

# Réseaux de neurones

masedki.github.io

jeu. 09 janv. 2020

# Plan

- Définition et caractéristiques des réseaux de neurones limitée aux perceptrons multicouches spécifiques pour la régression et la classification supervisée.
- Structure, fonctions de transfert, algorithme d'apprentissage par rétro-propagation du gradient, contrôles du sur-ajustement.
- Introduction à l'apprentissage profond.

## Un peu d'histoire (source : Philippe Besse)

L'Intelligence Artificielle, branche de l'Informatique fondamentale s'est développée avec pour objectif la simulation des comportements du cerveau humain. Les premières tentatives de modélisation du cerveau sont anciennes et précèdent même l'ère informatique. C'est en 1943 que Mc Culloch (neuro- physiologiste) et Pitts (logicien) ont proposé les premières notions de neurone formel. Ce concept fut ensuite mis en réseau avec une couche d'entrée et une sortie par Rosenblatt en 1959 pour simuler le fonctionnement rétinien et tacher de reconnaître des formes. C'est l'origine du **perceptron**. Cette approche dite *connexioniste* a atteint ses limites technologiques, compte tenu de la puissance de calcul de l'époque, mais aussi théoriques au début des années 70.

L'approche connexioniste à connaissance répartie a alors été supplantée par d'autres approches.

## Années 80 et 90 (source Philippe Besse)

L'essor technologique et quelques avancées théoriques :

- estimation du gradient par rétro-propagation de l'erreur (Hopkins, 1982)
- analogie de la phase d'apprentissage avec les modèles markoviens de systèmes de particules de la mécanique statistique (verres de spin) par (Hopfield, 1982)

au début des années 80 ont permis de relancer l'approche connexioniste. Celle-ci a connu au début des années 90 un développement considérable si l'on considère le nombre de publications et de congrès qui lui ont été consacrés mais aussi les domaines d'applications très divers où elle apparaît. La motivation initiale de simulation du cortex cérébral a été rapidement abandonnée alors que les méthodes qui en découlaient ont trouvé leur propre intérêt de développement méthodologique et leurs champs d'applications.

## Années 90 (source Philippe Besse)

Remis en veilleuse depuis le milieu des années 90 au profit d'autres algorithmes d'apprentissage machine ou plutôt statistique : *boosting*, *support vector machine*..., les réseaux de neurones connaissent un regain d'intérêt et même un énorme battage médiatique sous l'appellation d'apprentissage profond (deep learning). La taille des bases de données, notamment celles d'images issues d'internet, associée à la puissance de calcul disponible, permettent d'estimer les millions de paramètres du perceptron accumulant des dizaines voire centaines de couches de neurones aux propriétés très spécifiques. Ce succès médiatique est la conséquence des résultats spectaculaires obtenus par ces réseaux en reconnaissance d'image, jeux de go, traitement du langage naturel.

## Réseaux de neurones

Un réseau neuronal est l'association, en un graphe plus ou moins complexe, d'objets élémentaires, les neurones formels. Les principaux réseaux se distinguent par l'organisation du graphe (en couches, complets...), c'est-à-dire leur architecture, son niveau de complexité (le nombre de neurones, présence ou non de boucles de rétroaction dans le réseau), par le type des neurones (leurs fonctions de transition ou d'activation) et enfin par l'objectif visé : apprentissage supervisé ou non, optimisation, systèmes dynamiques...

# Neurone formel

De façon très réductrice, un neurone biologique est une cellule qui se caractérise par

- Des synapses, les points de connexion avec les autres neurones, fibres nerveuses ou musculaires.
- Des dendrites ou entrées du neurones.
- Les axones, ou sorties du neurone vers d'autres neurones ou fibres musculaires.
- Le noyau qui active les sorties en fonction des stimulations en entrée.

## Neurone formel

Par analogie, le neurone formel est un modèle qui se caractérise par un état interne  $s \in \mathcal{S}$ , des signaux d'entrée  $x_1, \dots, x_p$  et une fonction d'activation

$$s = h(x_1, \dots, x_p) = g\left(\alpha_0 + \sum_{j=1}^p \alpha_j x_j\right) = g(\alpha_0 + \boldsymbol{\alpha}^\top \mathbf{x})$$

La fonction d'activation opère une transformation d'une combinaison affine des signaux d'entrée,  $\alpha_0$ , **terme constant**, étant appelé le **biais du neurone**. Cette combinaison affine est déterminée par un vecteur des poids  $(\alpha_0, \dots, \alpha_p)$  associé à chaque neurone et dont les valeurs sont estimées dans la phase d'apprentissage. Ils constituent la *mémoire ou connaissance répartie* du réseau.

# Représentation d'une neurone formel

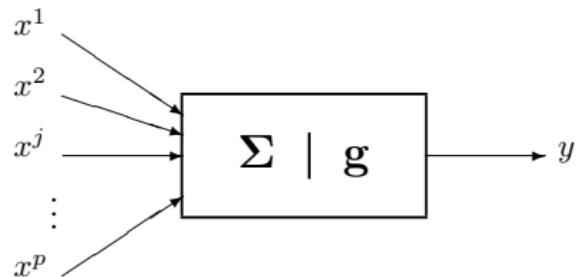


FIGURE 1 – *Représentation d'un neurone formel.*

# Différentes types de neurones

Les différents types de neurones se distinguent par la nature  $g$  de leur fonction d'activation. Les principaux types sont :

- *linéaire* si  $g$  est la fonction identité
- *seuil* si  $g(x) = \mathbf{1}_{[0,+\infty[}(x)$
- *sigmoïde* si  $g(x) = \frac{1}{1+e^x}$
- *ReLU* si  $g(x) = \max(0, x)$  (*rectified linear unit*)
- *softmax* si  $g(x)_j = \frac{e^{x_j}}{\sum_{k=1}^K e^{x_k}}$  pour tout  $k \in \{1, \dots, K\}$ .
- *radiale* si  $g(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ .
- *stochastique* si  $g(x) = 1$  avec probabilité  $\frac{1}{1+e^{-\frac{x}{H}}}$ , 0 sinon ( $H$  intervient comme paramètre dit de *température* dans l'algorithme de récuit simulé).

# Fonction d'activation

- Les modèles linéaires, sigmoïdaux, ReLU, softmax sont bien adaptés aux algorithmes d'apprentissage impliquant une rétro-propagation du gradient car leur fonction d'activation est différentiable; ce sont les plus utilisées.
- Le modèle à seuil est sans doute plus conforme à la réalité biologique mais pose des problèmes d'apprentissage.
- Enfin le modèle stochastique est utilisé pour des problèmes d'optimisation globale de fonctions perturbées ou encore pour les analogies avec les systèmes de particules (machine de Boltzmann pour l'apprentissage non-supervisé).

# Perceptron multicouche

- Nous nous intéresserons dans ce cours qu'à une structure élémentaire de réseau, celle dite statique ne présentant pas de boucle de rétroaction et dans un but d'apprentissage supervisé.
- Les systèmes dynamiques, avec boucle de rétroaction, les réseaux récurrents (LSTM) ainsi que les cartes de *Kohonen* ou cartes auto-organisatrices (*auto-encoders*) pour la classification non-supervisée ne sont pas abordés.

# Perceptron multicouche (représentation)

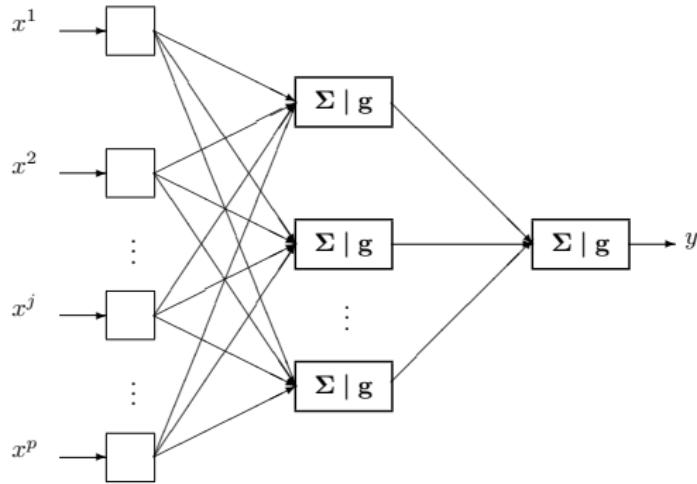


FIGURE 2 – Exemple de perceptron multicouche élémentaire avec une couche cachée et une couche de sortie.

## Architecture

- Le perceptron multicouche (PMC) est un réseau composé de couches successives.
- Une **couche** est un ensemble de neurones n'ayant pas de connexion entre eux
- Une couche d'entrée lit les signaux entrant (variables), un neurone par entrée  $x_j$ .
- Une couche de sortie fournit la réponse du système.
- Selon les auteurs, la couche d'entrée qui n'introduit aucune modification n'est pas comptabilisée.
- **Une ou plusieurs** couches cachées participent au transfert.
- Dans un perceptron, un neurone d'une couche cachée est connecté en entrée à chacun des neurones de la couche précédente et en sortie à chaque neurone de la couche suivante.

## Fonction de transfert

- Par souci de cohérence avec les autres parties du cours, les entrées d'un réseau sont encore notées  $X_1, \dots, X_p$  comme les variables explicatives d'un modèle tandis que les poids des entrées sont des paramètres  $\alpha$  et  $\beta$  à estimer lors de procédure d'apprentissage et que la *sortie* est la variable  $Y$  à expliquer ou variable réponse du problème.
- Un perceptron multicouche réalise donc une transformation des variables d'entrée

$$Y = f(X_1, \dots, X_p; \alpha)$$

où  $\alpha$  est le vecteur contenant chacun des paramètres  $\alpha_{jkl}$  de la  $j$ ème entrée du  $k$ ème neurone de la  $\ell$ ème couche; la couche d'entrée ( $\ell = 0$ ) n'est pas paramétrée, elle ne fait que distribuer les entrées sur tous les neurones de la couche suivante.

## Fonction d'activation

- En régression ( $Y$  quantitative), la dernière couche est constituée d'un seul neurone muni de la fonction d'activation identité tandis que les autres neurones (couche cachée) sont munis de la fonction sigmoïde.
- En classification binaire, le neurone de sortie est muni également de la fonction sigmoïde tandis que dans le cas de classification à  $K$  classes ( $Y$  qualitative), le neurone de sortie intègre une fonction d'activation *softmax* à valeurs dans  $[0, 1]$  de somme 1. Ces  $K$  valeurs sont assimilables à des probabilités d'appartenance à une classe.

Ainsi, en régression avec un perceptron à une couche cachée de  $q$  neurones et un neurone de sortie, cette fonction s'écrit :

$$y = f(\mathbf{x}; \boldsymbol{\alpha}, \boldsymbol{\beta}) = \beta_0 + \boldsymbol{\beta}^\top \mathbf{z}$$

avec  $z_k = g(\alpha_0 + \boldsymbol{\alpha}^\top \mathbf{x}); \quad k = 1, \dots, q.$

# Apprentissage

- Supposons que l'on dispose d'une base d'apprentissage de taille  $n$  d'observations  $(x_{i1}, \dots, x_{ip}; y_i)$  des variables explicatives  $X_1, \dots, X_p$  et de la variable réponse  $Y$ .
- Considérons le cas le plus simple de la régression avec un réseau constitué d'un neurone de sortie linéaire et d'une couche à  $q$  neurones dont les paramètres sont optimisés pour minimiser les moindres carrés. Ceci se généralise à toute fonction perte dérivable et donc à la classification à  $K$  classes.
- L'apprentissage est l'estimation des paramètres  $\alpha_{j=0,p;k=1,q}$  et  $\beta_{k=0,q}$  par minimisation de la fonction de perte quadratique ou de celle d'une fonction d'entropie en classification :

$$Q(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_{i=1}^n Q_i = \sum_{i=1}^n \left[ y_i - f(\mathbf{x}_i; \boldsymbol{\alpha}, \boldsymbol{\beta}) \right]^2 = \sum_{i=1}^n \left[ y_i - \beta_0 - \sum_{k=1}^q \beta_k g(\alpha_{k0} + \boldsymbol{\alpha}_k^\top \mathbf{x}_i) \right]^2$$

## Rétro-propagation de l'erreur

Différents algorithmes d'optimisation sont proposés, ils sont généralement basés sur une évaluation du gradient par rétro-propagation. Il s'agit donc d'évaluer la dérivée de la fonction coût en une observation et par rapport aux différents paramètres. Soit  $z_{ki} = g(\alpha_{k0} + \boldsymbol{\alpha}_k^\top \mathbf{x}_i)$  et  $\mathbf{z}_i = (z_{1i}, \dots, z_{qi})$ . Les dérivées partielles de la fonction perte quadratique s'écrivent

$$\begin{cases} \frac{\partial Q_i}{\partial \beta_k} = -2 \left( y_i - \beta_0 - \sum_{k=1}^q \beta_k g(\alpha_{k0} + \boldsymbol{\alpha}_k^\top \mathbf{x}_i) \right) z_{ki} = \delta_i z_{ki} \\ \frac{\partial Q_i}{\partial \alpha_{kj}} = -2 \left( y_i - \beta_0 - \sum_{k=1}^q \beta_k g(\alpha_{k0} + \boldsymbol{\alpha}_k^\top \mathbf{x}_i) \right) \beta_k g'(\alpha_{k0} + \boldsymbol{\alpha}_k^\top \mathbf{x}_i) x_{ij} = s_{ki} x_{ij}. \end{cases}$$

Les termes  $\delta_i$  et  $s_{ki}$  sont respectivement les termes d'erreur du modèle courant à la sortie et sur chaque neurone caché. Ces termes d'erreur vérifient les équations dites de rétro-propagation :

$$s_{ki} = g'(\alpha_{k0} + \boldsymbol{\alpha}_k^\top \mathbf{x}_i) \beta_k \delta_i$$

dont les termes sont évalués en deux passes.

## Évaluation des $s_{ki}$ et $\delta_i$

- Une *passe avant*, avec les valeurs courantes des poids : l'application des différentes entrées  $\mathbf{x}_i$  au réseau permet de déterminer les valeurs ajustées

$$\widehat{\beta}_0 + \sum_{k=1}^q \widehat{\beta}_k g\left(\widehat{\alpha}_{k0} + \widehat{\boldsymbol{\alpha}}_k^\top \mathbf{x}_i\right).$$

- La *passe retour* permet ensuite de déterminer les  $\delta_i$  qui sont rétro-propagés afin de calculer les  $s_{ki}$  et ainsi obtenir les évaluations des gradients.

## Pour une fonction softmax

Petit rappel

$$\text{softmax}(x_1, \dots, x_K) = \frac{(e^{x_1}, \dots, e^{x_K})}{\sum_{k=1}^K e^{x_k}}.$$

Sa dérivée

$$\frac{\partial \text{softmax}(\mathbf{x})_i}{\partial x_j} = \begin{cases} \text{softmax}(\mathbf{x})_i(1 - \text{softmax}(\mathbf{x})_i) & \text{si } i = j \\ -\text{softmax}(\mathbf{x})_i \text{softmax}(\mathbf{x})_j & \text{si } i \neq j. \end{cases}$$

## Algorithmes d'optimisation

Sachant évaluer les gradients, différents algorithmes, plus ou moins sophistiqués, sont implémentés. Le plus élémentaire est une utilisation itérative du gradient : en tout point de l'espace des paramètres, le vecteur gradient de  $Q$  pointe dans la direction de l'erreur croissante. Pour faire décroître  $Q$  il suffit donc de se déplacer en sens contraire. Il s'agit d'un algorithme itératif modifiant les poids de chaque neurone selon :

$$\begin{cases} \beta_k^{(r+1)} &= \beta_k^{(r)} - \tau \sum_{i=1}^n \frac{\partial q_i}{\partial \beta_k^{(r)}} \\ \alpha_{kj}^{(r+1)} &= \alpha_{kj}^{(r)} - \tau \sum_{i=1}^n \frac{\partial q_i}{\partial \alpha_{kj}^{(r)}} \end{cases}$$

Le coefficient de proportionnalité  $\tau$  est appelé le *taux d'apprentissage*. Il peut être fixe, à déterminer par l'utilisateur, ou encore varier en cours d'exécution selon certaines heuristiques. Il paraît en effet intuitivement raisonnable que, grand au début pour aller plus vite, ce taux baisse pour aboutir à un réglage plus fin au fur et à mesure que le système d'approche d'une solution. Si l'espace mémoire est suffisant, une version accélérée de l'algorithme fait intervenir à chaque itération un ensemble (*batch*) d'observations pour moyenner les gradients et la mise à jour des poids.

## Contrôle de la complexité par régularisation

Dans les réseaux élémentaires, une option simple pour éviter le sur-apprentissage consiste à introduire une terme de pénalisation ou régularisation, comme en régression *ridge*, dans le critère à optimiser. Celui-ci devient alors :

$$Q(\theta) + \gamma \|\theta\|^2.$$

Plus la valeur du paramètre  $\gamma$  est importante et moins les poids des entrées des neurones peuvent prendre des valeurs chaotiques contribuant ainsi à limiter les risques de sur-apprentissage.

## Choix des paramètres (exemple de la librairie keras)

L'utilisateur doit donc déterminer

- l'architecture du réseau : le nombre de couches cachées qui correspond à une aptitude à traiter des problèmes de non-linéarité, le nombre de neurones par couche cachée. Ces deux choix conditionnent directement le nombre de paramètres (de poids) à estimer et donc la complexité du modèle. Ils participent à la recherche d'un bon compromis biais/variance c'est-à-dire à l'équilibre entre qualité d'apprentissage et qualité de prévision.
- la taille des ensembles ou *batchs* d'observations considérés à chaque itération.

## Remarques

- Les champs d'application des PMC sont très nombreux : discrimination, prévision d'une série temporelle, reconnaissance de forme... Ils sont en général bien explicités dans les documentations des logiciels spécialisés.
- Les critiques principales énoncées à l'encontre du PMC concernent les difficultés liés à l'apprentissage (temps de calcul, taille de l'échantillon, localité de l'optimum obtenu) ainsi que son statut de boîte noir. En effet, contrairement à un modèle de discrimination ou un arbre, il est a priori impossible de connaître l'influence effective d'une entrée (une variable) sur le système dès qu'une couche cachée intervient. Néanmoins, des techniques de recherche de sensibilité du système à chacune des entrées permettent de préciser les idées et, éventuellement de simplifier le système en supprimant certaines des entrées.
- En revanche, ils possèdent d'indéniables qualités lorsque l'absence de linéarité et/ou le nombre de variables explicatives (images) rendent les modèles statistiques traditionnelles inutilisables. Leur flexibilité par l'introduction de couches spécifiques en apprentissage profond, alliée à une procédure d'apprentissage intégrant la pondération (le choix) des variables comme de leurs interactions peuvent les rendre très efficaces.

## Apprentissage profond : préambule

Pendant les années 90s et le début des années 2000, le développement de l'apprentissage machine s'est focalisé sur les algorithmes de machines à vecteurs supports et ceux d'agrégation de modèles. Pendant une relative mise en veilleuse du développement de la recherche sur les réseaux de neurones, leur utilisation est restée présente de même qu'une veille attendant le développement de la puissance de calcul et celle des grandes bases de données, notamment d'images.

Le renouveau de la recherche dans ce domaine est dû à Geoffrey Hinton, Yoshua Bengio et Yan le Cun qui a tenu à jour un célèbre site dédié à la reconnaissance des caractères manuscrits de la base MNIST. La liste des publications listées sur ce site témoigne de la lente progression de la qualité de reconnaissance, de 12% avec un simple perceptron à 1 couche jusqu'à moins de 0.3% en 2012 par l'introduction et l'amélioration incrémentale d'une couche de neurones spécifique appelée convolutional neural network (ConvNet). L'étude de ces données qui ont servi de benchmark pour la comparaison de très nombreuses méthodes sert maintenant de données jouet pour beaucoup de tutoriels des environnements dédiés (*tensorFlow, Keras, pyTorch, caffe...*)

## Trois familles de réseaux de neurones profonds

Schématiquement, trois grandes familles de réseaux d'apprentissage profond sont développées avec des ambitions industrielles en profitant du développement des cartes graphiques (GPU) pour paralléliser massivement les calculs au moment de l'apprentissage.

- *convolutional neural networks* (ConvNet) pour l'analyse d'images.
- *long-short term memory* (LSTM) lorsqu'une dimension temporelle ou plus généralement des propriétés d'autocorrélation sont à prendre en compte pour le traitement du signal ou encore l'analyse du langage naturel.
- *autoEncoder decoder* ou réseau *diabolo* en apprentissage non supervisé pour, par exemple, le débruitage d'images ou signaux, la détection d'anomalies.

On s'intéresse ici à la première famille de réseaux.

# Reconnaissance d'images

Une couche de neurones (ConvNet) ou plutôt un empilement de ces couches induit des propriétés locales d'invariance par *translation*. Ces propriétés sont indispensables à l'objectif de reconnaissance angles différents. C'est dans ce domaine que les résultats les plus spectaculaires ont été obtenus tandis que l'appellation *deep learning* était avancée afin d'accompagner le succès grandissant et le battage médiatique associé.

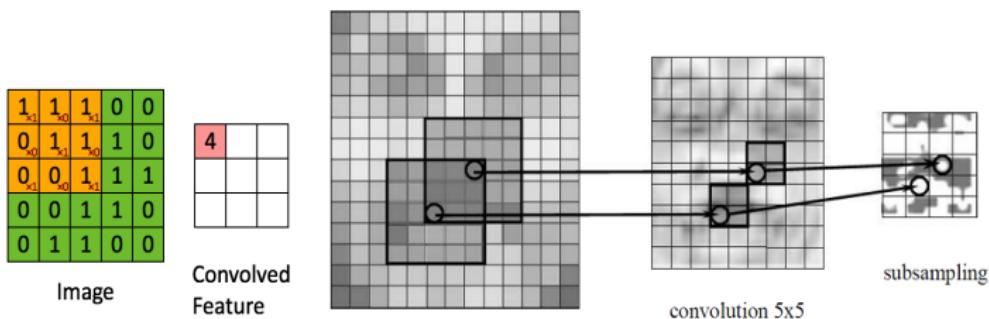


Figure 1: Principe d'une couche de convolution appliquée à une image

## Concours de reconnaissance d'images

La communauté de reconnaissance d'images se confronte chaque année depuis 2010 sur une jeu de données issues d'une base d'images labellisées : 15 millions d'images, 22000 catégories hiérarchisées. De cette base sont extraites 1.2 millions d'images pour l'apprentissage avec 1000 catégories. Les participants au concours doivent prévoir la catégorie de 15000 images de l'échantillon test. Ce projet à l'initiative de l'Université de Stanford est largement soutenu par Google. Comme pour les données de reconnaissance de caractères, une progression largement empirique à conduit à l'introduction et au succès d'un réseau empilant des couches de neurones aux propriétés particulières. C'est en 2012 qu'une équipe utilise pour la millions de paramètres. Une mise en œuvre simple sur des données spécifiques première fois un réseau de neurones profond contrairement à des traitements spécifiques et ad'hoc de l'analyse d'images utilisées jusque là. L'amélioration était telle que toutes les équipes ont ensuite adopté cette technologie pour une succession d'améliorations empiriques. En 2016 une équipe propose un réseau à 152 couches et atteint un taux d'erreur de 3%, mieux que les 5% d'un expert humain. Ce concours est depuis lors abandonné au profit de problèmes plus complexes de reconnaissance de scènes associant plusieurs objets ou thèmes.

# La base ImageNet

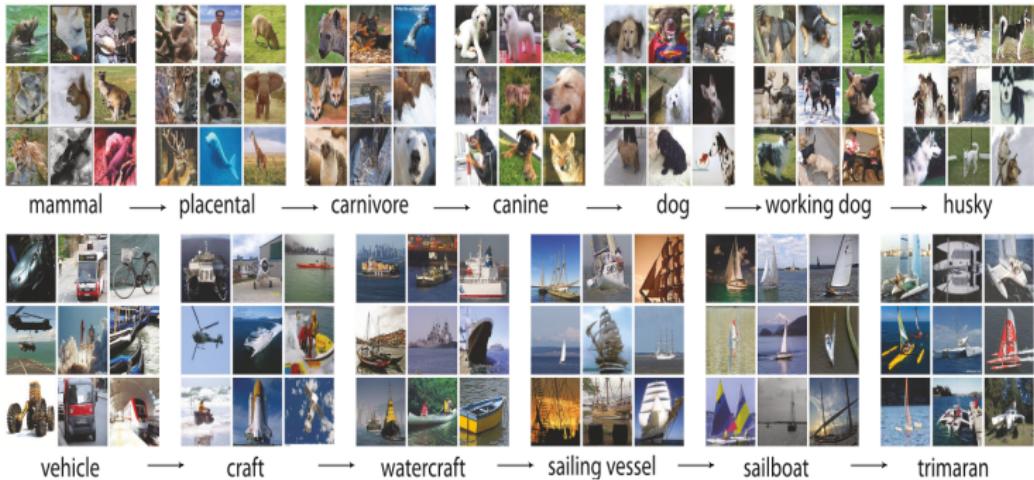


Figure 2: Échantillon de la base ImageNet.

# Concours de reconnaissance d'images

2012 Teams	%error	2013 Teams	%error	2014 Teams	%error
Supervision (Toronto)	15.3	Clarifai (NYU spinoff)	11.7	GoogLeNet	6.6
ISI (Tokyo)	26.1	NUS (singapore)	12.9	VGG (Oxford)	7.3
VGG (Oxford)	26.9	Zeiler-Fergus (NYU)	13.5	MSRA	8.0
XRCE/INRIA	27.0	A. Howard	13.5	A. Howard	8.1
UvA (Amsterdam)	29.6	OverFeat (NYU)	14.1	DeeperVision	9.5
INRIA/LEAR	33.4	UvA (Amsterdam)	14.2	NUS-BST	9.7
		Adobe	15.2	TTIC-ECP	10.2
		VGG (Oxford)	15.2	XYZ	11.2
		VGG (Oxford)	23.0	UvA	12.1

Figure 3: Classements successifs (Le Cun 2016) des équipes participant au concours ImageNet.  
En rouge, celles utilisant des neurones profonds.

## Les couches spécifiques à l'apprentissage profond

Construire un réseau d'apprentissage profond consiste à empiler des couches de neurones aux propriétés spécifiques rapidement résumées ci-dessous. Le choix du type, de l'ordre, de la complexité de chacune de ces couches ainsi que du nombre est complètement empirique et l'aboutissement de très nombreuses expérimentations nécessitant des moyens de calculs et bases de données considérables.

- **fully connected** Couche classique de perceptron et dernière couche d'un réseau profond qui opère la discrimination finale entre par exemple des images à reconnaître. Les couches précédentes construisant, extrayant, des caractéristiques (features) de celles-ci.
- **convolution** opère une convolution sur le signal d'entrée en associant une réduction de dimension.
- **pooling** réduction de dimension en remplaçant un sous-ensemble des entrées (sous-image) par une valeur, généralement le max.
- **normalisation** identique au précédent avec une opération de centrage et/ou de normalisation des valeurs.
- **drop out** les paramètres estimés sont les possibilités de supprimer des neurones d'une couche afin de réduire la dimension.

## Illustration des couches

- Une couche de type convolutif
- Une étape de max-pooling

# **Table of Contents**