# Deep Learning Courses

***Yann LeCun - NYU.***
***https://www.youtube.com/watch?v=0bMe_vCZo30&ab_channel=AlfredoCanziani***

## Cours 1.

**Histoire, motivation, évolution.**

Inspiration for Deep Learning: **The Brain! On peut comparer ça à du mimétisme biologique (avions => oiseaux, etc) = biomimicry.**
**On peut lier ce domaine à la cybernétique.**

**McCulloch & Pitts (1943**): networks of binary neurons can do logic (*un neurone calcule la somme de ses entrées et compare cette somme à un seuil, il s'allume si la somme est au-dessus du seuil sinon il s'éteint => vision TRÈS simplifiée de comment fonctionne un neurone*)

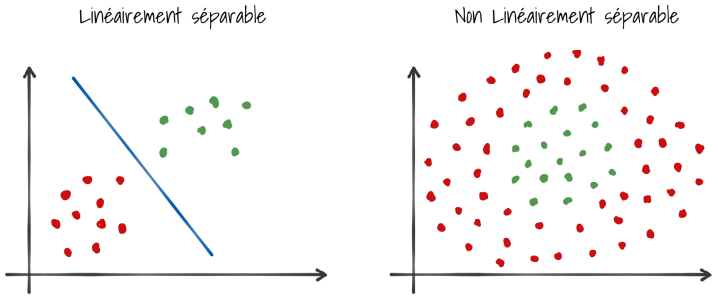
**Donald Hebb (1947)**: Hebbian synaptic plasticity (*le cerveau apprend en modifiant l'intensité des connexions entre les neurones : les synapses. Si deux neurones s'associent/fire ensemble, alors la connexion qui les lie augmente - sinon l'inverse.*)

**Norbert Wiener (1948)**: cybernetics, optimal filter, feedback, autopoïesis, auto-organisation. (*en ayant des systèmes qui ont des capteurs et des actuateurs, on peut avoir des feedback loops, des systèmes auto-régularisants. Exemple : on conduit notre voiture, on tourne le volant qui tourne les roues selon l'intensité voulue etc… tout ça constitue un mécanisme feedback.*)

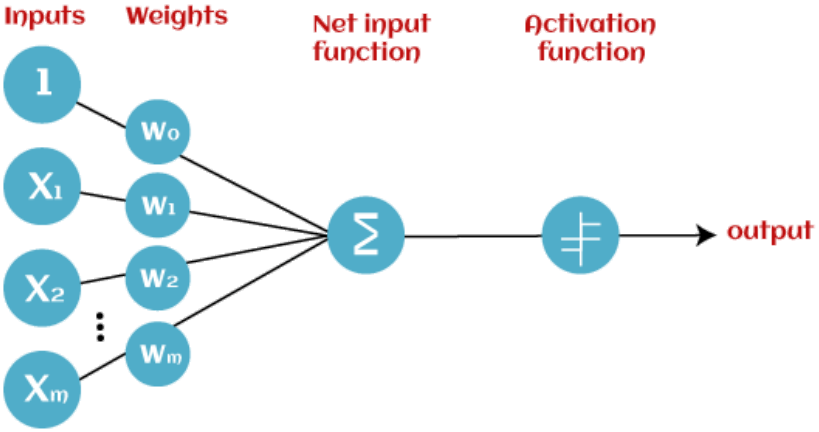
**Frank Rosenblatt (1957)**: Perceptron (*algorithme de classification qui s'inspire du neurone binaire (non continue) et qui modifie les poids des entrées de simple réseau de neurone*)
=> utile pour des données linéairement **séparables** et **bi-classe**. (**monocouche**, le perceptron multicouche permet de s'affranchir de la non-séparabilité linéaire des variables)

**En réalité, le Perceptron est une fonction mathématique. Les données d'entrée (x) sont multipliées par les coefficients de poids (w). Le résultat produit est une valeur.**

**Cette valeur peut être positive ou négative. Le neurone artificiel s'active si la valeur est positive.** Il ne s'active donc que si le poids calculé des données d'entrée dépasse un certain seuil.
Le résultat prédit est comparé avec le résultat connu. En cas de différence, l'erreur est **rétro propagée** afin de permettre d'ajuster les poids.




**Entrée** = nombre de features (chaque entrée a un **poid**) + le biais
**Sortie** = nombre de classe, ici binaire donc 1 ou -1
On ajuste les poids jusqu'à trouver une bonne frontière de décision.




***Démonstration en python :***
https://www.youtube.com/watch?v=pXc6usW3Txo&ab_channel=Arthurus

**Hubel & Wiesel (1960s)**: visual cortex architecture

Ensuite, le sujet est MORT jusqu'en 1984 (à part au Japon avec chercheurs isolés) car on ne pouvait pas faire grand chose avec les réseaux de neurones.
En 1985, le sujet est relancé avec la **BACKPROPAGATION (algorithme pour entraîner des réseaux de neurones multicouches, il fonctionne en utilisant une fonction d'activation CONTINUE)**.
Avant les années 60, cet algorithme n'a pas été trouvé puisque les chercheurs se basaient sur les **neurones de McCulloch-Pitts qui étaient des neurones binaires et non continues**. Comme la fonction d'activation n'était **ni dérivable ni continue**, on ne

pouvait pas utiliser la **descente de gradient** pour entraîner le modèle (rappel, il permet de trouver un minimum/maximum local. De plus, lorsqu'on a un réseau de neurones binaires, si le neurone est actif on a juste à ajouter le poids pondéré au poids total (si neurone inactif on ne fait rien). Lorsqu'on a des un réseau de neurones continues, cette fois-ci on raisonne en termes de contribution à la somme totale. Il faut donc faire des multiplications, ce qui était TRÈS long pour les ordinateurs de l'époque !

Puis le sujet est mort de nouveau entre 1995 et 2010, jusqu'à ce que les gens se rendent compte qu'ils pouvaient entraîner les réseaux de neurones avec la backpropagation pour améliorer la **reconnaissance vocale (Android!)**.
La même chose s'est passé avec la computer vision autour de 2012/2013 en particulier lorsque les gens ont réalisé que cela fonctionnait beaucoup mieux avec le deep learning, réseaux neuronaux à convolution.
Et enfin vers 2015/2016, même chose avec le NLP pour la traduction.
**La même chose arrive en robotique, contrôle etc.. dans le futur !**

L'apprentissage supervisé représente 90% des applications du Deep Learning.
Lorsqu'on donne un input à un modèle, on attend un output en sortie (une classe).

- Si elle est correcte, on ne fait rien.
- Si elle est fausse, on va jouer avec les paramètres.

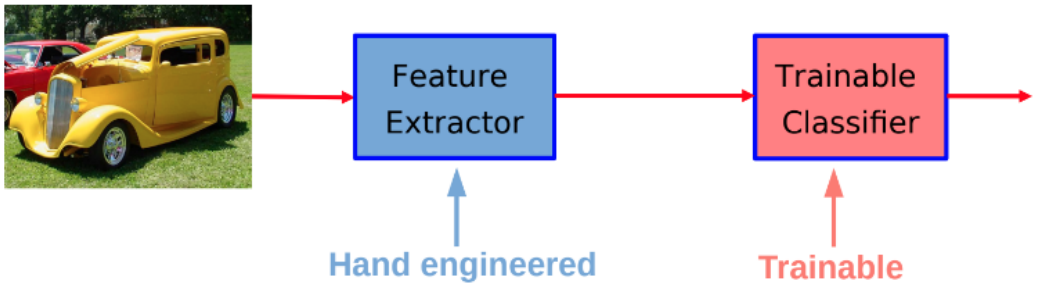
Les paramètres dans les réseaux de neurones sont les poids des neurones !
Mais comment les manipuler ? C'est à cela que sert la descente de gradient et la back propagation.

---

Il y a eu bcp de modèles basiques pour la classification:

- **Perceptron (réseau de neurone caché à 2 couches dans lequel la seconde couche est entraînable avec des poids variables, mais la première couche est fixe dont les entrées sont attachés à des neurones où le poids est initialisé aléatoirement => couche associative)**
- **Adaline**

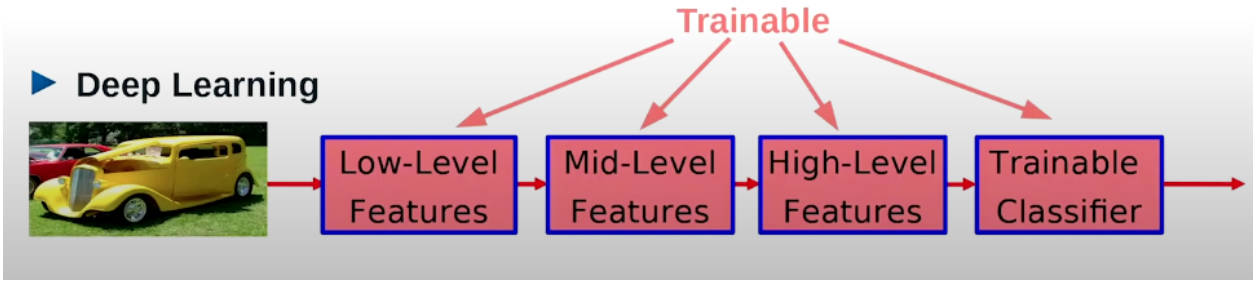
Ils sont basés sur la même architecture : calculer la somme des poids des inputs, comparer à un seuil, si > au seuil, allumer le neurone sinon l'éteindre.
**Machine Learning :**




**Feature Extractor** = Extraire les caractéristiques les plus importantes de l'input utiles pour notre tâche. On a donc un vecteur de feature que l'on va donner à notre classifieur.

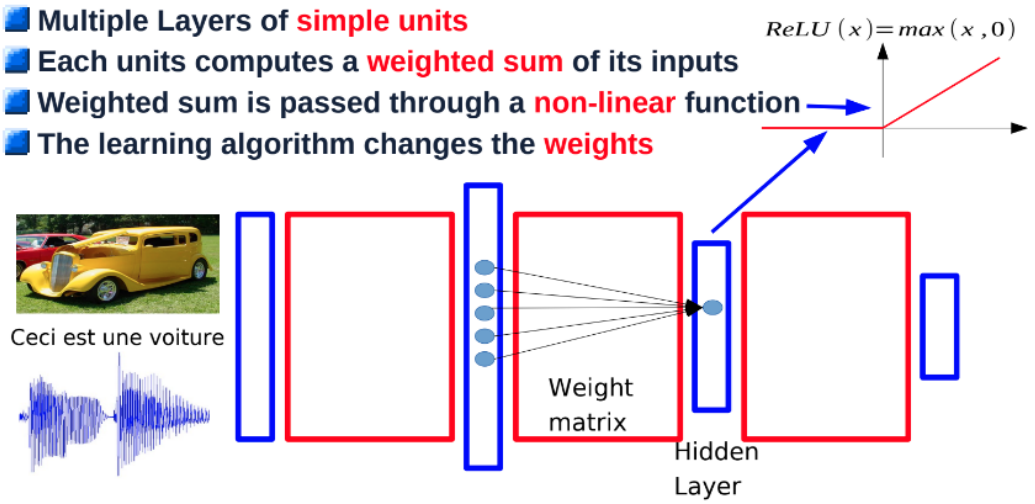
Le problème est que la majeure partie de la computer vision faisant de la reconnaissance de pattern repose sur cette partie plutôt que sur le classifieur en lui-même. Comment faire un feature extractor pour un problème en particulier ? + Preprocessing des images !

**Deep Learning (= multilayer, nothing much more !):**




La différence est que nous n'avons plus ce processus à 2 étapes (dont une à la main) que le ML avait, le DL apprend la tâche du début à la fin : cascade/séquence de module, où chacun a des paramètres à accorder.
La difficulté est de paramétrer les paramètres de chacun des modules de façon à se rapprocher le plus possible de l'output voulu : c'est le rôle de la **backpropagation** !
Les modules sont non-linéaires car si 2 modules successifs sont linéaires, on peut les fusionner ! Pas d'intérêt à avoir plusieurs couches, si ces couches sont linéaires… autant faire une seule couche !

**Quel est l'architecture multi-couche la plus simple, qui a des paramètres à bouger (comme des poids) et est non linéaire ?**
Ca va ressembler à ça :




- On prend une entrée (un vecteur).
- On multiplie ce vecteur par une matrice de poids dont les coefficients sont ajustables.
- On prend ce vecteur résultat (car **vecteur * matrice = vecteur**), et on fait passer chaque composante de ce vecteur par une fonction non linéaire (la plus facile est la RELU). On a donc un nouveau vecteur avec beaucoup de 0.
- On répète le processus ! On prend ce dernier vecteur, on le multiplie par une matrice de poids et on fait passer ce nouveau vecteur dans une fonction non linéaire => nouveau vecteur et ainsi de suite !

## RÉSEAUX DE NEURONES BASIQUES !

Pourquoi on appelle ça un réseau de neurones ? Parce que lorsqu'on prend un vecteur que l'on multiplie par une matrice, pour calculer chaque composante de la sortie (vecteur). Enfaite, on fait une somme pondérée des composantes de l'entrée par une ligne correspondante de la matrice. Donc on aura autant de composantes en sortie que de lignes de la matrice !

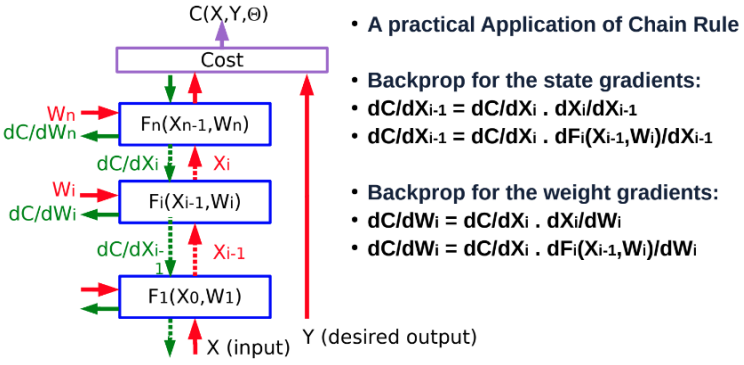
En apprentissage supervisé, on compare ensuite notre sortie à la sortie attendue. On va donc avoir une fonction **OBJECTIF** (module de perte ou **LOSS**) qui va calculer une distance (par exemple la distance euclidienne entre un vecteur cible et le vecteur du neural net), une divergence dont on va calculer la moyenne (**scalaire = point**). On veut **MINIMISER** cette moyenne sur le jeu d'entraînement par rapport aux paramètres ajustables (donc les poids) => on veut donc trouver la valeur des paramètres qui minimise l'erreur moyenne entre sorties qu'on a et les sorties qu'on veut.

La façon de minimiser cela est de faire une **descente de gradient** (qui donne la direction) :

- It's like walking in the mountains in a fog and following the direction of steepest descent to reach the village in the valley
- But each sample gives us a noisy estimate of the direction. So our path is a bit random.
- Stochastic Gradient Descent (SGD)

La convexité est importante et le pas aussi !
**Gradient stochastique :** la moyenne calculée de la fonction objectif est une moyenne sur beaucoup d'échantillons mais il est plus intéressant de ne prélever qu'un seul échantillon ou un petit groupe d'échantillons, calculer l'erreur de cet échantillon puis calculer le gradient de cette erreur par rapport aux paramètres et faire un petit pas. Un nouvel échantillon arrive, on récupère l'erreur et la gradient, on refait un petit pas. Si on continue à faire ça, on va réduire le coût mais de manière bruyante car beaucoup de fluctuations.
=> elle converge plus rapidement et plus facilement, surtout si on a un très grand jeu de données.
=> on prend en compte la redondance des données
=> meilleure généralisation

**Rétropropagation (règle de la chaîne) :** pourquoi ça s'appelle comme ça ? Car cela s'applique à **plusieurs couches**. On se rappelle un neural net comme un tas de module empilés, non linéaires entre eux, on peut les considérer comme des compositions de fonctions. On se rappelle également du calcul de la dérivée d'une fonction composée avec une autre fonction.
Si on veut calculer la dérivée de la différence entre la sortie voulue et la sortie attendue (= valeur de la fonction objectif) par rapport à toute variable à l'intérieur du réseau, alors il faut propager en arrière les dérivées et multiplier les choses en cours de route.

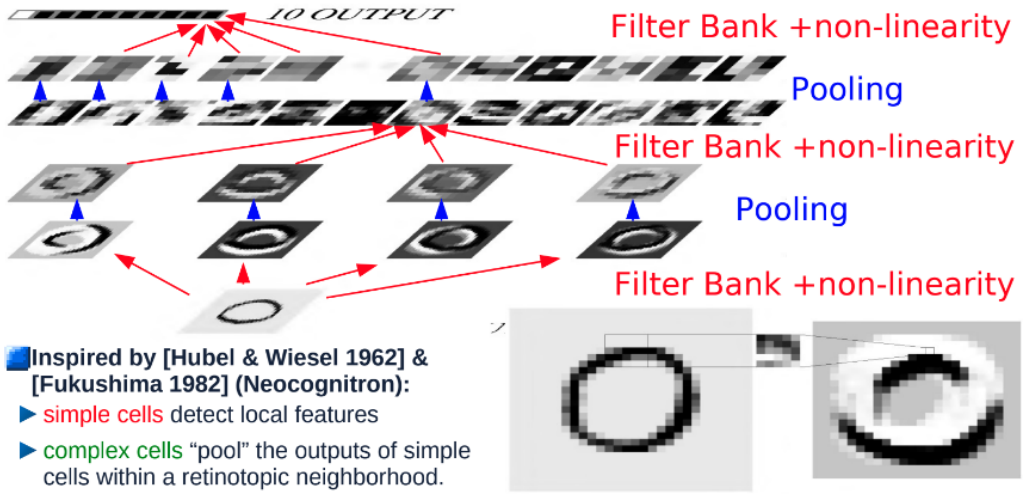



- A practical Application of Chain Rule
- Backprop for the state gradients:
- $dC/dX_{i-1} = dC/dX_i \cdot dX_i/dX_{i-1}$
- $dC/dX_{i-1} = dC/dX_i \cdot dF_i(X_{i-1},W_i)/dX_{i-1}$
- Backprop for the weight gradients:
- $dC/dW_i = dC/dX_i \cdot dX_i/dW_i$
- $dC/dW_i = dC/dX_i \cdot dF_i(X_{i-1},W_i)/dW_i$

## Hubel & Wiesel Model of the Architecture of the Visual Cortex :

EXEMPLE : on prend une image en entrée (256*256*3 en RGB), on a donc un vecteur d'environ 200k composantes. Il nous fait donc une matrice de 200k colonnes qui va multiplier ce vecteur, et selon le nombre d'unité que l'on a dans la première couche, ce sera 200k*un grand nombre. C'est une énorme matrice ! C'est impraticable, c'est d'ici que vient l'architecture basée sur le cortex visuel lorsqu'on veut traiter des images.
H&W ont placé des électrodes sur le cortex visuel de chats et ont essayé de comprendre ce que les neurones du cortex visuel ont fait.
Voici ce qu'ils ont découvert : des signaux arrivent à nos yeux, en frappant la rétine. Quelques couches de neurones **pré-traitent** le signal en le compressant, et relient ce signal au morceau de cerveau appelé "noyau géniculé latéral" qui **normalise les contrastes**, puis cela va dans des endroits plus complexes et plus spécifiques (V1 à V4).
***Donc avec l'idée que d'une certaine manière le contexte visuel peut faire de la reconnaissance de formes et semble avoir cette sorte de structure hiérarchique, structure multicouche. Il y a aussi l'idée que le processus visuel est essentiellement un processus « feed-forward ». Donc le processus par lequel vous reconnaissez un objet quotidien est très rapide. Il prend environ 100 millisecondes.***
Kunihiko Fukushima, a eu l'idée de s'inspirer de Hubel et Wiesel dans les années 70, et a en quelque sorte construit un modèle de réseau neuronal sur ordinateur. Et a eu cette idée, qu'il y avait d'abord des couches. Mais aussi l'idée que Hubel et Wiesel ont découverte, est que les neurones individuels ne réagissent qu'à une petite partie du champ visuel.
=> exemple : V1 = sélectivité de l'orientation.
Donc les neurones regardent un champ local et réagissent ensuite aux orientations. Et les groupes de neurones qui réagissent à de multiples orientations sont reproduits sur l'ensemble du champ visuel.
Alors Fukushima, a dit : "pourquoi ne pas construire un réseau neuronal qui fasse ça ?"
Il a également utilisé le concept de **cellules complexes**.

Tout cela s'appelle **CONVOLUTION & POOLING** dans les réseaux convolutifs (ConvNets). Ce sont des réseaux où les connexions sont locales et sont répliquées le long du champ visuel et où l'on interfère des couches de détection qui détectent ces caractéristiques locales avec une opération de pooling : chaque neurone d'une couche qui sont répliqués à travers les neurones de la couche



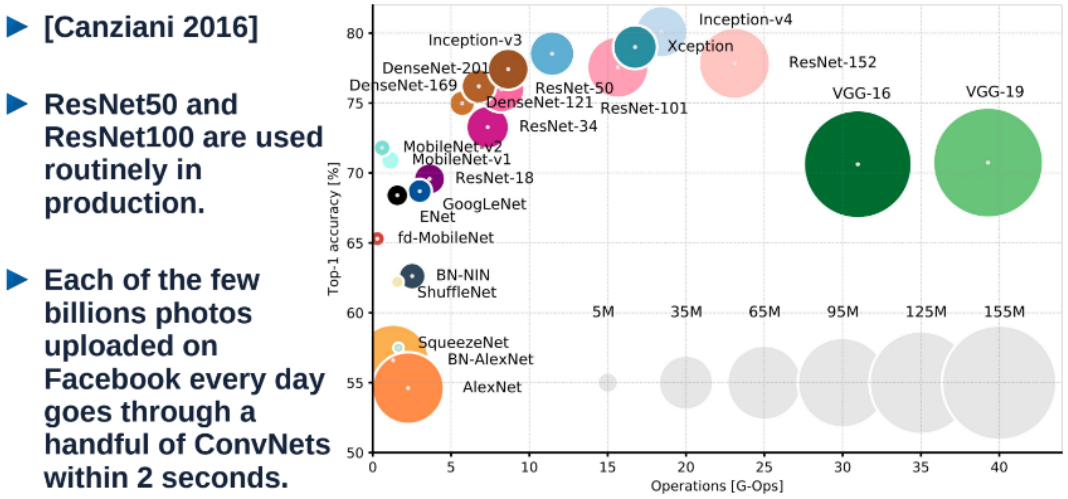

Les premières couches de ce réseau de neurones jouent un rôle d'extracteur de caractéristiques.

ConvNets font de la **segmentation sémantique** => utilisé dans les voitures autonomes.

- **33 categories**

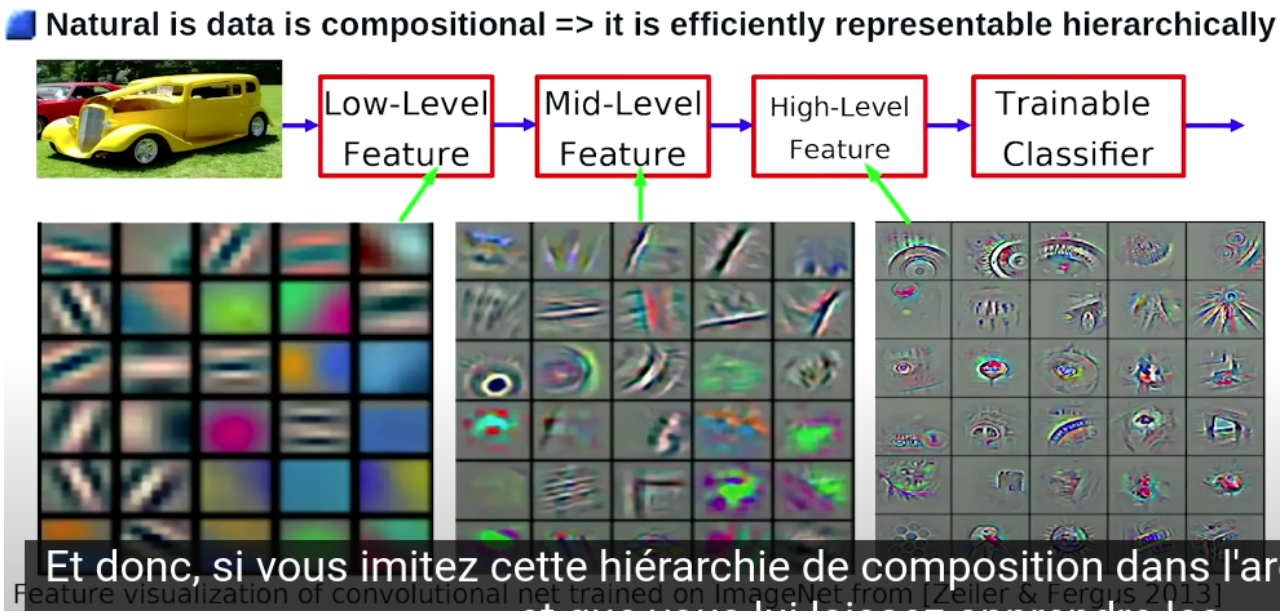
**En ce moment, le topic brûlant est les cartes électroniques accélératrices de NeuralNets (aujourd'hui sur GPU).**

Speech recognition: 2010
Image recognition: 2013
Natural language processing: 2015




Aujourd'hui, on veut **optimiser** la mémoire utilisée par ces réseaux.

**Les ConvNets exploitent la compositionnalité des données naturelles :** une scène est composée d'objets, qui sont composés de parties, qui elles-mêmes sont composées de sous-parties.




Les ConvNets imitent donc cette architecture : ils apprennent à représenter le monde et sa structure. On a donc été capable d'énormes progrès dans le **Computer Vision** !

**Pourquoi les ConvNets fonctionnent-ils si bien ?**

## Why does it work so well?

- **We can approximate any function with two layers**
  - Why do we need layers?
- **What is so special convolutional networks?**
  - Why do they work so well on natural signals?
- **The objective function are highly non-convex.**
  - Why doesn't SGD get trapped in local minima?
- **The networks are widely over-parameterized.**
  - Why do they not overfit?

En statistiques, on dit que si nous avons *n* données, nous ne devrions pas avoir plus de *n* paramètres. Les NeuralNets sont **sur-paramétrés** !

- **What we can have**
  - Safer cars, autonomous cars
  - Better medical image analysis
  - Personalized medicine
  - Adequate language translation
  - Useful but stupid chatbots
  - Information search, retrieval, filtering
  - Numerous applications in energy, finance, manufacturing, environmental protection, commerce, law, artistic creation, games,.....
- **What we cannot have (yet)**
  - Machines with common sense
  - Intelligent personal assistants
  - "Smart" chatbots"
  - Household robots
  - Agile and dexterous robots
  - Artificial General Intelligence (AGI)

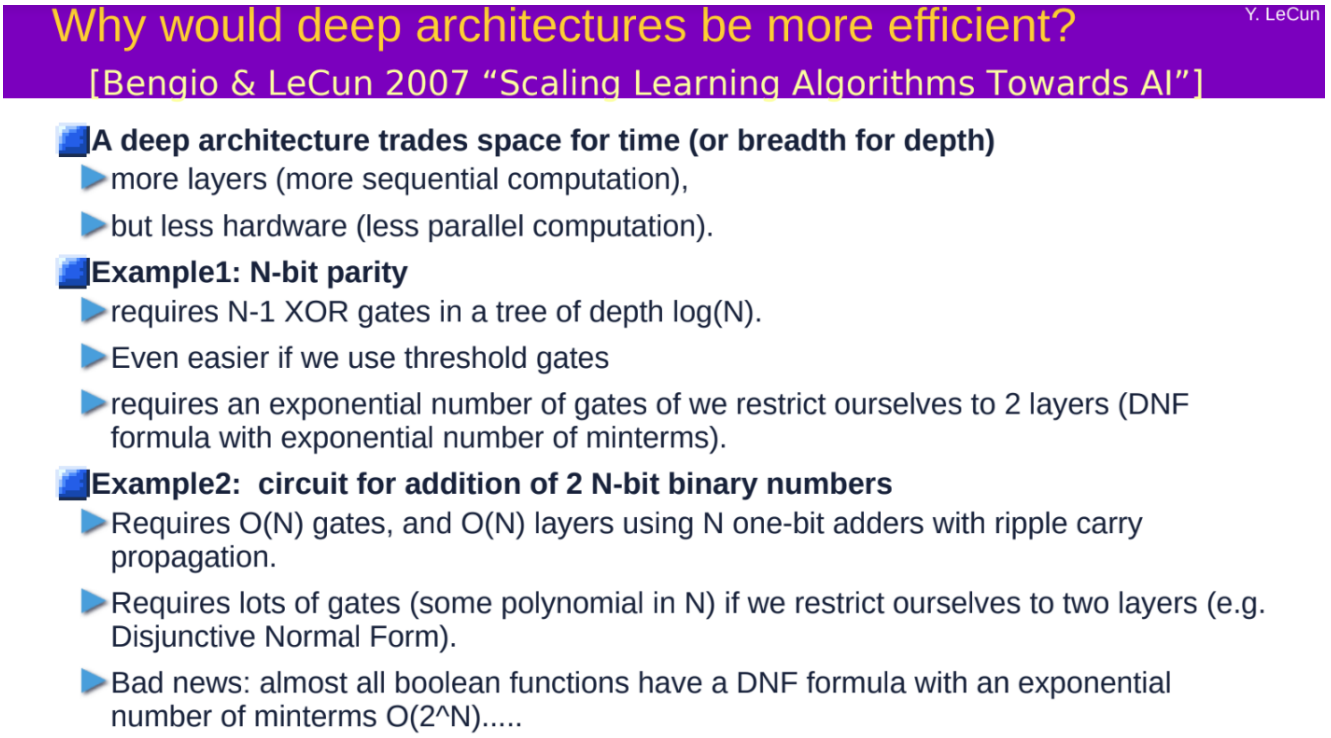
**Représentation de l'apprentissage :**

Quelles sont les représentations ? Nous avons des données brutes que nous voulons transformer en une forme utile pour la tâche que nous souhaitons.

**Exemple** :
Projection aléatoire => idée du perceptron => projection aléatoire car matrice aléatoire de poids qui a une dimension de sortie plus petite que la dimension d'entrée.

**Le dilemme du théoricien :** "Nous pouvons approcher n'importe quelle fonction aussi près que nous le voulons avec une architecture peu profonde.
**Pourquoi aurions-nous besoin d'une architecture profonde** ?"

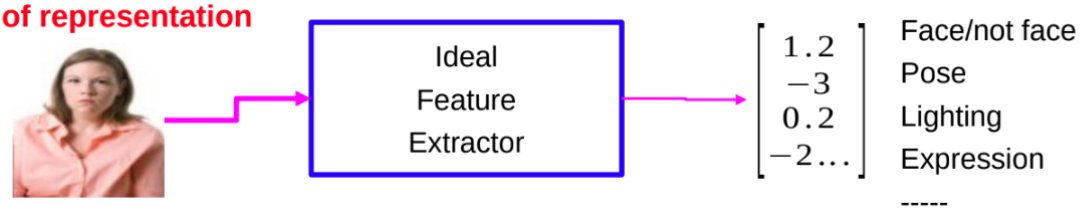
- **SVM (kernel machine)** => 2 layer NeuralNet => chaque unité de la première couche compare le vecteur d'entrée X à l'un des échantillons d'entraînement Xi => on prend le score à chaque fois => on calcule une somme pondérée => on va apprendre les poids (alphas). La première couche est fixée et la seconde couche peut-être entraînée linéairement.




Qu'est-ce qu'on appelle ***profond*** alors ?
SVM => pas profond

***Quelles sont les bonnes représentations ?***
**The Manifold (collecteur) Hypothesis: les données naturelles vivent dans un collecteur de faible dimension (non linéaire), parce que les variables des données naturelles sont mutuellement dépendantes.**

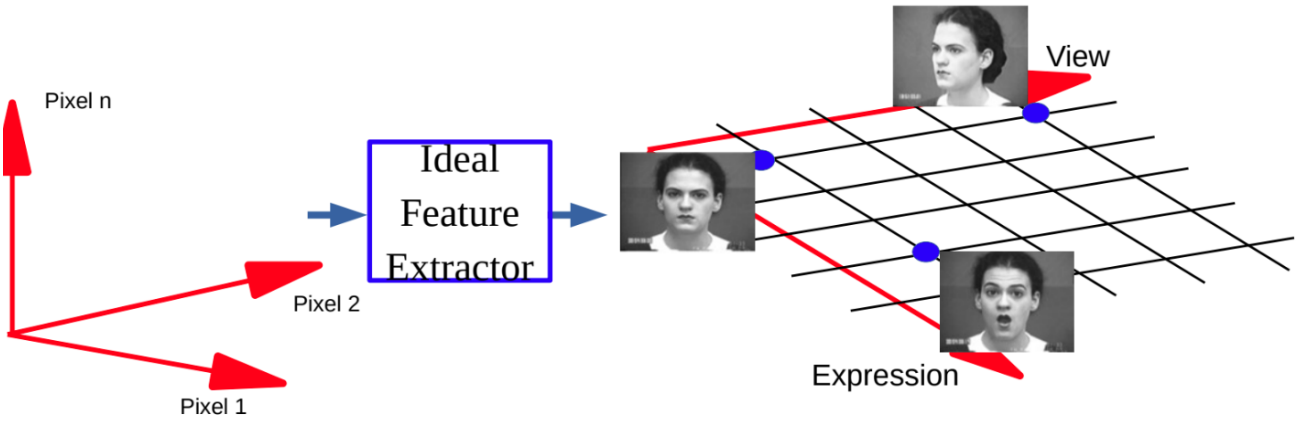
- **Example: all face images of a person**
  - 1000x1000 pixels = 1,000,000 dimensions
  - But the face has 3 Cartesian coordinates and 3 Euler angles
  - And humans have less than about 50 muscles in the face
  - Hence the manifold of face images for a person has <56 dimensions
- **The perfect representations of a face image:**
  - Its coordinates on the face manifold
  - Its coordinates away from the manifold
- **We do not have good and general methods to learn functions that turns an image into this kind of representation**




## Disentangling factors of variation

Y. LeCun

- **The Ideal Disentangling Feature Extractor**




**ATTENTION : la PCA ne gère que des relations LINÉAIRES.**

---

**Résumé** 👍

## L’inspiration de l’apprentissage profond et son histoire

Sur le plan conceptuel, l’apprentissage profond est inspiré par le cerveau. Cependant tous les détails du cerveau ne sont pas pertinents. À titre de comparaison, les avions ont

été inspirés par les oiseaux. Le principe du vol est le même, mais les détails sont extrêmement différents (les avions ne battant pas des ailes).
L'histoire de l'apprentissage profond remonte à un domaine qui a changé de nom pour devenir aujourd'hui la cybernétique. Elle a commencé dans les années 1940 avec McCulloch et Pitts. Ils ont eu comme idée que les neurones sont des unités de seuil avec des états de marche et d'arrêt. De ce fait, il est possible de construire un circuit booléen en connectant les neurones entre eux et d'effectuer des inférences logiques avec les neurones. Le cerveau est essentiellement une machine d'inférence logique car les neurones sont binaires. Les neurones calculent une somme pondérée d'entrées et comparent cette somme à son seuil. Un neurone s'allume s'il est supérieur au seuil et s'éteint s'il est inférieur. Ceci est une vue simplifiée du fonctionnement des réseaux neuronaux.
En 1947, Donald Hebb a eu l'idée que les neurones du cerveau apprennent en modifiant la force des connexions entre eux. C'est ce qu'on appelle l'hyper apprentissage, où si deux neurones sont activés ensemble alors la connexion entre eux augmente et s'ils ne sont pas activés ensemble alors la connexion diminue.
Plus tard, en 1948, la cybernétique a été proposée par Norbert Wiener. Il se base sur l'idée qu'en ayant des systèmes avec des capteurs et des actionneurs, alors il y a une boucle de rétroaction et un système d'autorégulation. Les règles du mécanisme de rétroaction d'une voiture sont toutes issues de son travail.
En 1957, Frank Rosenblatt a proposé le Perceptron, un algorithme d'apprentissage qui modifie les poids de réseaux neuronaux très simples.
Dans l'ensemble, cette idée d'essayer de construire des machines intellectuelles en simulant de nombreux neurones est née dans les années 1940. Elle a pris son essor dans les années 1950 et s'est complètement éteinte à la fin des années 1960. Les principales raisons de la disparition du domaine en 1960 sont les suivantes :

- Les chercheurs ont utilisé des neurones binaires. Cependant, la façon de faire fonctionner la rétropropagation est d'utiliser des fonctions d'activation qui sont continues. À cette époque, les chercheurs n'avaient pas l'idée d'utiliser des neurones continus et ils ne pensaient pas pouvoir réaliser l'entraînement avec des gradients car les neurones binaires ne sont pas différentiels.
- Avec les neurones continus, il faudrait multiplier l'activation d'un neurone par un poids pour obtenir une contribution à la somme pondérée. Cependant, avant 1980, la multiplication de deux nombres, en particulier des nombres à virgule flottante, était extrêmement lente. Cela n'encourageait donc pas à utiliser des neurones continus.

L’apprentissage profond a pris un nouvel essor en 1985 avec l’apparition de la rétropropagation. En 1995, le domaine a périclité à nouveau et la communauté de l’apprentissage machine a abandonné l’idée des réseaux neuronaux. Début 2010, les gens ont commencé à utiliser les réseaux de neurones dans la reconnaissance vocale avec une amélioration considérable des performances et, plus tard, ils ont été largement déployés dans le domaine commercial. En 2013, la vision par ordinateur a commencé à passer aux réseaux de neurones. En 2016, la même transition a eu lieu dans le traitement du langage naturel. Bientôt, des révolutions similaires se produiront dans la robotique, le contrôle et bien d’autres domaines.

### L’apprentissage supervisé

90% des applications d’apprentissage profond utilisent l’apprentissage supervisé. L’apprentissage supervisé est un processus par lequel, on collecte un tas de paires d’entrées et de sorties. Les entrées sont introduites dans une machine pour apprendre la sortie correcte. Lorsque la sortie est correcte, on ne fait rien. Si la sortie est incorrecte, on ajuste le paramètre de la machine afin d’obtenir une nouvelle sortie ressemblant davantage à celle désirée. L’astuce consiste à déterminer la direction et l’ampleur de l’ajustement du paramètre, ce qui revient à calculer le gradient et la rétropropagation. L’apprentissage supervisé est issu du Perceptron et de l’Adaline. L’Adaline est basé sur la même architecture avec des entrées pondérées. Au-dessus du seuil, cela s’allume et en dessous du seuil, cela s’éteint. Le Perceptron est un réseau de neurones à deux couches dont la première couche est fixe et la deuxième couche est entraînable. La plupart du temps, la première couche est déterminée de manière aléatoire et c’est ce qu’on appelle les couches associatives.

## Histoire de la reconnaissance des motifs et de l’introduction de la descente en gradient

La base conceptuelle de la reconnaissance des motifs précède le développement de l’apprentissage profond. Le modèle standard de reconnaissance des motifs comprend un extracteur de caractéristiques et un classifieur pouvant être entraîné. L’entrée va dans l’extracteur de caractéristiques où sont extraites les caractéristiques pertinentes comme la détection d’un œil lorsque le but est de reconnaître le visage. Ensuite, le vecteur de caractéristiques est envoyé au classifieur (pouvant être entraîné) pour calculer la somme pondérée et la comparer avec le seuil. Dans ce cas, un classifieur pouvant être entraîné peut être un perceptron ou un réseau neuronal unique. Le problème est que l’extracteur de caractéristiques doit être conçu à la main. Ce qui signifie que la reconnaissance des motifs et la vision par ordinateur se concentrent sur l’extracteur en considérant la façon

de le concevoir pour un problème particulier, ce qui n'est pas vraiment le cas d'un classifieur pouvant être entraîné.
Après l'émergence et le développement de l'apprentissage profond, le processus en deux étapes est passé à des séquences de modules. Chaque module a des paramètres réglables et une non-linéarité. Ensuite, il faut les empiler pour former plusieurs couches. C'est pourquoi on l'appelle « apprentissage profond ». La raison pour laquelle on utilise une non-linéarité plutôt qu'une linéarité est que deux couches linéaires reviennent à une seule couche linéaire puisque la composition de deux linéaires est linéaire.
L'architecture multicouche la plus simple avec des paramètres réglables et la non-linéarité pourrait être la suivante. L'entrée est représentée par un vecteur tel qu'une image ou un son. Cette entrée est multipliée par la matrice de poids dont le coefficient est un paramètre réglable. Ensuite, chaque composante du vecteur de résultat est passée par une fonction non linéaire telle que la ReLU (Rectified Linear Unit). En répétant ce processus, cela devient un réseau neuronal de base. La raison pour laquelle on parle de neurones est que cette architecture calcule la somme pondérée des composantes d'entrée par les lignes correspondantes d'une matrice.
Pour en revenir à l'apprentissage supervisé, nous comparons le résultat obtenu avec le résultat cible puis nous optimisons la fonction objectif qui est la perte du calcul de la distance/pénalité/divergence entre le résultat et la cible. Ensuite, nous faisons la moyenne de cette fonction de coût sur le jeu d'entraînement. C'est l'objectif que nous voulons minimiser. En d'autres termes, nous voulons trouver la valeur des paramètres qui minimisent cette moyenne.
La méthode pour trouver cette valeur est le calcul du gradient. Par exemple, si nous sommes perdus dans une montagne par une nuit brumeuse et nous voulons aller au village dans la vallée, un moyen pourrait être de faire demi-tour et de voir quel est le chemin le plus raide pour descendre puis de faire un petit pas vers le bas. La direction est le gradient (négatif). En supposant que la vallée est convexe, nous pouvons atteindre la vallée.
Le chemin le plus efficace est appelé la « descente de gradient stochastique » (SGD en anglais pour Stochastic Gradient Descent). Comme nous voulons minimiser la perte moyenne sur le jeu d'entraînement, nous prenons un échantillon ou un petit groupe d'échantillons et nous calculons l'erreur. Puis nous utilisons la descente de gradient. Ensuite, nous prenons un nouvel échantillon et obtenons une nouvelle valeur pour l'erreur. Nous avons alors le gradient qui est normalement une direction différente. Deux des principales raisons d'utiliser la SGD sont qu'elle permet à un modèle de converger rapidement si le jeu d'entraînement est très grand et qu'elle permet une meilleure généralisation, c'est-à-dire qu'elle donne des performances similaires sur différents jeux de données.

### Calculer les gradients par rétropropagation

Le calcul des gradients par rétropropagation est une application pratique de la règle de la chaîne. L'équation de rétropropagation pour les gradients d'entrée est la suivante :

$$\frac{\partial C}{\partial \boldsymbol{x}_{i-1}} = \frac{\partial C}{\partial \boldsymbol{x}_i} \frac{\partial \boldsymbol{x}_i}{\partial \boldsymbol{x}_{i-1}}$$
$$\frac{\partial C}{\partial \boldsymbol{x}_{i-1}} = \frac{\partial C}{\partial \boldsymbol{x}_i} \frac{\partial f_i(\boldsymbol{x}_{i-1}, \boldsymbol{w}_i)}{\partial \boldsymbol{x}_{i-1}}$$

Dans la même logique, l'équation de rétropropagation pour les gradients des poids est la suivante :

$$\frac{\partial C}{\partial \boldsymbol{w}_i} = \frac{\partial C}{\partial \boldsymbol{x}_i} \frac{\partial \boldsymbol{x}_i}{\partial \boldsymbol{w}_i}$$
$$\frac{\partial C}{\partial \boldsymbol{w}_i} = \frac{\partial C}{\partial \boldsymbol{x}_i} \frac{\partial f_i(\boldsymbol{x}_{i-1}, \boldsymbol{w}_i)}{\partial \boldsymbol{w}_i}$$

Notez qu'au lieu d'entrées scalaires, il s'agit d'entrées vectorielles. Plus généralement, il s'agit d'entrées multidimensionnelles. La rétropropagation permet de calculer la dérivée de la différence entre la sortie souhaitée et la sortie obtenue (qui est la valeur de la fonction objectif) par rapport à toute valeur à l'intérieur du réseau. Enfin, la rétropropagation est essentielle car elle s'applique à plusieurs couches.

Il est important de réfléchir à la manière d'interpréter les entrées. Par exemple, une image de $256 \times 256$ nécessite une matrice d'une valeur de $200000$. On aurait alors d'énormes matrices que les couches du réseau neuronal auraient à manipuler. Il n'est donc pas pratique d'utiliser de telles matrices. Par conséquent, il est important de faire des hypothèses sur la structure de la matrice.

## Représentation hiérarchique du cortex visuel

Les expériences de Fukushima nous ont permis de comprendre comment notre cerveau interprète les données transmises à nos yeux. En résumé, les neurones situés devant notre rétine compriment l'entrée (connue sous le nom de normalisation du contraste) et le signal voyage de nos yeux à notre cerveau. Ensuite, l'image est traitée par étapes et certains neurones sont activés pour certaines catégories. Ainsi, le cortex visuel effectue une reconnaissance des motifs de manière hiérarchique.
Des expériences dans lesquelles des chercheurs ont enfoncé des électrodes dans des zones spécifiques du cortex visuel, en particulier la zone V1, ont permis de réaliser que certains neurones réagissent à des motifs qui apparaissent dans une très petite zone du champ visuel et de la même façon avec les neurones voisins et les zones voisines du

champ visuel. En outre, les neurones qui réagissent au même champ visuel réagissent à différents types de bords de manière organisée (par exemple, les bords verticaux ou horizontaux). Il est également important de noter qu'il y a aussi l'idée que le processus visuel est essentiellement un processus feed forward. Par conséquent, une reconnaissance rapide peut être effectuée d'une manière ou d'une autre, sans que certaines connexions récurrentes soient nécessaires.

## Evolution des ConvNets

Dans le cerveau des animaux, les neurones réagissent aux bords qui ont des orientations particulières. Les groupes de neurones qui réagissent aux mêmes orientations sont reproduits sur l'ensemble du champ visuel.
En se basant sur deux concepts, Fukushima (1982) a construit un réseau de neurones qui fonctionne de la même manière que le cerveau. Premièrement, les neurones sont répliqués sur l'ensemble du champ visuel. Deuxièmement, il existe des cellules complexes qui regroupent les informations provenant de cellules simples (unités d'orientation-sélection). Par conséquent, le déplacement de l'image modifie l'activation des cellules simples mais n'influence pas l'activation intégrée de la cellule complexe (le pooling convolutif).
Yann Le Cun (1990) a utilisé la rétropropagation pour entraîner un ConvNet à reconnaître les chiffres manuscrits. Il existe une démo de 1993 où l'algorithme reconnaît les chiffres de n'importe quel style. La reconnaissance de caractères/motifs à l'aide d'un modèle entraîné de bout en bout était nouvelle à l'époque. Auparavant, les gens utilisaient des extracteurs de caractéristiques avec un modèle supervisé par-dessus.
Ces nouveaux systèmes ConvNets pouvaient reconnaître plusieurs caractères dans l'image en même temps. Pour ce faire, les gens utilisaient une petite fenêtre de saisie pour un ConvNet et la glissaient sur toute l'image. Si elle s'activait, cela signifiait qu'un caractère particulier était présent.
Plus tard, cette idée a été appliquée à la détection des visages/personnes et à la segmentation sémantique (classification au niveau du pixel). Citons par exemple Hadsell (2009) et Farabet (2012). Cette méthode a fini par devenir populaire dans l'industrie, utilisée dans des applications de conduite autonome telles que le suivi de voie.
Les capacités de calcul pour entraîner un ConvNet étaient un sujet brûlant dans les années 1980, puis l'intérêt a chuté, et est maintenant redevenu populaire.
La révolution de l'apprentissage profond (bien que le terme ne soit pas utilisé à l'époque) a commencé en 2010-2013. Les chercheurs se sont concentrés sur l'invention d'algorithmes qui pourraient aider à entraîner plus rapidement les gros ConvNets. Krizhevsky (2012) a conçu AlexNet, qui était un ConvNet beaucoup plus grand que ceux utilisés auparavant, et l'a entraîné sur ImageNet (1,3 million d'échantillons) en utilisant

des GPUs. Après quelques semaines de fonctionnement, AlexNet a largement dépassé les performances des meilleurs systèmes concurrents avec un taux d'erreur pour les 5 premières sorties du modèle (Top-5 error rate) de 16,4% contre 25,8% pour les autres.
Après avoir vu le succès d'AlexNet, la communauté de la vision par ordinateur (souvent abrégée en CV en anglais pour Computer Vision) a été convaincue que les ConvNets fonctionnent. Alors que tous les articles de 2011-2012 mentionnant les ConvNets étaient alors rejetés, depuis 2016 la plupart des papiers acceptés utilisent les ConvNets.
Au fil des ans, le nombre de couches utilisées a augmenté : 7 pour LeNet, 12 pour AlexNet, 19 pour VGG, 50 pour ResNet. Toutefois, il y a un compromis à faire entre le nombre d'opérations nécessaires pour calculer le résultat, la taille du modèle et sa précision.
Ainsi, un sujet actuellement populaire est de savoir comment comprimer les réseaux pour rendre les calculs plus rapides.

## Apprentissage profond et extraction de caractéristiques

Les réseaux multicouches ont du succès parce qu'ils exploitent la structure des données naturelles (la parole, les images, le texte). Dans la hiérarchie des compositions, les combinaisons d'objets à une couche de la hiérarchie forment les objets de la couche suivante. Si nous imitons cette hiérarchie sous forme de couches multiples et que nous laissons le réseau apprendre la combinaison de caractéristiques appropriées, nous obtenons ce que l'on appelle l'architecture d'apprentissage profond. Ainsi, les réseaux d'apprentissage profond sont de nature hiérarchique.
Les architectures d'apprentissage profond ont conduit à un progrès incroyable dans les tâches de vision par ordinateur, allant de l'identification et de la génération de masques précis autour d'objets à l'identification des propriétés spatiales d'un objet. Les architectures Mask-RCNN et RetinaNet ont principalement conduit à cette amélioration.
Mask-RCNN est utile dans la segmentation d'objets individuels, c'est-à-dire la création de masques pour chaque objet d'une image. L'entrée et la sortie sont toutes deux des images. L'architecture peut également être utilisée pour faire de la segmentation d'instance, c'est-à-dire identifier différents objets du même type dans une image.
Detectron, un système logiciel de Facebook AI Research (FAIR), met en œuvre tous ces algorithmes de détection d'objets de pointe et est open source.
Certaines des applications pratiques des ConvNets sont l'aide à la conduite et l'analyse d'images médicales.
Bien que les sciences et les mathématiques qui sous-tendent l'apprentissage profond soient assez bien comprises, il reste quelques questions intéressantes qui nécessitent davantage de recherche. Ces questions sont les suivantes :

- Pourquoi les architectures à plusieurs couches sont-elles plus performantes, étant donné que nous pouvons approximer n'importe quelle fonction avec deux couches ?
- Pourquoi les ConvNets fonctionnent-ils aussi bien avec des données naturelles telles que la parole, les images et le texte ?
- Comment sommes-nous capables d'optimiser aussi bien les fonctions non convexes ?
- Pourquoi les architectures sur paramétrées fonctionnent-elles ?

L'extraction de caractéristiques consiste à élargir la dimension de représentation de telle sorte que les caractéristiques élargies aient plus de chances d'être linéairement séparables (en raison de l'augmentation du nombre de plans de séparation possibles). Auparavant, les experts en apprentissage machine s'appuyaient sur des caractéristiques de haute qualité, artisanales et spécifiques à une tâche afin de construire des modèles d'intelligence artificielle. Mais avec l'avènement de l'apprentissage profond, les modèles sont capables d'extraire automatiquement les caractéristiques génériques. Certaines approches courantes utilisées dans les algorithmes d'extraction de caractéristiques sont mises en évidence ci-dessous :

- Pavage de l'espace (Space tiling)
- Projections aléatoires (Random Projections)
- Classifieur polynomial (Polynomial Classifier)
- Fonctions radiales (Radial basis functions)
- Machines à noyaux (Kernel Machines)

En raison de la nature compositionnelle des données, les traits appris ont une hiérarchie de représentations avec un niveau d'abstraction croissant. Par exemple :

- pour les images:
  Au niveau le plus granulaire, les images peuvent être considérées comme des pixels. La combinaison de pixels constitue des bords qui, lorsqu'ils sont combinés, forment des textons (formes à bords multiples). Les textons forment des motifs et les motifs font partie de l'image. En combinant ces parties ensemble, on obtient l'image finale.
- pour le texte:
  De même, il existe une hiérarchie inhérente aux données textuelles. Les caractères forment des mots, lorsque nous combinons des mots ensemble, nous obtenons des groupes de mots, puis en les combinant nous obtenons des phrases. Les phrases nous disent finalement quelle histoire est véhiculée.
- pour la parole:
  Dans la parole, les échantillons composent des bandes, qui composent des sons,

puis des phonèmes, puis des mots entiers, puis des phrases, montrant ainsi une hiérarchie claire dans la représentation.

## Apprendre les représentations

Certaines personnes rejettent l'apprentissage profond car si nous pouvons approcher n'importe quelle fonction avec deux couches, pourquoi en avoir plus ?
Par exemple, les SVMs (de l'anglais support vector machines, pouvant être traduit en français par machines à vecteurs de support) trouvent un hyperplan de séparation dans l'étendue des données. Cela signifie que les prédictions sont basées sur des comparaisons avec des exemples d'entraînement. Les SVMs peuvent être vues comme un réseau neuronal à deux couches très simpliste, où la première couche définit des « pochoirs » et la seconde couche est un classifieur linéaire. Le problème avec deux couches est que la complexité et la taille de la couche intermédiaire sont exponentielles en N (pour réussir une tâche difficile, il faut BEAUCOUP de modèles). Mais si vous augmentez le nombre de couches à log(N), les couches deviennent linéaires en N. Il y a un compromis entre le temps et l'espace.
Une analogie consiste à concevoir un circuit pour calculer une fonction booléenne avec un maximum de deux couches de portes : nous pouvons calculer toute fonction booléenne de cette façon ! Mais, la complexité et les ressources de la première couche (nombre de portes) deviennent rapidement inutilisables pour les fonctions complexes.
Qu'est-ce que la « profondeur » ?

- Une SVM n'est pas profonde car ne comporte que deux couches
- Un arbre de classification n'est pas profond car chaque couche analyse les mêmes caractéristiques (brutes)
- Un réseau profond comporte plusieurs couches et les utilise pour construire une hiérarchie de caractéristiques d'une complexité croissante

Comment les modèles peuvent-ils apprendre les représentations (les bonnes caractéristiques) ?
L'ensemble des images possibles est essentiellement infini, l'ensemble des images « naturelles » est un minuscule sous-ensemble. Par exemple : pour une image d'une personne, l'ensemble des images possibles est de l'ordre de grandeur du nombre de muscles du visage qu'elle peut bouger (degrés de liberté) soit environ 50. Un extracteur de caractéristiques idéal (et irréaliste) représente tous les facteurs de variation (chacun des muscles, éclairage, etc.). C'est l'hypothèse de la multiplicité : les données naturelles vivent dans une multiplicité à faible dimension.

**Réponses aux questions d'étudiants :**

**Pour l’exemple du visage, l’ACP qui est une technique de réduction de la dimensionnalité pourrait-elle extraire ces traits ?**

Cela ne fonctionnerait que si la surface est un hyperplan, ce qui n’est pas le cas.