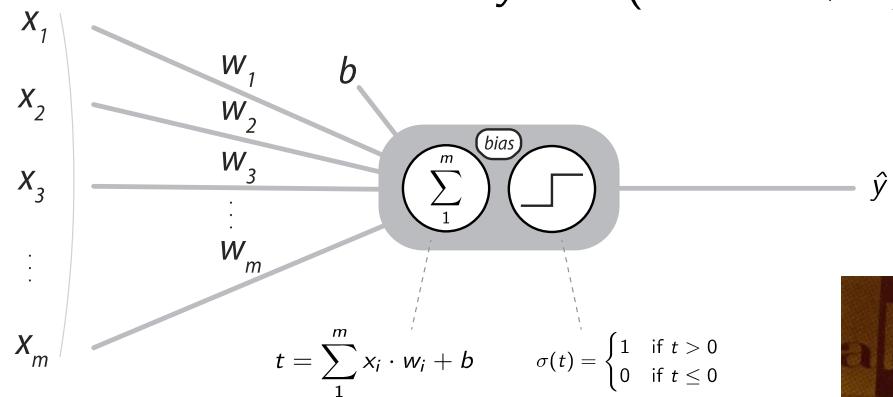


| Input | Bias / Weight | Activation function | Output |
|-------|---------------|---------------------|-----------|
| X | Θ | $\sigma(t)$ | \hat{y} |

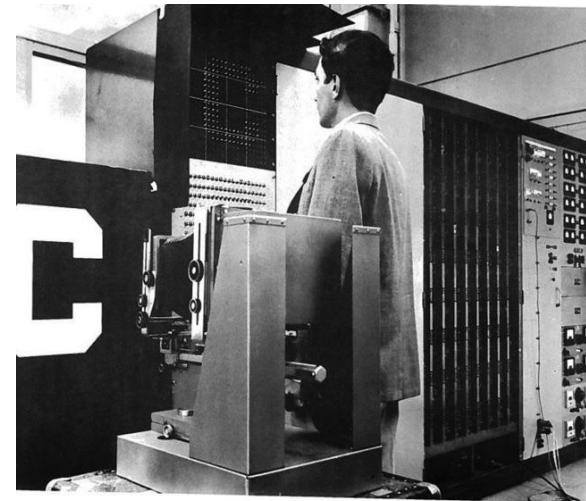
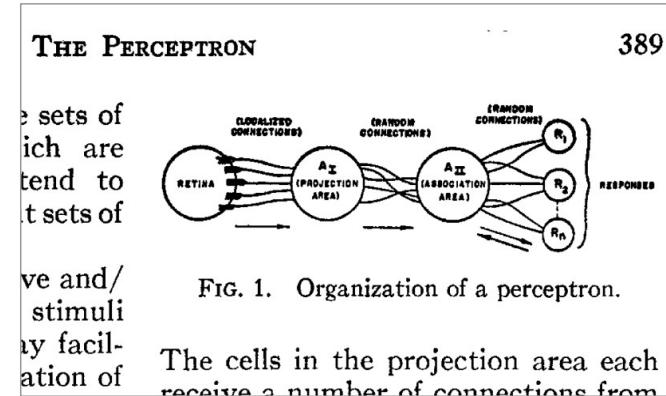
Surprise !!! Ceci est le schéma d'un neurone artificiel (perceptron). On a un réseau d'un neurone !!

L'HISTOIRE DES RÉSEAUX DE NEURONES

- Création du "Perceptron" : Frank Rosenblatt (1957)



<https://romeo.univ-reims.fr/chps>



NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI)—The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's \$2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of \$100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptrons will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

UN PERCEPTRON DANS SKLEARN

■ L'exemple de l'Iris

```
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np

iris = datasets.load_iris()

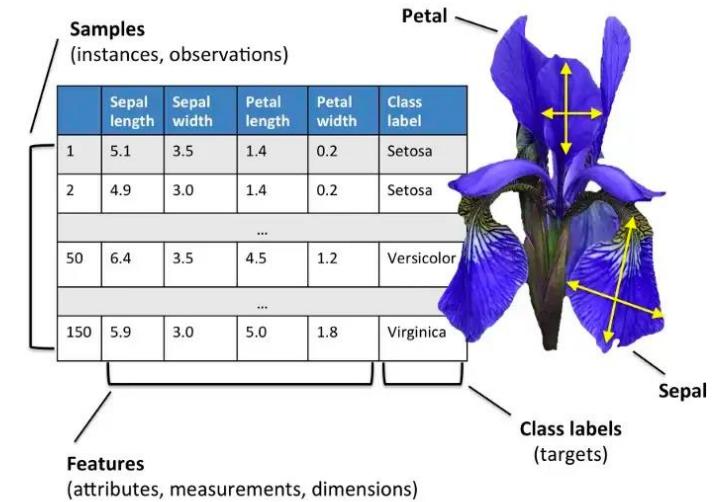
# Split the data into 70% training data and 30% test data
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.3)

# Train the scaler, which standarizes all the features to have mean=0 and unit variance
sc = StandardScaler()
sc.fit(X_train)

# Apply the scaler to the X training and test data
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

# Create a perceptron object 40 iterations (epochs) and a learning rate of 0.1
ppn = Perceptron(max_iter=40, eta0=0.1, random_state=0)

# Train the perceptron
ppn.fit(X_train_std, y_train)
```



AFFICHAGE DES RÉSULTATS

```
: # Apply the trained perceptron on the X data to make predicts for the y test data
y_pred = ppn.predict(X_test_std)

: y_pred
array([2, 0, 2, 0, 0, 2, 0, 2, 2, 0, 1, 0, 1, 0, 2, 2, 0, 0, 0, 2, 1, 1,
       2, 0, 0, 0, 1, 2, 2, 0, 1, 0, 2, 0, 1, 2, 2, 1, 0, 1, 2, 0, 2, 0,
       0])

: y_test
array([2, 0, 1, 0, 0, 1, 0, 2, 2, 0, 1, 0, 1, 0, 2, 2, 0, 0, 1, 2, 1, 1,
       2, 0, 0, 0, 1, 1, 2, 0, 1, 0, 2, 0, 1, 1, 2, 2, 1, 1, 2, 0, 2, 0,
       0])

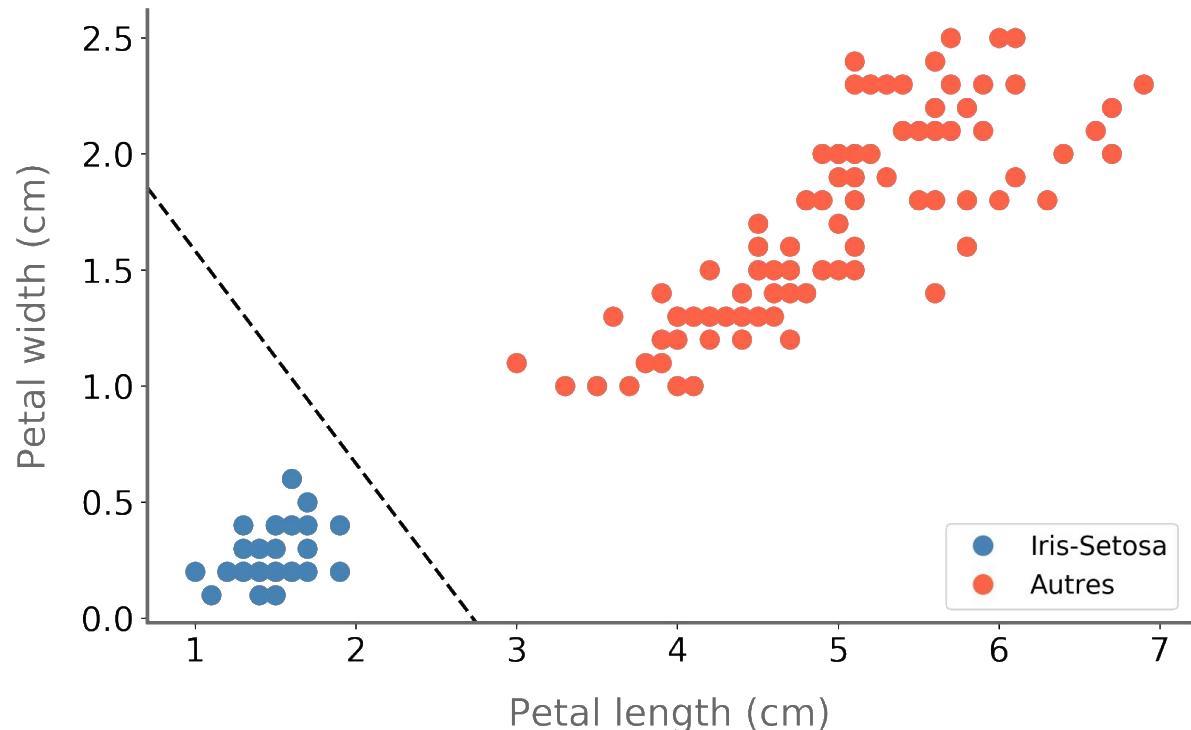
: # View the accuracy of the model, which is: 1 - (observations predicted wrong / total observations)
print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
```

Accuracy: 0.84

LE PERCEPTRON MARCHE (LINÉAIREMENT) !

Iris plants dataset

Dataset from : Fisher, RA "The use of multiple measurements in taxonomic problems" Annual Eugenics, 7, Part II, 179-188 (1936)



EXERCICE 1

1. Exécuter l'exemple disponible à cette adresse
 - Accéder ce site : <https://t.ly/jPuBM>
2. Cet exemple n'utilise que les données des pétales
 - Modifier l'exemple pour utiliser aussi les données des sépales
3. La classe setosa est facile à séparer (accuracy 100%)
 - Essayez de changer la classe cible, avez-vous la même précision ?



You are not a data scientist..

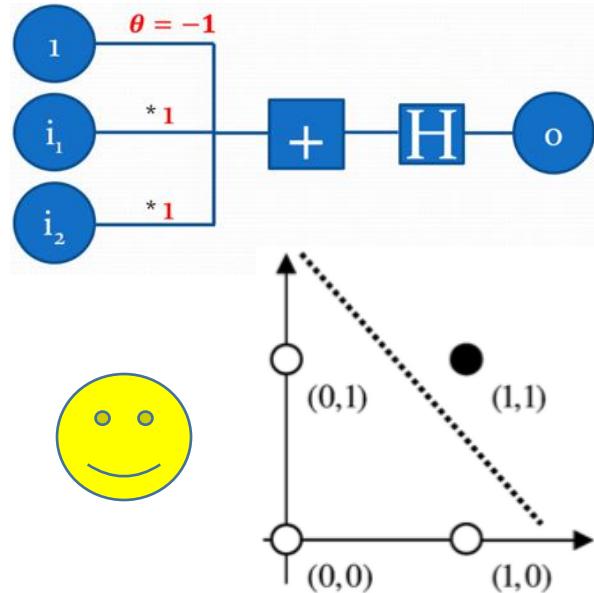


if you don't know this flower

LES RÉSEAUX DE NEURONES PROFONDES

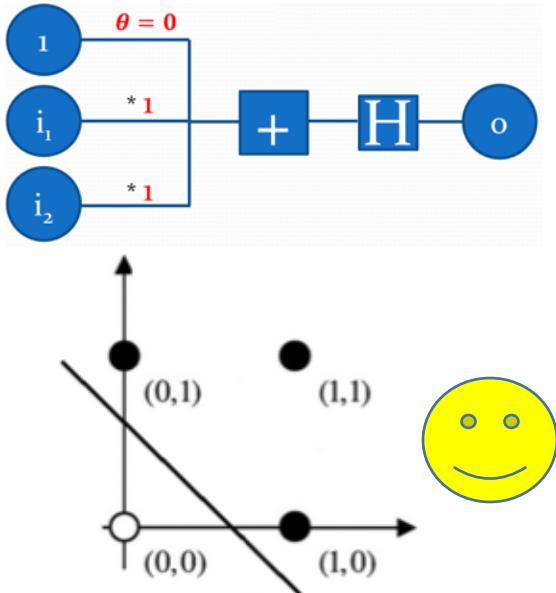
- Un perceptron peut apprendre tant que les résultats sont "linéairement séparables" :
- Ex ET logique

| i₁ | i₂ | o |
|----------------------|----------------------|----------|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



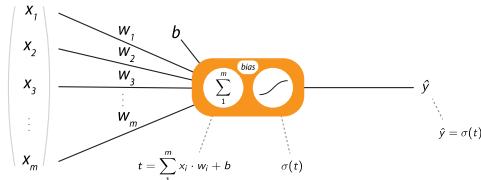
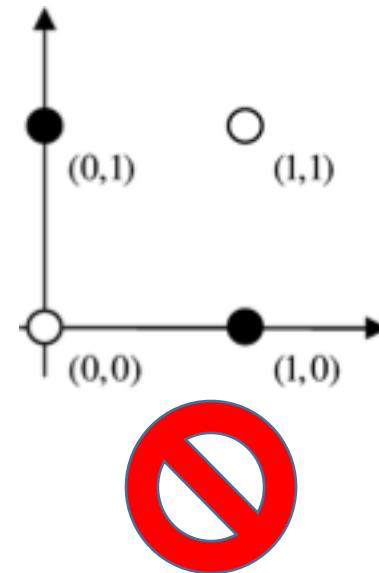
Ex OU logique

| i₁ | i₂ | o |
|----------------------|----------------------|----------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

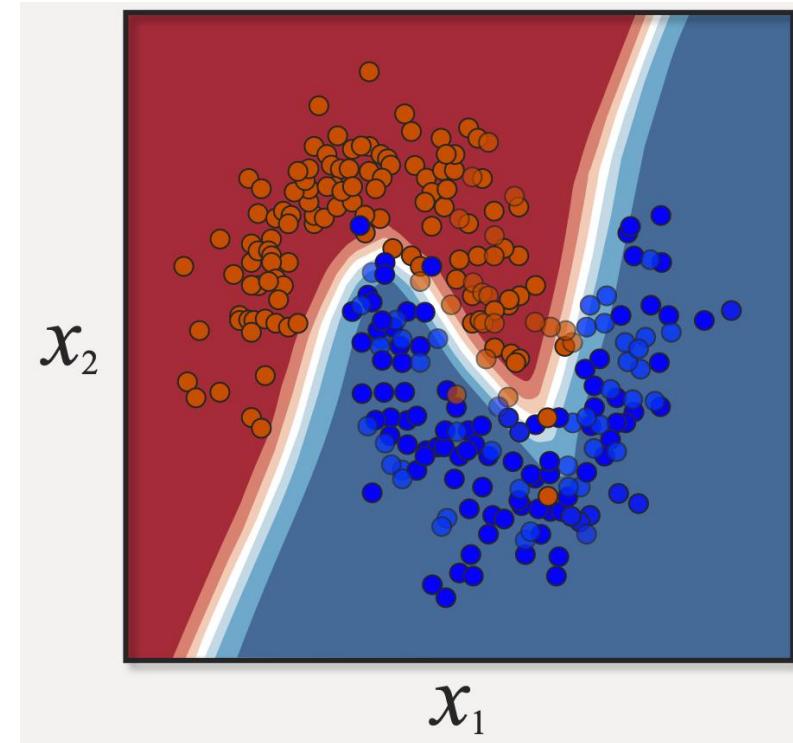
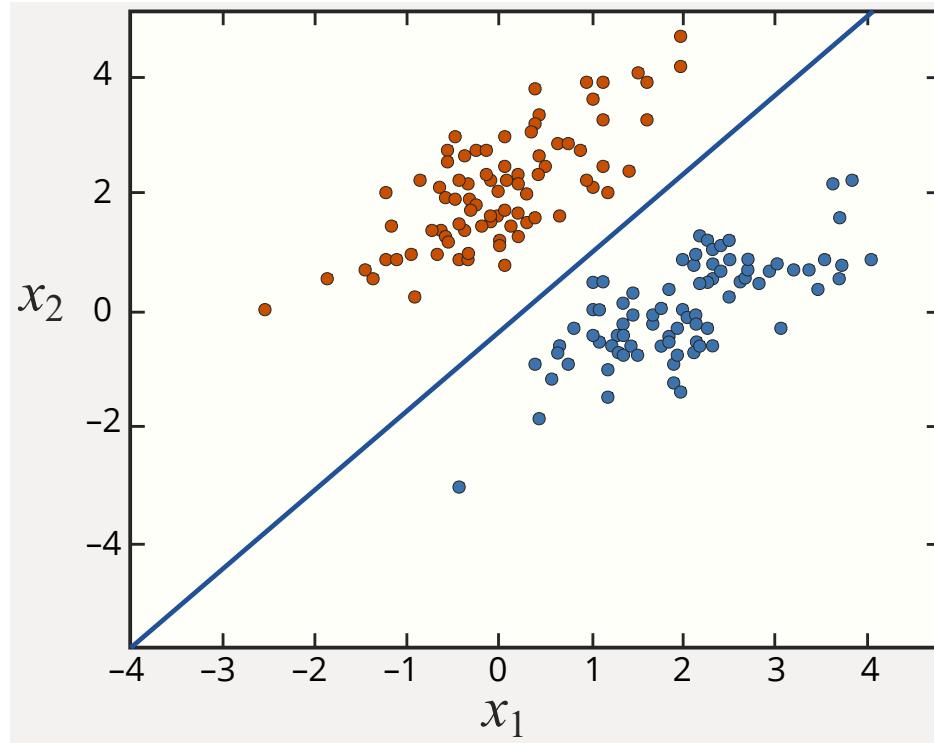


Ex XOU logique

| | E₁ = 0 | E₁ = 1 |
|--------------------------|--------------------------|--------------------------|
| E₂ = 0 | 0 | 1 |
| E₂ = 1 | 1 | 0 |



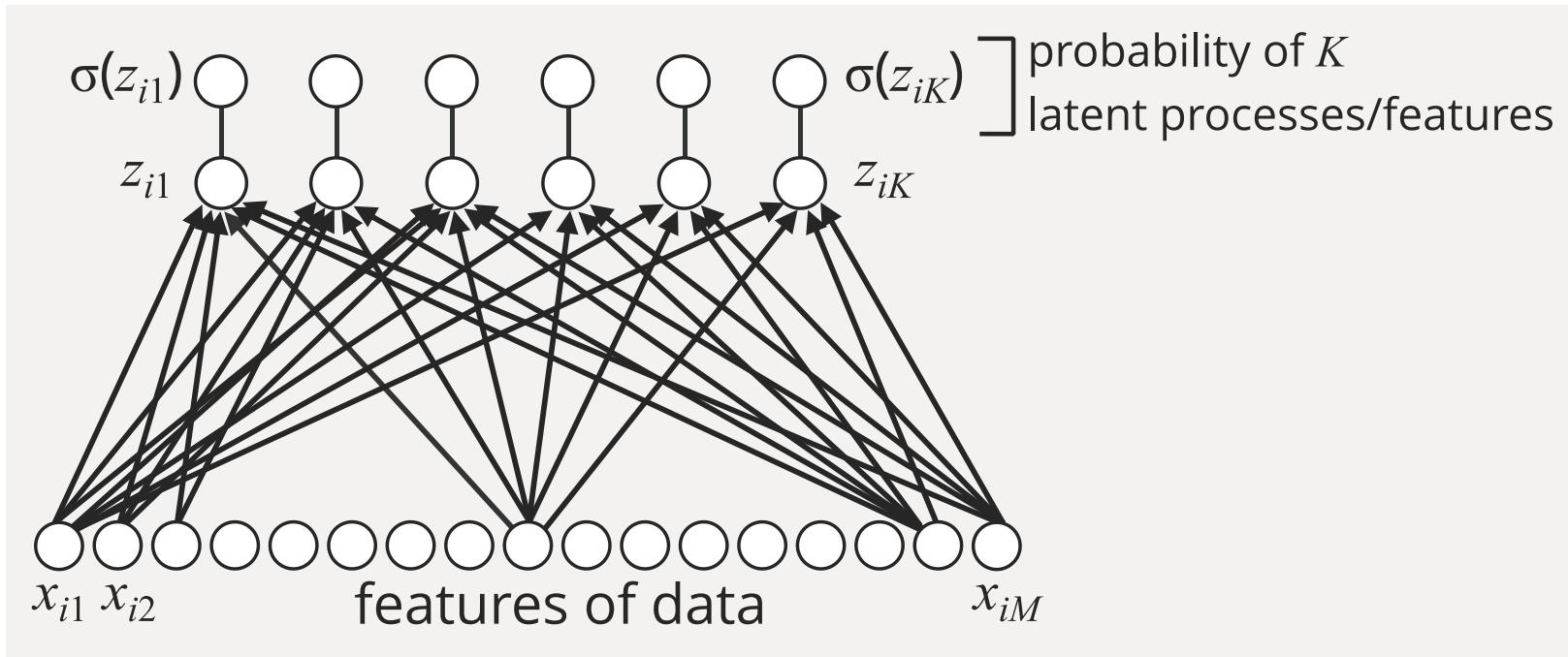
UN SEUL NEURONE NE FAIT PAS TOUT



GÉNÉRALISATION DE LA RÉGRESSION LOGISTIQUE

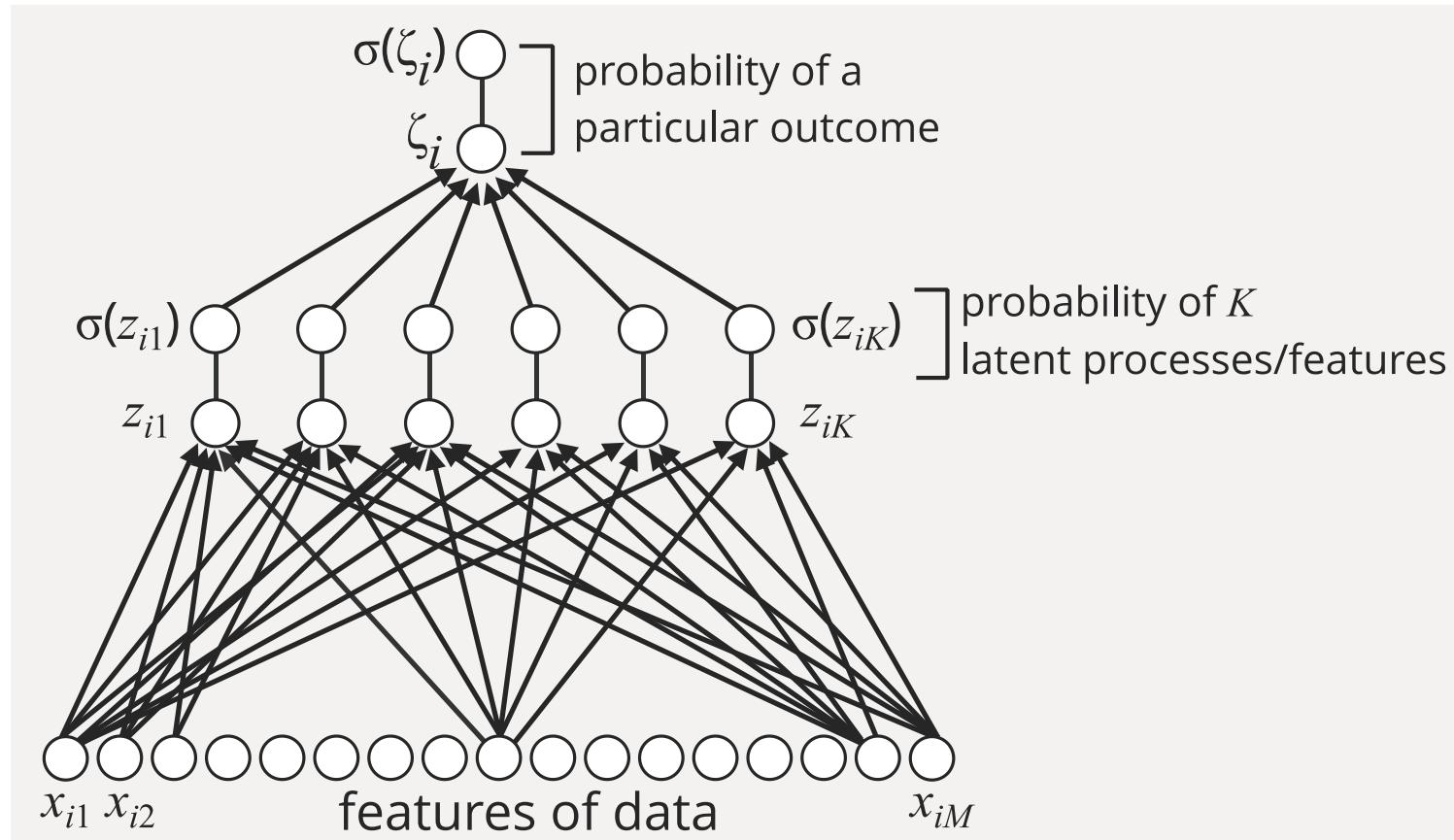
- APPRENTISSAGE DE PLUSIEURS FEATURES

- On pourrait multiplier le nombre de neurones afin d'avoir des "vues" différentes sur les données



EXTENSION DE LA RÉGRESSION LOGISTIQUE

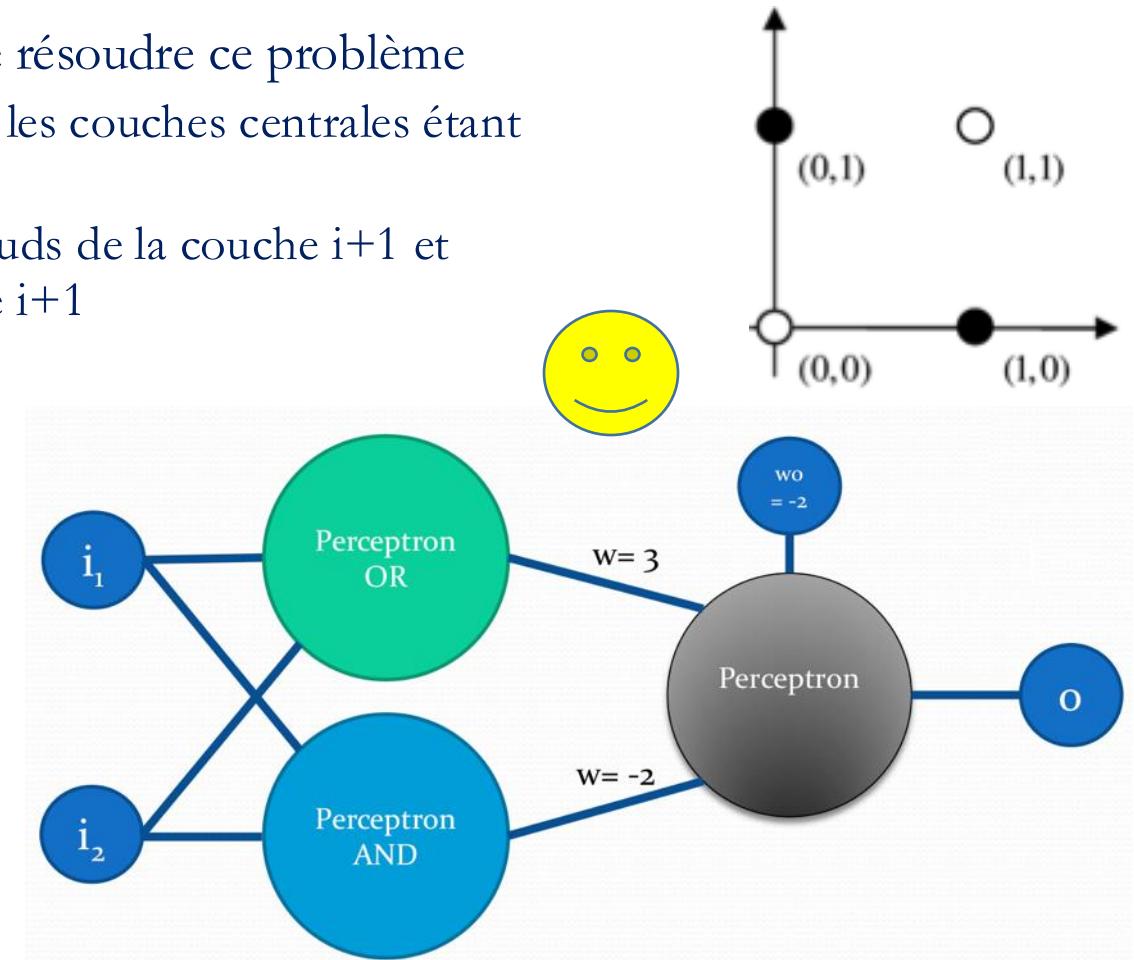
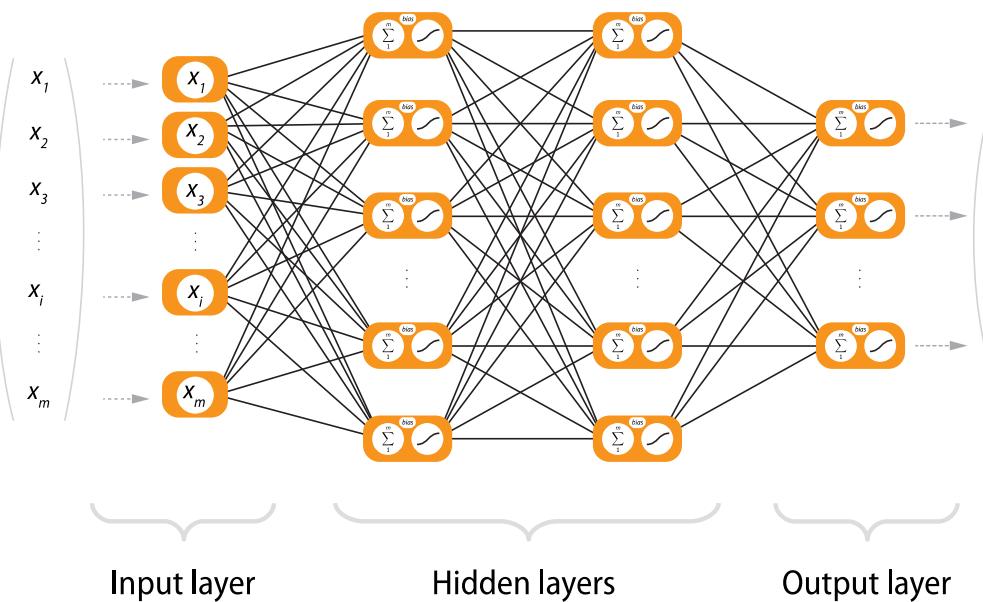
- Et si on faisait un 2^{ème} niveau de régression logistique ?



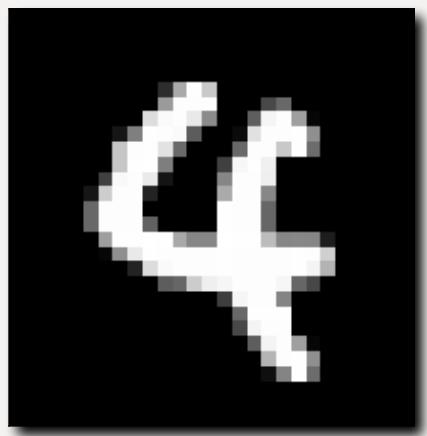
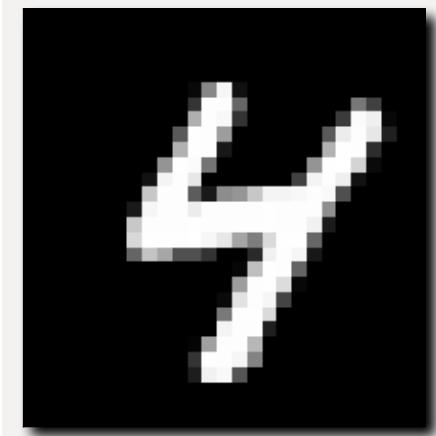
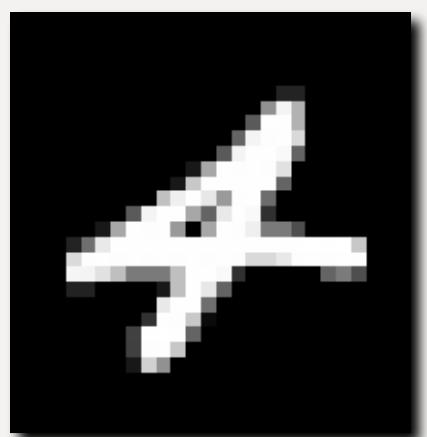
On a une régression logistique sur K features latentes, plutôt qu'une régression sur M données brutes

RÉSEAUX DE NEURONES

- Le concept de perceptron multicouche (MLP) permet de résoudre ce problème
 - On utilise plusieurs perceptrons en couche successives, les couches centrales étant dites cachées
 - Chaque nœud de la couche i est connecté à tous les nœuds de la couche $i+1$ et l'information circule toujours de la couche i à la couche $i+1$

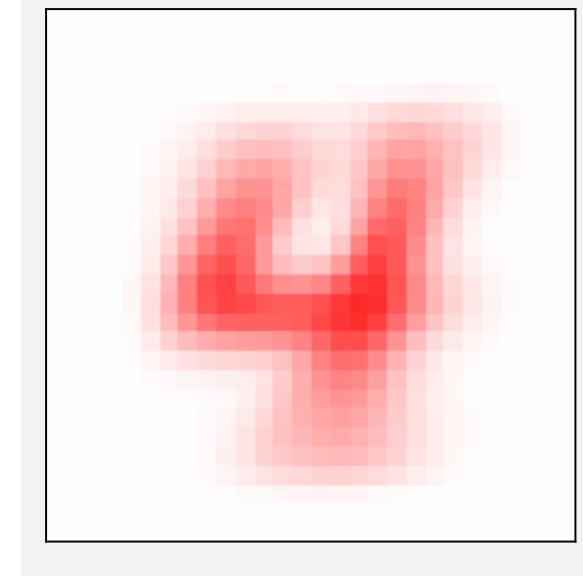


PLUSIEURS FAÇONS D'ÉCRIRE 4

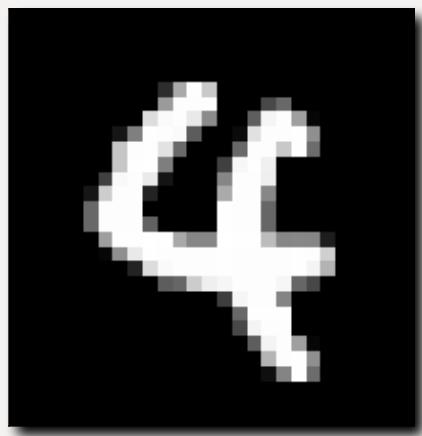
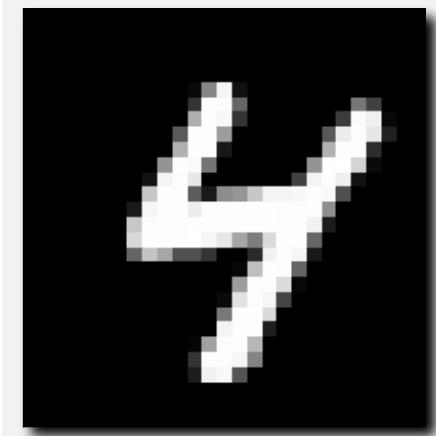
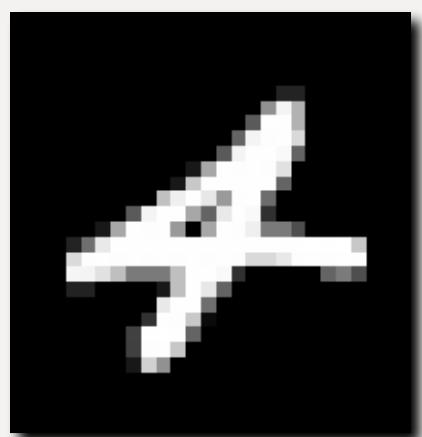


Filtre unique (shallow)

- pas très précis, adapté juste au cas moyen

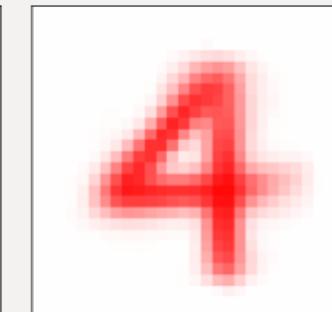
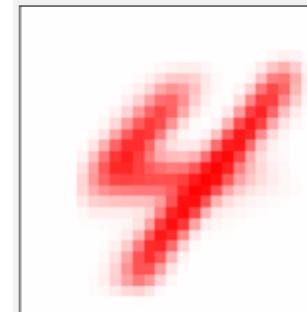
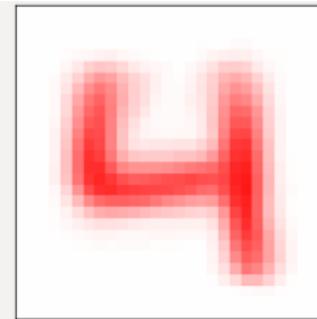


PLUSIEURS FAÇONS D'ÉCRIRE 4

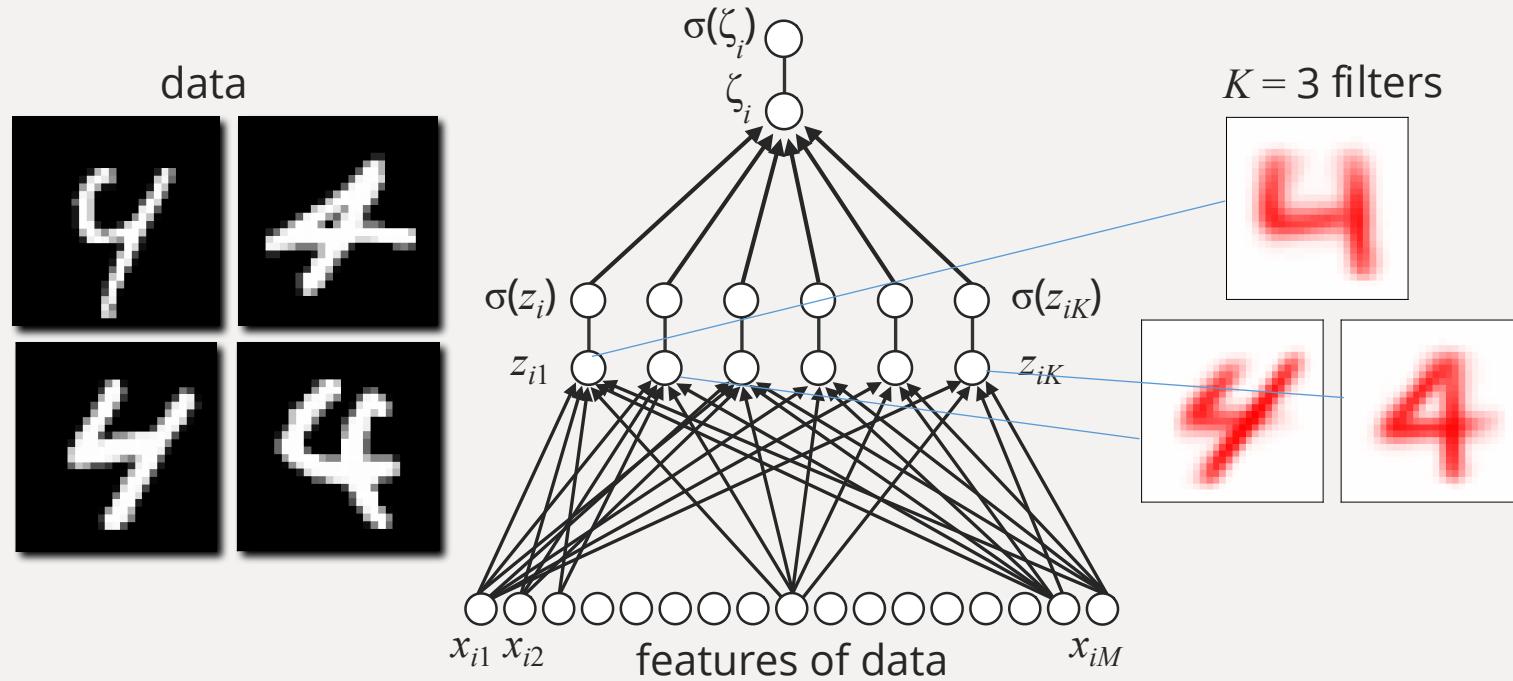


Plusieurs filtres

- Chacun est spécialisé sur une forme d'écrire 4



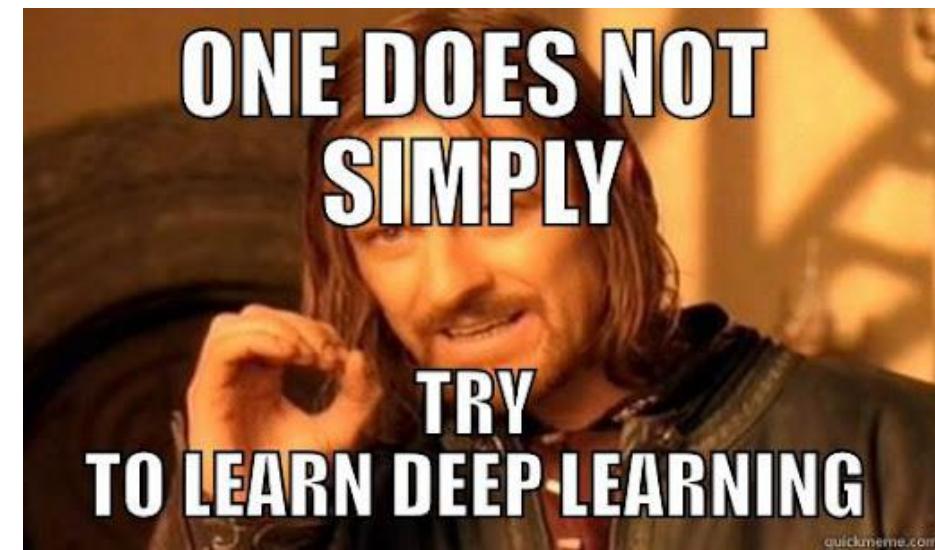
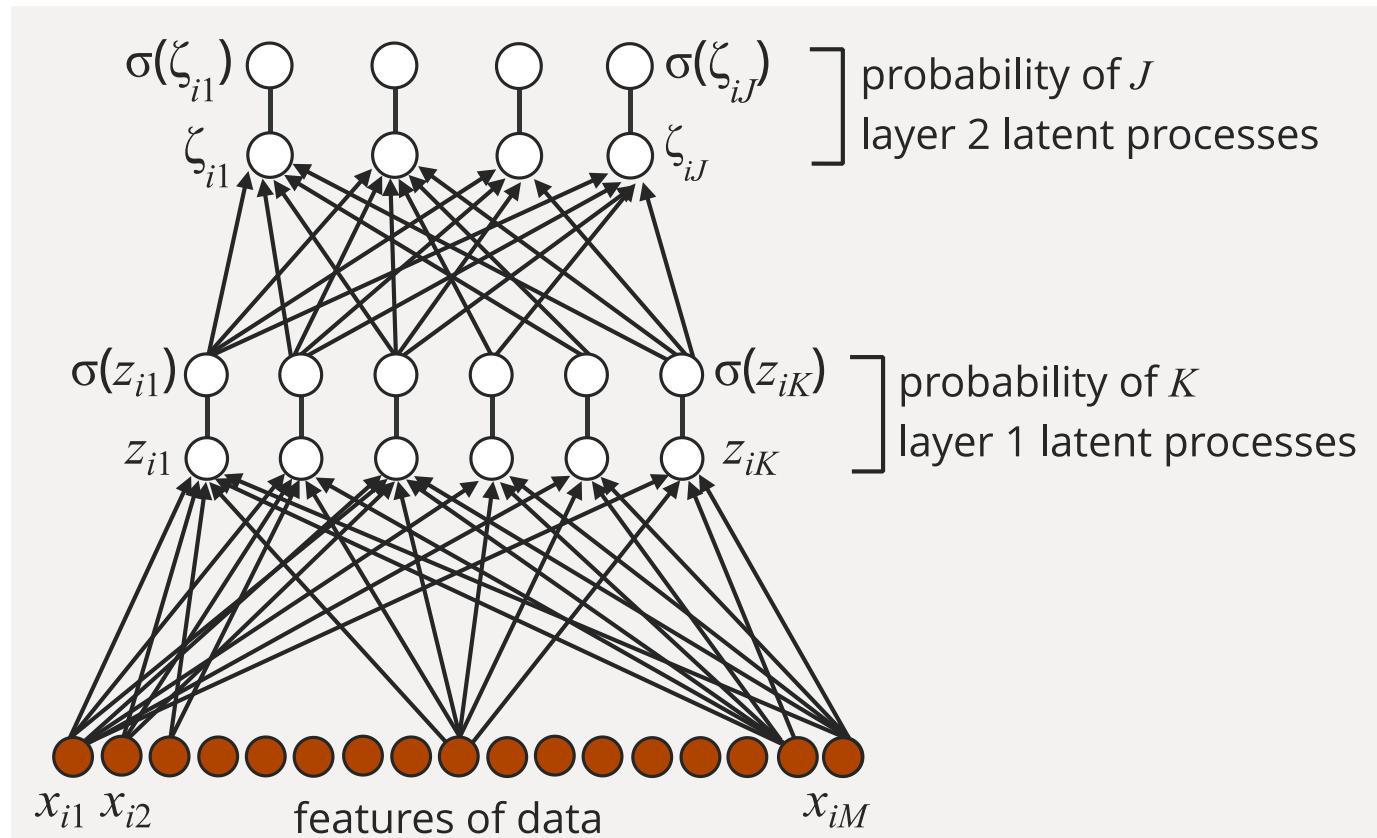
RÉSEAU DENSE DE NEURONES



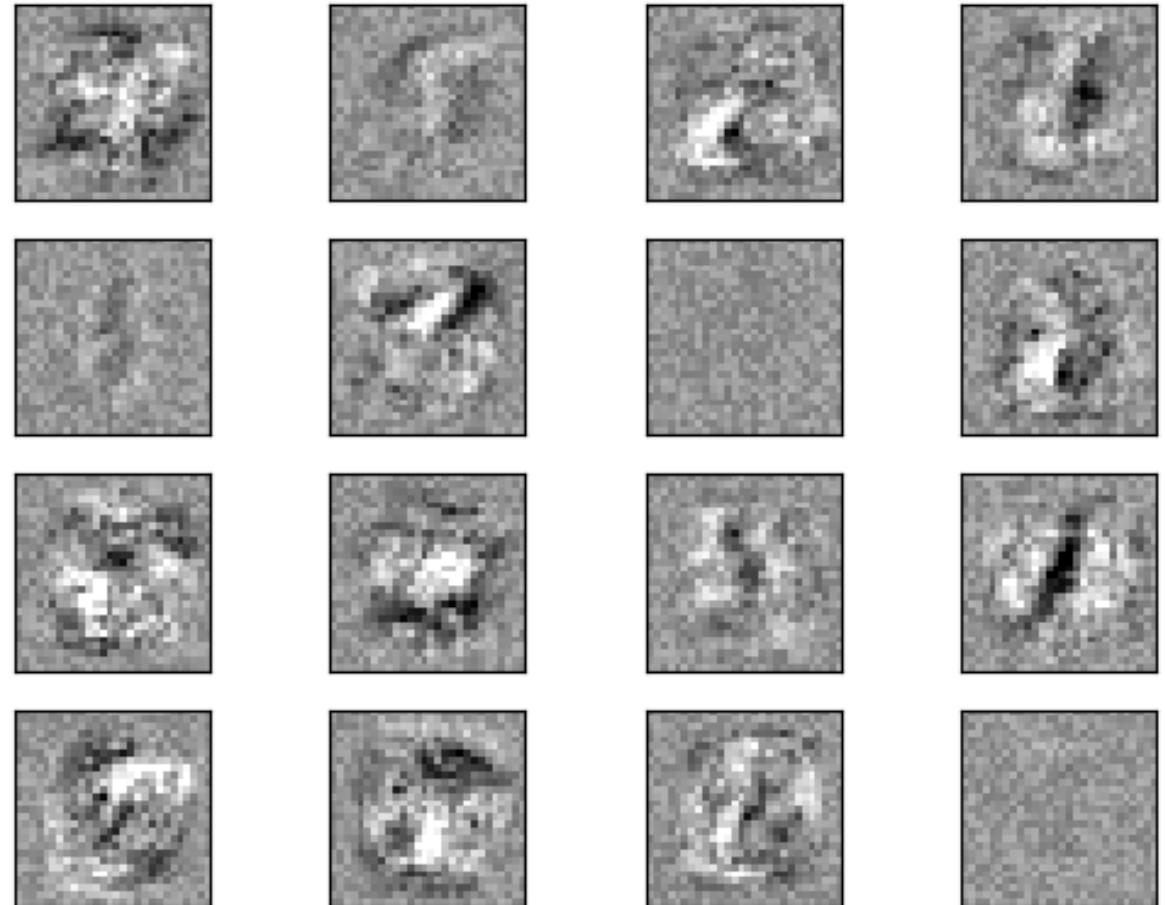
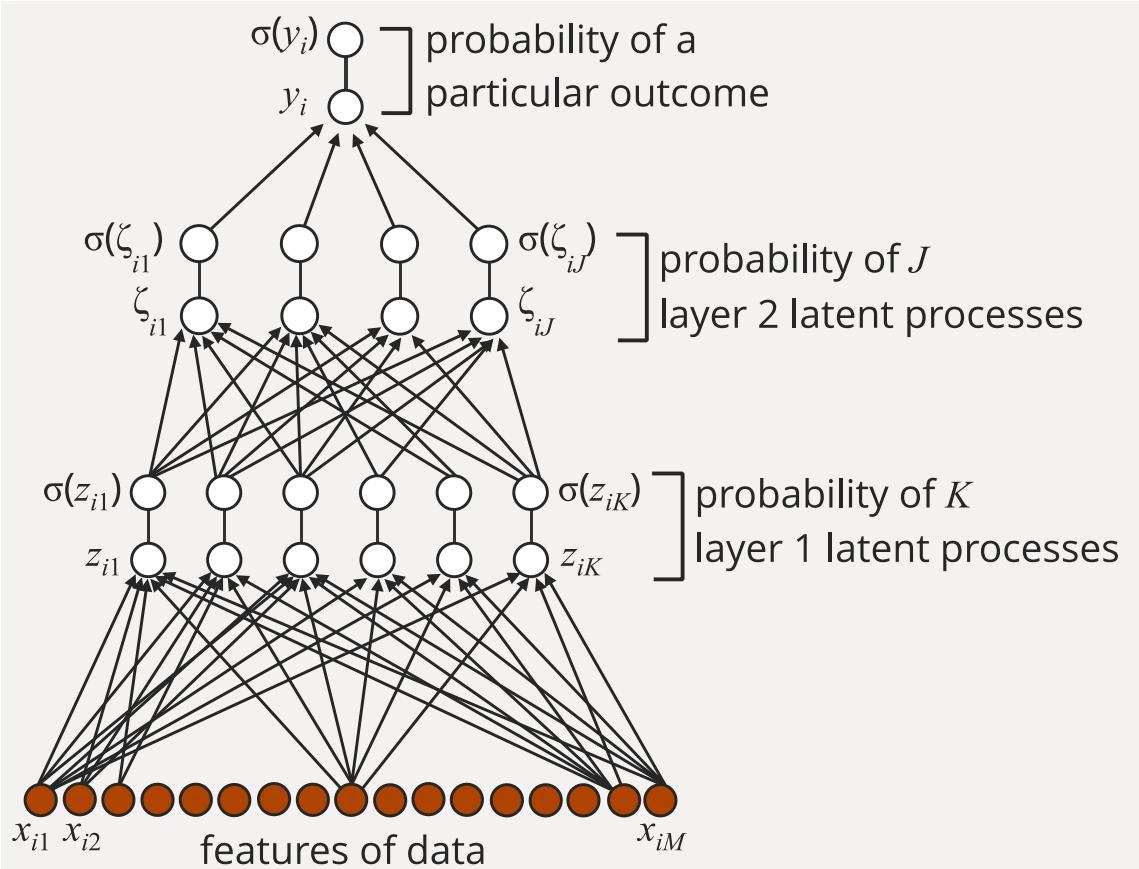
- Dans la pratique, les neurones sont initialisés avec des poids aléatoires
- L'entraînement permet à chacun de regarder les données avec des points de vue différents
- C'est la somme de ces points de vue qui donne une sortie

POURQUOI S'ARRÊTER LÀ ?

- Différentes couches peuvent mettre en valeur des features distinctes



PERCEPTRON MULTI-COUCHES



MAIS COMMENT LE RÉSEAU APPREND ?

- Perceptron multicouches :
 - Chaque neurone reçoit le vecteur de toutes les sorties de neurones de la couche précédente (on parle de réseau dense -> *dense layer*)
- Comment le réseau apprend ?
 - Grâce à **La fonction coût!**
- Il s'agit de l'erreur entre le résultat du réseau de neurones et le résultat attendu
 - Elle est liée à la forme du résultat recherché (régression, classification)
 - Continue, dérivable selon chacune de ses variables (poids du réseau)

FONCTION COÛT

Principales fonctions de coût:

$$x = (x_i) : \text{inputs} \quad y = (y_i) : \text{outputs voulu} \quad z = (z_i) : \text{outputs du DNN}$$

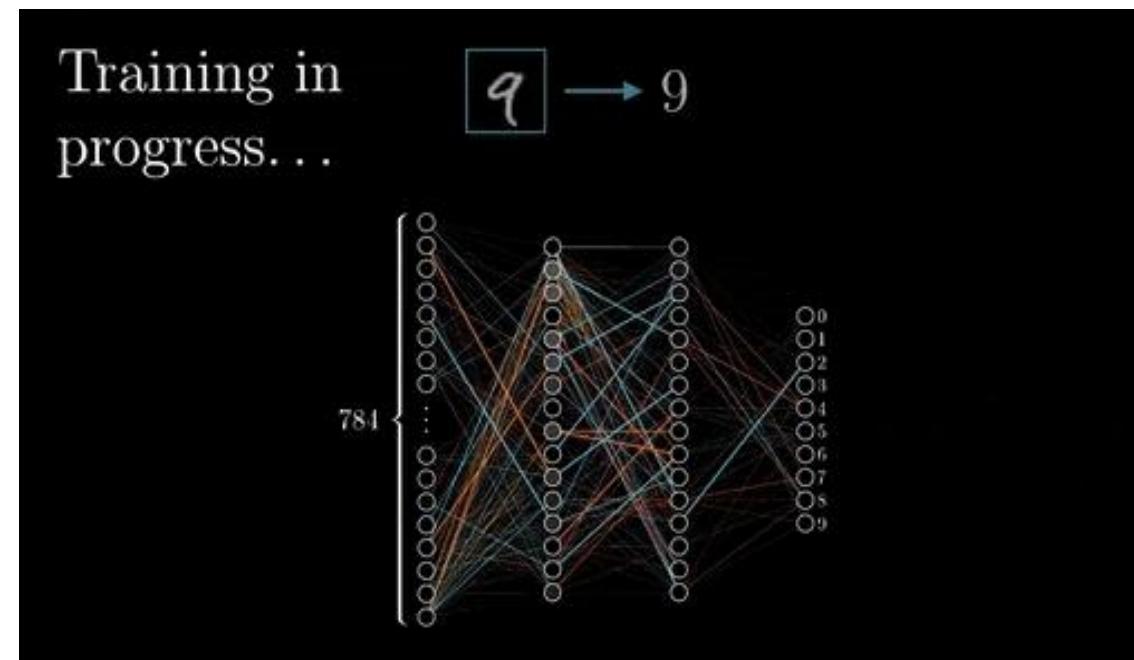
- **Mean Square Error :** $MSE = \frac{1}{n} \sum_{i=1}^n (y_i - z_i)^2$  Régression
- **Cross Entropy :** $CE = \frac{1}{n} \sum_{i=1}^n y_i \log(z_i)$  Classification

La **Cross entropy** mesure la distance entre deux distributions de probabilités

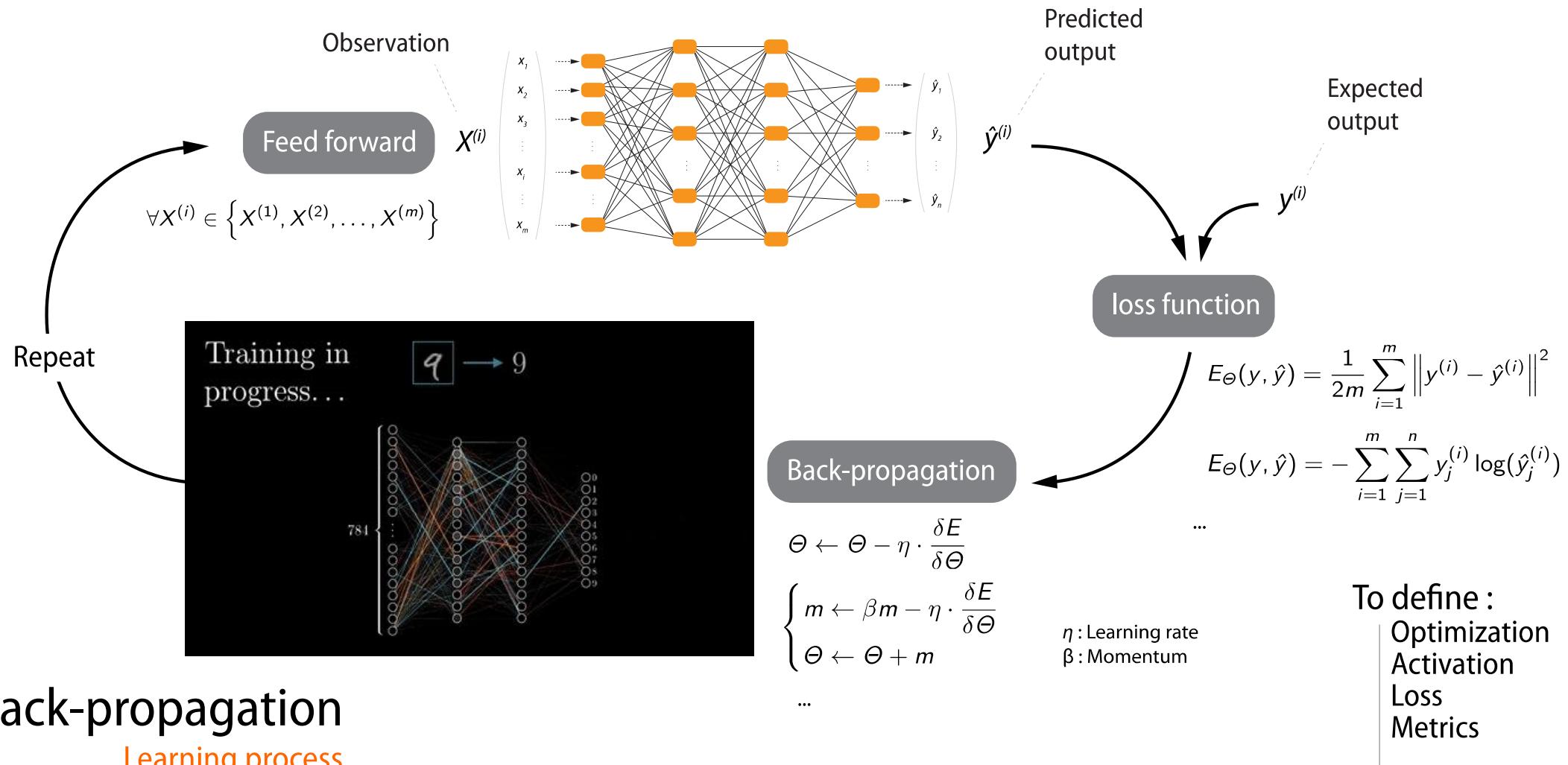
Elle pénalise fortement les grandes erreurs d'estimation

RÉTROPROPAGATION

- Le problème dans un réseau de neurones est de construire un ensemble de Poids (W) qui minimisera la fonction coût
 - Initialement les poids sont inconnus, notre tâche est de les estimer en tenant compte des données
- Dans une itération, notre réseau va donc :
 1. Calculer les résultats issus des données d'entrée (forward pass)
 2. Estimer l'erreur de ces résultats à l'aide de la fonction coût
 3. Mettre à jour (modifier) les poids (W) des différentes couches en dérivant cette erreur par rapport à ces poids



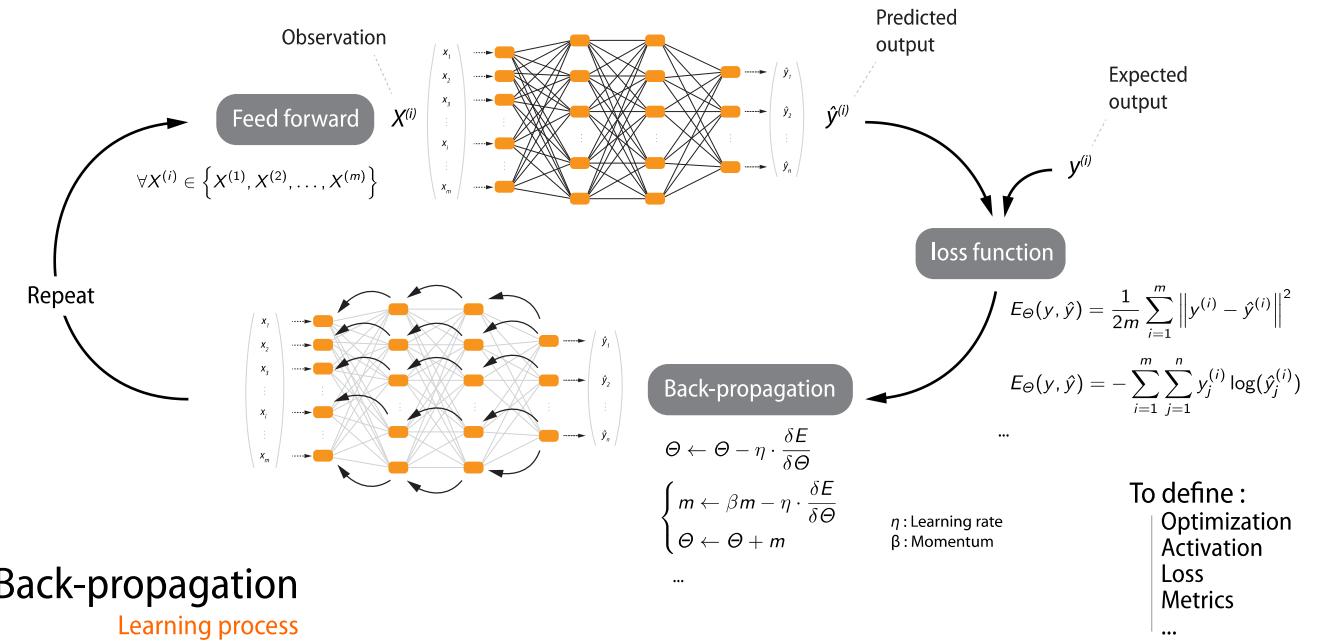
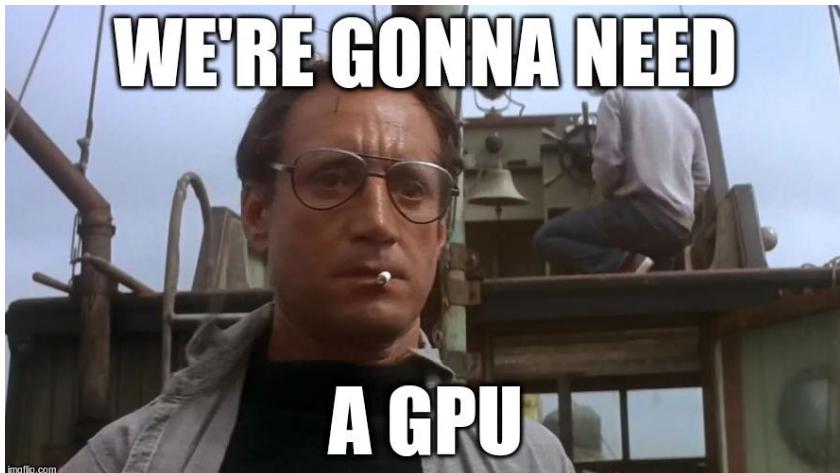
ENTRAÎNEMENT PAR RETROPROPAGATION (1986)



Back-propagation
Learning process

COÛT D'UN ENTRAÎNEMENT COMPLET

- La rétropropagation s'effectue sur l'ensemble des couches
- Opération lourde, surtout si le réseau est profond et le dataset grand et complexe
 - Peu pratique sans des GPU



ET POUR UTILISER L'IA ?

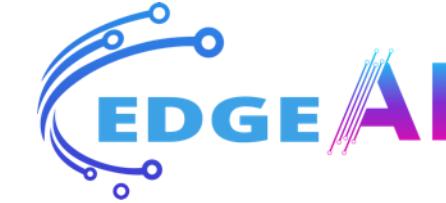
- Un modèle entraîné est souvent simple à utiliser
 - Nous avons tous des applications IA dans nos téléphones



- Le cloud est également une option
 - Nécessite un réseau fonctionnel



- Un modèle IA peut être déployé à proximité de l'utilisateur final
- → Edge AI computing



PROBLÈMES DES DNN - SCALABILITÉ

For a fully connected layer of (only)
1000 neurons, we would need to



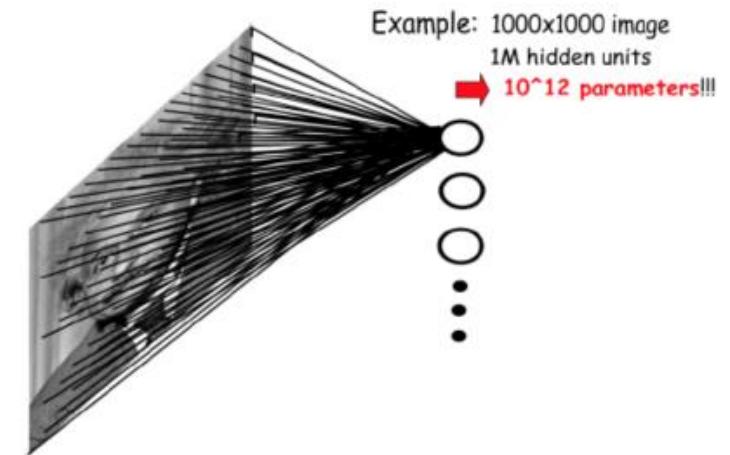
0.0008 M pixels
28x28, 8 bits

→ 785.000 params



24 M pixels
(r,v,b) 3x8 bits

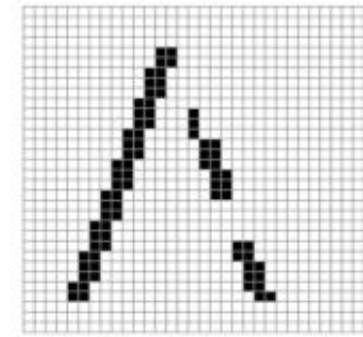
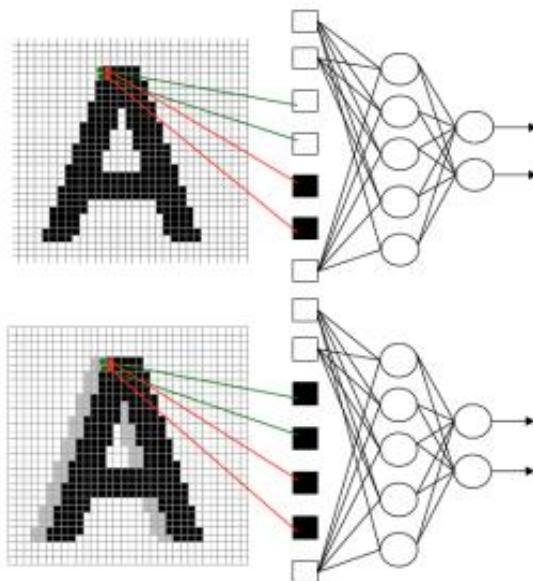
→ 72. 10^{E9} params...



DEUXIÈME PROBLÈME

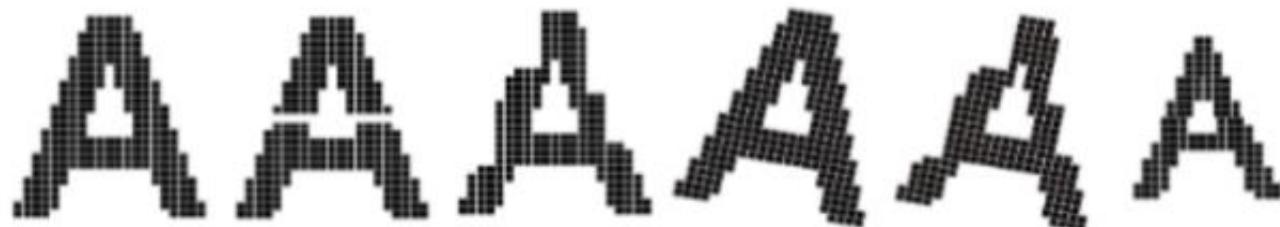
- Invariance et Stabilité

- **Invariance & stability**
- Expectations:
 - Small deformation \Rightarrow similar representations
 - Large deformation \Rightarrow dissimilar
- Translation invariance difficult with Fully Connected Networks \sim local scale, rotation, deformations, etc



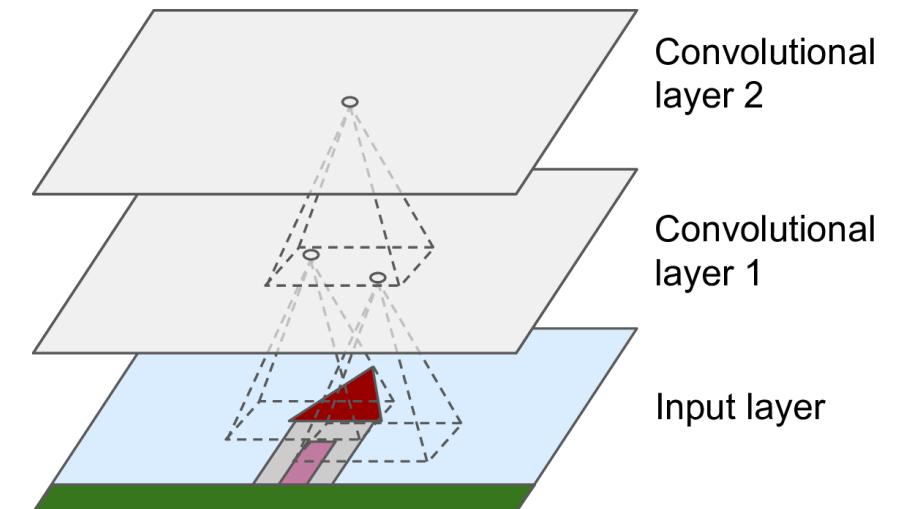
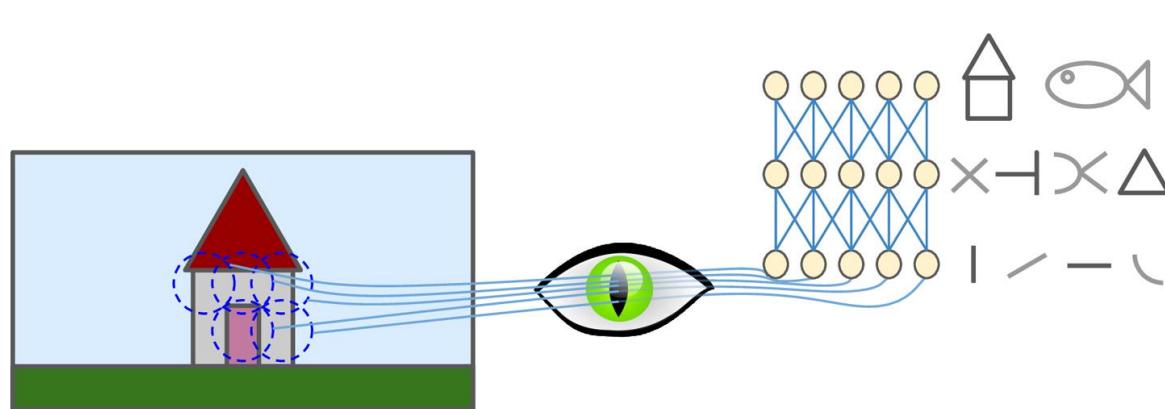
154 input change
from 2 shift left
77 : black to white
77 : white to black

@LeCun



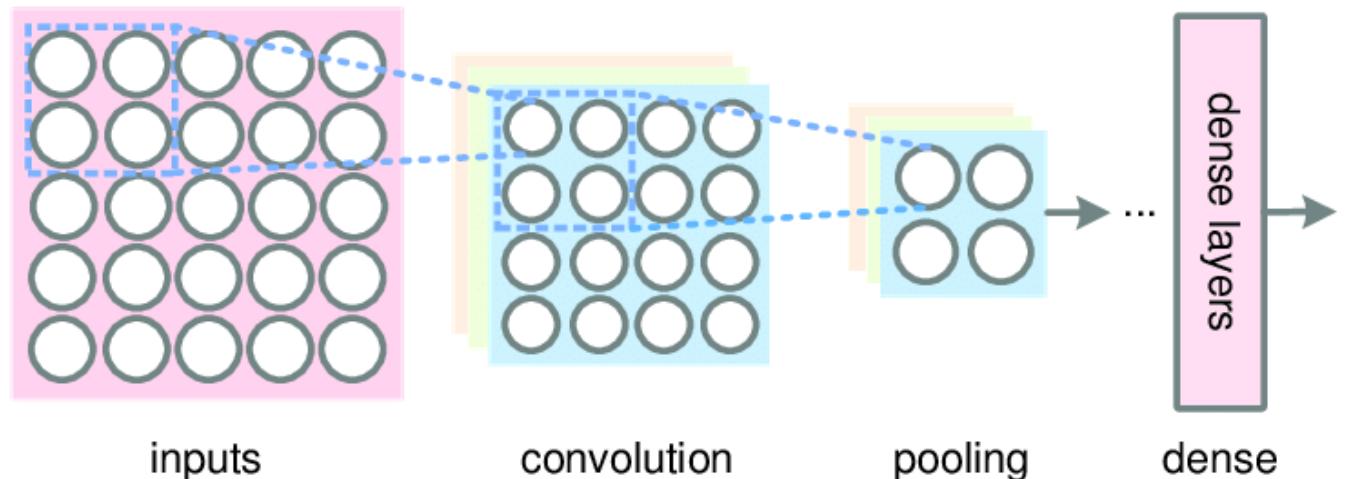
L'ARCHITECTURE DU CORTEX VISUEL

- Des travaux faits par des biologistes/neurologistes dans les années 50 ont montré que plusieurs neurones ont un champ d'activation limité
 - Ne réagissent qu'à des certaines zones du champ visuel
 - La réceptivité de ces neurones peut se superposer
 - Certains neurones ne répondent qu'à certains motifs : ex : lignes horizontales
 - D'autres étaient plus "avancés", combinant de signaux des neurones plus basiques



LES RÉSEAUX CONVOLUTIFS (CNN)

- Introduits par Yan Lecun en 1998 dans l'architecture LeNet-5
 - Utilisé encore pour la reconnaissance de l'écriture manuelle
- Utilisation d'éléments déjà connus :
 - Réseaux totalement connectés
 - Fonctions d'activation (sigmoid, notamment)
- Introduction de deux nouveaux éléments :
 - Les couches convolutionnelles
 - Les couches de pooling



C'EST QUOI UNE CONVOLUTION ?

filter (W)

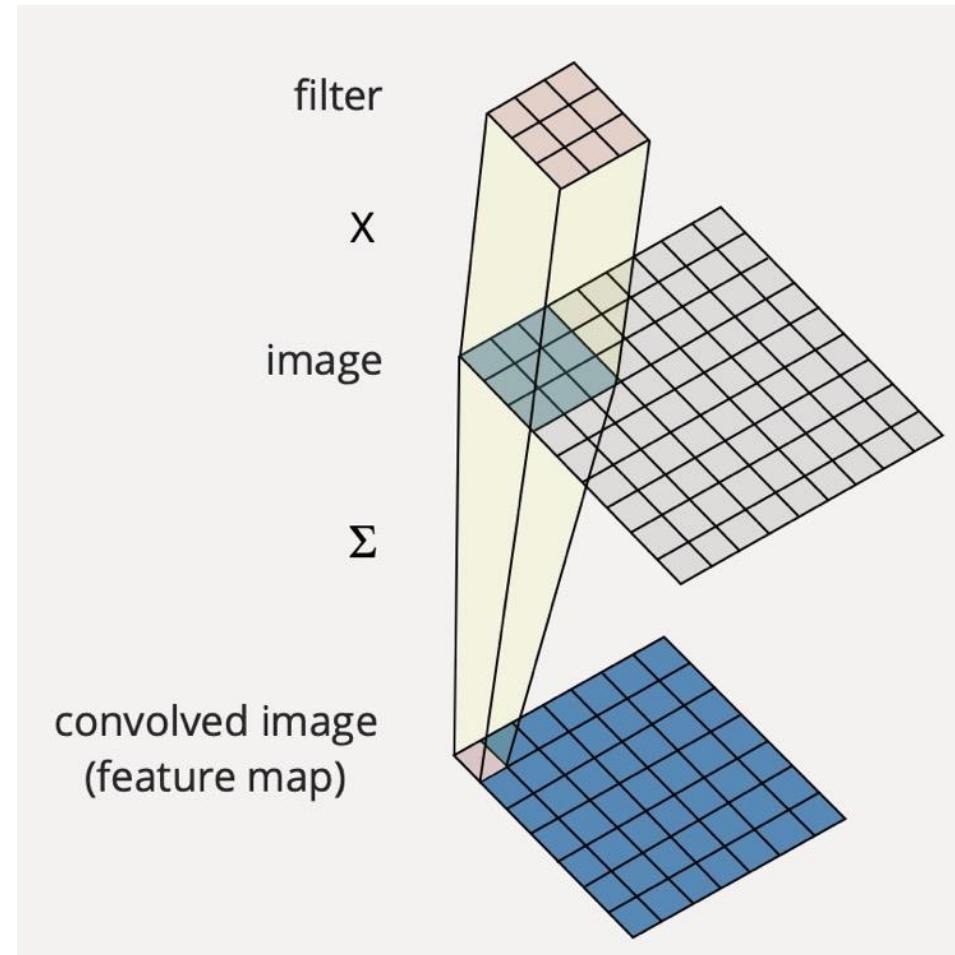
| | | |
|---|----|---|
| 0 | 1 | 0 |
| 1 | -4 | 1 |
| 0 | 1 | 0 |

input (i)

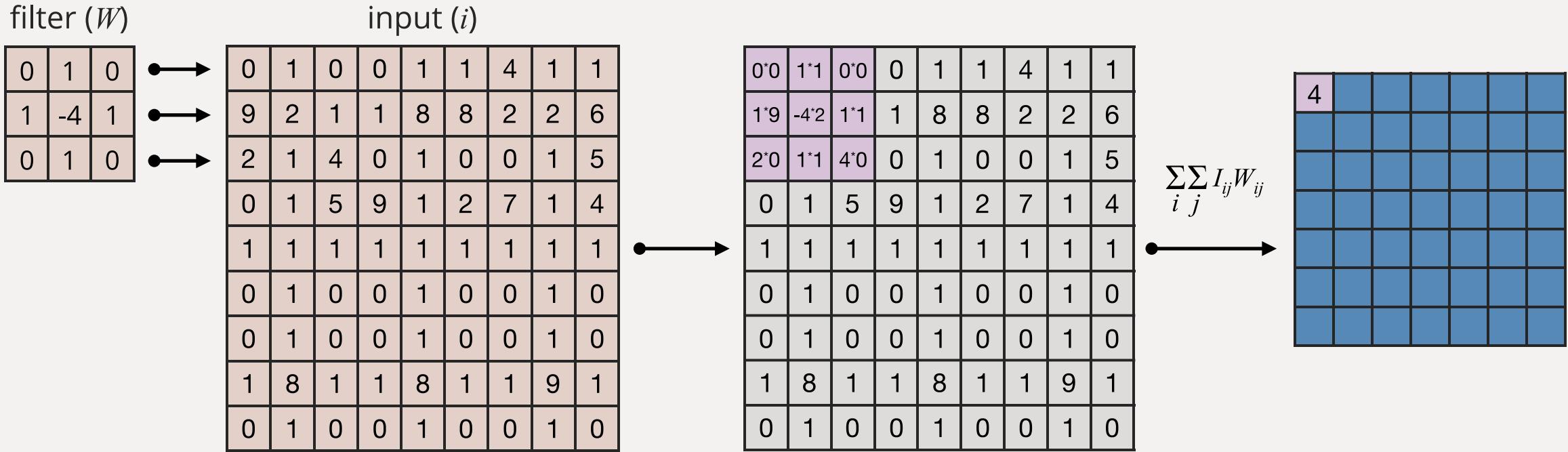


LE CAS DE LA CONVOLUTION

- Une convolution (2D dans cet exemple) est une opération de type multiplication de matrices
- Un filtre "balaye" la matrice de données, résumant l'information selon un motif défini (ou aléatoire)
- Un filtre peut avoir différentes tailles
 - Ex : 5×5 , 3×3 , 4×6 ...

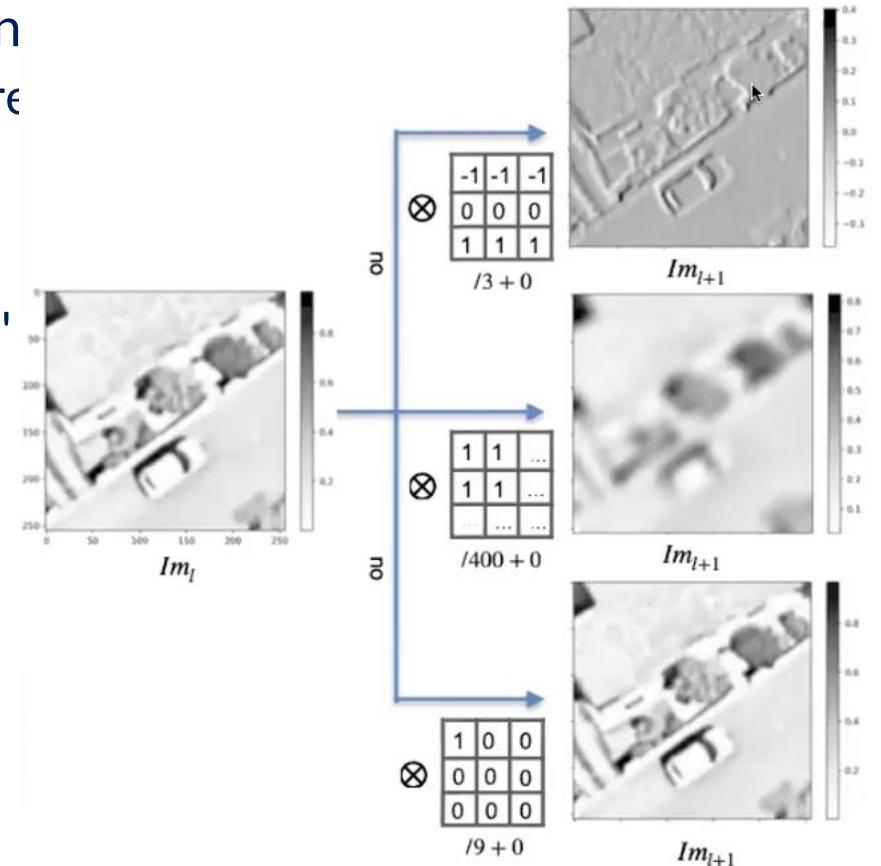


LE CAS DE LA CONVOLUTION 2D



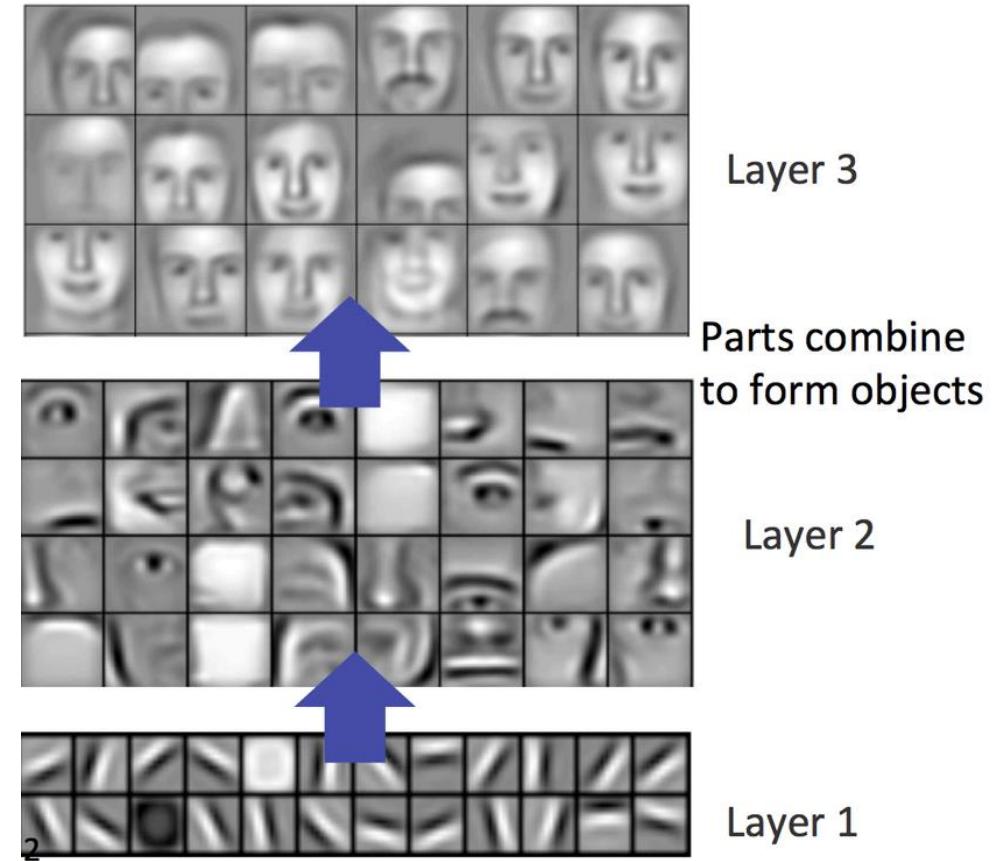
L'INTÉRÊT DES CONVOLUTIONS

- Les filtres permettent de chercher des caractéristiques différentes
 - On peut dire que chaque neurone est responsable par un filtre
- Souvent, les filtres sont initialisés de manière aléatoire
 - Certains se dédoublent, d'autres ne servent à rien
- En spécialisant les neurones des couches initiales, on "enrichit" l'observation sans trop augmenter le nombre de paramètres
 - 1 – chaque convolution produit une sortie "réduite"
 - 2 – les neurones intermédiaires ne seront pas "figés" sur une seule caractéristique
 - 3 – comparé à un DNN, on aura moins de connexions



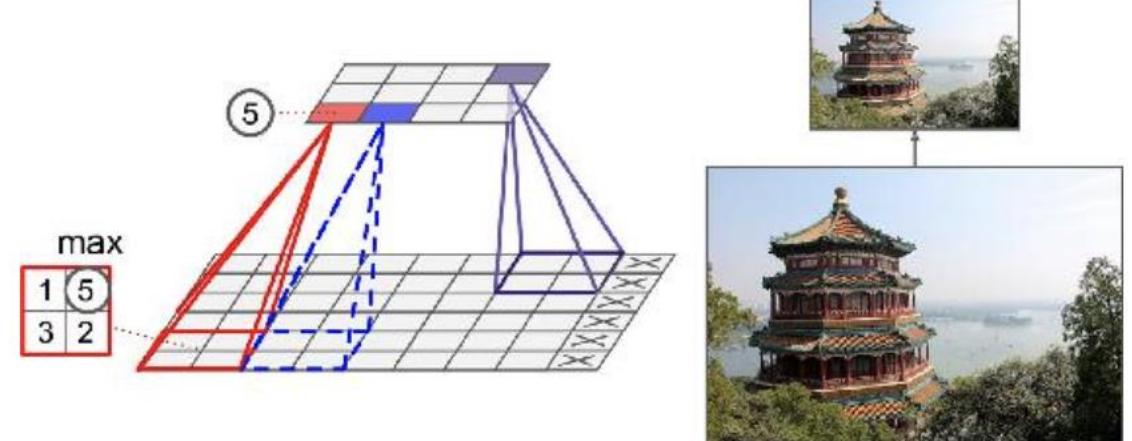
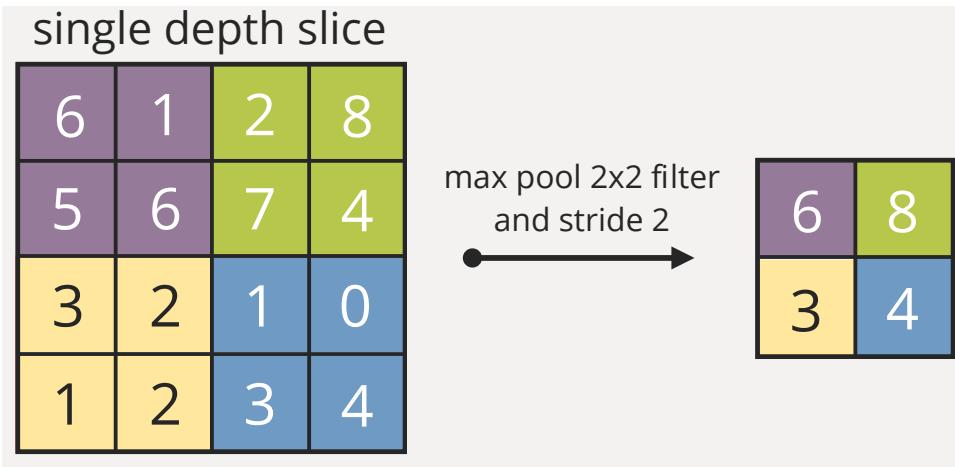
EST-CE SI REVOLUTIONNAIRE ?

- Comme on a vu avant, il faut un nombre important de filtres pour représenter différentes features
- Les neurones pour des feature maps différents utilisent des poids différents
- Un DNN ne peut reconnaître le motif là où il l'a appris
- Un CNN apprend à identifier un motif, il est capable d'identifier ce motif partout dans l'image



LA COUCHE POOLING

- Similaire à une convolution, sert à réduire la taille de la matrice (et donc la complexité du modèle)
- On utilise typiquement un filtre "max" (*maxpooling*)
 - Permet de réduire la taille des matrices tout en gardant les valeurs plus importantes
- On peut également utiliser la "moyenne"

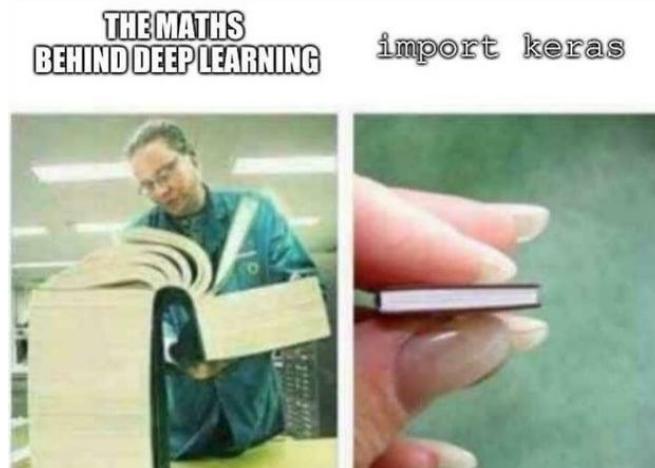




Comment programmer un
Deep Learning ?

DEUX GRANDS FRAMEWORKS

- Scikit Learn est une référence pour des algorithmes machine learning classiques
 - Régressions, arbres de décision
 - Il n'a pas été développé pour explorer les réseaux de neurones
 - Pas de support GPU, par exemple
- En ce moment, deux frameworks dominent le "marché" du Deep Learning
- Tensorflow
 - Initialement développé par Google
 - Bien testé et disposant d'une base importante d'outils
 - Dispose d'une API "haut niveau" : Keras
- PyTorch
 - Initialement développé par Facebook/Meta
 - Plus récent, support croissant
 - Quelques APIs permettent un usage haut niveau
 - Fast.ai, PyTorch Lightning, Skorch...



Keras

- Keras a été créé comme une interface "haut niveau" pour faire des réseaux de neurones
 - Jusqu'à la version 2.3 il supportait plusieurs types de bibliothèques
 - TensorFlow
 - Microsoft Cognitive Toolkit (CNTK)
 - Theano
 - PlaidML
 - Depuis la version 2.4 (Juin 2020), support prioritaireTensorflow
 - Ça rend Tensorflow un peu plus simple...
- La version 3 de Keras supporte également PyTorch

```
hidden1      = 100
hidden2      = 100

model = keras.Sequential([
    keras.layers.Input((28, 28)),
    keras.layers.Flatten(),
    keras.layers.Dense(hidden1, activation='relu'),
    keras.layers.Dense(hidden2, activation='relu'),
    keras.layers.Dense(10,      activation='softmax')
])

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

batch_size   = 512
epochs       = 16

history = model.fit(x_train, y_train,
                     batch_size      = batch_size,
                     epochs         = epochs,
                     verbose        = 1,
                     validation_data = (x_test, y_test))

score = model.evaluate(x_test, y_test, verbose=0)
```

KERAS : API SEQUENTIAL

- En Keras on a souvent utilisé l'API Sequential, qui est un simple empilement de couches
- Ex : API Sequential

```
model = keras.Sequential()
model.add( keras.Input(shape=(28,28)) )
model.add( keras.layers.Flatten() )
model.add( keras.layers.Dense( hidden1, activation='relu') )
model.add( keras.layers.Dense( hidden2, activation='relu') )
model.add( keras.layers.Dense( 10, activation='softmax') )
```

KERAS : API FUNCTIONAL

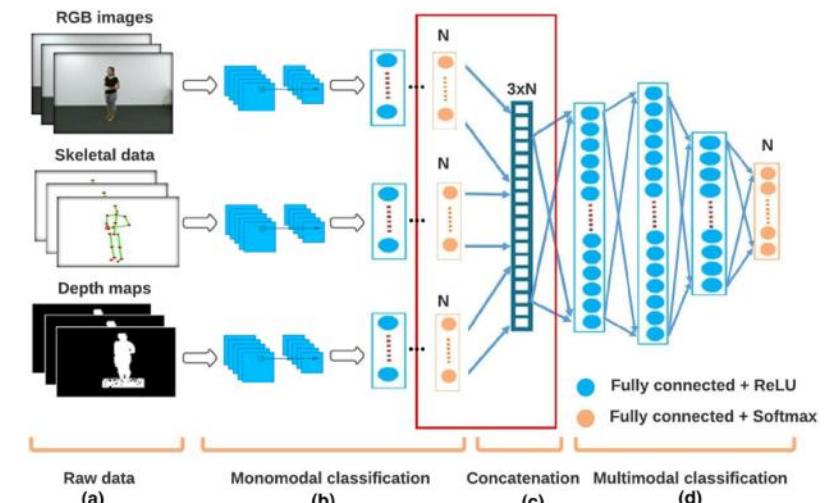
- Autre manière d'exprimer le modèle utilise l'API Functional

- Intéressante si on veut reutiliser des éléments ou créer des branchements

```
inputs = keras.Input(shape=(28,28))

x      = keras.layers.Flatten()(inputs)
x      = keras.layers.Dense( hidden1, activation='relu')(x)
x      = keras.layers.Dense( hidden2, activation='relu')(x)
outputs = keras.layers.Dense( 10,      activation='softmax')(x)

model = keras.Model(inputs=inputs, outputs=outputs)
```



KERAS : ENTRAÎNEMENT

- Pour la compilation et l'entraînement, on doit spécifier au moins une fonction de loss

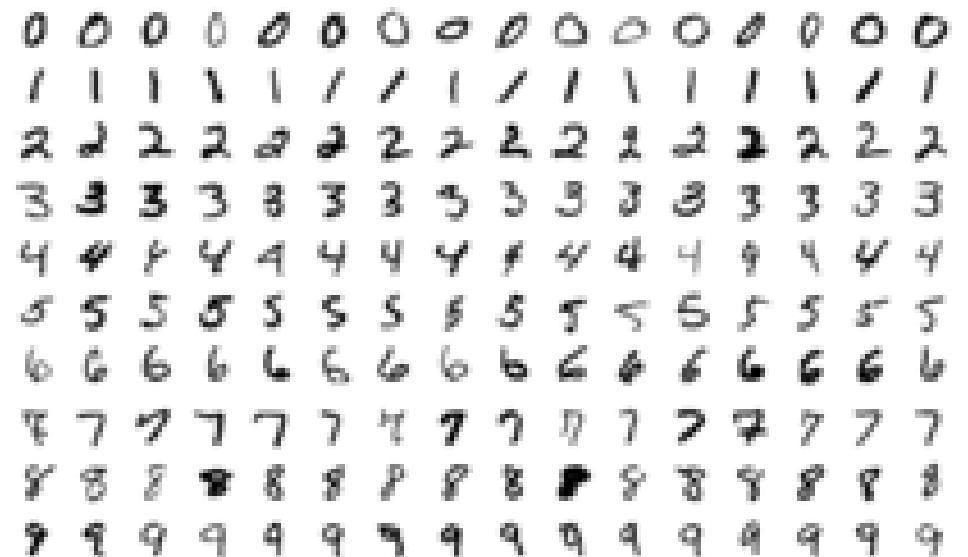
```
model.compile( optimizer = 'adam',
                loss      = [ categorical_crossentropy, mse ]
                loss_weights = [ 0.25, 1. ] )
```

- Également, on doit passer une liste lorsqu'on a plusieurs entrées

```
model.fit( [ x_train1, x_train2], [y_train1, y_train2],
            epochs = 10, batch_size = 64 )
```

EXERCICE 2

- Ici, nous allons apprendre à programmer avec Keras en reproduisant l'expérience d'Yan Lecun en 1998
 - Le dataset MNIST
- Utilisation de réseaux denses (DNN)
- Accéder le site <https://t.ly/iY4qG>



EXERCICE 3

- Et si on faisait de la classification de vêtements ?
 - Le dataset Fashion-MNIST (Zalando)
- Utilisation de réseaux convolutionnels (CNN)
- Accéder le site <https://t.ly/ceAZ->

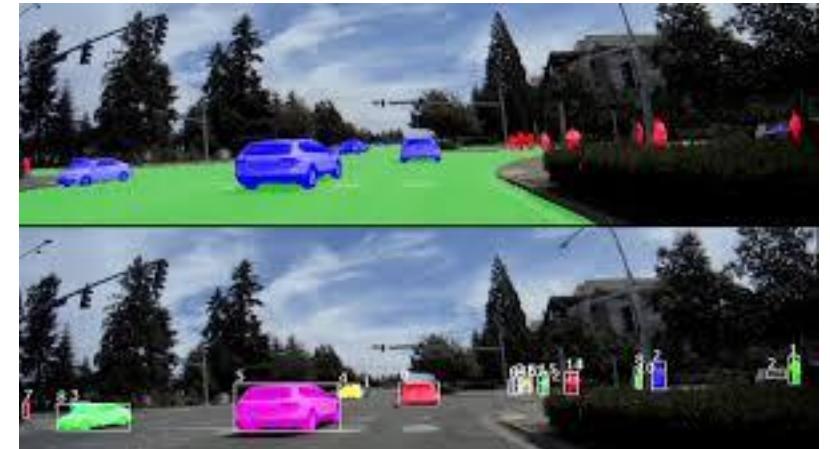


Reposer sur les
épaules
des géants



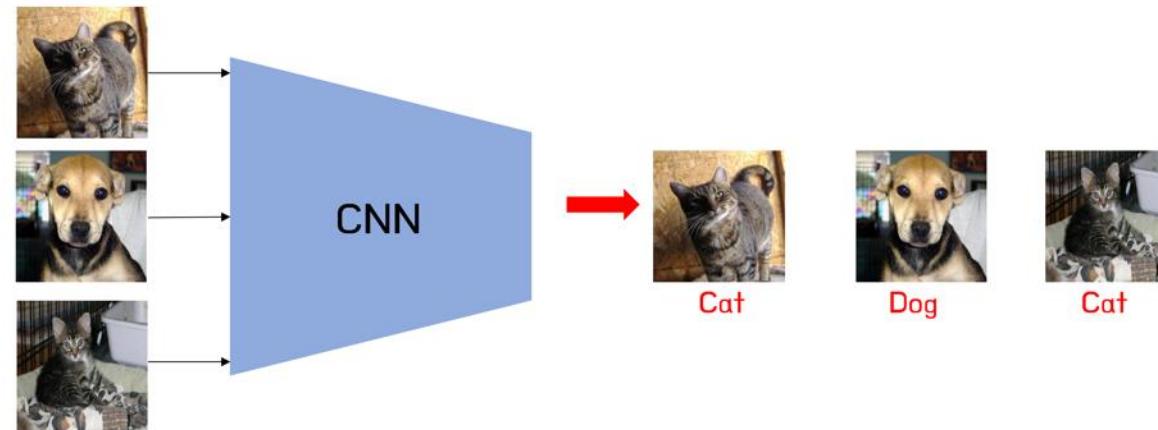
OPTION 1 : UTILISER UN MODÈLE PROCHE

- Plusieurs auteurs proposent des modèles extensivement entraînés sur des datasets publics
 - Ex : ImageNet -> 1000 classes, des millions d'images
- On peut facilement "emprunter" un modèle prêt et juste utiliser les classes qui nous intéressent
- Plusieurs sources disponibles
 - Tensorflow Hub
 - Nvidia NGC
 - Github
 - Une recherche sur Google...



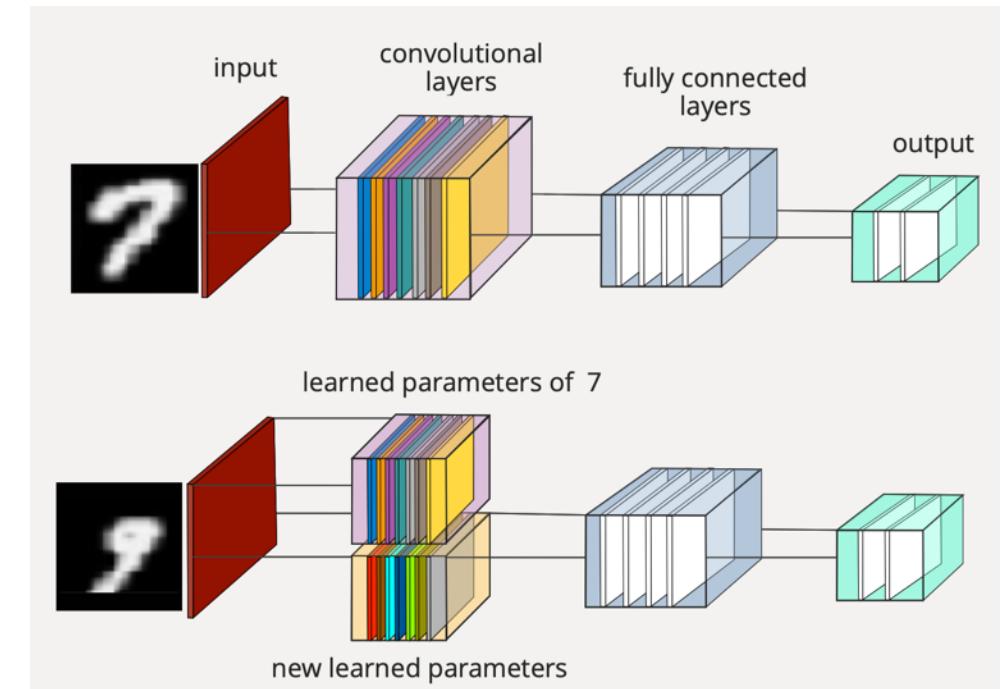
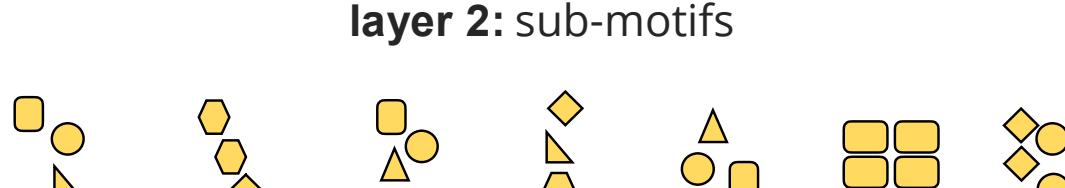
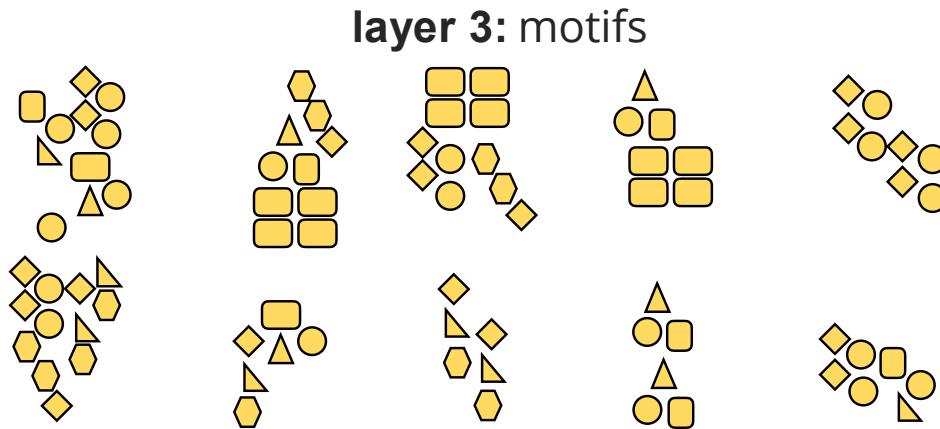
EXERCICE 4

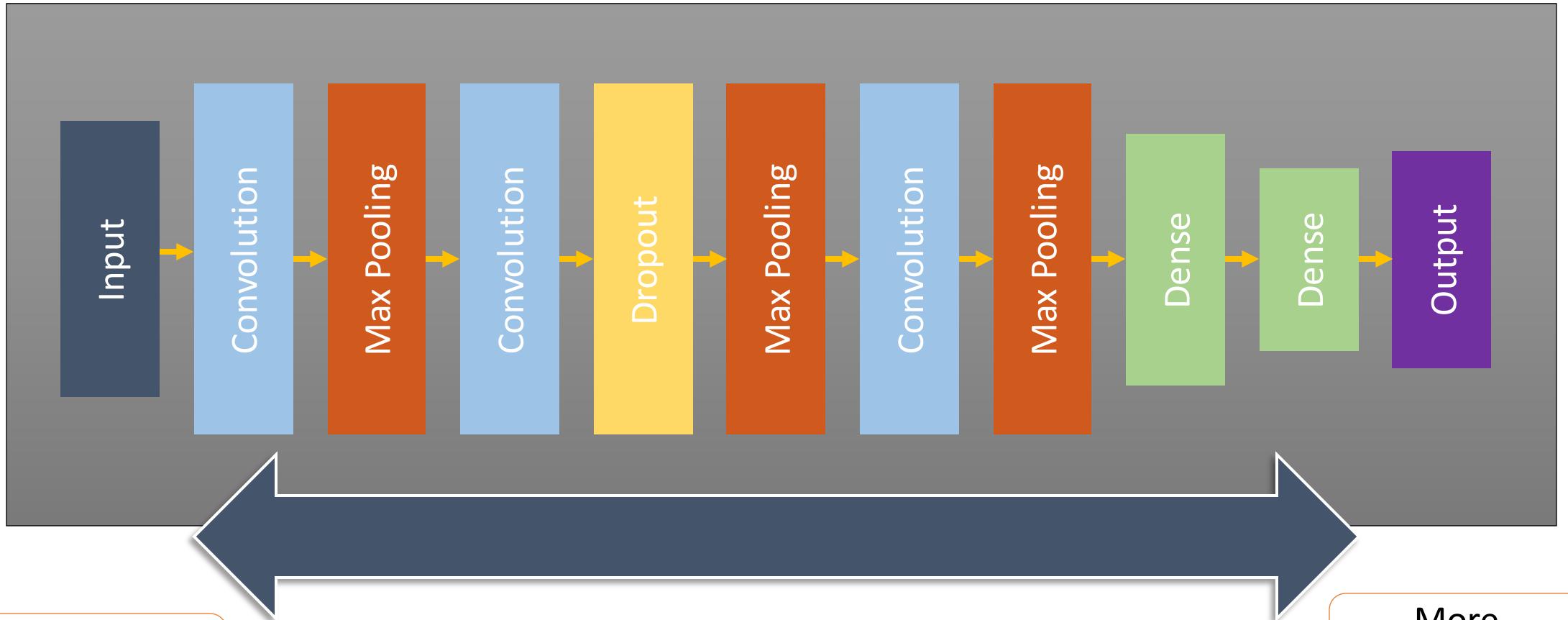
- Utiliser un modèle existant pour identifier Chiens x Chats
- Utilisation du modèle VGG 16
- Accéder le site <https://t.ly/NfirD>



OPTION 2 - TRANSFER LEARNING

- Les avantages d'une organisation hiérarchique des features

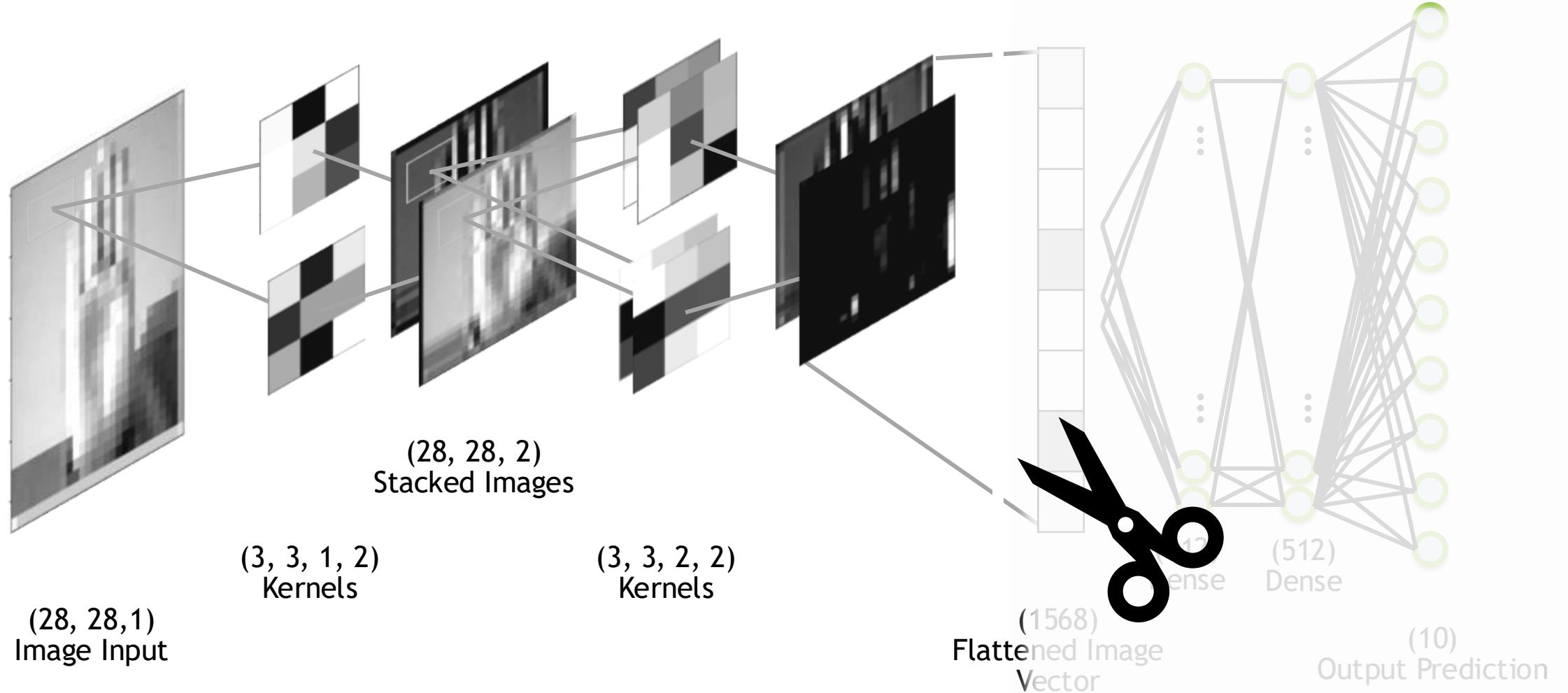




More
Generalized

More
Specialized

TRANSFER-LEARNING À PARTIR DE MODÈLES PRÉ-ENTRAÎNÉS



EXERCICE 5

- Mieux qu'identifier Chiens vs Chats, détecter un chien spécifique !!
 - Bo, le chien de Barack Obama
- Accéder le site <https://t.ly/kkbTz>



EXERCICE 6 (ANCIEN "DEVOIR 2023")

- Aider à organiser la défense civile
- Grâce à des images aériennes, identifier des zones touchées par un ouragan
- Comparer la performance de différents modèles de classification
- Accéder le site <https://t.ly/M5Bs1>



UN EXERCICE POUR LA MAISON

- On peut faire également des régressions avec les CNN.
- Exécuter l'exercice 07b (MNIST regression avec Keras)
 - <https://t.ly/babkB>
 - Proposer un cas d'étude où l'on pourrait appliquer ce type de régression
 - Ex : estimer la précipitation à partir d'une image radar