Introduction au deep learning

CONSULTING BSTORM

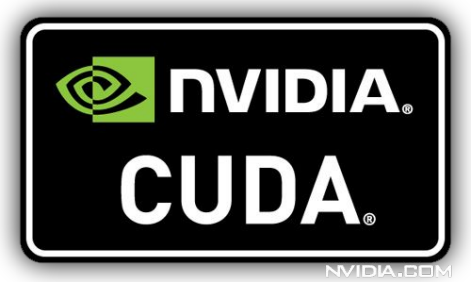
1. Introduction

1957 => naissance du perceptron

depuis 2015 => explosion des applications de deep learning:

- vision par ordinateur
- médecine
- finance
- …

1. Introduction



PyTorch

1. Introduction

Puissance de calcul:

CPU vs **GPU**

=> **CPU** sont très efficaces pour exécuter une succession d'opérations

=> **GPU** sont très efficaces pour exécuter des opérations **simultanément**

Le deep learning est **hautement parallélisable**

1. Introduction

Les données:

=> Internet a rendu possible l'**échange** de données

=> Intérêt croissant pour les données et leur collecte

=> Nouvelles technologies (e.g. NoSQL)

Le deep learning nécessite de beaucoup de données pour être efficace **><** machine learning classique

1. Introduction

Les librairies **open source**:

TensorFlow (Google):

- première version: 2015
- TF 2.0: septembre 2019

PyTorch (Facebook):

- première version: 2016

=> Outils très jeunes

1. Introduction

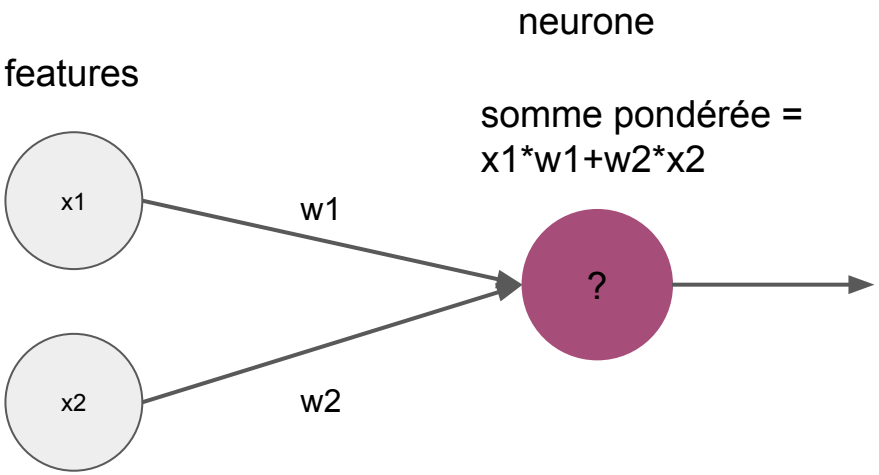
Le deep learning pour:

=> De gros volume de données

=> Des données peu ou pas structurées (image, son, texte)

=> Pour générer de nouvelles données

2. Le neurone



2. Le neurone

Un neurone est constitué:

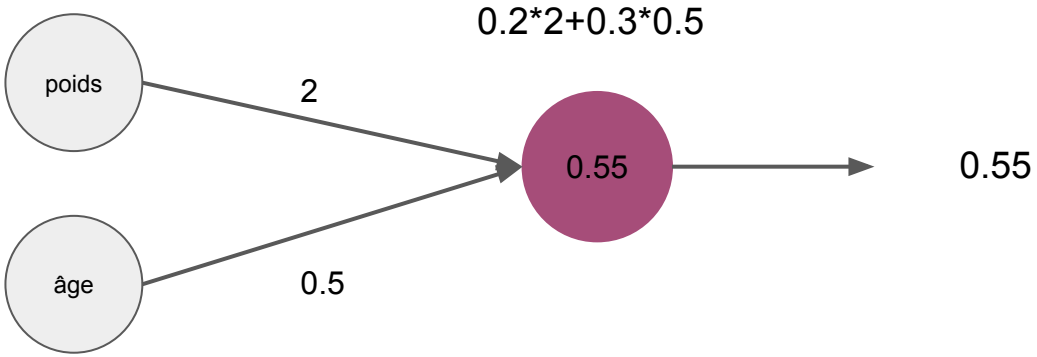
- d’entrées
- et de poids associés à ces entrées
- une somme pondérée (somme des entrées*leur poids)

Imaginons une base de données avec des enfants et adultes accompagnés de leur poids et âge

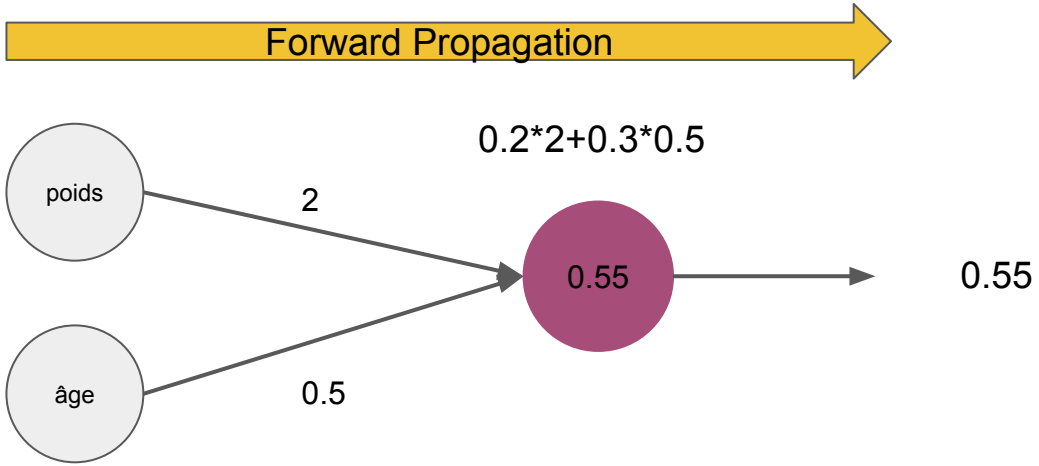
On souhaiterait que notre neurone retourne selon l’âge et le poids:

- 1 pour des adultes
- 0 pour des enfants

2. Le neurone

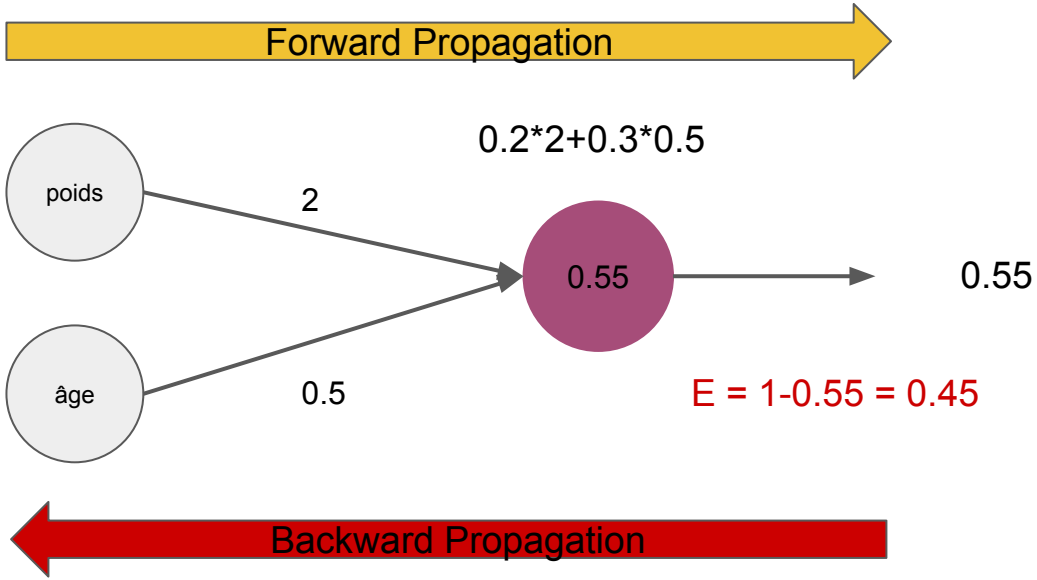


2. Le neurone



On obtient 0.55, autrement dit, notre neurone ne parvient pas à faire de différence entre un enfant et un adulte

2. Le neurone

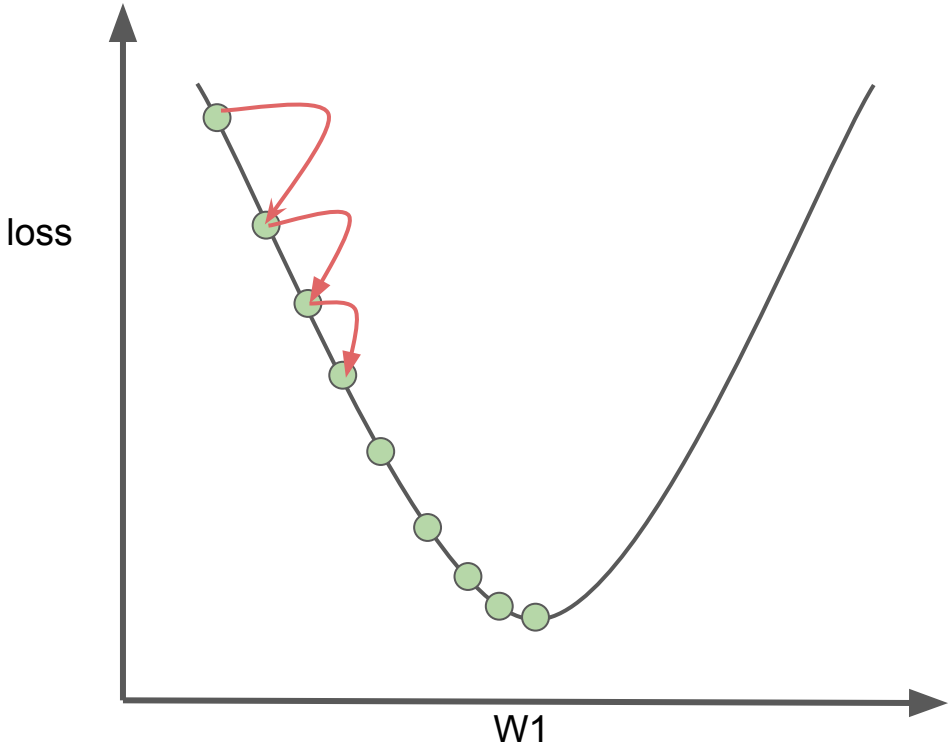


On calcule une erreur de manière à modifier nos poids

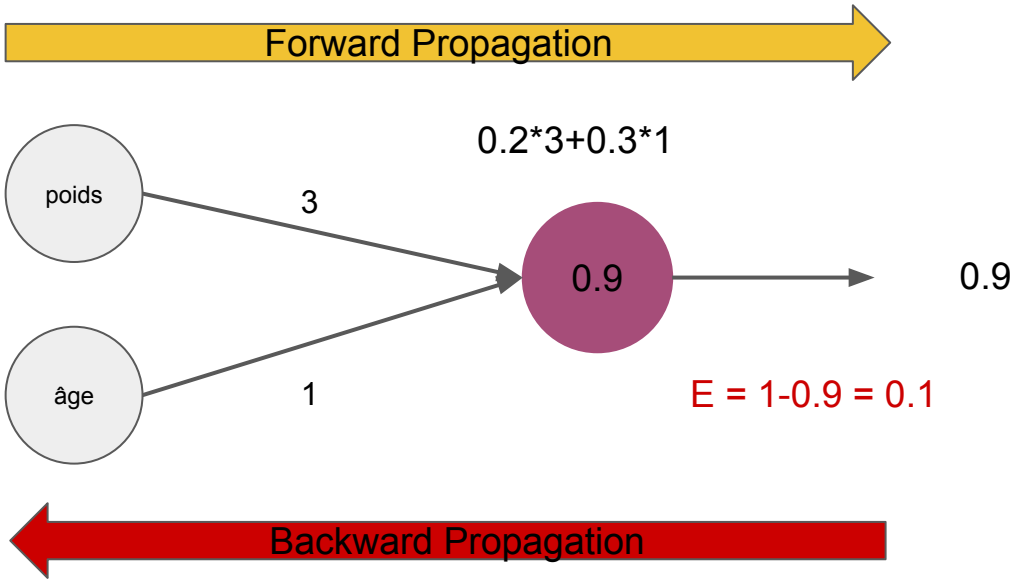
On répète cette opération jusqu'à obtenir de petites erreurs

=> Descente de gradient

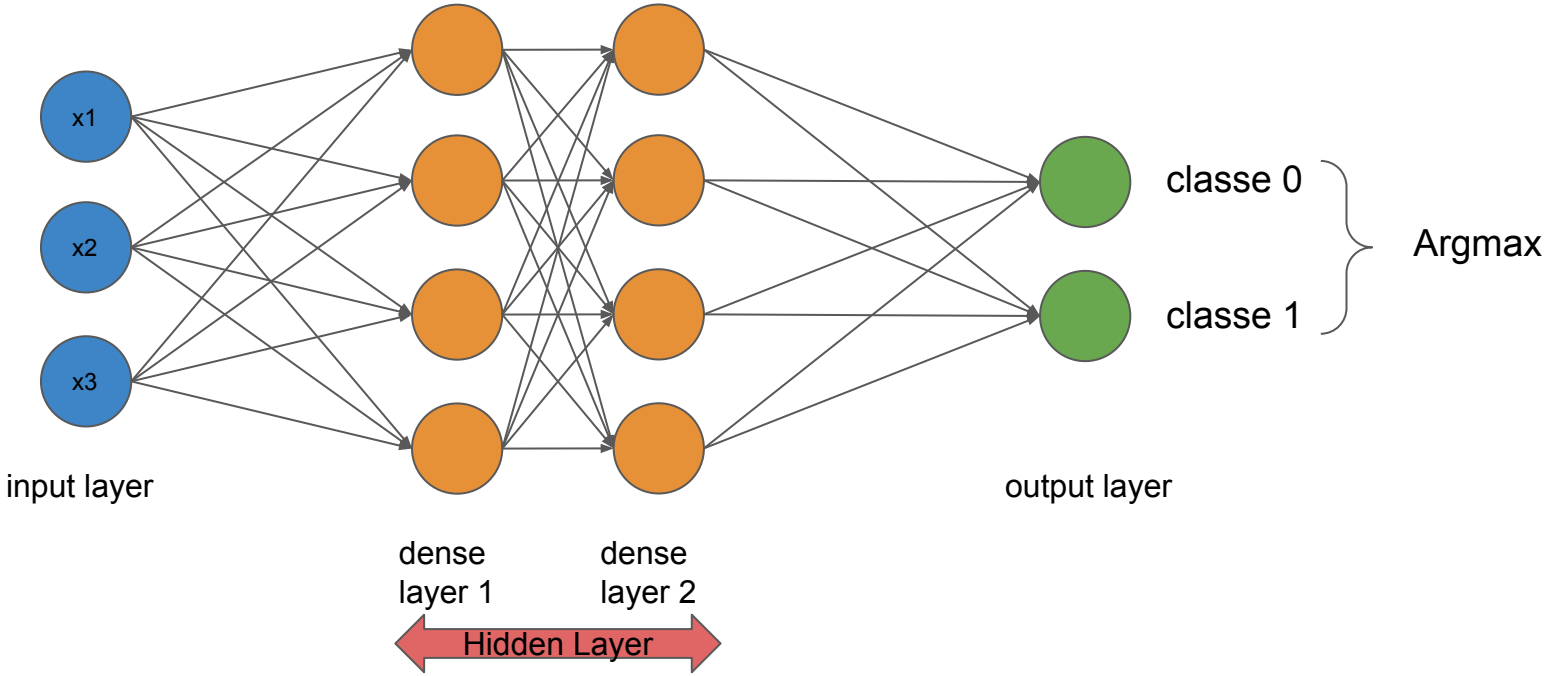
2. Le neurone



2. Le neurone


Forward Propagation
0.2*3+0.3*1
poids
3
0.9
0.9
âge
1
E = 1-0.9 = 0.1
Backward Propagation

3. Les réseaux de neurones



4. Batch, batch size, epoch

En pratique, on ne calcule pas une erreur après chaque sample, on préfère travailler par batch de samples avant de calculer une erreur et de corriger les poids

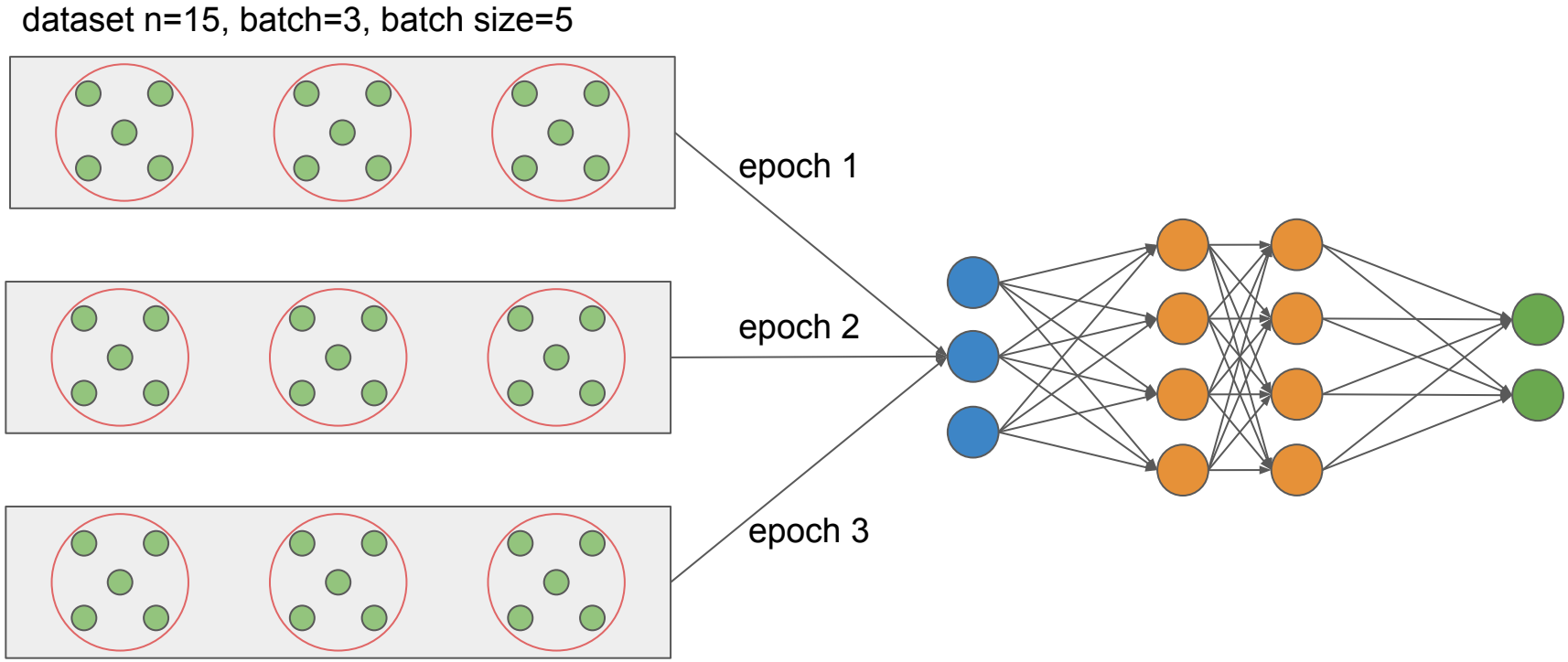
batch = un sous ensemble du dataset

batch size = le nombre de samples par batch

Aussi, on passe plusieurs fois notre dataset dans le réseau

epochs = le nombre de fois que le réseau ‘voit’ les données

4. Batch, batch size, epoch



5. Les fonctions d'activation

Nous avons vu comment calculer la somme pondérée

Il est possible de moduler cette somme avec des fonctions d'activations:

- Linéaire
- Sigmoïde
- Softmax
- Tangente hyperbolique
- ReLU (Rectified Linear Unit)
- Leaky ReLU

Pourquoi ? => Non linéarité, normalisation des sommes pondérées

5.1. Sigmoïde et softmax

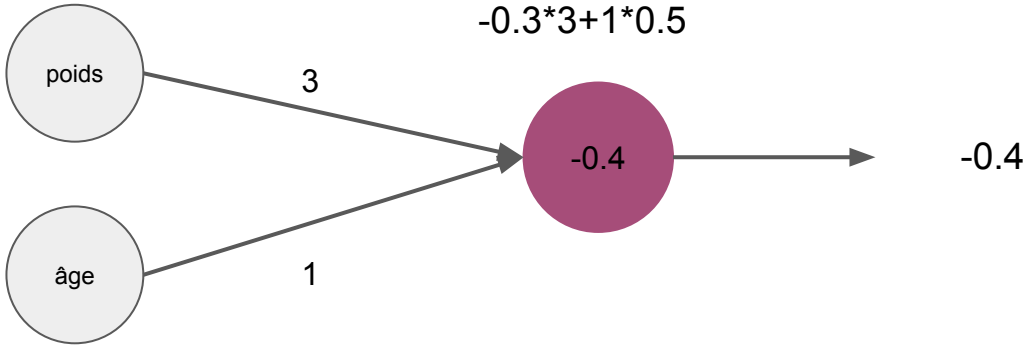
Dans notre exemple du perceptron, nous partions du principe que la valeur est sortie était comprise entre 0 et 1

En réalité, la sortie d'un neurone n'est pas par défaut comprise entre 0 et 1

Une autre question se pose également, que faire si nous avons plus que deux classes ?

=> Il faut pouvoir normaliser la sortie à l'aide d'une fonction d'activation

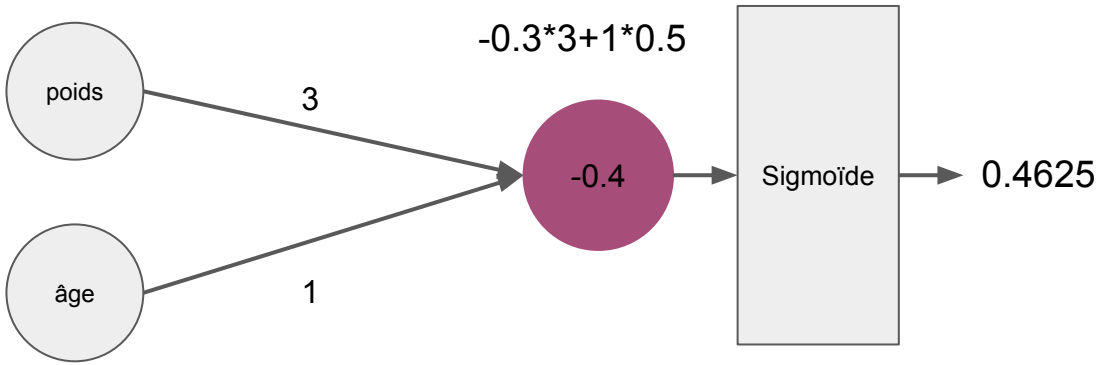
5.1. Sigmoïde et softmax



Si on change la valeur de nos entrées on obtient -0.4

=> difficile à interpréter

5.1. Sigmoïde et softmax

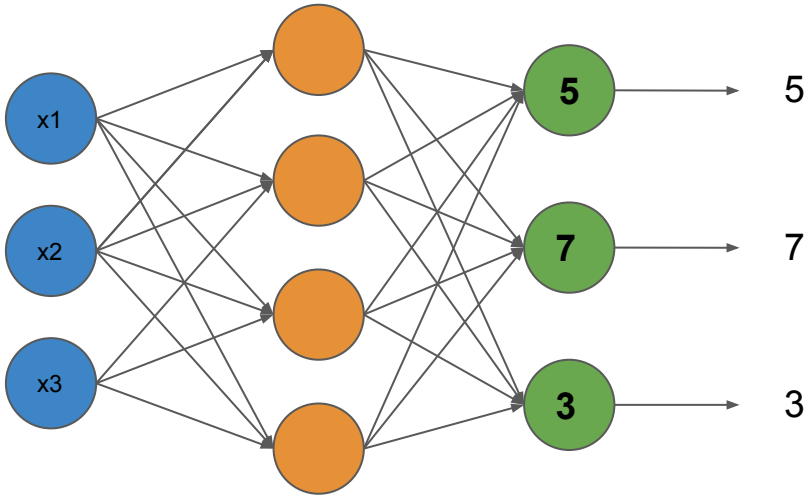


La fonction sigmoïde permet de remettre les valeurs sur une échelle comprise entre 0 et 1

=> Interprétable

=> classification binaire (soit 0 soit 1)

5.1. Sigmoïde et softmax

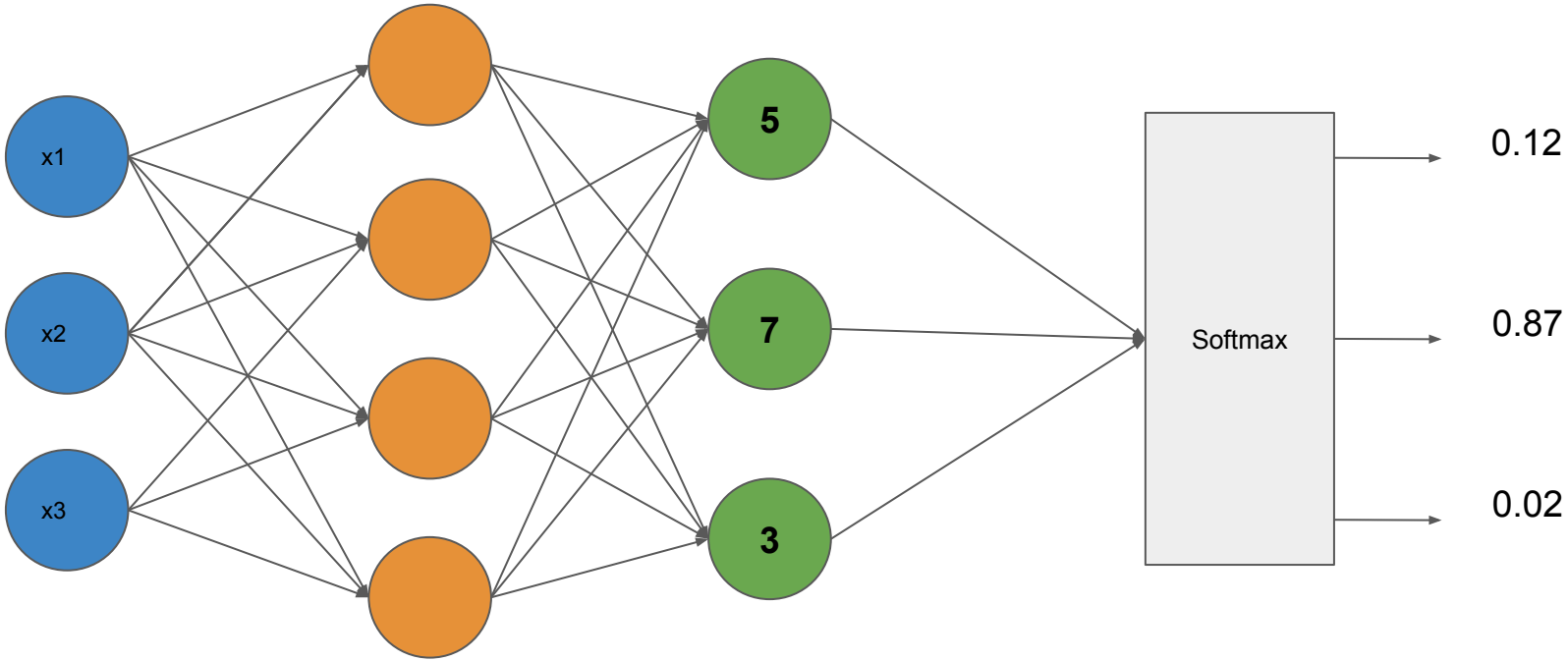


On ne peut plus utiliser la fonction sigmoïde au delà de 2 classes

Il faut trouver une autre solution pour exprimer ceci sous forme de probabilité

=> softmax

5.1. Sigmoïde et softmax

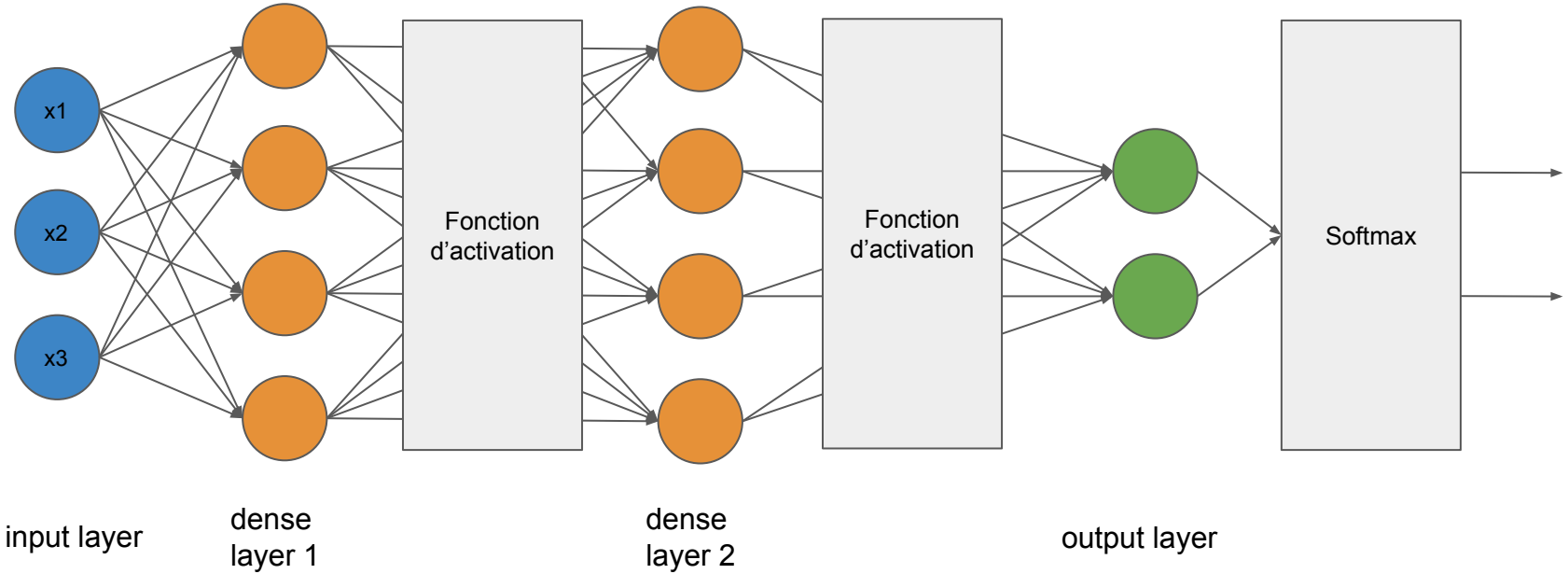


5.2. Non linéarité

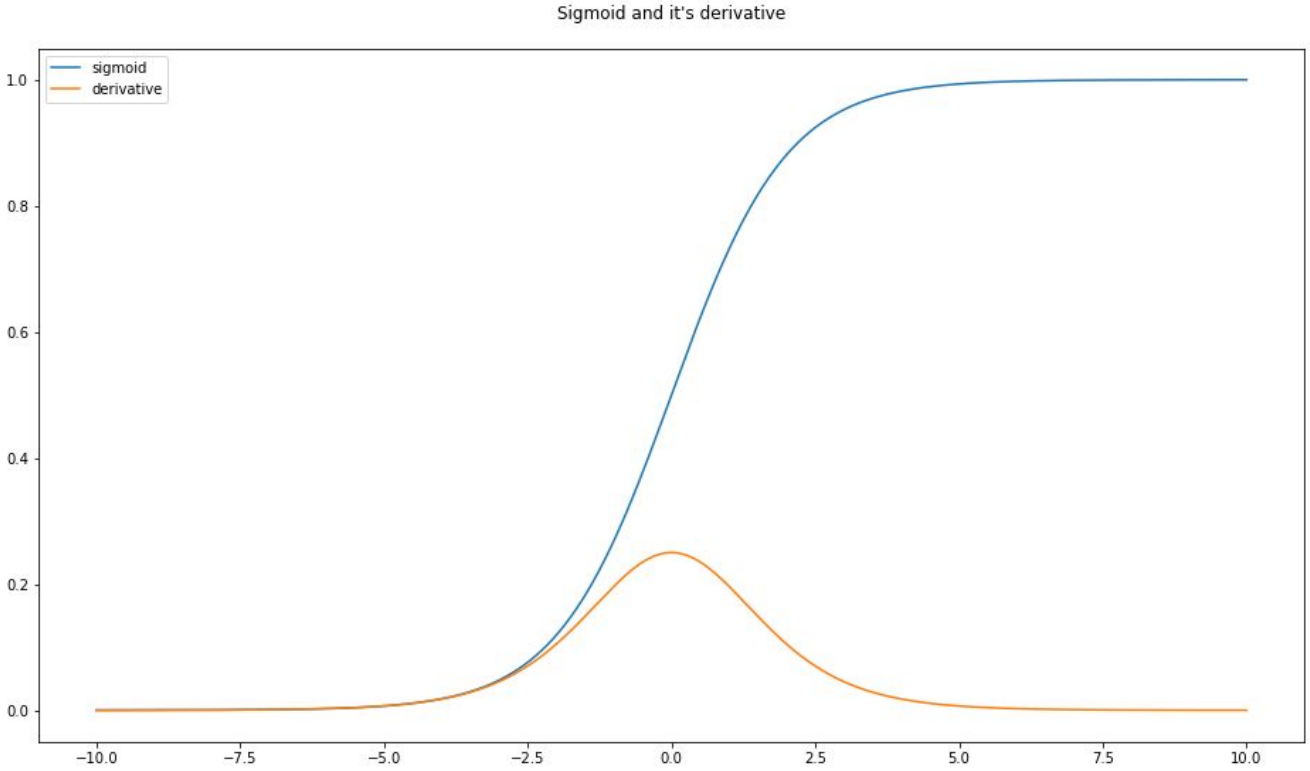
Jusqu'ici on utilisait les fonctions que pour moduler la couche de sortie mais on peut également les utiliser pour moduler les sorties des couches cachées

=> pour apporter de la non linéarité

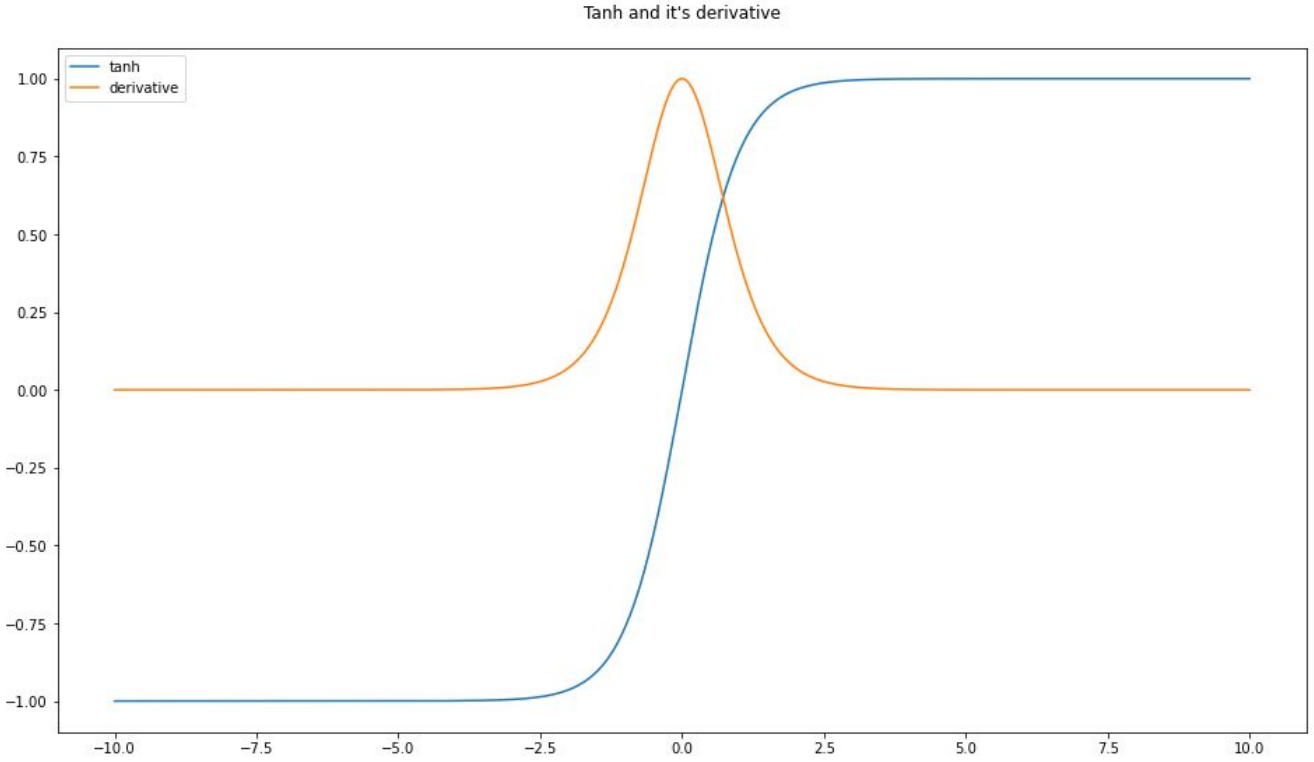
5.2. Non linéarité



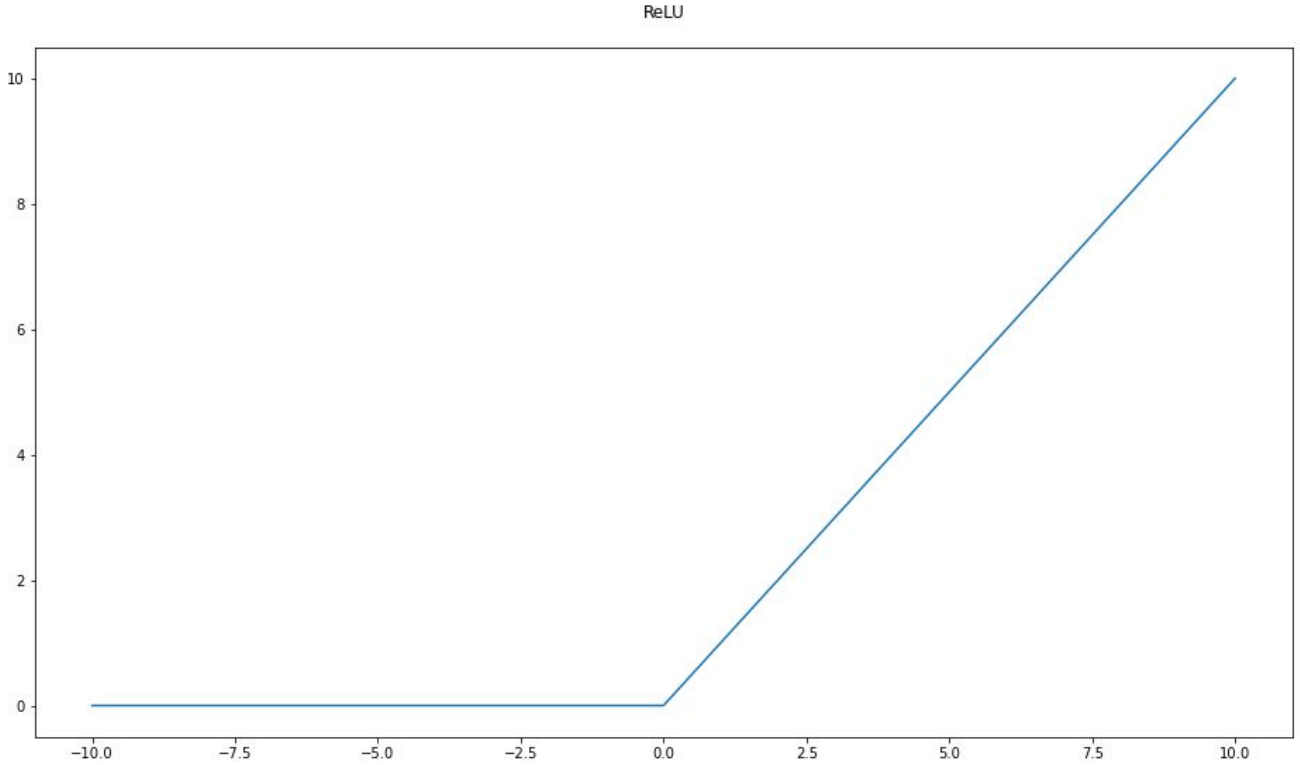
5.2. Non linéarité



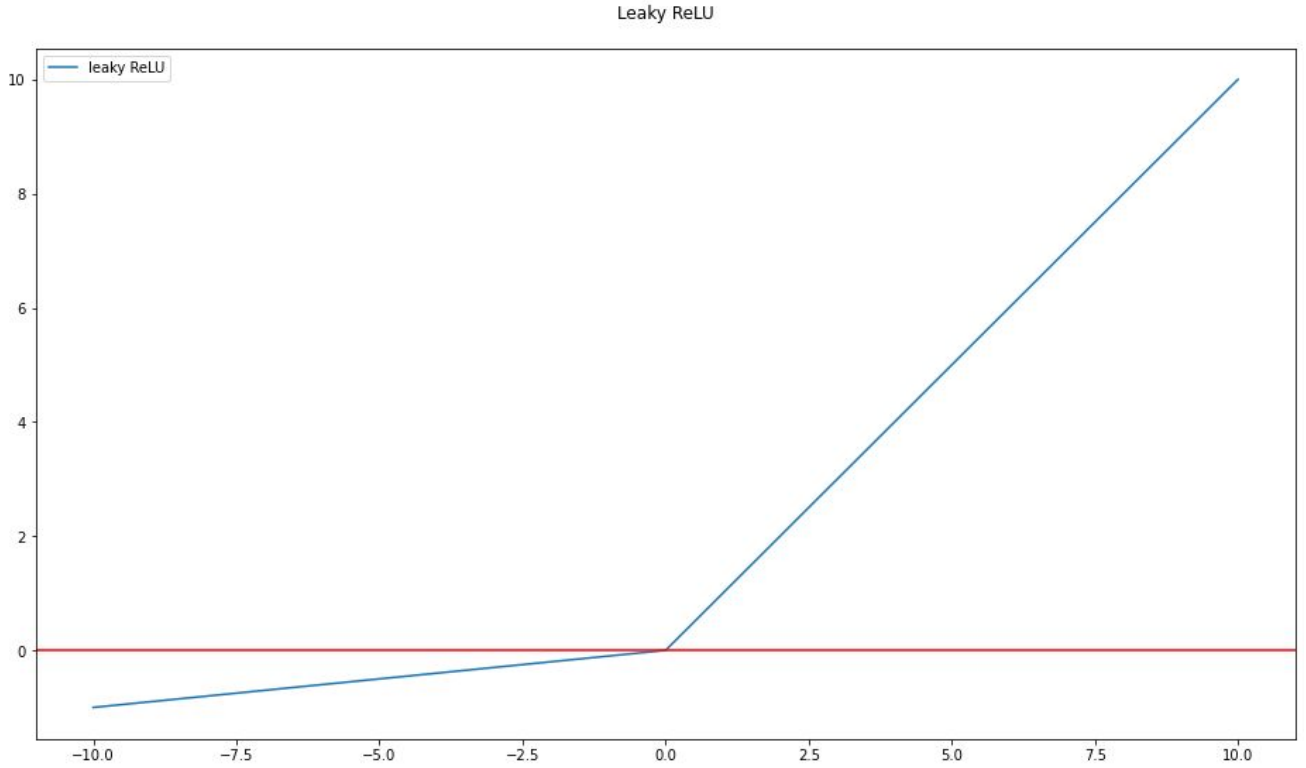
5.2. Non linéarité



5.2. Non linéarité



5.2. Non linéarité



6. Les architectures

Il existe des architectures de réseaux de neurones particuliers:

=> Convolution Neural Network (**CNN**)

=> Recurent Neural Network (**RNN**)

=> Long Short Term Memory (**LSTM**)

=> Generative Adversarial Network (**GAN**)

=> ...

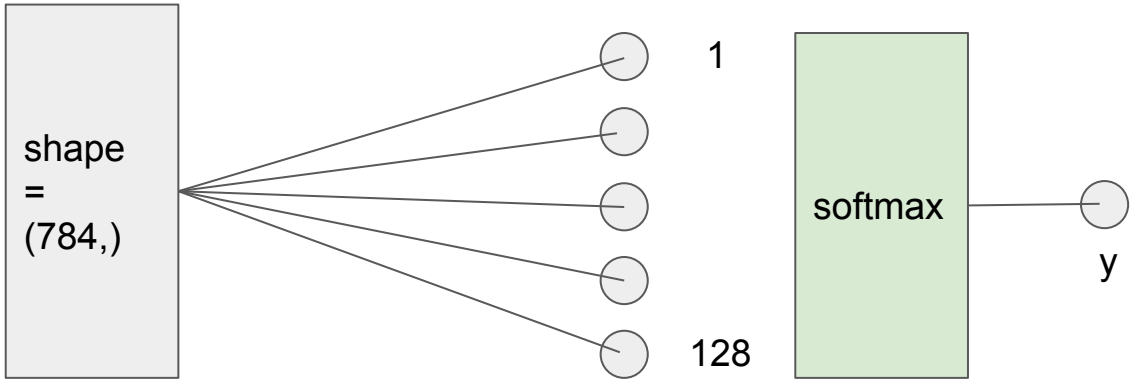
CNN

Convolutional Neural Networks

1. Introduction

Fashion mnist noir et blanc:

- shape = (28,28,1)



1. Introduction

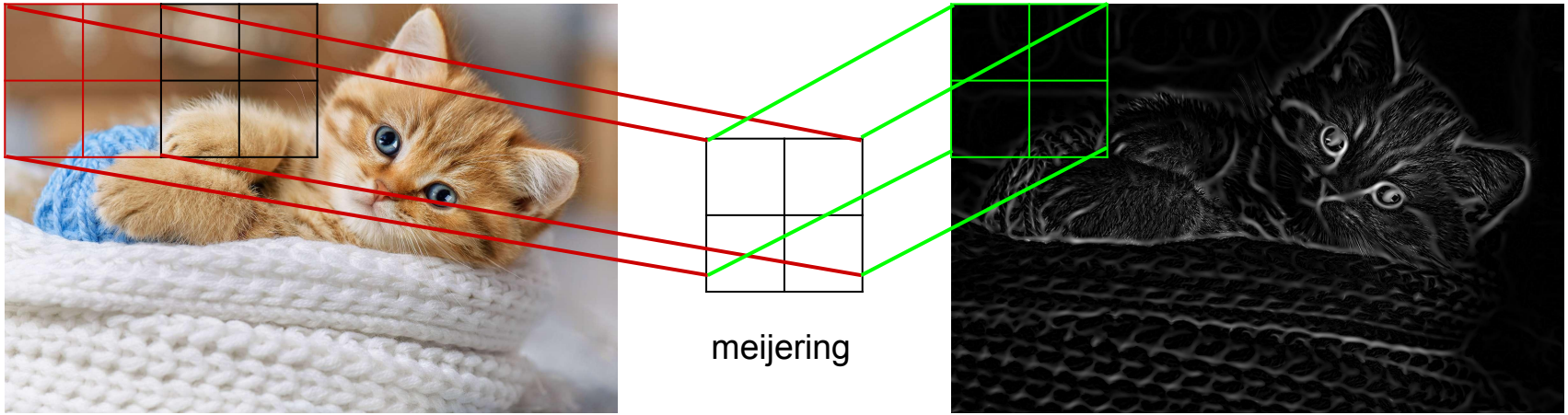
L'approche avec de simples couches denses pose un problème. En effet, le nombre de paramètres à calculer explose très vite:

- Des images avec une shape (28,28,1) connectées à un couche dense de 128 unités:
  - 28*28*1*128+128 = 100480 paramètres
- Image shape (28,28,3):
  - 28*28*3*128+128 = 301184
- Image shape = (224,224,3)
  - 224*224*3*128+128 = 19267712

2. La convolution

Une convolution désigne le fait de balayer une image avec un filtre

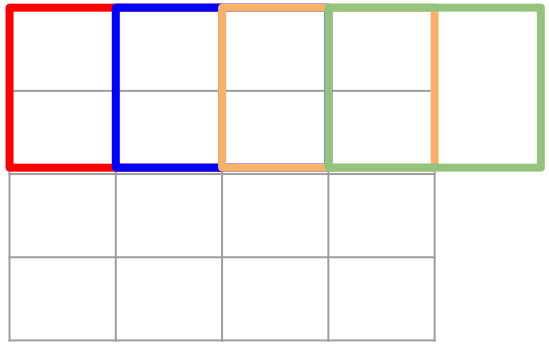
Ce filtre est petit réseau de neurones qui cherche à “résumer” l’image



2. La convolution

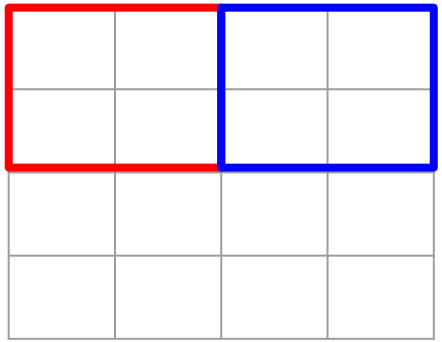
Le stride: désigne le décalage du filtre, à quelle vitesse parcourir l'image

Stride = 1



kernel size = (2,2)

Stride = 2

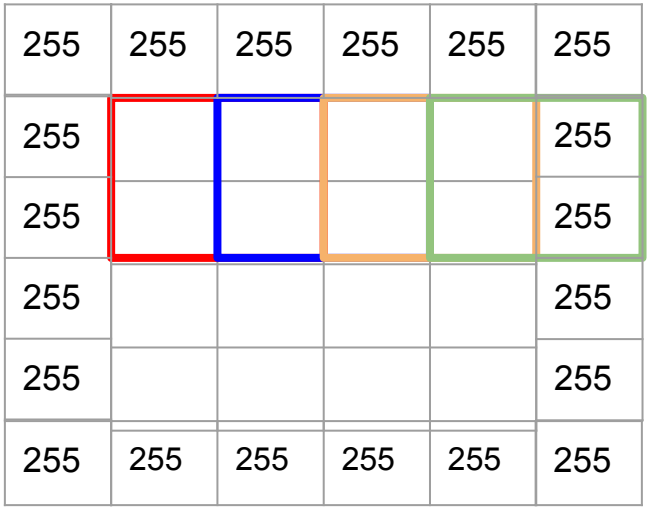


kernel size = (2,2)

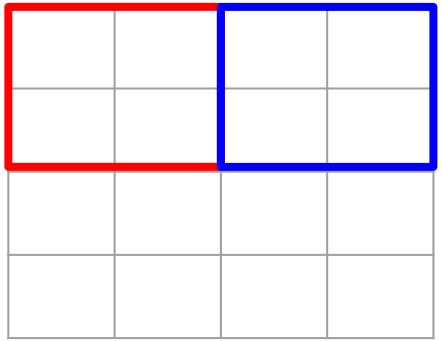
2. La convolution

Le padding consiste à ajouter des pixels afin de conserver

padding = ‘valid’



padding = ‘same’

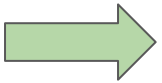


2. La convolution

Le pooling désigne l'opération de regrouper des pixels

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

Max Pooling



5	8
6	9

shape pooling = (2,2)

3. CNN

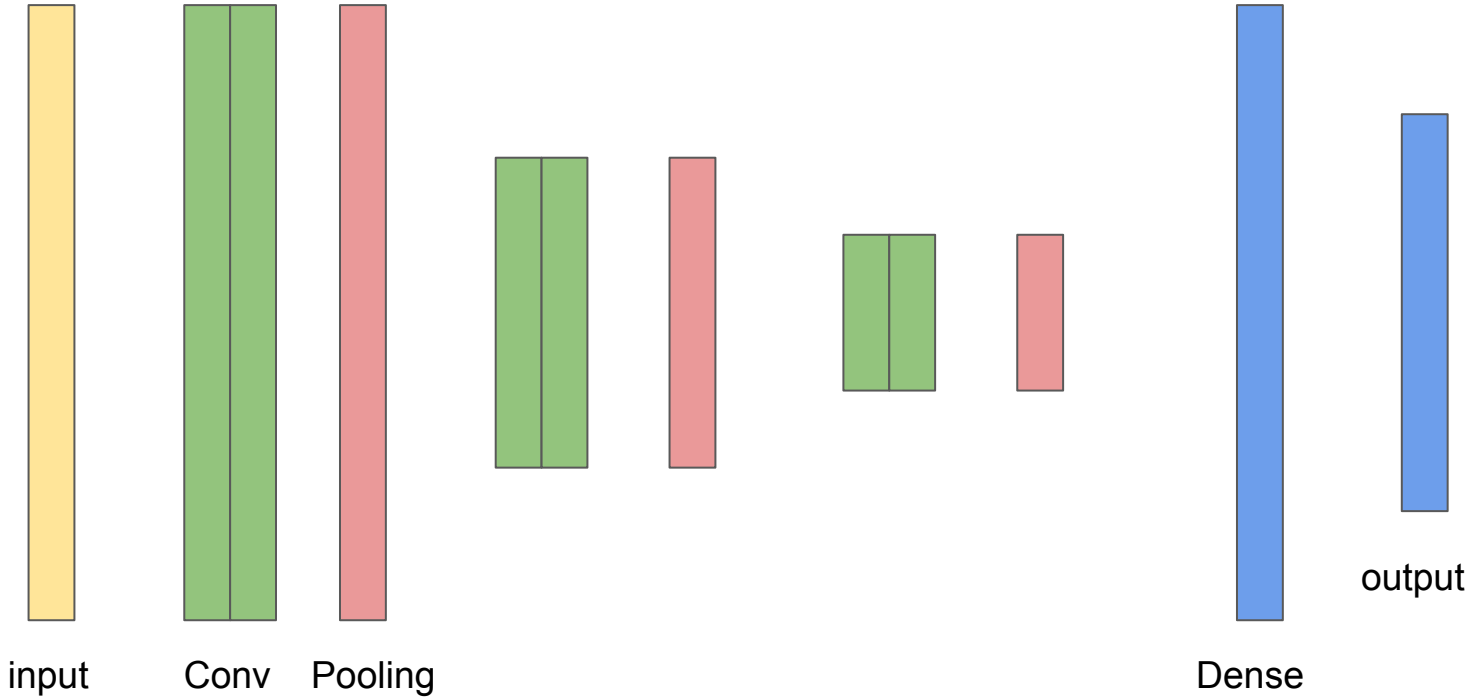
En pratique:

- On constitue plusieurs filtres
- On peut enchaîner les convolutions
- Le pooling permet de réduire la quantité de pixels
- On relie toutes nos convolutions à une simple couche dense

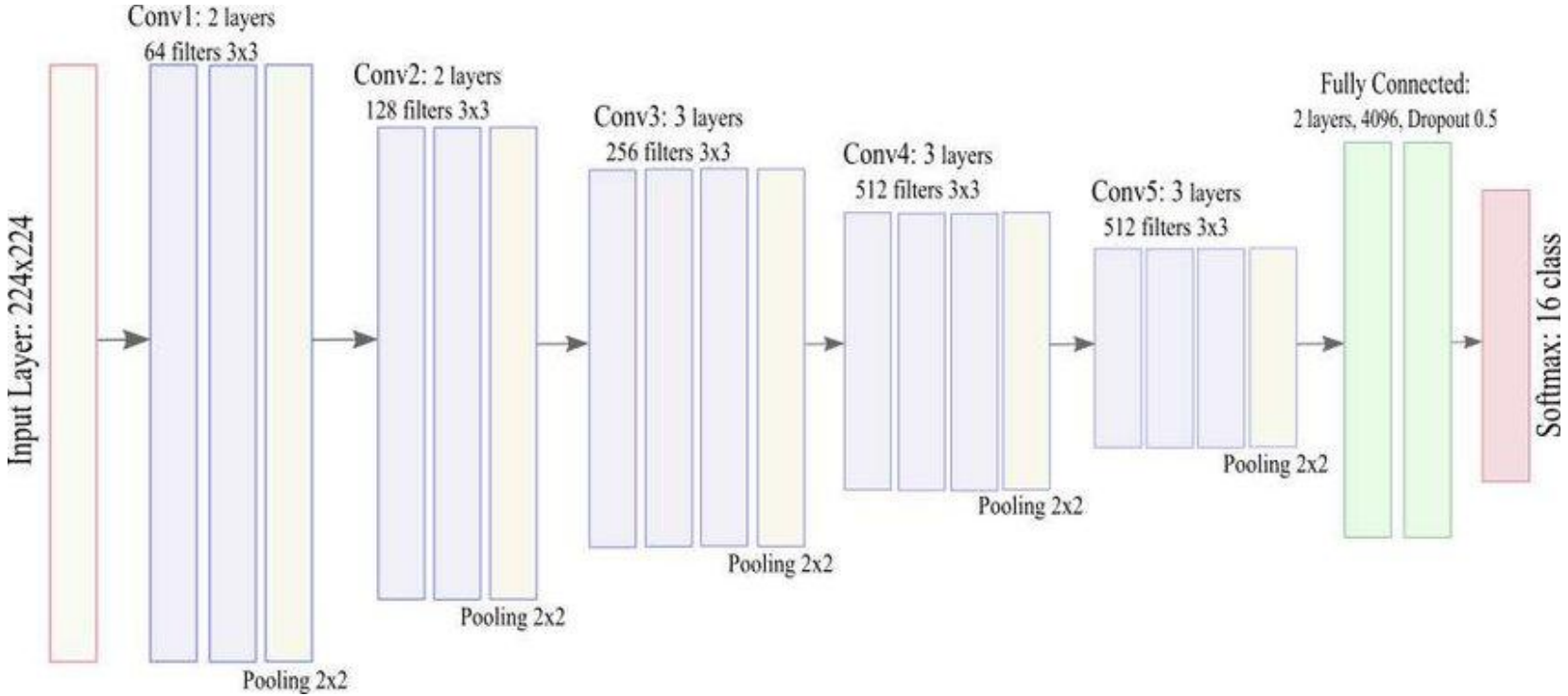
=> Permet d'extraire les données importantes des informations.

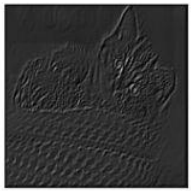
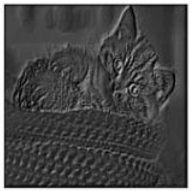
Ex: **VGG16**

3. CNN



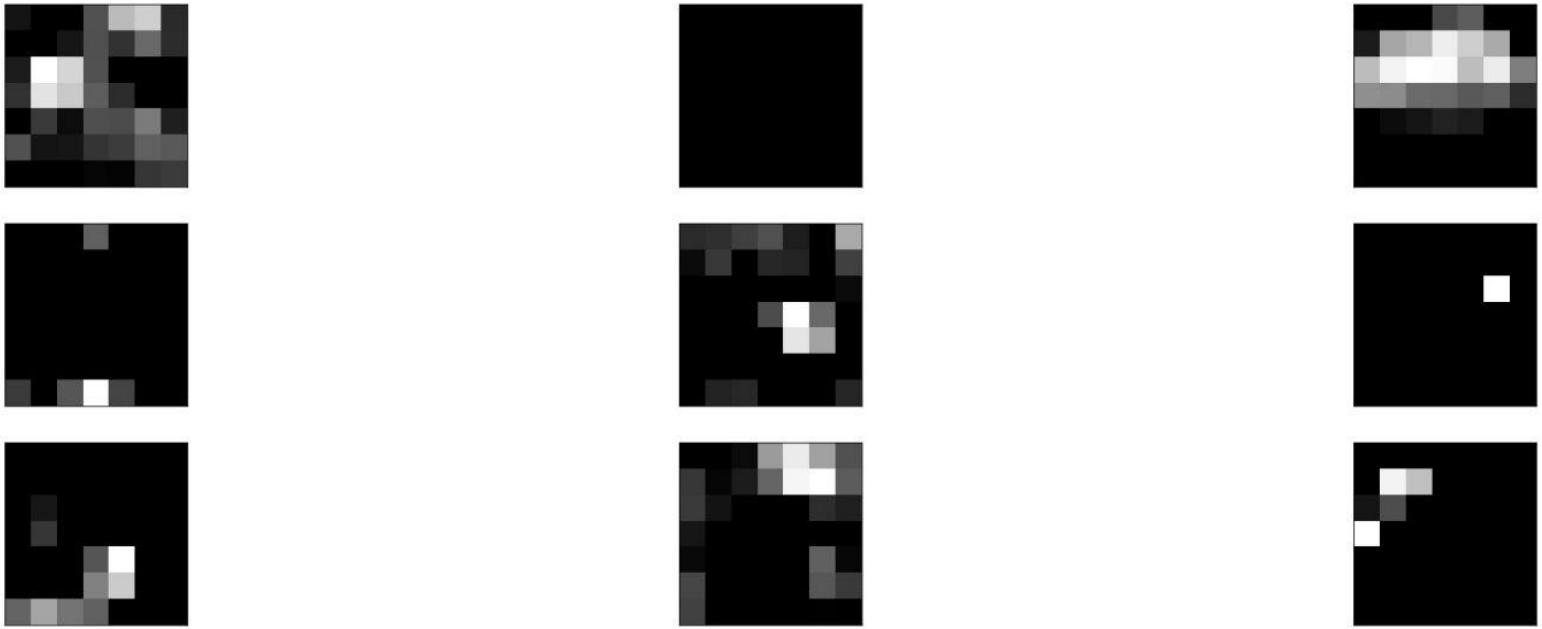
3. CNN


Input Layer: 224x224
Conv1: 2 layers
64 filters 3x3
Pooling 2x2
Conv2: 2 layers
128 filters 3x3
Pooling 2x2
Conv3: 3 layers
256 filters 3x3
Pooling 2x2
Conv4: 3 layers
512 filters 3x3
Pooling 2x2
Conv5: 3 layers
512 filters 3x3
Pooling 2x2
Fully Connected:
2 layers, 4096, Dropout 0.5
Softmax: 16 class





4. Data Augmentation

La Data Augmentation permet de créer des copies des images originales tout en les modifiant un peu.

=> Permet de réduire l'overfitting

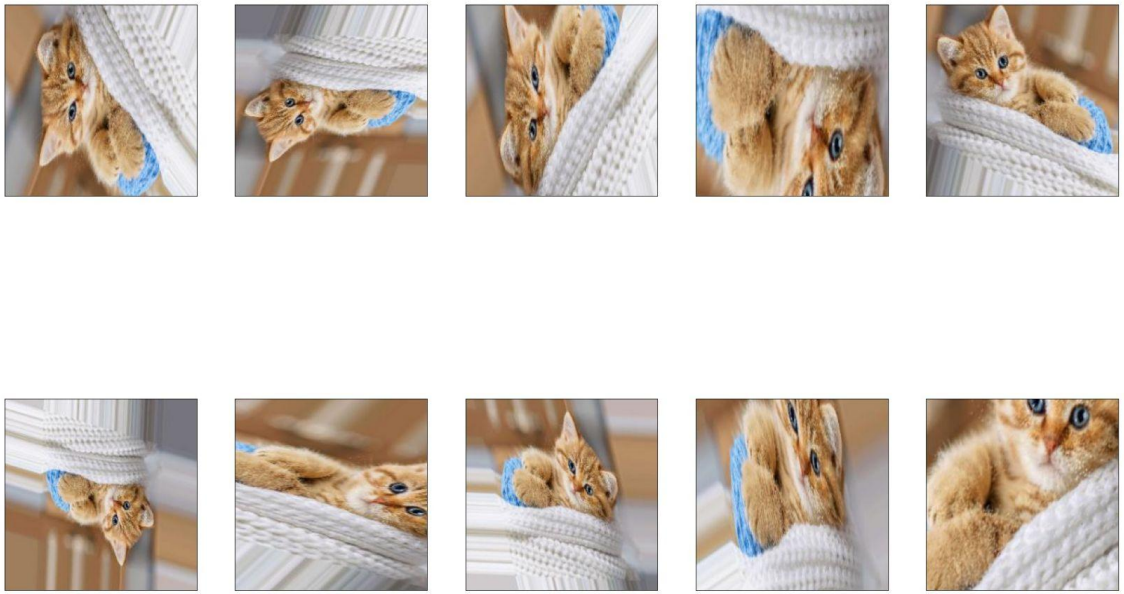
=> Permet d'obtenir artificiellement plus de données

Comment modifier des images:

- rotation
- inverser haut/bas, droite/gauche
- distorsion
- Inverser les canaux de couleurs

4. Data Augmentation

A chaque epoch, le modèle est entraîné avec des images qui diffèrent légèrement.



5. Autres architectures

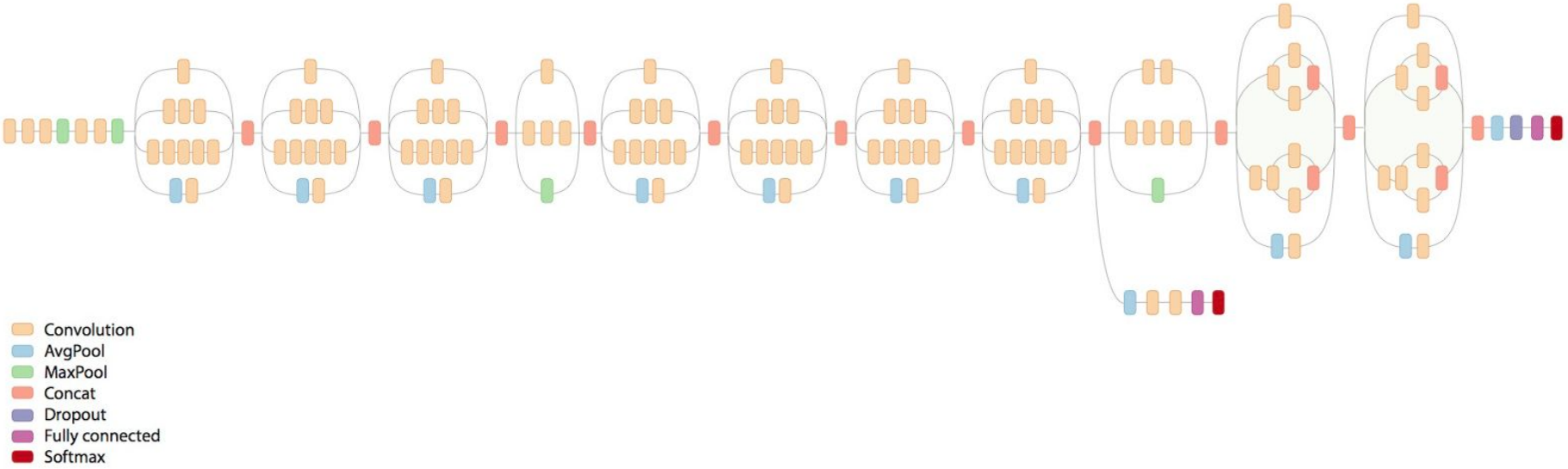
Jusqu'ici, nous avons utilisé des modèles séquentiels relativement simples mais il existe d'autres architectures plus complexes.

Ces architectures ne sont pas de simples modèles séquentiels et ne peuvent se construire qu'en passant par l'API fonctionnelle de Keras.

- inception
- resnet
- mobilenet
- CNN + XGBOOST

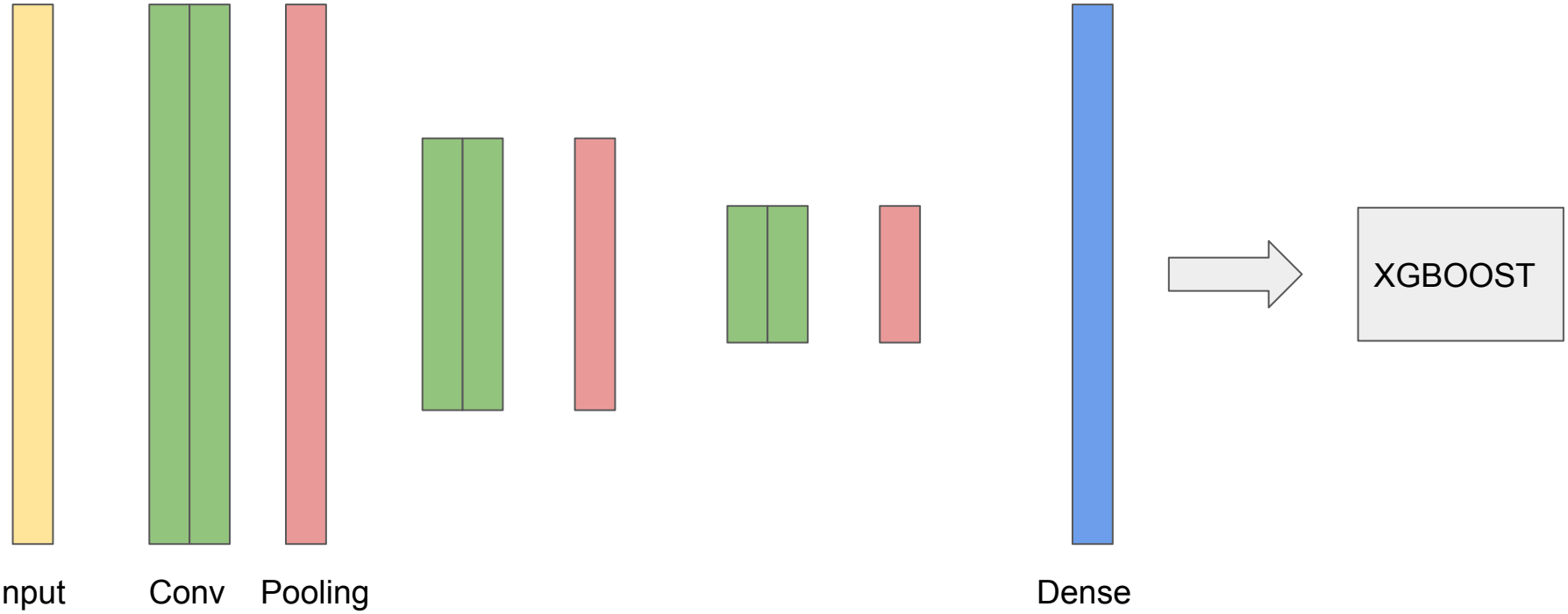
5. Autres architectures

InceptionV3



5. Autres architectures

CNN + XGBOOST



AE

auto encoders

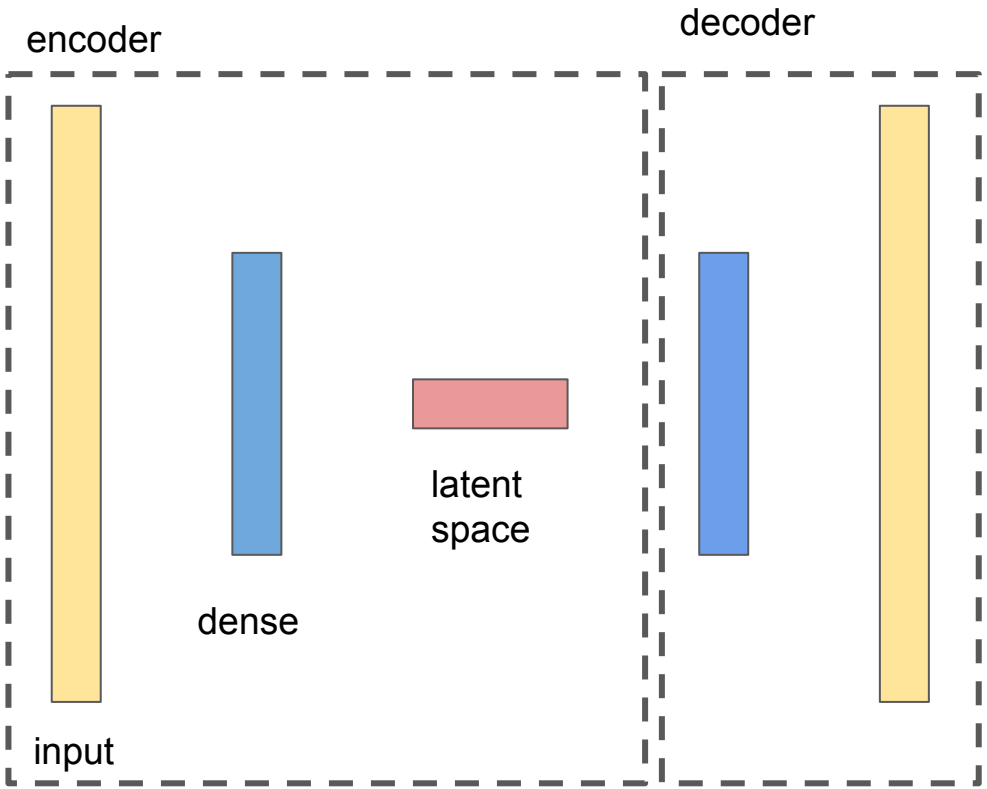
1. Introduction

Les auto encodeurs correspondent à une architecture particulière de réseau de neurones qui possède deux entités:

- L'encodeur qui a pour vocation de projeter l'information dans un nouvel espace réduit (espace latent). Il s'agit donc d'une réduction de la dimensionnalité
- Le décodeur reconstruit l'information sur base de l'espace latent

=> Il s'agit d'un apprentissage non supervisé

1. Introduction



Représentation classique d'un AE avec une couche d'entrée projetée dans un espace latent à l'aide d'une simple couche dense et ensuite reconstruite avec une autre couche dense

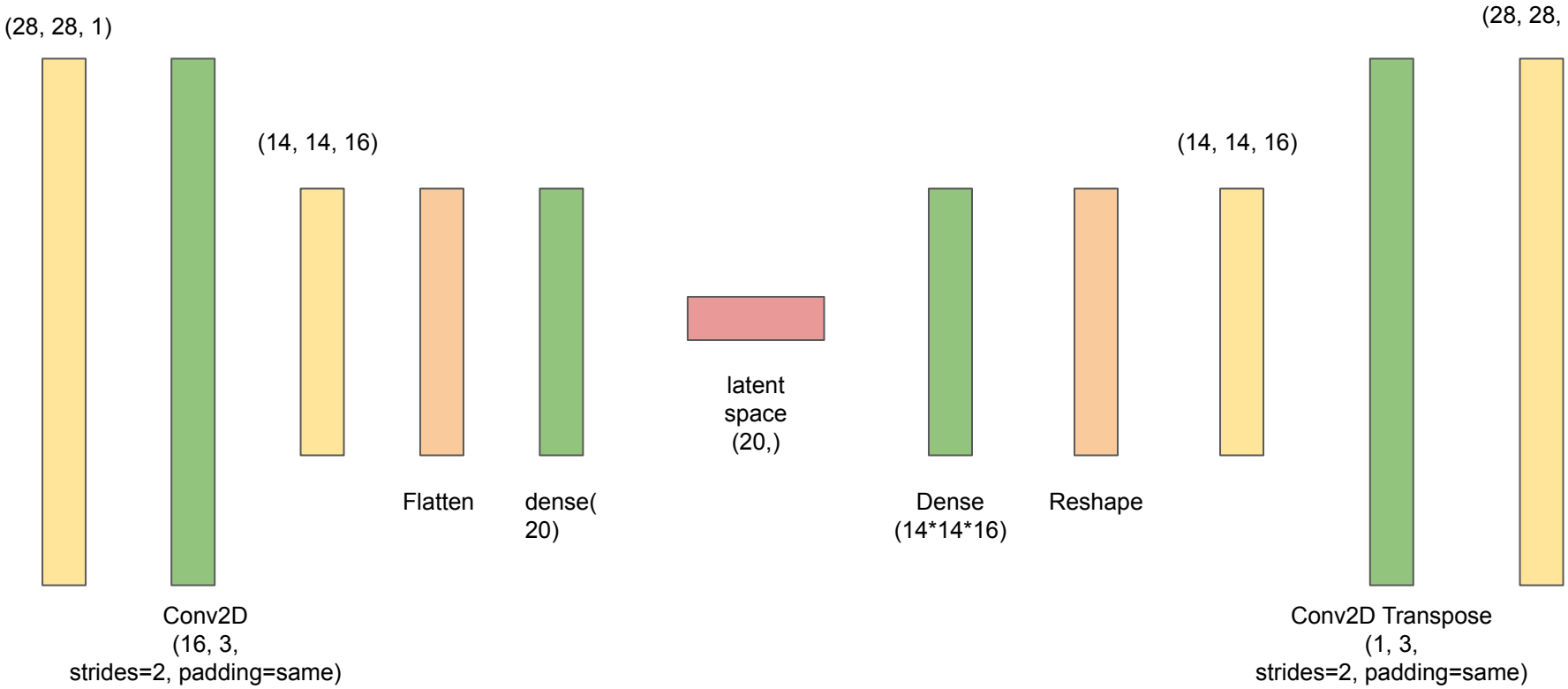
2. Application aux images

Dans le cadre des images, les auto encoders peuvent servir de débruiteur

L'encoder sera constitué de couches de convolution avec comme objectif de projeter notre image dans un espace latent

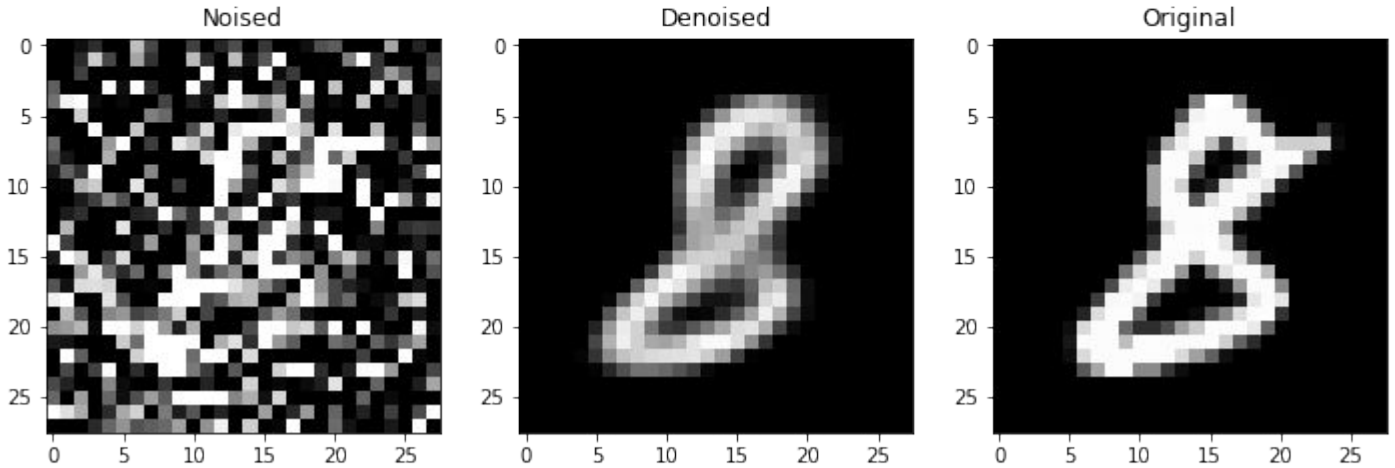
Le decodeur lui reconstruira une image débruitée

2. Application aux images



2. Application aux images

8



Autres architectures

intuition, exemples

1. RNN

Capacité à gérer des séquences:

- Des lettres, des mots
- des images
- du son, des paroles
- …

1. RNN

Capacité à gérer des séquences:

- Des lettres, des mots
- des images
- du son, des paroles
- …

1. RNN

Capacité à gérer des séquences:

- Des lettres, des mots
- des images
- du son, des paroles
- …

1. RNN

Capacité à gérer des séquences:

- Des lettres, des mots
- des images
- du son, des paroles
- …

1. RNN

=> Il faut avoir une mémoire pour prédire le déplacement d'un objet

Où les utilise-t-on ?

- Reconnaissance vocale
- voiture autonome
- génération de texte
- génération de musique

LSTM, BI-LSTM sont des **RNN** améliorés

2. GAN

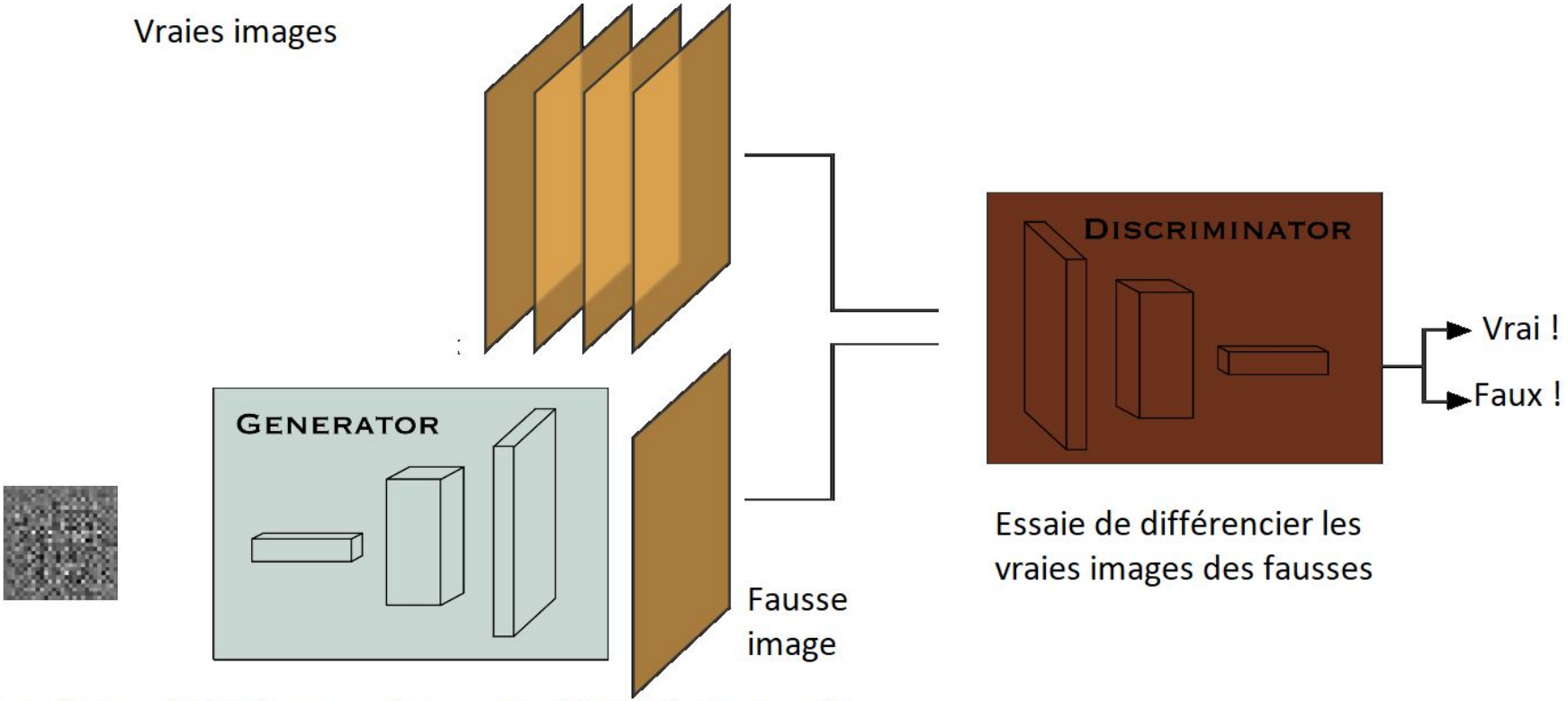
Architecture récente (2014)

L'idée est simple on met deux réseaux en compétition:

- Un générateur
- Un discriminateur

L'objectif du générateur est de produire de l'information que le discriminateur ne pourra pas identifier comme fausse

2. GAN


Vraies images
GENERATOR
DISCRIMINATOR
Vrai !
Faux !
Fausse image
Essaie de différencier les vraies images des fausses
Transforme du bruit en une image aussi "vraie" que possible

2. GAN

Où les utilise-t-on ?

- Deepfake:
  - Vidéo 1
  - Vidéo 2
- Faux visages
- Génération de collections de mode
- Modélisation 3D (Architecture, chimie, pharmacie)
- ...

3. Reinforcement Learning

Cas particulier puisque le réseau de neurone doit se débrouiller:

- sans donnée
- sans règle

Les données sont inhérentes à l'environnement

On ne fournit qu'une seule chose: une **récompense**

3. Reinforcement Learning

Difficile à mettre en oeuvre dans la réalité:

- simulation
- réalité

Très souvent appliqué au jeu:

E.g. société DeepMind:

- AlphaGo
- Starcraft 2