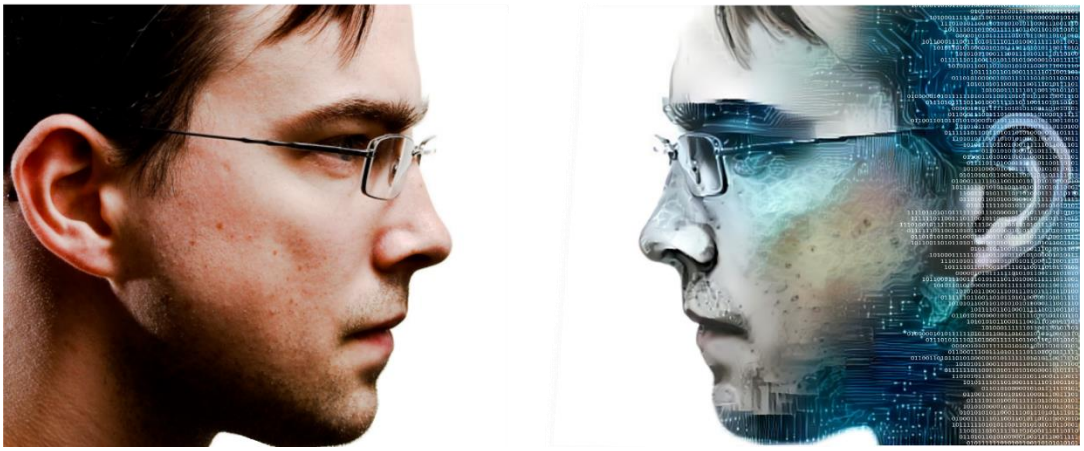

*Figure 1 : Image générée en partie par Deep Dream Generator [1]*


## Résumé :

Aujourd'hui l'apprentissage automatique fait partie de notre quotidien. Elle permet d'identifier vos amis sur les photos, de comprendre vos requêtes vocales faites à votre assistant personnel. Dans cet article, nous allons voir les différentes formes d'apprentissage automatique et nous présenterons un cas concret d'apprentissage de modèle basé sur les réseaux de neurones.


# Apprentissage automatique et réseaux de neurones

**L'apprentissage automatique** est utilisé lorsque le problème à résoudre est trop complexe pour le décrire à une machine en lui donnant une série d'instructions ou des règles simples. Prenons l'exemple du jeu de Go dont l'IA AlphaGo de Deep Mind a battu le champion du monde, Ke Jie, en 2017 [2]. Ce jeu est réputé difficile car le nombre de possibilités rend impossible une recherche exhaustive afin de définir la meilleure stratégie à jouer, même pour les machines les plus puissantes (le nombre d'états possibles du jeu est estimé à $10^{170}$ !). Une solution peut consister à définir un ensemble de règles à la main afin d'orienter la machine dans ses choix, mais la définition manuelle de ces règles est soit trop complexe à mettre en œuvre, soit insuffisante pour vaincre des joueurs confirmés. Cette difficulté se retrouve dans l'analyse d'image, le traitement des langues naturelles ou la reconnaissance de parole. Ces trois domaines associent des espaces de représentation de grande dimension avec des éléments sémantiques contenant une grande variabilité (Figure 2).

*Figure 2: Echantillon d'images représentant des "Joystick" provenant de la base de données ImageNet [3]. Nous pouvons noter la grande diversité de ces images et donc la difficulté de décrire ce qu'est un joystic dans une image.*

Au lieu d'exprimer à la machine sa manière de réagir en fonction de tous les cas possibles, nous lui donnons des exemples concrets sur lesquels elle doit acquérir **une expérience** par apprentissage. De cette expérience, nous espérons qu'elle sera capable de résoudre des cas qu'elle n'a jamais vu, c'est-à-dire de **généraliser**. Par exemple, être capable de jouer au jeu de Go dans une configuration pas encore rencontrée, ou détecter un nouveau visage humain dans une photo.

Nous pouvons dénoter 3 grandes familles d'apprentissage qui dépendent de l'expérience donnée à la machine :

- *L'apprentissage supervisé* : L'expérience consiste en une base de données $B$ contenant un ensemble de couples $\{(x^{(k)}, y^{(k)})|k \in [1, \dots, K]\}$ dont $x^{(k)}$ est un cas d'entrée possible pour la machine (une image, un état du jeu de go, …) et $y^{(k)}$ la réponse attendue (un label, le choix du prochain coup, …). Son but est d'apprendre à associer les entrées $x \in X$ à la bonne sortie $y \in Y$. Plus formellement, elle doit apprendre la fonction $f : X \rightarrow Y$ qui $\forall x, f(x) = y$ en utilisant les exemples fournis dans $B$ (voir la Figure 3).
- *L'apprentissage non-supervisé* : Comme dans l'apprentissage supervisé, la machine a accès à une base de données $B$. Cependant, les exemples $x^{(k)}$ donnés n'ont pas de sortie attendue. Le but de la machine est de comprendre la structure sous-jacente des données $x \in X$. Cela permet par exemple de trouver des groupes de données proches les unes des autres (*clustering*), de détecter des données anormales (détection d'anomalies) ou simplement d'encoder les données dans un espace de représentation plus petit (compression).
- *L'apprentissage par renforcement* : La machine est placée dans un environnement sur lequel elle peut faire des actions. L'apprentissage se fait par des récompenses obtenues en fonction de ses choix. Elle doit donc apprendre quelles séries d'actions entreprendre afin de maximiser ses récompenses aux cours du temps. C'est le type d'apprentissage utilisé par AlphaGo afin d'améliorer ses performances. Dans ce cas, la machine joue contre elle-même de multiple fois et récompense positivement les séries d'actions gagnantes afin d'apprendre une stratégie.

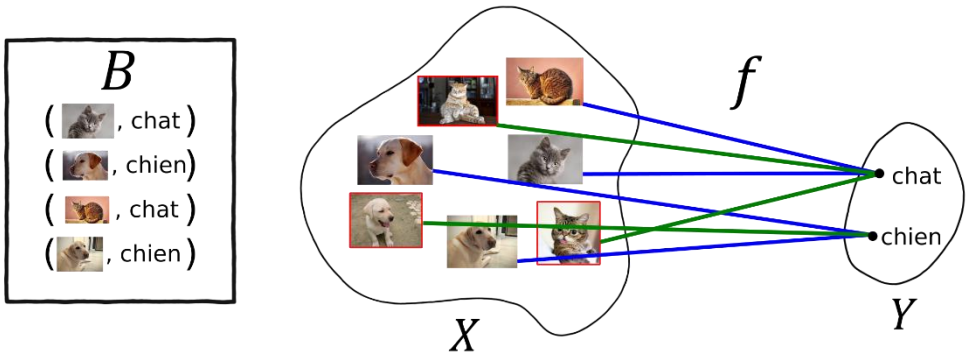


*Figure 3 : Problème de classification d'images d'animaux (ensemble X) dans l'espace des labels (ensemble Y). L'apprentissage consiste à apprendre le mapping correct entre les données dans X et les labels dans Y grâce à la base d'apprentissage B. Les liens en bleu sont ceux données par B tandis que les liens en vert doivent être découvert par la machine (généralisation).*

## Classification de mails avec un réseaux de neurones.

Dans cet article, nous prenons le cas de l'apprentissage supervisé d'un réseau de neurones afin classer automatiquement des mails comme étant important ou non (Figure 4).

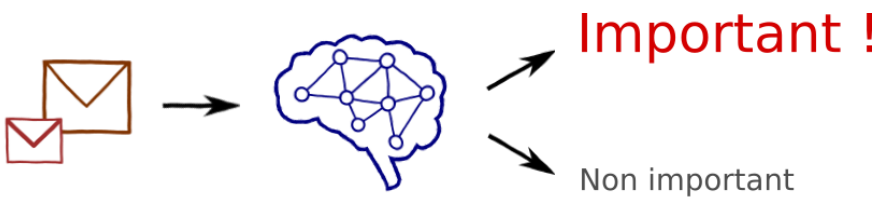


*Figure 4 : Tâche de classification de mails avec un réseau de neurones*

Un mail est caractérisé par le vecteur $x \in \{0,1\}^n$, dans laquelle chaque entrée représente une caractéristique possible, comme « Contient une pièce jointe », et sa valeur passe à 1 si la caractéristique est présente dans le mail (elle reste à 0 sinon). Par exemple, le mail représenté par le vecteur

$$x = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \begin{matrix} \text{"Contient une pièce jointe"} \\ \text{"Envoyé par un contact"} \\ \text{"Contient le mot "Urgent" en objet"} \end{matrix},$$

contient une pièce jointe et le mot «Urgent » en objet mais n'a pas été envoyé par un contact.

Supposons qu'il existe une fonction $f$ capable de classifier les mails comme étant important ou non en fonction d'une entrée $x$. Plus formellement :

$$f(x) = \begin{cases} 0, & \text{si le mail est important,} \\ 1, & \text{sinon.} \end{cases}$$

Pour apprendre cette fonction, nous avons besoin d'un modèle d'apprentissage automatique. Dans notre cas, nous allons prendre le modèle du **perceptron** (Figure 5).

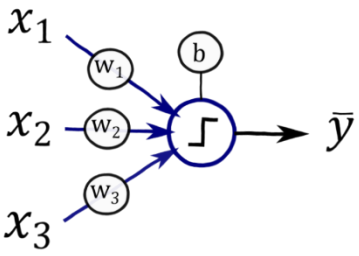


*Figure 5 : Exemple de perceptron avec 3 entrées*

Ce modèle est composé d'un unique neurone artificiel. Celui-ci qui prend en entrée les signaux $x_1, \dots, x_n$ qui sont pondérés par des **poids** $w_1, \dots, w_n$. Plus le poids associé à une entrée est important et plus cette entrée va influer sur la sortie du neurone $\bar{y}$. Le neurone calcule la somme des poids pondérés $\sum_{i=1}^{n} x_i w_i$ : si celle-ci dépasse la valeur du paramètre $b$, appelé **biais**, il renvoi la valeur $\bar{y} =$

1. En pratique, il est préférable d'utiliser une fonction en forme de S (tel que la sigmoïde) plutôt qu'un seuil pour calculer l'activation du neurone (Figure 6).

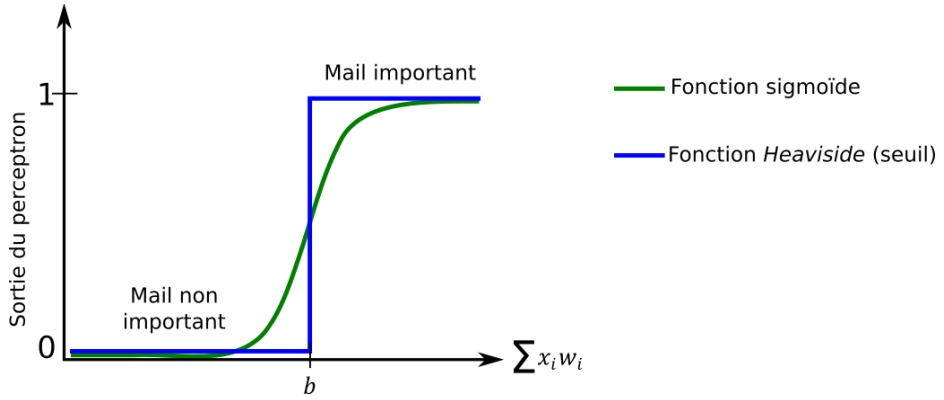


*Figure 6 : Sortie du perceptron en fonction de la somme pondérée des signaux d'entrée*

## Apprentissage du réseau de neurones

Le perceptron est donc capable de représenter notre fonction $f$ à condition que ses **paramètres**, c-à-d les poids $w_1, \dots, w_n$ et le biais $b$ soient bien fixés. Plutôt que les choisir à la main (ce qui est fastidieux si le nombre de paramètres dépasse le million), nous faisons appel à une **méthode d'optimisation** qui se base sur l'expérience apporté au modèle, c-à-d. une base de données d'apprentissage $B$ contenant un ensemble d'exemples concrets $\left(x^{(k)}, y^{(k)}\right)$.

Notons $\theta$ l'ensemble des paramètres du réseau de neurones et $L(\theta)$ l'erreur du réseau de neurones sur la base d'apprentissage $B$ en fonction de $\theta$. Nous pouvons choisir, par exemple, la distance moyenne entre la sortie du perceptron $\bar{y}\left(x^{(k)}\right)$ et la sortie attendue $y^{(k)}$ :

$$L(\theta) = \frac{1}{|B|}\sum_{k=1}^{|B|} ||\bar{y}\left(x^{(k)}\right) - y^{(k)}||.$$

Notre but va être de trouver le minimum de cette fonction $L$ dans l'espace des paramètres $\theta$. Pour ce faire nous utilisons la méthode de la **descente de gradient**. Imaginez que vous soyez perdu en montages à cause d'un brouillard épais et que vous devez rejoindre le village en bas dans la vallée (Figure 7). Vos coordonnées sur la carte représentent votre espace de recherche et l'attitude est la fonction à minimiser (car le village se trouve au point le plus bas des environs). A cause du brouillard, vous n'avez pas de vision d'ensemble des alentours. Cependant, localement vous connaissez la pente du terrain. La meilleure stratégie consiste à vous diriger là où la pente descend pour minimiser l'altitude à laquelle vous vous trouvez. Avec un peu de chance, vous finirez au point le plus bas, c'est-à-dire au village.

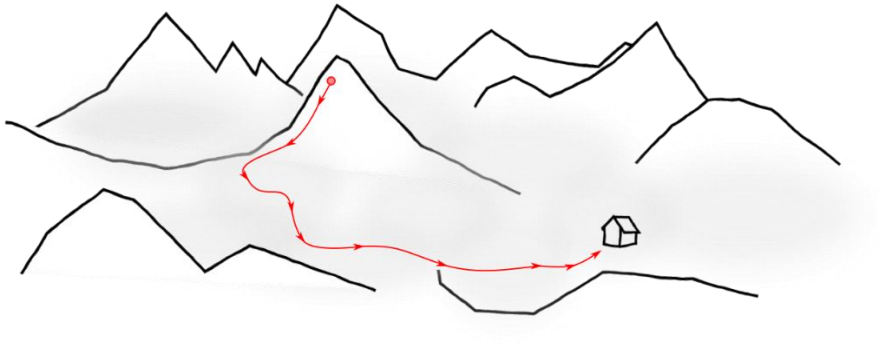

*Figure 7 : Illustration de la descente de gradient*

L'algorithme de descente de gradient fonctionne de la même manière. Nous démarrons à une position $\theta_0$, puis nous calculons la pente sur chacune des dimensions de $\theta$ : c'est-à-dire le gradient :

$$\Delta\theta_0 = \nabla L(\theta_0) = \begin{bmatrix} \frac{\mathrm{d}L(\theta_0)}{\mathrm{d}w_1} \\ \vdots \\ \frac{\mathrm{d}L(\theta_0)}{\mathrm{d}w_n} \end{bmatrix}.$$

Une fois calculé, nous avançons d'un ''pas'' dans la direction de la descente :

$$\theta_1 = \theta_0 - \alpha\Delta\theta_0$$

Le coefficient $\alpha$ permet d'influencer la taille du pas effectué. Une fois à la position $\theta_1$, nous pouvons évaluer de nouveau le gradient $\Delta\theta_1$et faire un pas à la position $\theta_2$, et ainsi de suite jusqu'à atteindre un minimum. Selon la forme de la fonction $L$, rien ne garantit que le minimum ne soit pas qu'un minimum local (comme nous pourrions nous retrouver dans une cuvette alors que nous cherchons le village), cependant la descente de gradient a fait ses preuves en pratique sur l'apprentissage des réseaux de neurones, et de nombreuses variantes existent afin d'accélérer cette optimisation (voir les algorithmes du Momentum, de RMSprop [3], AdaDelta [4], Adam [5], …).

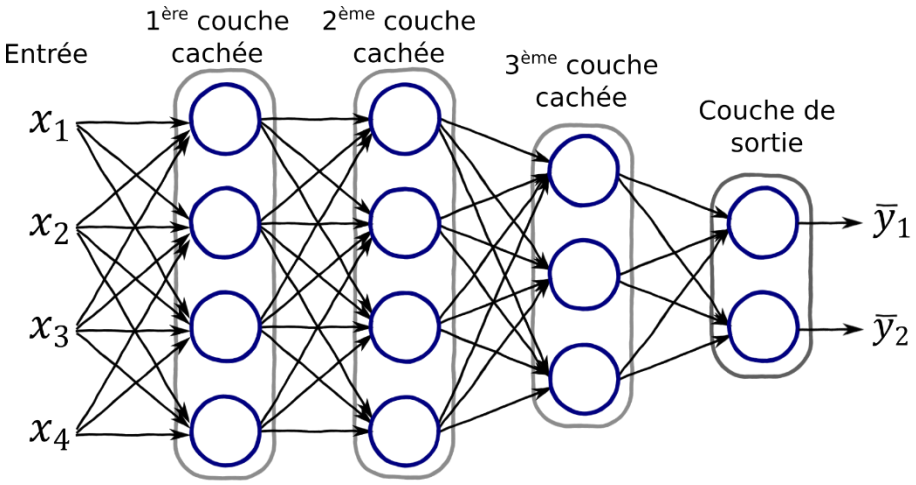


*Figure 8 : Exemple de perceptron multicouche avec 3 couches cachées et une couche de sortie. Les poids des connexions et les biais ne sont pas représentés pour plus de lisibilité.*

## Réseau de neurones multicouches

Dans notre exemple précédent, nous avons vu le modèle du perceptron. Cependant, il reste relativement simple et ne permet pas de répondre à des problèmes plus complexes. Nous allons voir maintenant un type de réseau de neurones plus général appelé **perceptron multicouche**. L'idée consiste à organiser des neurones en plusieurs couches connectées les unes aux autres (Figure 8). Les neurones de la première couche sont connectés avec les entrées du réseau de neurones. De la même manière que le perceptron, l'activation de chaque neurone $i$ de la première couche est calculée telle que :

$$\bar{y}_i^{(1)}(x_1, \dots, x_n) = \sigma\left( b_i^{(1)} + \sum_{j=1}^{n} x_j w_{ij}^{(1)} \right),$$

avec $\sigma$ la fonction d'activation de chaque neurone. Les neurones de la couche suivante peuvent prendre en entrée le vecteur d'activation $[y_1^{(1)} \quad \cdots \quad y_k^{(1)}]^\top$ de la première couche afin de calculer leur activation. Ce processus d'**inférence** continue ainsi jusqu'à la dernière couche, appelée **couche de sortie**. La sortie du perceptron multicouche à $L$ couches est calculée comme une suite de transformation :

$$\bar{y}(x) = \bar{y}^{(L)} \circ \bar{y}^{(L-1)} \circ \cdots \circ \bar{y}^{(1)}(x) = \bar{y}^{(L)}\left(\bar{y}^{(L-1)}\left(\cdots \bar{y}^{(1)}(x) \cdots\right)\right),$$

avec $\bar{y}^{(k)}$ la fonction qui permet de calculer l'activation de tous les neurones de la couche $k$ en fonction de l'activation de la couche $k-1$.

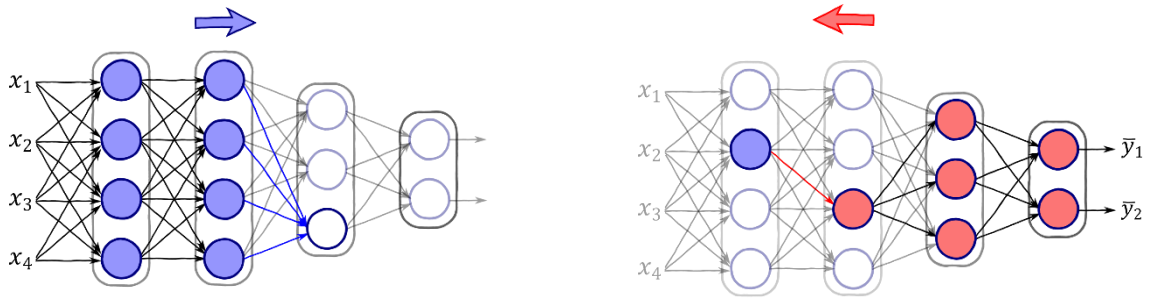


*Figure 9 : Etapes d'Inférence (gauche) et de rétropropagation du gradient (droite). Calculer le gradient sur un poids (flèche rouge) nécessite la valeur d'activation du neurone émetteur (bleu) et l'erreur commise par le neurone récepteur (en rouge). L'erreur d'un neurone dépends de l'erreur des neurones de toutes les couches supérieures.*

Contrairement au perceptron, le calcul du gradient de la fonction d'erreur par rapport à chaque paramètres (poids et biais) n'est plus immédiat. En effet, l'erreur faite par un neurone dans une couche basse dépend de l'erreur de tous les neurones impactés par celui-ci, c'est-à-dire tous les neurones des couches supérieures. Pour calculer le gradient nous utilisons la formule suivante :

$$\frac{\mathrm{d}L(\theta)}{\mathrm{d}w_{ij}} = \delta_j^{(l)} \bar{y}_i^{(l-1)},$$

avec $\delta_j^{(l)}$, l'erreur du neurone $j$ de la couche $l$ calculé tel que :

$$\delta_j^{(l)} = \begin{cases} 2\left(\bar{y}_j^{(l-1)} - y_j\right)\sigma'\left(b_i^{(l)} + \sum_{k=1}^{n} x_k w_{kj}^{(l)}\right) & \text{si le neurone j est dans la dernière couche,} \\ \left(\sum_p \delta_j^{(l+1)} w_{jp}^{(l)}\right)\sigma'\left(b_i^{(l)} + \sum_{k=1}^{n} x_k w_{kj}^{(l)}\right) & \text{si le neurone j est dans une couche cachée.} \end{cases}$$

Cette formule implique de calculer l'erreur de chaque neurone par récurrence en commençant par les neurones de couches supérieures. Cette étape est appelée l'étape de rétropropagation. Une fois effectuée, il est possible de modifier l'ensemble des paramètres du réseau de neurones de la même manière que le perceptron.

Avec l'arrivée du *Deep Learning*, de nombreux travaux de recherches ont été effectués dans ce domaine, apportant des nouveaux types de couches de neurones (tels que les couches convolutives ou les *Long-Short Term Memory* [6]), de nouvelles architectures de réseaux (tels que les GAN [7]) et des améliorations dans les techniques d'apprentissage (ex : apprentissage distribué, apprentissage *privacy-preserving*, …). Les futures articles auront pour but de faire un tour d'horizon de tous ces travaux autour du Deep Learning.

## Références


[1] «Deep Dream Generator,» [En ligne]. Available: https://deepdreamgenerator.com/. [Accès le 16 10 2019].

[2] «DeepMind,» [En ligne]. Available: https://deepmind.com/alphago-china. [Accès le 16 10 2019].

[3] G. Hinton, N. Srivastava et K. Swersky, «Neural networks for machine learning lecture 6a overview of mini-batch gradient descent,» 2012.

[4] D. M. Zeiler, «ADADELTA : An Adaptative Learning Rate Method,» ArXiv, 2012.

[5] D. Kingma et J. Ba, « Adam: A method for stochastic optimization,» ArXiv, 2014.

[6] S. Hochreiter et J. Schmidhuber, «Long short-term memory,» chez *Neural computation*, 1997.

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville et Y. Bengio, «Generative adversarial nets,» chez *Advances in neural information processing systems*, 2014.