



Introduction à l'Apprentissage Statistique

Réseau de Neurones

M2 Actuariat – ISFA – 2021/2022

Pierrick Piette
Actuaire à Seyna
pierrick.piette@gmail.com

Seyna.

In God we trust, all others bring data
- William Edwards Deming



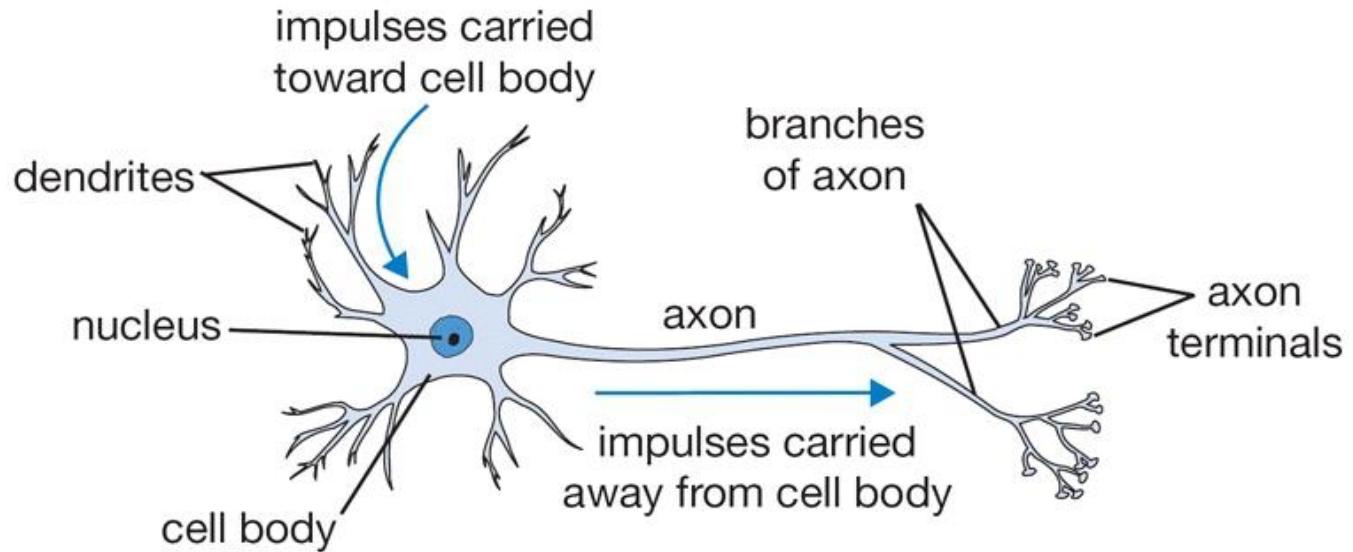


Perceptron

Inspiration : le cerveau

Objectif : simuler le comportement du cerveau humain

- Approche connexionniste des neurones
- 10^{11} neurones
- 10^4 connexions pour chaque neurone



Historique

L'idée initiale est ancienne !

- Modèle simple : McCulloch & Pitts (1943)
- Perceptron : Rosenblatt (1961)
- Fonction d'activation : Minsky & Papert (1969)

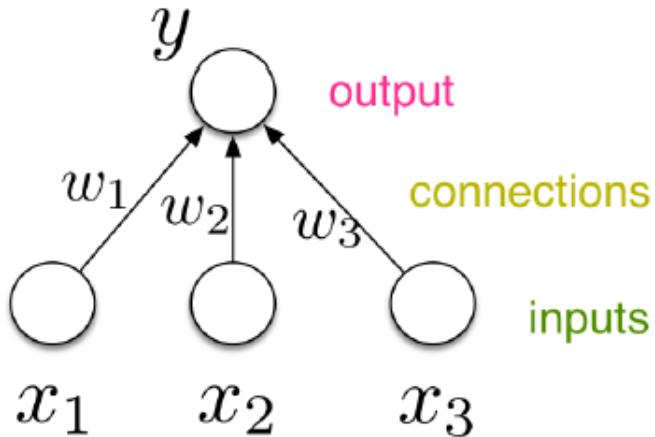
... mais il faut attendre les développements de l'informatique, dont l'utilisation de GPU vers 2010, pour des structures plus complexes

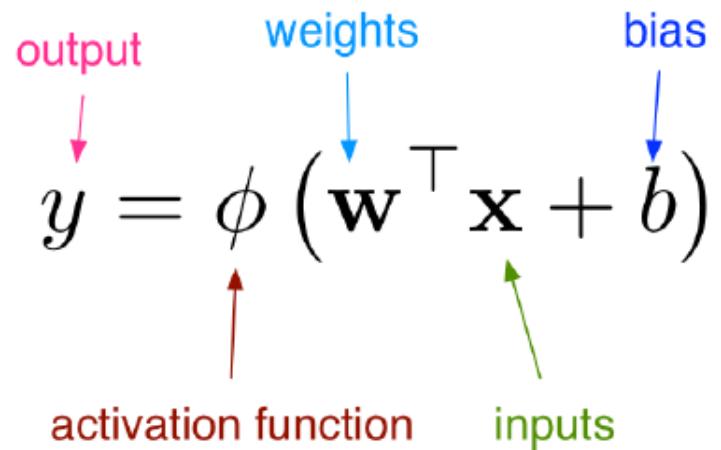
- Retropropagation de l'erreur : Dreyfus (1973)
- Convolutional Neural Network : Fukushima (1980)
- LSTM : Hochreiter & Schmidhuber (1995)
- Yann LeCun, Andrew Ng

Perceptron

Le perceptron est la brique élémentaire

- Equivalent au neurone dans le cerveau
- L'assemblage de ces briques permet de créer un réseau de neurones



$$y = \phi(\mathbf{w}^\top \mathbf{x} + b)$$


A diagram illustrating the mathematical formula for a perceptron. The formula is $y = \phi(\mathbf{w}^\top \mathbf{x} + b)$. The components are labeled as follows:

- output**: pink arrow pointing down to y .
- weights**: blue arrow pointing down to \mathbf{w}^\top .
- bias**: blue arrow pointing down to b .
- activation function**: red arrow pointing up to ϕ .
- inputs**: green arrow pointing up to \mathbf{x} .

Fonctions d'activation

Linear

$$\phi(x) = x$$

Soft ReLU

$$\phi(x) = e^x$$

Rectified Linear Unit (ReLU)

$$\phi(x) = \max(0, x)$$

Hard Threshold

$$\phi(x) = \mathbb{I}_{[0, +\infty[}(x)$$

Logistic sigmoid

$$\phi(x) = (1 + e^{-x})^{-1}$$

Radiale

$$\phi(x) = \sqrt{1/2\pi} \exp\left(-\frac{1}{2}x^2\right)$$

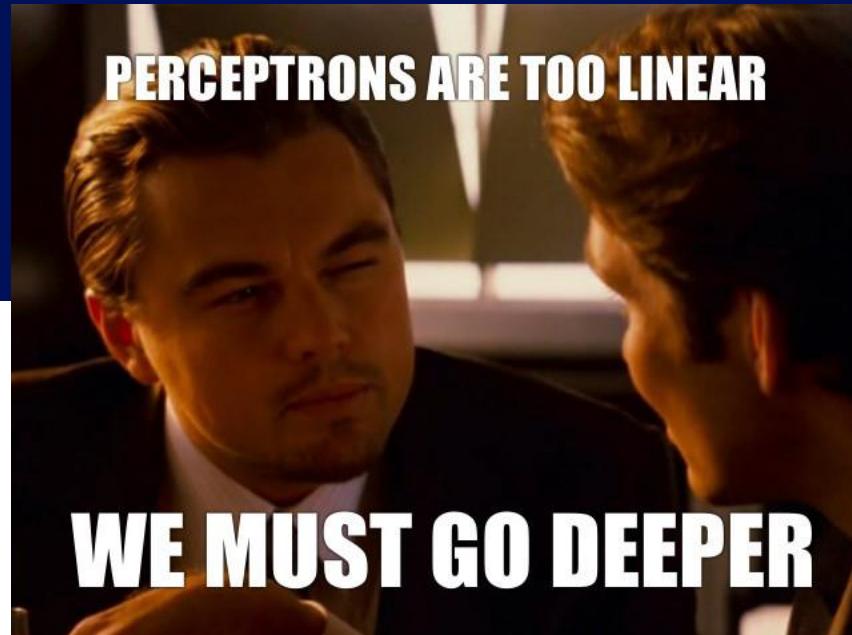
Hyperbolic Tangent

$$\phi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Sinus

$$\phi(x) = \sin(x)$$

Perceptron Multicouches



Perceptron Multicouches (PMC)

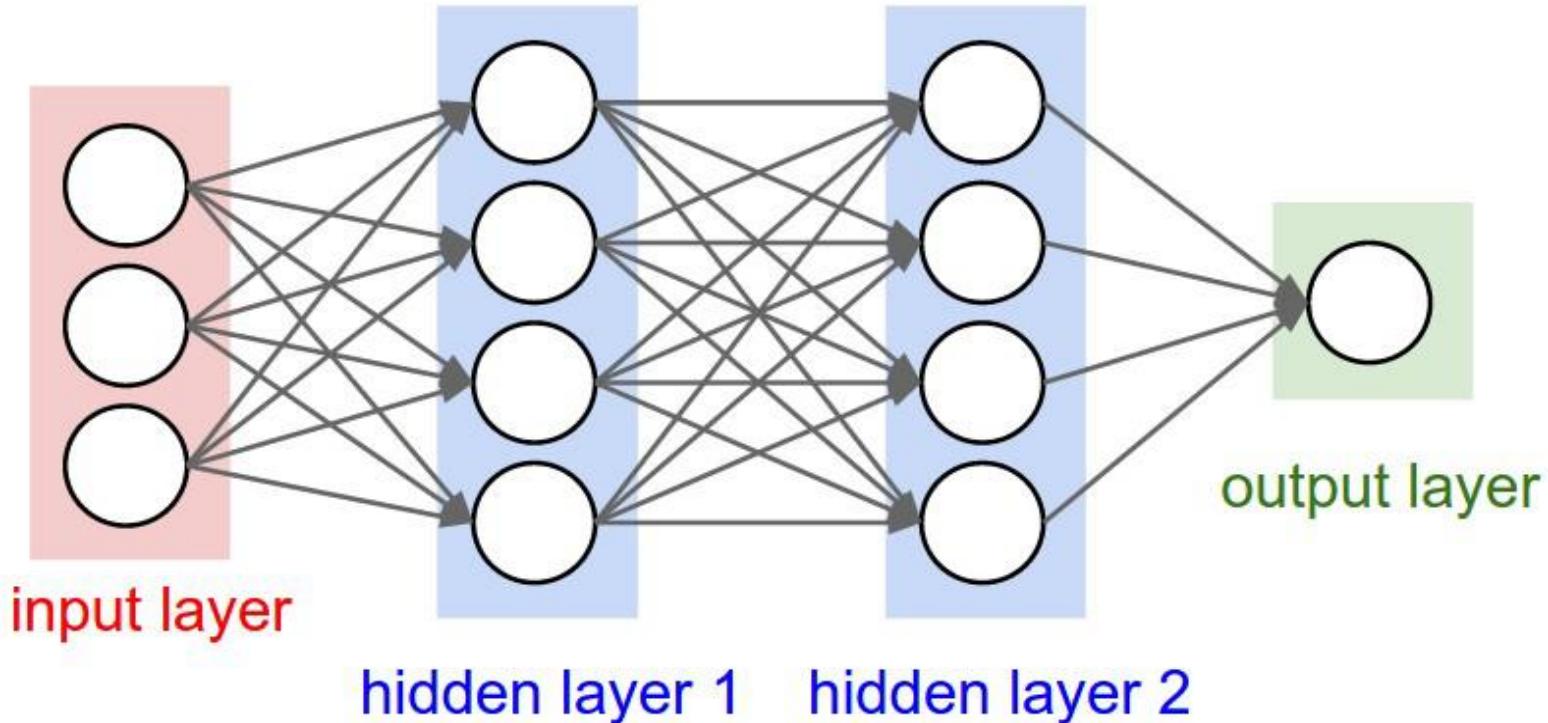
On peut connecter plusieurs neurones entre eux dans un graphé orienté acyclique (*Feed-forward Neural Network*)

- Neurones d'entrée (les covariables)
- Neurones de sortie (l'estimation de la variable d'intérêt)
- Couches cachées (calculs intermédiaires)

Dans un perceptron multicouches, les neurones sont organisés en couches successives

- Les neurones d'une même couche n'ont pas de connexion entre eux
- Les connexions se font entre couches successives
- Une couche d'entrée et une couche de sortie

Perceptron Multicouches



Caractéristiques d'un PMC

Un PMC est caractérisé par

- Nombre de couches
- Nombre de neurones dans chaque couche
- Fonctions d'activation de chaque couche
- (les connexions entre les neurones)

Sur la dernière couche, on aura typiquement

- en régression : un seul neurone avec la fonction identité
- en classification binaire : un seul neurone avec sigmoid
- en classification à m classes : m neurones avec sigmoid

Fonction d'un PMC

La fonction de sortie d'un PMC devient rapidement plus complexe qu'un perceptron simple : on parlera communément de **deep learning** lorsque le nombre de couches cachées augmente.

Exemple avec un PMC avec deux couches cachées

- K variables d'entrées
- Une première couche cachée avec M neurones
- Une deuxième couche cachée avec H neurones
- 1 neurone de sortie

$$F(x) = f \left(\sum_{h=1}^H \omega_h f_h \left(\sum_{m=1}^M \omega_{h,m} f_{h,m} \left(\sum_{k=1}^K \omega_{h,m,k} x_k + b_{h,m} \right) + b_h \right) + b \right)$$

Théorème d'approximation universelle

Théorème

Toute fonction régulière peut être approchée uniformément avec une précision arbitrairement petite et dans un domaine fini de l'espace des variables par un réseau de neurones avec une couche de neurones cachés (en nombre fini et possédant une fonction d'activation non linéaire) et un neurone de sortie de type linéaire

Limites du théorème

- Le nombre de neurones nécessaire est exponentiel
- Risque d'overfitting
- Besoin d'une représentation compacte de la solution

Retropropagation



Estimation des paramètres

Soit $(x_i, y_i)_{i=1,\dots,n}$ notre base d'apprentissage de n observations de $x \in \mathbb{R}^p$

Prenons l'exemple d'une régression faite par un NN

- une couche d'entrée avec q neurones ;
- une sortie linéaire ;
- même fonction d'activation f pour tous les neurones.

Les paramètres du modèle sont les poids ω des différentes fonctions

- on note $\alpha = (\alpha_{j,k})_{j=0,\dots,p ; k=1,\dots,q}$ les poids des neurones cachés
- et $\beta = (\beta_k)_{k=0,\dots,q}$ les poids du neurone de sortie

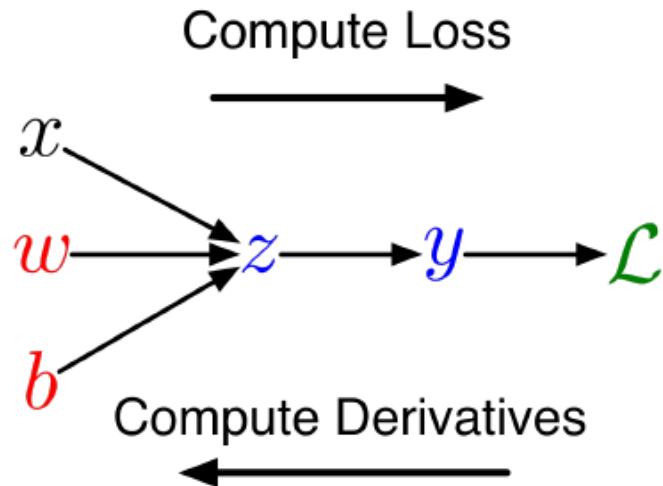
On estime les paramètres en minimisant la perte quadratique

$$\mathcal{L}(\alpha, \beta) = \sum_i^n (y_i - F(x_i, \alpha, \beta))^2$$

Retropagation

Une technique usuelle d'optimisation est la descente de gradient

- les fonctions d'activation simples sont dérivables
- on cherche à calculer $\partial \mathcal{L} / \partial \omega$
- mais l'imbrication des neurones complexifie le calcul des gradients
- Calcul couche par couche du gradient



Estimation des gradients (1/2)

On note le vecteur des sorties des neurones cachés \mathbf{z}_i

$$z_{k,i} = f(\alpha_{0,k} + \boldsymbol{\alpha}_k^T \mathbf{x}_i)$$

Erreur du modèle courant à la sortie pour l'individu i : δ_i

$$\frac{\partial \mathcal{L}_i}{\partial \beta_k} = \delta_i \mathbf{z}_{k,i}$$

Erreur sur chaque neurone caché pour l'individu i : $s_{k,i}$

$$\frac{\partial \mathcal{L}_i}{\partial \alpha_{j,k}} = s_{k,i} \mathbf{x}_{j,i}$$

Estimation des gradients (2/2)

Ces deux termes vérifient les équations de rétropropagation de l'erreur

$$s_{k,i} = f'(\alpha_k^T x_i) \beta_k \delta_i$$

Pour estimer les gradients, on a besoin d'évaluer δ_i et $s_{k,i}$. On le fait en deux étapes

1. Une passe avant : en utilisant la valeur courante des poids, on détermine la sortie $\hat{F}(x_i)$
2. Une passe retour :
 - a. avec $\hat{F}(x_i)$ on évalue δ_i ,
 - b. puis $s_{k,i}$ par rétropropagation des δ_i .

Descente de gradient

Maintenant que l'on sait évaluer les gradient, il ne reste plus qu'à appliquer un algorithme adapté pour l'optimisation. Typiquement une descente itérative du gradient.

$$\beta_k^{(r+1)} = \beta_k^{(r+1)} - \gamma \sum_i \frac{\partial \mathcal{L}_i}{\partial \beta_k^{(r)}}$$

$$\alpha_{j,k}^{(r+1)} = \alpha_{j,k}^{(r)} - \gamma \sum_i \frac{\partial \mathcal{L}_i}{\partial \alpha_{j,k}^{(r)}}$$

γ : taux d'apprentissage

Algorithme de retropropagation

Initialisation

Tirage aléatoire uniforme sur $[-1,1]$ pour les poids ω

Itération

1. Calcul de la perte \mathcal{L}
2. Calcul des gradients en fonction des poids de la dernière couche
3. Calcul des gradients des autres couches par rétropropagation itérative
4. Descente stochastique du gradient

Arrêt

- quand la fonction de perte atteint un seuil minimal
- quand le nombre d'itération dépasse le nombre d'itération maximal

Overfitting

Le taux d'apprentissage est un paramètre de tuning

- Fixé par l'utilisateur au début de l'algo ou varie en cours d'exécution
- Si γ est grand alors on converge vite, mais la prédiction est moins précise
- Et inversement...

Possibilité d'ajouter un terme de régularisation (*weight decay*)

- Analogie avec la régression ridge : pénalisation en norme L_2 sur les poids ω

$$R(\omega) = \mathcal{L}(\omega) + \lambda \sum \omega^2$$

- avec λ un nouveau paramètre de tuning

Paramètres d'un réseau de neurones

Variables d'entrée et de sortie

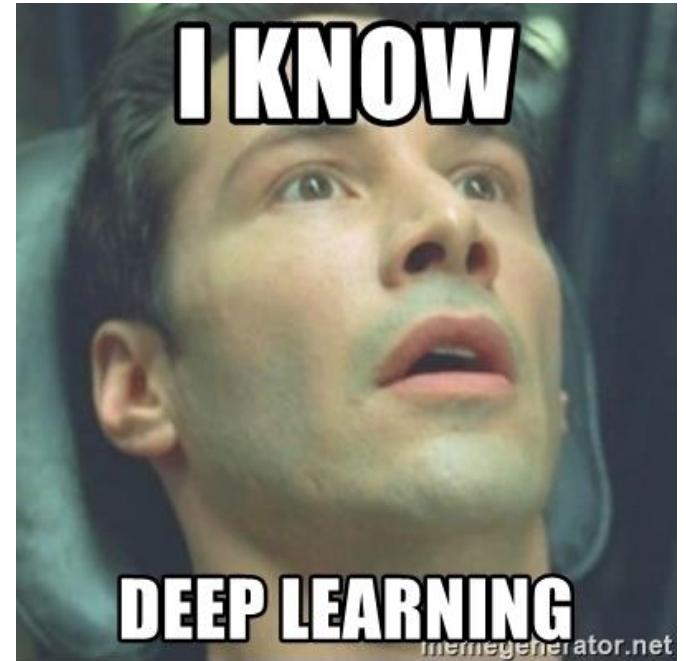
- Besoin potentiel de normaliser les variables

Architecture du réseau

- Nombre de couches
- Nombre de neurones dans chaque couche
- Fonctions d'activation

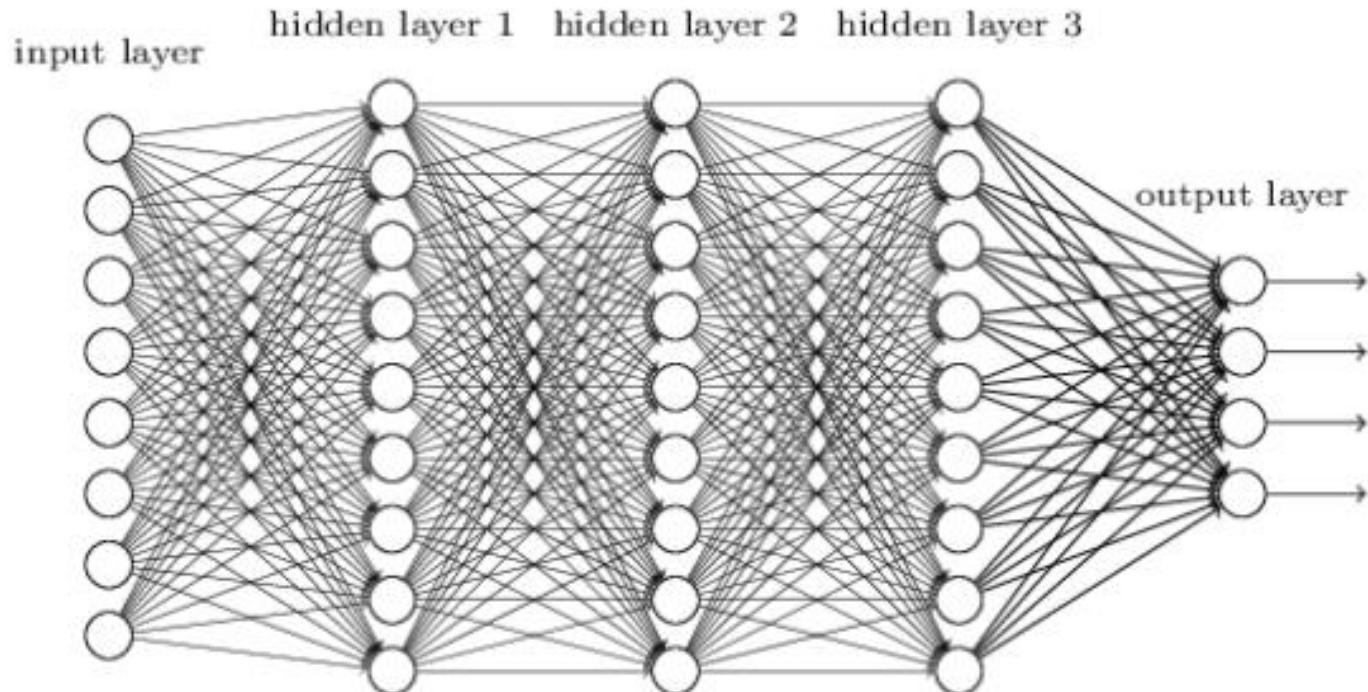
Hyperparamètres (à tuner !)

- Erreur minimale tolérée
- Nombre maximal d'itérations
- Taux d'apprentissage
- Terme de régularisation



Deep Learning

Deep neural network



Deep Learning

Fonction d'optimisation non-convexe

- Potentiels minimum locaux
- La descente de gradient n'est théoriquement pas armée pour cela

Extrême complexité

- Nombre de paramètres très important
- Besoin de beaucoup de données et donc beaucoup de capacité de calcul
- Augmente la dimension VC

Mais en pratique ça fonctionne !

- Même si ça ne devrait pas...



CNN

Reconnaissance d'images

Les humains sont très forts en terme de reconnaissance d'objet

- Même si ça nous paraît facile, c'est très difficile pour un ordinateur

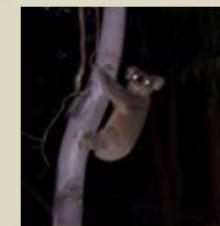
Quelques raisons de cette difficulté pour les ordinateurs

- Segmentation : les scènes réelles ne sont pas forcément ordonnées
- Invariance : les humains sont très agiles pour ignorer toute sorte de variations
- Déformation : les objets peuvent avoir une multitude de variations

Dimension des variables

- Input : une image est composée de millions de pixels
- Output : un objet parmi des dizaines de milliers de classes

Reconnaissance d'images

**occlusion****scale****deformation****clutter****illumination****viewpoint****object pose**

How to confuse machine learning



Couche de convolution

On découpe l'image d'entrée en sous-images se chevauchant



Chaque sous-image est en entrée de « petits » réseau de neurones

- Les poids et les biais ne sont pas modifiés selon la sous-image, i.e. l'espace
- Chaque réseau, ou filtre, permet de trouver une caractéristique spécifique
- Récupérer des statistiques similaires sur l'ensemble de l'espace

Pooling et analyse

Le nombre de filtres et les différentes localisations peuvent donner un nombre encore important de résultats

- Besoin d'une réduction de la dimension avant l'analyse

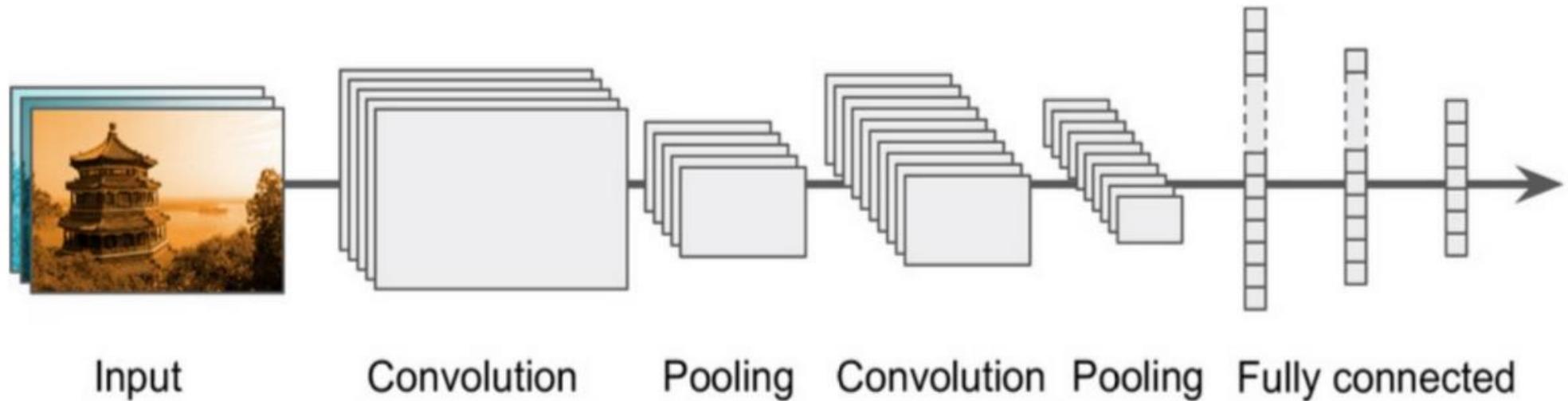
Couche de pooling

- Réduit la dimension, notamment spatiale, pour plus de robustesse
- Fait en sorte que le réseau soit invariant par des petites transformations
- Pas de poids ni de biais : *max pooling, average pooling, etc.*

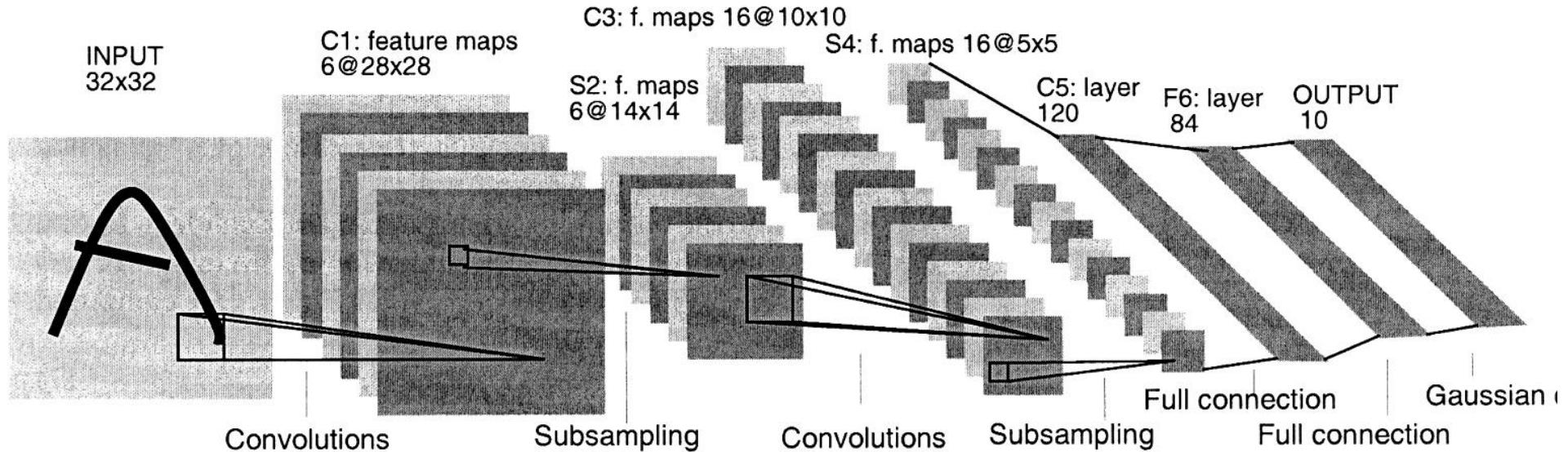
Analyse finale

- Le dernier type de couche est simplement un PMC *fully connected*

Convolution Neural Network



LeNet (1998)



Appliquée à la reconnaissance de chiffres écrits manuellement

- 61,470 paramètres
- 98,9% de précision : suffisant pour lire automatiquement les chiffres sur les chèques

Evolution des algorithmes

La reconnaissance d'objet est plus complexe

- Besoin d'aller plus en profondeur

AlexNet 2012

- 8 couches – 60 millions de paramètres

InceptionNet 2014

- 22 couches – 7 millions de paramètres

ResNet 2015

- 152 couches – 60 millions de paramètres

EfficientNet 2019

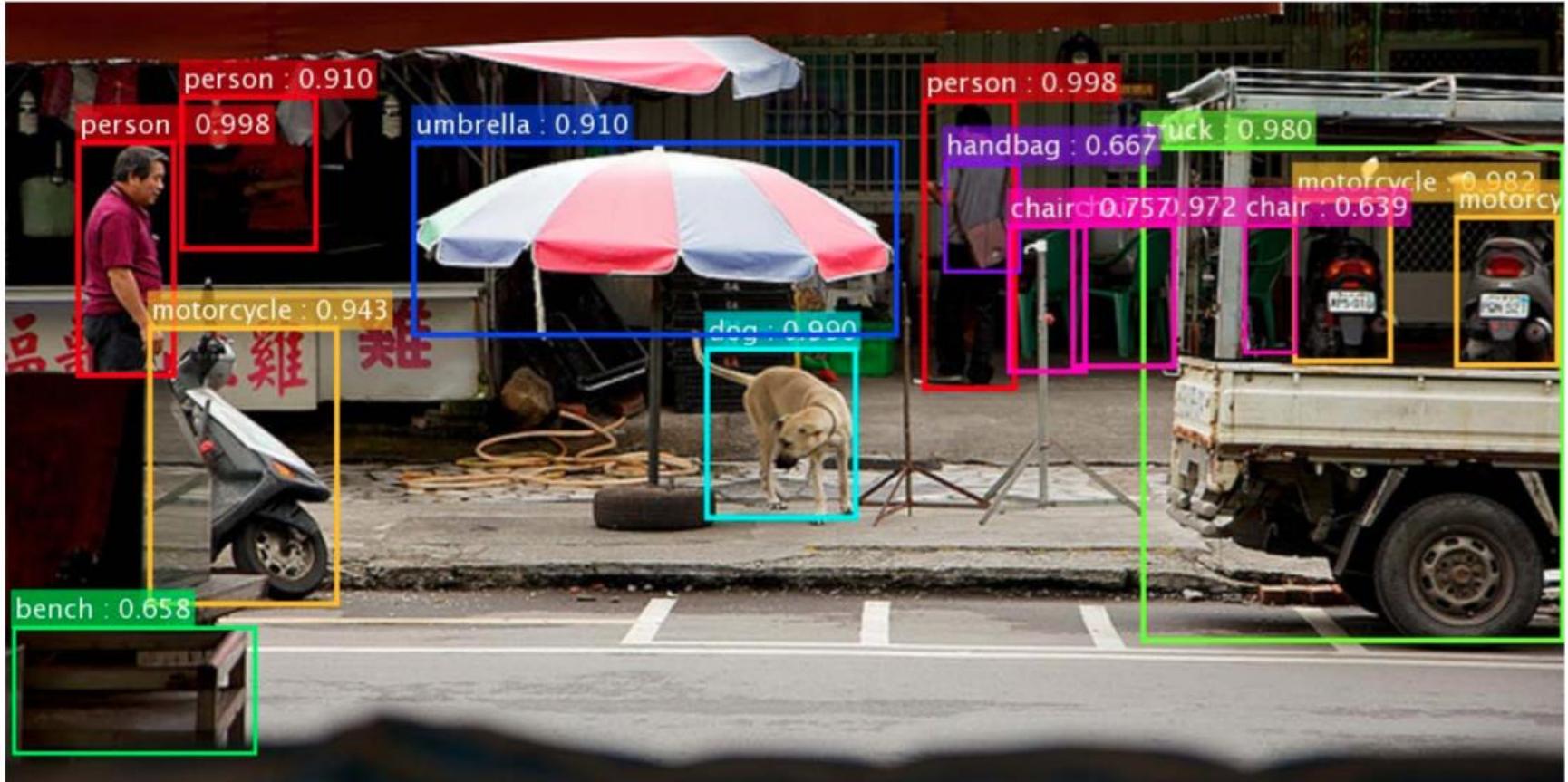
- 813 couches – 66 millions de paramètres

how's your ML Engineering job going?

me:



Reconnaissance d'objet





GAN



Generative Adversarial Network

Reconnaitre c'est bien, créer c'est mieux

- Generative Adversarial Network : Goodfellow (2014)

Architecture utilisant deux réseaux de neurones

- Un réseau générateur
- Un réseau de discrimination

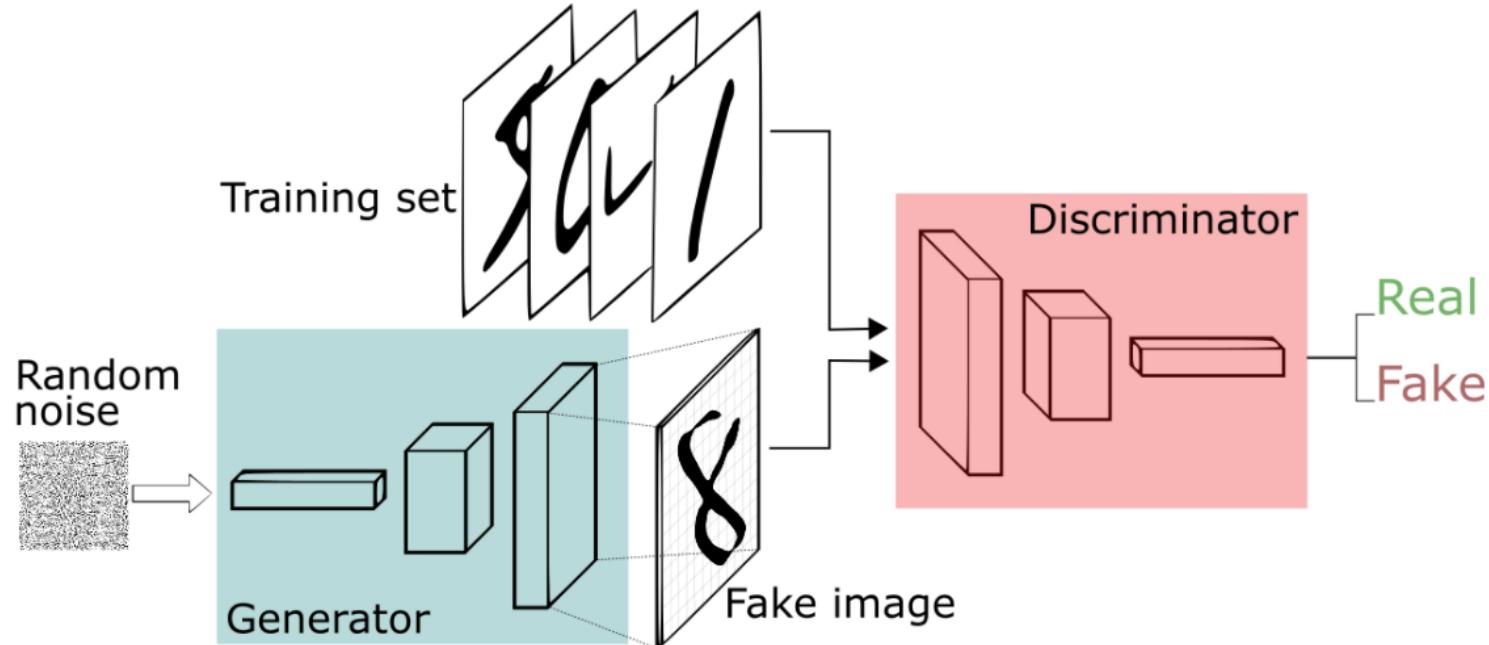
Utilisation pour la génération de différents types de contenus

- Image
- Video
- Voix

Fonctionnement d'un GAN

Compétition des deux réseaux l'un contre l'autre

- le générateur créer une nouvelle observation
- le discriminant décide si cette observation appartient à la base d'entraînement



Evolution des GAN



2014



2015



2016



2017

Portrait d'Edmond de Belamy

