Université Nouveaux Horizons

Invariance des transformations géométriques et qualités des objets dans la detection des plaques d'immatriculation

---

Auteur :
TSHELEKA KAJILA Hassan

Directeur :
Prof. MASAKUNA Jordan

Mémoire présenté à la Faculté des Sciences Informatiques en vue de l'obtention du grade de Licencié en informatique.

en

Calcul Scientifique

6 juillet 2022

RÉSUMÉ

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES ACRONYMES

ML Machine Learning

CV Computer Vision

OCR Optical character recognition

ANPR Automatic number-plate recognition

ALPR Automatic license plate recognition

GD Gradient Descent

SGD Stochastic Gradient Descent

ADALINE ADAptative LInear NEuron

CNN Convolutional Neural Network

VGG Visual Geometry Group

ILSVRC ImageNet Large Scale Visual Recognition Challenge

NAG Nesterov Accelerated Gradient

API Application Programming Interface

UML Unified Modeling Language

$\mathbb{N}$	Ensemble des entiers naturels
$\mathbb{R}^n$	Ensemble des réels ou Espace euclidien de dimension n
$\mathbb{B}^n = \{0,1\}^n$	Espace booléen de dimension n
$\mathcal{O}(\cdot)$ ou $\Omega(\cdot)$	L'ordre de grandeur maximal de complexité d'un algorithme
$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$	Un vecteur
$x = (x_1, \vdots, x_n)^T$	Un vecteur
$x^T = (x_1, \vdots, x_n)^T$	Un vecteur transposé
$\langle xy \rangle = x^T y$	Le produit vectoriel
$\| x \|$	La norme du vecteur
M^{-1}	La matrice inverse d'une matrice M
M^T	La matrice transposée
$\frac{\partial}{\partial x} f(x,y)$	La dérivée partielle par rapport à x de al fonction f des deux variable x et y
$\nabla_A J(A,B)$	Le vecteur dérivé par rapport au vecteur A de la fonctionnelle J des deux vecteurs A et B

Les éléments en apprentissage

$\mathcal{S}$	L'échantillon d'apprentissage (un ensemble ou une suite d'exemple)
$\hat{y}$	La valeurs prédite après l'entrainement d'un modèle d'apprentissage automatique
$\hat{y}_i \in \mathcal{Y}$	La prédiction, ou sortie désirée, d'un exemple
$\mathcal{H}$	Espace des hypothèses d'apprentissages
$h \in \mathcal{H}$	Une hypothèses produite par un algorithme d'apprentissage
$y = h(x) \in \mathcal{Y}$	La prédiction faite par l'hypothèse h sur la description s d'un exemple
$\ell(f(x), h(x))$	La fonction perte ou fonction coût entre la fonction cible et une hypothèse sur x d'un exemple
$\mathcal{C}$	L'ensemble des classes
C	Le nombre de classes
$c_i \in \mathcal{C}$	Une sous classe de $\mathcal{C}$

0 INTRODUCTION

0.1 APERÇU GÉNÉRALE

L'intelligence désigne communément le potentiel des capacités mentales et cognitives d'un individu, animal ou humain, lui permettant de résoudre un problème ou de s'adapter à son environnement. L'intelligence nous fait ressentir ce besoin d'apprendre pour arriver à nos fins, extresinquement l'intelligence c'est l'apprentissage. Pour que nous puissions dire qu'une machine est intelligente, premièrement elle doit passer par une phase d'apprentissage. Apprendre à résoudre des problèmes ou à réaliser des tâches par lui-même d'une façon autonome. Dans le IA nous parlons de l'apprentissage automatique (en anglais : machine Learning, ML), nous utilisons plusieurs paradigmes d'apprentissage automatique : apprentissage supervisé, apprentissage non supervisé, apprentissage par renforcement, apprentissage en profondeur.

L'apprentissage supervisé représente une grande partie de l'activité de recherche en apprentissage automatique et de nombreuses techniques de ce paradigme ont trouvé une application dans le traitement de contenu multimédia [11]. La caractéristique qui définit ce type d'apprentissage est la disponibilité de données d'apprentissage annotées.
Les algorithmes d'apprentissage supervisé font l'expérience d'un ensemble de données contenant des caractéristiques, et chaque exemple est également associé à une étiquette ou à une cible [17].

L'application de cette étude dans l'apprentissage supervisé est orientée vers la reconnaissance automatique des plaques d'immatriculation (en anglais : automatic number plate recognition, ANPR) dans les images. Une des applications intéressantes parmi tant d'autres dans l'intelligence artificielle. Nous présentons une étude approfondie sur les algorithmes de minimisation de la fonction coût (en anglais : loss function) d'un modèle d'apprentissage appliqué à l'ANPR.

Lorsque nous voulons faire une application dans le traitement de reconnaissance des formes dans des vidéos, les ensembles de données d'entraînement pour les problèmes de détection d'objets sont généralement très volumineux et les capacités des méthodes d'apprentissage automatique statistique sont limitées par le temps de calcul plutôt que par la taille de l'échantillon [7]. Par exemple, pour entraîner une machine à reconnaître des plaques d'immatriculation de voiture, elle doit recevoir de grandes quantités d'images de plaques d'immatriculation et d'éléments liés aux plaques pour apprendre les différences et reconnaître une plaque, en particulier la voiture qui porte une plaque sans défaut. Plus nous avons des données, plus nous gagnons en précision et plus la complexité en temps augmente.
Des contraintes d'exploitation découlent des observations citées ci-dessus, parmi lesquelles nous citerons celles qui sont liées à la reconnaissance des objets dans les vidéos et images. Par exemple, de nos jours, un très grand nombre de caméras est déployé exclusivement pour la surveillance vidéo [1] . Souvent, le contenu de ces vidéos est interprété par des opérateurs humains qui engendrent des coûts exorbitants pour le suivi

et l'analyse du contenu, sans mentionner les erreurs qui peuvent être induites par la fatigue et l'inattention humaine.

0.2 PROBLÉMATIQUE

La complexité de calcul de l'algorithme d'apprentissage devient le facteur limitant critique lorsque l'on envisage un grand ensemble de données, un ensemble d'image par exemple. C'est à ce point critique qu'entre en jeu cette étude, la minimisation des erreurs sans alourdir la complexité en temps et espace de l'algorithme d'apprentissage. Les ensembles de données d'entraînement pour les problèmes de détection d'objets dans des images sont généralement très volumineux. Minimiser les erreurs dans ces modèles d'apprentissage est une tâche très importante pour renforcer la fiabilité de notre **modèle entraîné** [19].

Les modèles entraînés doivent être invariants sous des transformations géométriques et qualités des objets observés.

- Invariance à la rotation : l'objet dans une image doit être reconnue même après une rotation, sous différent angle.
- Invariance à l'échelle : le même objet dans une image doit être reconnue même sous une échelle différente.

Établir un algorithme d'apprentissage qui s'adapte au mieux à notre modèle, pour en reproduire un modèle entraîné et invariant à la transformation géométrique de l'objet dans l'image. Selon la nature du problème métier traité, il existe différentes approches qui varient selon le type de modèle à entraîné et la quantité des paramètres du modèle en question.

L'un des piliers de l'apprentissage automatique est l'optimisation mathématique [9] qui, dans ce contexte, implique le calcul numérique de minimisation des paramètres d'un système conçu pour prendre des décisions en fonction des données disponibles. Ces paramètres sont choisis pour être optimaux par rapport à notre problème d'apprentissage.

Dans l'ensemble, ce document tente d'apporter des réponses aux questions suivantes.

1. Comment les problèmes de minimisation surviennent-ils dans les applications d'apprentissage automatique ?
2. Quelles ont été les méthodes de minimisation les plus efficaces pour la reconnaissance automatique de plaque d'immatriculation ?
3. Quel sera le comportement du modèle entraîné par rapport à la transformation géométrique des objets ?
4. Comment des algorithmes d'apprentissage supervisé arrivent-t-ils résoudre le problème de l'invariance des transformations géométriques des objets ?

0.3 HYPOTHÈSE

Le cas des problèmes d'apprentissage à grande ou à petite échelle implique la complexité de calcul de l'algorithme d'optimisation sous-jacent de manière non triviale.

En effet, dans ce travail, nous analysons les algorithmes de descente de gradient stochastique parce qu'ils montrent des performances d'optimisation incroyables pour les problèmes à grande échelle. [7].

Le travail de Léon Bottou et al (e. g., dans [7] [31] [8]), présente *la descente de gradient stochastique comme un algorithme d'apprentissage fondamental*.
Une analyse plus précise révèle des compromis qualitativement différents pour le cas des problèmes d'apprentissage à grande échelle [9]. Des algorithmes d'optimisation improbables, tels que la SDG, montrent des performances étonnantes pour les problèmes à grande échelle, lorsque l'ensemble d'apprentissage est volumineux. En particulier, le gradient stochastique du second ordre et le gradient stochastique moyennée sont asymptotiquement efficaces après un seul passage sur l' ensemble d'entraînement [7].

Les optimiseurs axés sur la SGD n'utilisent qu'un seul nouvel échantillon d'apprentissage à chaque itération. De plus, ces optimiseurs sont utilisés pour faire une rétropropagation dans un réseau des neurones pour alléger les poids et biais de celui-ci.

0.4 OBJECTIFS ET DIVISION DU TRAVAIL

Nous nous proposons dans ce mémoire d'aborder sur l'utilisation des algorithmes d'optimisation numérique, précisément de minimisation. Ils seront appliqués à l'apprentissage automatique qui permettra aux ordinateurs et aux systèmes informatiques de dériver des informations significatives à partir d'images numériques, avec un coût plus bas que possible.

En fait, nous faisons la reconnaissance des plaques d'immatriculation des véhicules à l'aide d'un classificateur et optimiseurs de la famille de descente de gradient stochastique implémentés dans un réseau de neurones convolutifs (CNN). On s'en servira pour mesurer l'efficacité des optimiseurs par rapport à notre contrainte d'invariance à la transformations géométriques.
Pour minimiser la fonction de coût du classificateur, les optimiseurs SGD adoptent un modèle d'optimisation convexe [13]. De plus, pour augmenter la vitesse de convergence du classificateur, la descente de gradient stochastique, à chaque étape, tire un échantillon aléatoire de l'ensemble des paramètres de la fonction objectif [? ?].

Pour chaque algorithme, nous examinons l'efficacité de l'invariance à la transformation géométrique et comparons le score (accuracy, loss) pour différents cas.

En dehors de cette introduction, la partie conclusive et l'annexe, ce mémoire est organisé en quatre chapitres comme suit.

> ***Chapitre 1*** est consacré à quelques rappels des matières sur lesquels je me base pour constituer l'ensemble de ce travail. Nous traitons des considérations de méthodes numériques et mathématiques impliquées dans la résolution de problèmes de minimisation des erreurs d'apprentissage. Certaines discussions sur les modèles de régression linéaire convexe et de classification dans d'apprentissage supervisé. Nous discutons également du réseau neuronal convolutif le plus adapté pour analyser l'imagerie visuelle.
>
> ***Chapitre 2*** explore une méthodologie parmi tant d'autres, pour entraîner les modèles d'apprentissage automatique de façon optimale, qui nous permettra par la suite de faire une prédiction d'images pour reconnaissance automatique de plaque

d'immatriculation.
Pour la minimisation de la fonction coût nous utilisons des algorithmes comme ASGD, ADAM, ADADELTA, NAG. Puis faire une étude comparative de leurs performances.

Chapitre 3, Ici nous construirons des modèles à partir d'une base de données annotée pour l'apprentissage et pour les tests de reconnaissance de plaque d'immatriculation sur l'image. Les résultats concluants de cette étude pourront conduire à un déploiement de notre système dans les domaines comme celui de la surveillance vidéo de voitures dans une entrée de parking. Des métriques connues pour mesurer les erreurs et en déduire le score du classificateur seront utilisées pour évaluer la qualité de la reconnaissance automatique des plaques d'immatriculation (ALPR) par notre approche.

1

CONCEPTS ET ÉLÉMENTS MATHÉMATIQUES DE L'APPRENTISSAGE PROFOND

1.1 LES BASES D'OPTIMISATION NUMÉRIQUE ET STATISTIQUE

1.1.1 *Éléments de calcul différentiel*

A *Convexité*

DÉFINITION : (ENSEMBLE CONVEXE) Une partie $\mathcal{C} \subset \mathbb{R}^n$ est dite convexe si et seulement si pour tout $(x, y) \in \mathcal{C}^2$, et pour tout $\alpha \in [0, 1]$, $\alpha x + (1 - \alpha)y \in \mathcal{C}$ combinaison convexe [23].

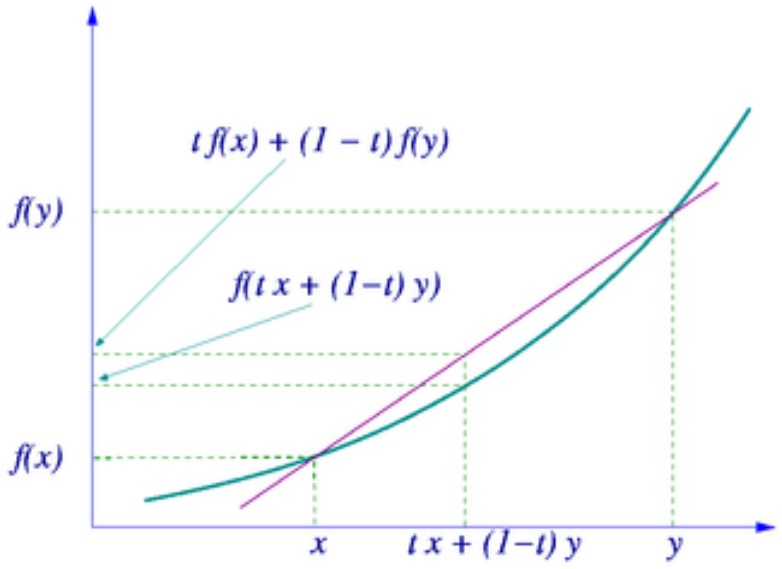


FIGURE 1 : Illustration d'une fonction convexe, avec $\alpha = t$.

DÉFINITION : (FONCTION CONVEXE) Une fonction f d'un intervalle réel $I \in \mathcal{C}$ est dite fonction convexe lorsque, $\forall (x, y)$ de I tel que $(x, y) \in \mathcal{C}^2$ et tout $\alpha \in [0, 1]$ on a :

$$f(\alpha x + (1 - \alpha)y) \leqslant \alpha f(x) + (1 - \alpha)f(y) \tag{1}$$

et si

$$f(\alpha x + (1 - \alpha)y) < \alpha f(x) + (1 - \alpha)f(y) \tag{2}$$

on dit que la fonction est strictement convexe dans $\mathcal{C}$, [23].

EXTREMUM D'UNE FONCTION [10] : Soit $I \to \mathbb{R}$ une fonction et a un point de I ($a \in I$).

+ On dit que m est un **minimum local** de f, si pour tout $x \in I$, $f(x) \leqslant f(a)$ ou s'il existe $\alpha > 0$ tel que m soit le minimum de f restreinte à $I \cap]a-\alpha, a+\alpha[$.
+ On dit que M est un **maximum local** de f, si pour tout $x \in I$, $f(x) \geqslant f(a)$ ou s'il existe $\alpha > 0$ tel que M soit le maximum de f restreinte à $I \cap]a-\alpha, a+\alpha[$.

La recherche des extrema est liée au calcul différentiel, grâce notamment au théorème suivant.

Théorème 1.1.1 *Soit* I *un intervalle ouvert et* $f : I \to \mathbb{R}$ *dérivable. Si* f *admet un extremum local en* a *alors* $f'(a) = 0$.

La réciproque de ce théorème est fausse comme le montre l'exemple de la fonction cube, dont la dérivée s'annule en 0, mais qui ne possède pas d'extremum en ce point. En général, on étudie la fonction, et notamment le signe de f' pour déterminer si a est effectivement un extremum, et si c'est un maximum ou un minimum.

Pour les fonctions de plusieurs variables, on remplace la dérivée par la différentielle et on affine l'étude avec les dérivées partielles secondes. On a ainsi le résultat suivant.

Théorème 1.1.2 *Soit* $f : \mathbb{R}^2 \to \mathbb{R}$ *et* $a \in \mathbb{R}^2$.

- *Si* f *est différentiable, et si* f *admet extremum local en* a, *alors*

$$\frac{\partial f}{\partial x}(a) = 0 \quad \text{et} \quad \frac{\partial f}{\partial y}(a) = 0.$$

  On dit alors que a *est un* ***point stationnaire*** *ou* ***point critique*** *de* a.

- *Si* $f \in I^2$ *et, si* a *est point stationnaire de* f, *on pose*

$$r = \frac{\partial^2 f}{\partial x^2}(a), \quad s = \frac{\partial^2 f}{\partial x \partial y}(a), \quad t = \frac{\partial^2 f}{\partial y^2}(a).$$

  On distingue les cas suivants

  1. *Si* $rt - s^2 > 0$ *et* $r > 0$, f *admet un minimum relatif en* a.
  2. *Si* $rt - s^2 > 0$ *et* $r < 0$, f *admet un maximum relatif en* a.
  3. *Si* $rt - s^2 < 0$, f *n'admet pas maximum en* a *on parle de* ***point col***, *ou de* ***point selle***.
  4. *Si* $rt - s^2 = 0$, *on ne peut pas conclure.*

Donc nous pouvons dire qu'une fonction convexe à un unique point minimum.

B *Développement limité*

En physique et en mathématiques, un développement limité (noté DL) d'une fonction en un point est une *approximation polynomiale* de cette fonction au voisinage de ce point, c'est-à-dire l'écriture de cette fonction sous la forme de la somme d'une fonction polynomiale et d'un reste négligeable au voisinage du point considéré [4, 10].

Soit f une fonction à valeurs réelles définie sur un intervalle I, et $x_0 \in I$. On dit que f admet un développement limité d'ordre n^2 (abrégé par DL_n) en x_0, s'il existe $n+1$ réels $a_0, a_1, \ldots, a_n$ tels que la fonction $R : I \to \mathbb{R}$ définie par :

$$f(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)^2 + \cdots + a_n(x-x_0)^n + R(x) = \sum_{i=0}^{n} a_i(x-x_0)^i + R(x)$$

vérifie : $R(x)$ tend vers 0 lorsque x tend vers x_0, et ce plus rapidement que le dernier terme de la somme, c'est-à-dire que :

$$\lim_{x \to x_0} \frac{R(x)}{(x-x_0)^n} = 0.$$

La fonction reste $R(x)$ vérifiant ceci est notée $o((x–x0)^n)$ (selon la notation de Landau). On écrit donc :

$$f(x) = \sum_{i=0}^{n} a_i(x-x_0)^i + R(x) = \sum_{i=0}^{n} a_i(x-x_0)^i + o((x-x_0)^n)$$

Il est fréquent d'écrire un développement limité en posant $x = x_0 + h$ on aura :

$$f(x_0+h) = \sum_{i=0}^{n} a_i h^i + o(h^n) \tag{3}$$

CONSÉQUENCES IMMÉDIATES

- Si f admet un DL_0 en x_0, alors $a_0 = f(x_0)$. [10]
- Si f admet un DL_n en x_0, alors elle admet un DL_k en x_0 pour tout entier $k < n$ [10].
- Une condition nécessaire et suffisante pour que f admette un DL_n en x_0 est l'existence d'un polynôme P tel que $f(x) = P(x) + o((x–x_0)^n)$ [10]. S'il existe un tel polynôme P, alors il en existe une infinité d'autres, mais un seul d'entre eux est de degré inférieur ou égal à n : le reste de la division euclidienne de $P(X)$ par $(X–x_0)^{n+1}$. On l'appelle la partie régulière, ou partie principale, du DL_n de f en x_0.

Le théorème de Taylor-Young assure [10] qu'une fonction f dérivable n fois au point x_0 (avec $n \geqslant 1$ admet un DL_n en ce point :

$$f(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{f''(x_0)}{2!}(x-x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n + o((x-x_0)^n)$$

soit en écriture abrégée

$$f(x) = \sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!}(x-x_0)^i + o((x-x_0)^n)$$

Le développement d'ordre 0 en x_0 revient à écrire que f est continue en x_0 :

$$f(x) = f(x_0) + o((x - x_0)^0) = f(x_0) + o(1)$$

Le développement limité d'ordre 1 en x_0 revient à approcher une courbe par sa tangente en x_0 on parle aussi d'approximation affine :

$$f(x) = f(x_0) + f'(x_0) \cdot (x - x_0) + o(x - x_0) \tag{4}$$

c *Gradient*

DÉFINITION : Le gradient d'une fonction de plusieurs variables en un certain point est un vecteur qui caractérise la variabilité de cette fonction au voisinage de ce point. Défini en tout point où la fonction est différentiable, il définit un champ de vecteurs, également dénommé gradient. Le gradient est la généralisation à plusieurs variables de la dérivée d'une fonction d'une seule variable [4, 5].

DÉFINITION MATHÉMATIQUE : Dans un système de coordonnées cartésiennes, le gradient d'une fonction $f(x_1, x_2, \ldots, x_n)$ est le vecteur de composantes $\partial f/\partial x_i$ $(i = 1, 2, \ldots, n)$, c'est-à-dire les dérivées partielles de f par rapport aux coordonnées [23].

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix} \in \mathbb{R}^n \tag{5}$$

GRADIENT SOUS FORME DE DÉVELOPPEMENT LIMITÉ : *Si une application admet un gradient en un point, alors on peut écrire ce développement limité du premier ordre (voir le point b, équation 3 et 4).*

$$f(x + h) = f(x) + \langle \nabla f(x) \mid h \rangle + o(h)$$

ou

$$f(x - h) = f(x) - \langle \nabla f(x) \mid h \rangle + o(h)$$

Numériquement, il est très intéressant de faire ensuite la demi-différence des deux développements pour obtenir la valeur du gradient et on note que celui-ci ne dépend pas en fait de la valeur de la fonction au point x : $f(x)$. Cette formule a l'avantage de tenir compte des gradients du 2e ordre et est donc beaucoup plus précise et numériquement robuste [23]. L'hypothèse est, en pratique, de connaitre les valeurs "passé" et "futur" de la fonction autour d'un petit voisinage du point x [5].

DÉFINITION NUMÉRIQUE : Une fonction multivariée (à variable vectorielle) $f(x) : \mathbb{R}^n \to \mathbb{R} : x \to f(x)$ définie sur un ouvert $O \in \mathbb{R}^n$ est dite dérivable (au sens de Fréchet [23]) en x ssi il existe un vecteur noté $\nabla f(x) \in \mathbb{R}^n$ tel que

$$f(x + h) = f(x) + \nabla f(x)^T h + o(||h||) \tag{6}$$

$\nabla f(x) \in \mathbb{R}^n$ et où l'on a posé que le reste $o(||h||) = ||h||\epsilon(h) \in \mathbb{R}^n$, avec $h \in \mathbb{R}^n$

$$\epsilon(h) : \mathbb{R}^n \to \mathbb{R}, \qquad \lim_{||h||\to 0} \epsilon(h) = 0.$$

Le vecteur $\nabla f(x)$ est unique et nommé **gradient** de $f(x)$ en x. Le gradient s'adresse aux fonctions scalaires à variables vectorielles.

A PROPOS DE LA NOTATION $o(||h||)$: La notation de Bachmann-Landau $o(||h||)$ traduit le comportement d'une fonction de h qui tend vers 0 d'un ordre de grandeur plus vite que $||h||$ [10]. Elle est infiniment plus petit que h dans le voisinage de 0.

D *Hessienne*

DÉFINITION MATHÉMATIQUE : Étant donnée une fonction f à valeurs réelles

$$f : \mathbb{R}^n \to \mathbb{R}; (x_1, ..., x_n) \mapsto f(x_1, ..., x_n)$$

dont toutes les dérivées partielles secondes existent, le coefficient d'indice i, j de la **matrice hessienne**[1] $H(f)$ vaut $H_{ij}(f) = \frac{\partial^2 f}{\partial x_i \partial x_j}$ [23].
Autrement dit,

$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial {x_1}^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial {x_2}^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial {x_n}^2} \end{bmatrix}.$$

DÉFINITION NUMÉRIQUE : Supposons que $f : \mathbb{R}^n \to \mathbb{R}$ définie sur un ouvert $\mathcal{O} \in \mathbb{R}^n$. La fonction $f(x)$ est dite 2 fois continûment dérivable (au sens de Fréchet) si en tout $x \in \mathcal{O}$ on a

$$f(x + h) = f(x) + \nabla f(x)^T h + \frac{1}{2} h^T \nabla^2 f(x) h + o(||h||^2) \tag{7}$$

avec$\nabla f(x) \in \mathbb{R}^{n\times n}$ et où on a posé que le reste $o(||h||^2) = ||h||\epsilon(h) \in \mathbb{R}$ avec $\lim_{||h||\to 0} \epsilon(h) = 0$ La matrice carrée symétrique $\nabla^2 f(x)$ appelée **Hessien** de $f(x)$ en x [5]. Remarque :

$$\lim_{||h||\to h} \frac{o(||h||^2)}{||h||} = 0 \in \mathbb{R}$$

La Hessienne s'adresse aux fonctions scalaires à variables vectorielles.

E *Jacobienne*

DÉFINITION MATHÉMATIQUE : Soit F une fonction d'un ouvert de $\mathbb{R}^n$ à valeurs dans $\mathbb{R}^m$ ($F : \mathbb{R}^n \to \mathbb{R}^m$). Une telle fonction est définie par ses m fonctions composantes à valeurs réelles :

1 En mathématiques, la matrice hessienne (ou simplement la hessienne) d'une fonction numérique f est la matrice carrée, notée $H(f)$, de ses dérivées partielles secondes.

$$F : \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \longmapsto \begin{pmatrix} f_1(x_1, \ldots, x_n) \\ \vdots \\ f_m(x_1, \ldots, x_n) \end{pmatrix}.$$

Les dérivées partielles de ces fonctions en un point M, si elles existent, peuvent être rangées dans une matrice à m lignes et n colonnes [23], appelée **matrice jacobienne**[2] de F :

$$J_F(M) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}.$$

La case sur la ligne i et la colonne j contient $\frac{\partial f_i}{\partial x_j}$ qui est la dérivée partielle de f_i selon la variable x_j. Cette matrice est notée :

$$J_F(M), \qquad \frac{\partial(f_1, \ldots, f_m)}{\partial(x_1, \ldots, x_n)} \qquad \text{ou} \qquad \frac{D(f_1, \ldots, f_m)}{D(x_1, \ldots, x_n)}$$

Pour $i = 1, \ldots, m$, la i-ème ligne de cette matrice est la transposée du vecteur **gradient** (voir le point c) au point M de la fonction f_i, lorsque celui-ci existe. La matrice jacobienne est également la matrice de la différentielle de la fonction, lorsque celle-ci existe [23].

DÉFINITION NUMÉRIQUE : Soit $f(x) : \mathbb{R}^n \to \mathbb{R}^m$ définie sur un ouvert $\mathcal{O} \subset \mathbb{R}$. On dit que $f(x)$ est dérivable (au sens de Fréchet) en x, si chacune des composantes $f_i(x)$ est dérivable en x [5]. On a alors

$$f(x + h) = f(x) + D_f(x)h + o(||h||) \tag{8}$$

avec $D_f(x) \in \mathbb{R}^{n \times m}$ et/où $o(||h||) = ||h||\epsilon(h) \in \mathbb{R}^m$ avec $\lim_{||h|| \to 0} \epsilon(h) = 0$. Remarque :

$$\lim_{||h|| \to h} \frac{o(||h||^2)}{||h||} = 0 \in \mathbb{R}$$

Soient $x = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T \in \mathbb{R}^n$ et $f(x) = \begin{bmatrix} f_1(x) & f_2(x) & \cdots & f_n(x) \end{bmatrix}^T \in \mathbb{R}^m$

$$D_f(x) = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m(x)}{\partial x_1} & \cdots & \frac{\partial f_m(x)}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla f_1(x)^T \\ \nabla f_2(x)^T \\ \vdots \\ \nabla f_m(x)^T \end{bmatrix} \in \mathbb{R}^{n \times m},$$

La matrice $D_f(x) \in \mathbb{R}^{n \times m}$ est appelée **Jacobienne** de f(x) en x. La Jacobienne s'adresse aux fonctions vectorielles à variables vectorielles [23].

2 En analyse vectorielle, la matrice jacobienne est la matrice des dérivées partielles du premier ordre d'une fonction vectorielle en un point donné.

NOTE : Lorsque $m = 1$ (m : nombre des lignes), la Jacobienne est la même que le gradient car il s'agit d'une généralisation du gradient.

1.1.2 *Échantillonnage (statistique)*

En statistiques, l'échantillonnage est la sélection d'un sous-ensemble (un échantillon statistique) d'individus au sein d'une population statistique pour estimer les caractéristiques de l'ensemble de la population.
Sur un échantillon, on peut calculer différents paramètres statistiques de position ou de dispersion issus de la statistique descriptive, de la même manière que l'on peut déterminer des paramètres statistiques d'une population par son recensement exhaustif [26].

On peut également déduire des propriétés de la population à partir de celles de l'échantillon par inférence statistique. D'après la loi des grands nombres, plus la taille de l'échantillon augmente, plus ses propriétés seront proches de celles de la population. En particulier, on peut estimer une probabilité sur les individus d'une population par la fréquence observée sur un échantillon si sa taille est suffisamment grande [18, 26].

Cette méthode présente plusieurs avantages : une étude restreinte sur une partie de la population, un moindre coût, une collecte des données plus rapide que si l'étude avait été réalisée sur l'ensemble de la population, la réalisation de contrôles destructifs, etc. Un dataset assez large permettra ainsi à une machine d'effectuer un bon apprentissage automatique sur cet ensemble de données constituant l'échantillon de la population a apprendre.

On peut procéder de différentes manières pour collecter les données de l'échantillon, il existe en effet plusieurs méthodes d'échantillonnage [26] :

- **Échantillonnage aléatoire et simple** : le tirage des individus de l'échantillon est aléatoire, c'est-à-dire que chaque individu a la même probabilité d'être choisi, et simple, c'est-à-dire que les choix des différents individus sont réalisés indépendamment les uns des autres.
- **Échantillonnage systématique** : le premier individu est choisi de manière aléatoire, puis les suivants sont déterminés à intervalle régulier. Par exemple, dans un verger, on choisit au hasard le 7e pommier, puis les 27e, 47e, 67e, etc.
- **Échantillonnage stratifié** : on subdivise la population en plusieurs parties avant de prendre l'échantillon1.
- **Échantillonnage par quotas** : la composition de l'échantillon doit être représentative de celle de la population selon certains critères jugés particulièrement importants. On utilise cette méthode pour réaliser les sondages d'opinions.

A *La collecte de données*

La collecte de données est le processus de collecte et de mesure des informations sur des variables ciblées dans un système établi, qui permet ensuite de répondre aux questions pertinentes et d'évaluer les résultats.

Une bonne collecte de données implique :

- Suivre le processus d'échantillonnage défini
- Garder les données dans l'ordre du temps
- Noter les commentaires et autres événements contextuels
- Enregistrement des non-réponses

ERREUR D'ÉCHANTILLONNAGE : Dans les statistiques, les erreurs d'échantillonnage se produisent lorsque les caractéristiques statistiques d'une population sont estimées à partir d'un sous-ensemble, ou échantillon, de cette population. Étant donné que l'échantillon n'inclut pas tous les membres de la population, les statistiques de l'échantillon (souvent appelées estimateurs), telles que les moyennes et les quartiles, diffèrent généralement des statistiques de l'ensemble de la population (appelées paramètres). La différence entre la statistique d'échantillon et le paramètre de population est considérée comme l'erreur d'échantillonnage [26].

1.2 CONCEPTS DE LA MODÉLISATION ET CLASSIFICATION DES DONNÉES

1.2.1 *Introduction*

A *Les ingrédients d'apprentissage*

Résoudre un problème d'apprentissage, c'est d'abord le comprendre, c'est-à-dire discuter longuement avec les experts, ou consulter les ouvrages, du domaine concerné pour identifier quelles sont les "entrées", les "sorties" ou résultats désirés, les connaissances disponibles, les particularités des données, par exemple : valeurs manquantes, taux de bruit dans les mesures des attributs de description, proportions des classes, stationnarité ou pas de l'environnement.
C'est aussi réaliser un gros travail de *préparation des données* : nettoyage, ré-organisation, enrichissement, intégration avec d'autres sources de données, etc. Ces étapes de compréhension du problème, de préparation des données, de mise au point du protocole d'apprentissage et des mesures d'évaluation des résultats, prennent, et de loin, la plus grande partie du temps pour (tenter de) résoudre un problème d'apprentissage [3]. Nous avons toujours tendance à largement sous-estimer ces étapes et à vouloir se concentrer uniquement sur la phase excitante de l'essai de méthodes d'apprentissage sur des données supposées bonnes à la consommation [12].

B *Concepts de la modélisation*

La modélisation est la conception et l'utilisation d'un *modèle*. Selon son objectif et les moyens utilisés, la modélisation est dite mathématique, géométrique, 3D, etc.
En informatique, la modélisation permet de concevoir l'architecture globale d'un système d'information, ainsi que l'organisation des informations à l'aide de la modélisation des données [21].

MODÈLE (INFORMATIQUE) : En informatique, un modèle a pour objectif de structurer les informations et activités d'une organisation : données, traitements, et flux d'informations entre entités [12, 21].

MODÈLE (MATHÉMATIQUE) : Un modèle mathématique est une description d'un système utilisant des concepts et un langage mathématiques [12].

Un modèle peut aider à expliquer un système et à étudier les effets de différents composants, et à faire des prédictions sur le comportement [18].

- *Modèles non paramétriques*

E. g. : Prenons l'exemple de données décrites dans l'espace d'entrée $\mathcal{X} = \mathbb{R}^n$ avec n variables réelles et supposons-les étiquetées par $\times$ ou par $*$. On cherche donc une fonction de décision h, appelée hypothèse ou modèle, telle qu'elle soit capable d'étiqueter toute entrée $x \in \mathcal{X}$, $h : x \to \{\times, *\}$. Reste à définir l'espace des hypothèses ou modèles $\mathcal{H}$ que l'on est prêt à considérer.

Toujours en considérant le problème de prédiction basique (présenté ci-dessus), on pourrait définir une hypothèse par une procédure qui examine les trois plus proches voisins du point à étiqueter x et qui choisit l'étiquette majoritaire parmi ces trois points pour étiqueter x. Il n'y a évidemment plus de paramètres pour définir les modèles possibles [3].

Un **modèle non paramétrique** est construit selon les informations provenant des données. Dans [3, 6] il est expliqué que : La régression non paramétrique exige des tailles d'échantillons plus importantes que celles de la régression basée sur des modèles paramétriques parce que les données doivent fournir la structure du modèle ainsi que les estimations du modèle.

Un **modèle paramétrique** est, s'il est approximativement valide, plus puissant qu'un modèle non paramétrique, produisant des estimations d'une fonction de régression qui ont tendance à être plus précises que ce que nous donne l'approche non paramétrique [21]. Cela devrait également se traduire par une prédiction plus précise.

Dans [3], il est expliqué que nous pouvons construire un modèle d'apprentissage, ou l'espaces des hypothèses d'apprentissage, par :

- La classification
- La régression
- Les distributions de probabilités
- Les arbres de décisions
- Les réseaux bayésiens
- Etc.

La table suivante présente d'abord les qualités des différentes représentations des hypothèses en fonction des critères cités ci-dessus.

	Fonctions séparatrices	Distributions de probabilités	Arbres de décision	Hiérarchies de concepts	Réseaux bayésiens	Chaînes de Markov
Concept	√	√	√	√	–	–
Classes multiples	√	√	√	√	–	–
Ontologies	–	–	√	√	–	–
Régression	–	√	√	–	–	–
Évolutions temporelles	–	√	–	–	–	√
Apprentissage non supervisé	√	√	√	√	–	–
Données continues	√	√	√	–	–	√
Connaissances relationnelles				√	√	–
Degré de certitude	–	√	–	–	√	√
Degré d'imprécision	–	√	–	–	√	–
Transparence, intelligibilité	–	–	–	√	√	√

- *Entraînement du modèle*

Tout modèle, où toutes les informations nécessaires ne sont pas disponibles, contient certains paramètres qui peuvent être utilisés pour adapter le modèle au système qu'il est censé décrire. Si la modélisation est effectuée par un réseau de neurones artificiels ou un autre apprentissage automatique, l'optimisation des paramètres est appelée **entraînement** (en anglais : **training**), tandis que l'optimisation des hyperparamètres du modèle est appelée **réglage** (en anglais : **tuning**) et utilise souvent la validation croisée [17]. Dans une modélisation plus conventionnelle à travers des fonctions mathématiques explicitement données, les paramètres sont souvent déterminés par ajustement de courbe (voir le chapitre **??**, section **??**).

Une partie cruciale du processus de modélisation consiste à évaluer si oui ou non un modèle mathématique donné décrit un système avec précision. Il peut être difficile de répondre à cette question car elle implique plusieurs types d'évaluation différents [17, 21].

1.2.2 *Les problèmes de régressions*

L'algorithme d'apprentissage automatique est défini comme un algorithme capable d'améliorer les performances d'un programme informatique à certaines tâches via l'expérience est quelque peu abstraite. Pour rendre cela plus concret, Une des méthode d'apprentissage automatique basique est *la régression linéaire* [17].

Dans la modélisation statistique, l'analyse de régression est un ensemble de processus statistiques permettant d'estimer les relations entre une variable dépendante et une ou plusieurs variables indépendantes [21].
En statistique, la régression linéaire est une approche linéaire pour modéliser (voir la section 1.2.1, point b) la relation entre une réponse scalaire et une ou plusieurs variables explicatives (également appelées variables dépendantes et indépendantes). Le cas d'une variable explicative est appelé régression linéaire simple ; pour plus d'un, le processus est appelé régression linéaire multiple [12].

Dans la régression linéaire, les relations sont modélisées à l'aide de *fonctions prédictives*[3] linéaires dont les paramètres de modèle inconnus sont estimés à partir des données [21]. De tels modèles sont appelés modèles linéaires.

La régression linéaire a de nombreuses utilisations pratiques. Si l'objectif est la prédiction, la prévision ou la réduction des erreurs, la régression linéaire peut être utilisée pour ajuster un modèle prédictif à un ensemble de données observées de valeurs de la réponse et de variables explicatives [12]. Après avoir développé un tel modèle, si des valeurs supplémentaires des variables explicatives sont collectées sans valeur de réponse d'accompagnement, le modèle ajusté peut être utilisé pour faire une prédiction de la réponse [18].

Dans ce type de tâche, le programme informatique est invité à prédire une valeur numérique à partir d'une entrée donnée. Pour résoudre cette tâche, l'algorithme d'apprentissage est invité à sortir une fonction $f : \mathbb{R}^n \to \mathbb{R}$. Ce type de tâche est similaire à la **classification**, sauf que le format de sortie est différent [17].

A *Le cas de la régression linéaire*

On appelle problèmes de régression de tels problèmes, dans lesquels la sortie est numérique, généralement un vecteur de réels, supposé dépendre de la valeur d'un certain nombre de facteurs en entrée [12, 21].

Le vecteur d'entrée $x = (x_1, x_2, ..., x_n)^T$ est souvent appelé variable indépendante, tandis que le vecteur de sortie y est appelé variable dépendante. On formalise le problème en supposant que la sortie résulte de la somme d'une fonction déterministe f de l'entrée et d'un bruit aléatoire :

$$y = f(x) + \epsilon \tag{9}$$

où $f(x)$ est la fonction inconnue que nous souhaitons approcher par un estimateur $h(x|w)$, où h est défini à l'aide d'un vecteur w de paramètres [2]. Si l'on suppose que le bruit ϵ est nulle et de variance constante σ^2, c'est-à-dire $\epsilon = \mathcal{N}(0, \sigma^2)$, alors, en plaçant notre estimateur $h(\cdot)$ à la place de la fonction inconnue, on devrait avoir la densité conditionnelle réelle $p(y|x)$ vérifiant :

$$p(y|x) = \mathcal{N}(h(x|w), \sigma^2) \tag{10}$$

On peut estimer le vecteur de paramètres w grâce au principe de maximisation de la vraisemblance. On suppose que les couples (x_t, y_t) de l'échantillon d'apprentissage sont

3 En statistique et en apprentissage automatique, une fonction de prédicteur linéaire est une fonction linéaire d'un ensemble de coefficients et de variables explicatives, dont la valeur est utilisée pour prédire le résultat d'une variable dépendante.

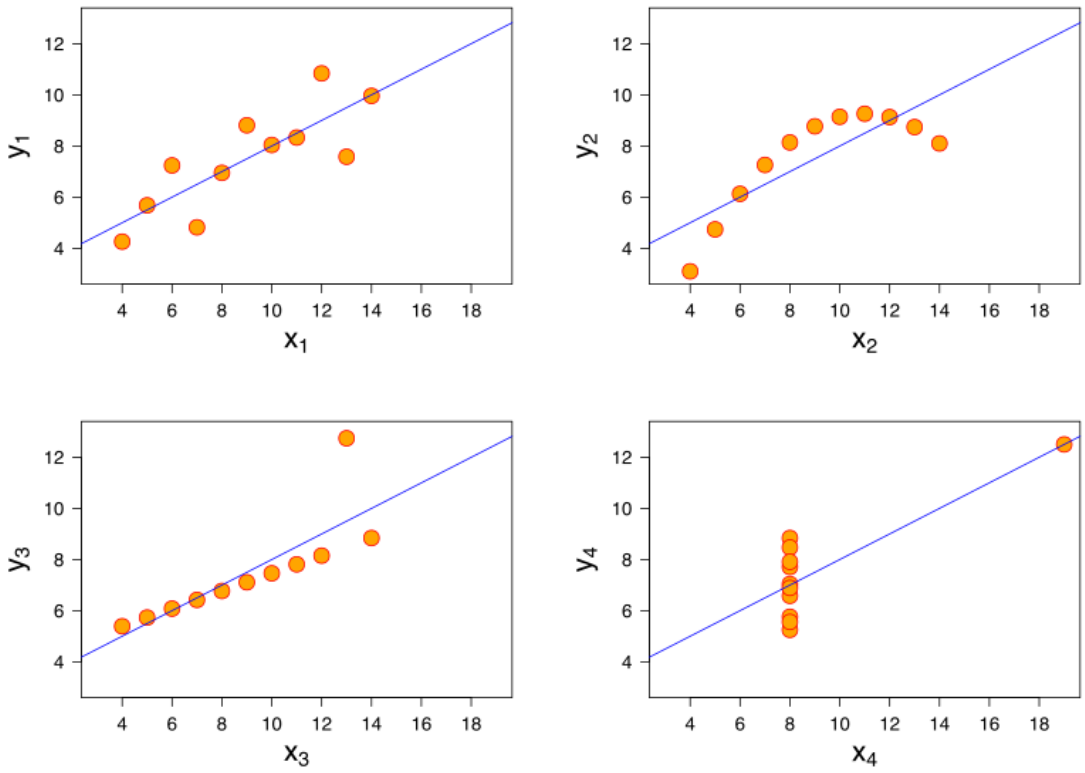


FIGURE 2 : Images illustrant l'efficacité de la régression linéaire sur plusieurs type de modèle.

tirés par tirages indépendants d'une distribution de probabilités jointes inconnue $p(x, y)$, qui peut s'écrire :

$$p(y|x) = p(y|x)p(x)$$

où $p(y|x)$ est la probabilité de la sortie étant donnée l'entrée et $p(x)$ est la densité de probabilité sur les entrées [21].

Étant donné un échantillon d'apprentissage $S = \langle (x_t, y_t) \rangle_{1 \leqslant t \leqslant m}$ supposé tiré de manière indépendante et identiquement distribuée. Maximiser l'expression résultante revient alors à minimiser la somme de carrés des erreurs (SCE) [3] :

$$\mathrm{SCE}(w|S) = \frac{1}{2} \sum_{t=1}^{m} [y_t - h(x_t|w)]^2 \tag{11}$$

B *Le cas de la régression générale*

La plupart des modèles de régression proposent que Y_i est une fonction de X_i et w, avec ϵ_i représentant un terme d'erreur additif ou bruit statistique aléatoire qui peut remplacer des déterminants non modélisés de Y_i :

$$Y_i = f(X_i, w) + \epsilon_i \tag{12}$$

L'objectif est d'estimer la fonction $f(X_i, w)$ qui correspond le mieux aux données.

Pour effectuer une analyse de régression, la forme de la fonction f doit être spécifié. Parfois, la forme de cette fonction est basée sur la connaissance de la relation entre Y_i et X_i. Si ces connaissances ne sont pas disponibles, un formulaire souple ou pratique pour f est choisi. Par exemple, une simple régression univariée peut proposer

$$f(X_i, w) = w_0 + w_1 X_i$$

ou

$$Y_i = w_0 + w_1 X_i + e_i$$

être une approximation raisonnable du processus statistique générant les données.

Différentes formes d'analyse de régression fournissent des outils pour estimer les paramètres. w. Par exemple, les moindres carrés trouvent la valeur de w qui minimise la somme des carrés des erreurs [13].

$$\sum_{i}^{n} (Y_i - f(X_i, w))^2$$

Étant donné un ensemble de données $\{y_i, x_{i1}, \ldots, x_{ip}\}_{i=1}^{n}$ de n unités statistiques, un modèle de régression linéaire suppose que la relation entre la variable dépendante y et le vecteur p des régresseurs x est linéaire. Cette relation est **modélisée** par un terme de perturbation ou une variable d'erreur ϵ : une variable aléatoire non observée qui ajoute du "bruit" à la relation linéaire entre la variable dépendante et les régresseurs [3, 12]. Ainsi le modèle prend la forme

$$y_i = w_0 + w_1 x_{i1} + \cdots + w_n x_{in} + \varepsilon_i = \mathbf{x}_i^T \boldsymbol{w} + \varepsilon_i, \qquad \text{avec} \quad i = 1, \ldots, n,$$

Souvent, ces n équations sont empilées et écrites en notation matricielle comme

$$\mathbf{y} = X\boldsymbol{w} + \boldsymbol{\varepsilon}, \tag{13}$$

où

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad X = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix}, \quad \boldsymbol{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{pmatrix}, \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}.$$

$\mathbf{y}$ est un vecteur de valeurs observées y_i $(i = 1, \ldots, n)$ de la variable appelée variable mesurée ou variable dépendante.

X peut être vu comme une matrice de vecteurs-lignes $\mathbf{x}_i$ ou de vecteurs-colonnes à n dimensions X_j, appelées régresseurs, variables explicatives, variables d'entrée, variables prédictives ou variables indépendantes. La matrice X est parfois appelée la matrice de conception.

$\boldsymbol{w}$ est un vecteur de paramètre de dimension $(p + 1)$, où w_0 est le terme d'interception, s'il n'est pas inclus dans le modèle $\boldsymbol{w}$ est de dimension p. Ses éléments sont appelés

coefficients de régression [3]. En régression linéaire simple, $p = 1$, et le coefficient est appelé **pente** de régression.

L'estimation statistique et l'inférence dans la régression linéaire se concentrent sur w. Les éléments de ce vecteur de paramètres sont interprétés comme les dérivées partielles de la variable dépendante par rapport aux différentes variables indépendantes [12].

En définissant les vecteurs et matrice ci dessous, $\mathbf{X}$, $\boldsymbol{w}$ et $\mathbf{y}$ (avec $S_y = y$) [3] ; le critère de la somme des carrées des erreurs s'écrit alors :

$$\mathrm{SCE}(w|S) = \frac{1}{2}(\mathbf{S}_y - \mathbf{X}\boldsymbol{w})^T(\mathbf{S}_y - \mathbf{X}\boldsymbol{w}) \tag{14}$$

Il suffit de prendre la dérivée de la somme des carrés des erreurs (équation 11) par rapport à w, qui est maintenant remplacer par w, pour obtenir les équations :

$$\frac{\partial \mathrm{SCE}}{\partial w} = -\mathbf{X}^T(\mathbf{S}_y - \mathbf{X}\boldsymbol{w})$$

$$\frac{\partial^2 \mathrm{SCE}}{\partial^2 w \partial^2 w^T} = -\mathbf{X}^T\mathbf{X}$$

En supposant que la matrice X est non singulière, et donc que X^TX est positive définie, et en posant que la dérivée première est nulle, on obtient :

$$X^TXw = X^TS_y \tag{15}$$

à partir de quoi on peut calculer l'unique solution par :

$$\hat{w} = (X^TX)^{-1}X^TS_y \tag{16}$$

La valeur $\hat{y}$ prédite pour une entrée x_n est donc :

$$\hat{y} = \hat{w} \cdot x_n = (X^TX)^{-1}X^TS_yx_n$$

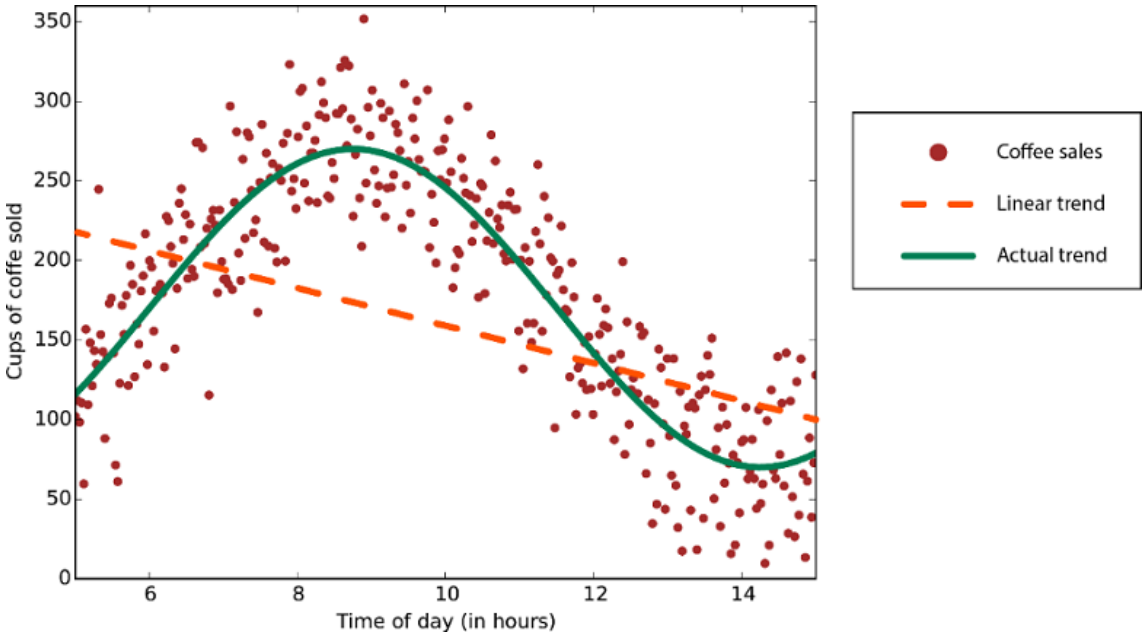


FIGURE 3 : L'efficacité de la régression non linéaire par rapport à une régression linéaire..

RÉGRESSION LINÉAIRE MULTIPLE La régression linéaire multiple est une généralisation de la régression linéaire simple au cas de plus d'une variable indépendante, et un cas particulier des modèles linéaires généraux, limités à une variable dépendante.

1.2.3 *Les problèmes de classifications*

En apprentissage automatique, les classifieurs linéaires sont une famille d'algorithmes de classement statistique. Le rôle d'un classifieur est de classer dans des groupes (des classes) les échantillons qui ont des propriétés similaires, mesurées sur des observations. Un classifieur linéaire est un type particulier de classifieur, qui calcule la décision par combinaison linéaire des échantillons [3].

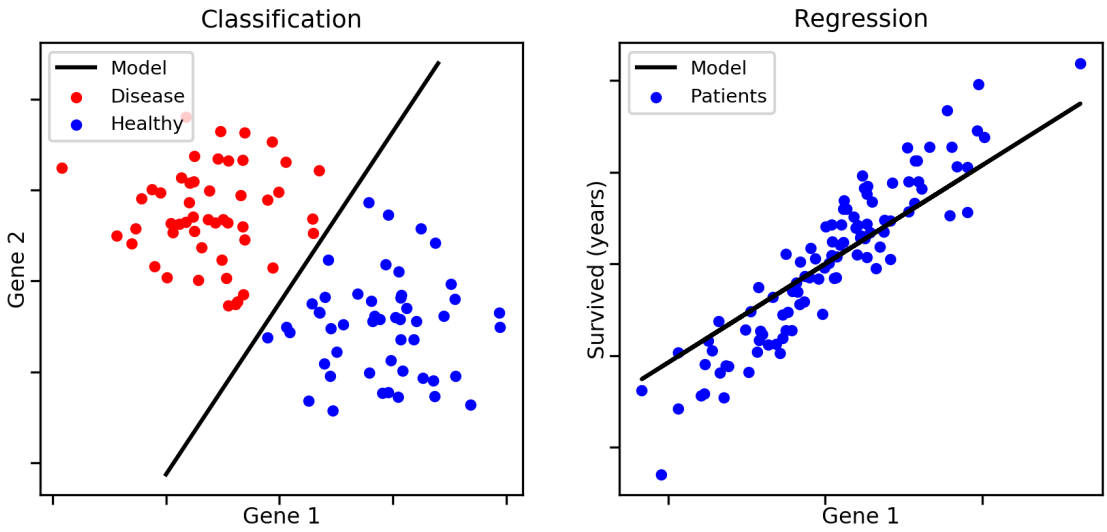


FIGURE 4 : Classification vs régression [27].

Nous nous plaçons dans le cadre où la variable dépendante ou à prédire prend ses valeurs dans un ensemble fini que l'on associe généralement à un ensemble de classes. A la différence de la régression linéaire où l'ensemble de valeurs à prédire est infini.

Lorsque l'on se place dans un espace de représentation euclidien, on peut librement faire des hypothèses sur la géométrie des classes ou sur celles de leurs surfaces séparatrices. La plus simple d'entre elles est de supposer que deux classes peuvent être séparées par une certaine surface, définie par une équation ; les paramètres qui régissent cette équation sont alors les variables à apprendre.

Le nombre de paramètres à calculer est minimal si l'on suppose cette surface linéaire ; aussi est-ce l'hypothèse qui prévaut souvent, en particulier lorsque l'échantillon de données est de taille réduite par rapport à la dimension de l'espace d'entrée, d'autant qu'elle permet de mener des calculs faciles et de visualiser précisément le résultat obtenu [25].

Dans $\mathbb{R}^n$, une surface linéaire est un hyperplan A, défini par l'équation :

$$a_0 + a^T x = 0$$

avec a vecteur de dimension n et a_0 scalaire. Si deux classes $\mathcal{C}_1$ et $\mathcal{C}_2$ sont *séparables* par A, tous les points de la première classe sont par exemple tels que :

$$x \in \mathcal{C}_1 \implies a_0 + a^T x > 0 \tag{17}$$

et ceux de la seconde vérifient alors :

$$x \in \mathcal{C}_2 \implies a_0 + a^T x \leqslant 0 \tag{18}$$

Dans un espace de dimension $d = 1$, une séparation linéaire se réduit à la comparaison à un seuil. Prenons ce cas particulier pour donner deux exemples où un problème de discrimination à deux classes ne peut pas en pratique être complètement résolu par une séparatrice linéaire.

SÉPARATRICE LINÉAIRE : On appelle hyperplan séparateur ou séparatrice linéaire un hyperplan qui sépare parfaitement deux classes, c'est-à-dire qui vérifie les équations 17 et 18 ; en particulier, il sépare parfaitement leurs points d'apprentissage. Un hyperplan discriminant est un classificateur linéaire pour deux classes qui ne sont pas linéairement séparables [3].

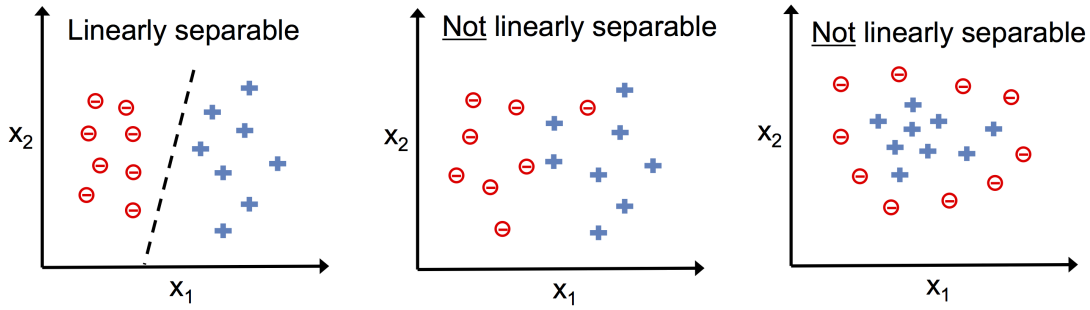


FIGURE 5 : Classes linéairement séparables [27]

A *Le modèle de la régression logistique*

Ce qu'il est convenu d'appeler *régression logistique* concerne en fait une méthode de classification binaire, à l'instar du perceptron (voir la section 1.3.1). A la différence du perceptron, cependant, nous allons chercher à apprendre une hypothèse h définie de $\mathbb{R}^n$ dans [0,1], et non pas dans 0,1, une motivation étant d'interpréter $h(x)$ comme étant la probabilité que l'entrée x appartienne à la classe d'intérêt que nous notons $\mathcal{C}_1$. [3]

Dans le cas à deux classes, nous sommes intéressés par le rapport de probabilités conditionnelles :

$$\frac{\mathbf{P}(y = \mathcal{C}_1|\mathbf{x})}{\mathbf{P}(y = \mathcal{C}_2|\mathbf{x})} = \frac{\mathbf{P}(y = \mathcal{C}_1)}{\mathbf{P}(y = \mathcal{C}_2)} \frac{\mathbf{p}(\mathbf{x}|y = \mathcal{C}_1)}{\mathbf{p}(\mathbf{x}|y = \mathcal{C}_2)} \tag{19}$$

Bien entendu, on affecte l'entrée x à la classe $\mathcal{C}_1$ si le rapport 19 est > 1, et à la classe $\mathcal{C}_1$ sinon [3].

Le terme $\frac{\mathbf{p}(\mathbf{x}|y=\mathcal{C}_1)}{\mathbf{p}(\mathbf{x}|y=\mathcal{C}_2)}$ est n'est facile à estimer à partir des fréquences mesurées des classes $\mathcal{C}_1$ et $\mathcal{C}_2$. Pour estimer ce terme, il faut faire des hypothèses sur sa forme. Dans la régression logistique, on fait l'hypothèse que le logarithme du rapport est de forme linéaire :

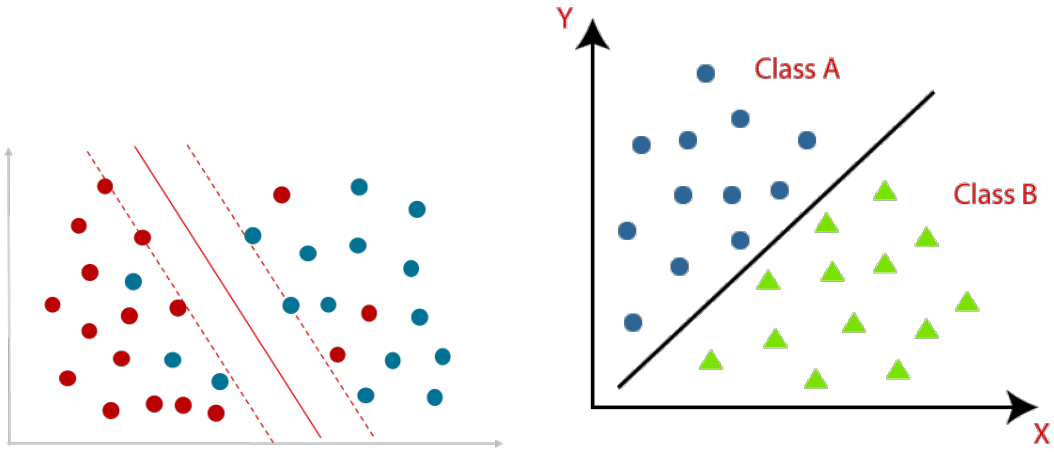
$$\log\left\{\frac{\mathbf{p}(\mathbf{x}|y=\mathcal{C}_1)}{\mathbf{p}(\mathbf{x}|y=\mathcal{C}_2)}\right\} = w^T x + b_0$$

Il est possible de ré-exprimer l'équation 19 en utilisant les propriétés logarithmique, rappel :

$$\log(a.b) = \log(a) + \log(b) \qquad \log(x)^n = n\log(x)$$

et en utilisant la règle de Bayes, l'équation 19 deviendra :

$$\log\frac{\mathbf{P}(\mathcal{C}_1|\mathbf{x})}{1-\mathbf{P}(\mathcal{C}_1|\mathbf{x})} = \log\frac{\mathbf{P}(\mathcal{C}_1)}{\mathbf{P}(\mathcal{C}_2)} + \log\frac{\mathbf{p}(\mathbf{x}|y=\mathcal{C}_1)}{\mathbf{p}(\mathbf{x}|y=\mathcal{C}_2)} = w^T x + b \tag{20}$$



(a) La zone de décision et la droite séparatrice. (b) Classification linaire de deux classes A et B

FIGURE 6 : Exemple de la classification

La fonction de la droite séparatrice comme l'illustre la figure 6a et 6b s'écrit :

$$z = w_1 x_1 + \cdots + w_n x_n + b \tag{21}$$

avec $i = 1, \ldots, n$, et w_i et b des paramètres de la droite.

$$\begin{cases} \hat{y} = 0 \quad (y \in \mathcal{C}_1) & \text{si } z < 0 \\ \hat{y} = 1 \quad (y \in \mathcal{C}_2) & \text{si } z \geqslant 0 \end{cases}$$

La fonction logistique est une fonction sigmoïde , qui prend n'importe quelle entrée réelle t, et renvoie une valeur comprise entre zéro et un [27]. La fonction logistique standard $\sigma : \mathbb{R} \to (0, 1)$ est défini comme suit :

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}} \tag{22}$$

Supposons que t est une fonction linéaire (comme la droite de la formule 21) $t = z$. Et la fonction logistique générale $p : \mathbb{R} \to (0, 1)$ peut maintenant l'écrire :

$$p(x) = \sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-(w_1 x_1 + \cdots + w_n x_n + b)}} \tag{23}$$

Dans le modèle logistique, $p(x)$ est interprété comme la probabilité de la variable dépendante Y équivalant à un succès/cas-oui plutôt qu'à un échec/non-cas. Il est clair que les variables de réponse Y_i ne sont pas identiquement répartis :$P(Y_i = 1 \mid X)$ diffère d'un point de données X_i à l'autre, bien qu'ils soient indépendants étant donné la matrice de conception X et paramètres partagés w [3].

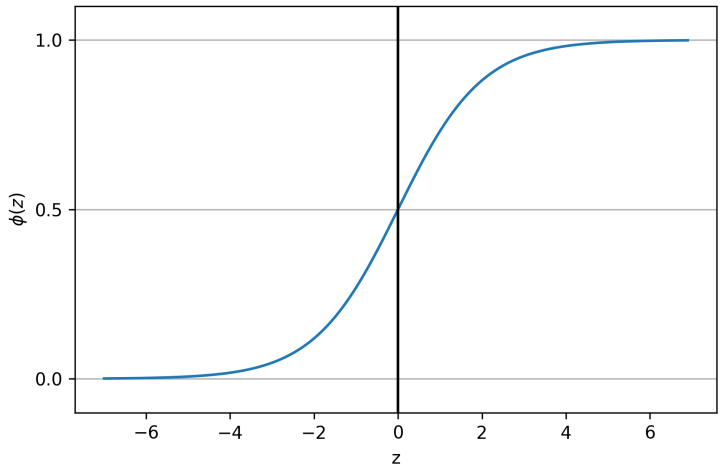


FIGURE 7 : Graphique représentant fonction sigmoïde logistique ajustée aux données (x_n, y_n). [27]

LA VRAISEMBLANCE : Indique la plausibilité du modèle vis-a-vis du vraies données. Soit l'échantillon $\mathcal{S} = (\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)$, avec $y_i \in \{\mathcal{C}_1, \mathcal{C}_2\}, \forall_i \in (1, ..., m)$. Sa vrai semblance en fonction des paramètres à apprendre s'écrit :

$$L = \prod_{i=1}^{m} p_i{}^{y_i} (1 - p_i)^{1 - y_i} \tag{24}$$

où m est le nombre d'exemples d'apprentissage appartenant à la classe.

Dans [3], il est montré que ces paramètres peuvent être obtenus par maximisation de la vraisemblance des paramètres conditionnellement aux exemples. Il a été de plus montré que, sous des conditions très générales [25], le maximum de L est unique. La

maximisation de la vraisemblance se fait sois en passant par le logarithme, pour obtenir la log-vraisemblance :

$$\begin{aligned}\log(L) &= \log(\prod_{i=1}^{m} {p_i}^{y_i}(1-p_i)^{1-y_i}) \\ &= \sum_{i=1}^{m} \log({p_i}^{y_i}) + \log((1-p_i)^{1-y_i}) \\ &= \sum_{i=1}^{m} y_i \log(p_i) + (1-y_i)\log(1-p_i)\end{aligned} \tag{25}$$

Comme en Machine Learning on est plus habile à minimiser qu'à maximiser et que maximiser une fonction $f(\cdot)$ consiste à minimiser $-f(\cdot)$ alors la log-vraisemblance s'écrira :

$$\mathcal{L} = -\frac{1}{m}\sum_{i=1}^{m} y_i \log(p_i) + (1-y_i)\log(1-p_i) \tag{26}$$

avec le terme $\frac{1}{m}$ pour augmenter la précision. Avec cette fonction, nous allons maximiser la vraisemblance L en minimisant $-\log(L)$.

La log-vraisemblance négative est égale à la perte logarithmique (Log-loss) sous une distribution de probabilité de Bernoulli [??].

1.3 RÉSEAU DE NEURONES, APPRENTISSAGE EN PROFONDEUR

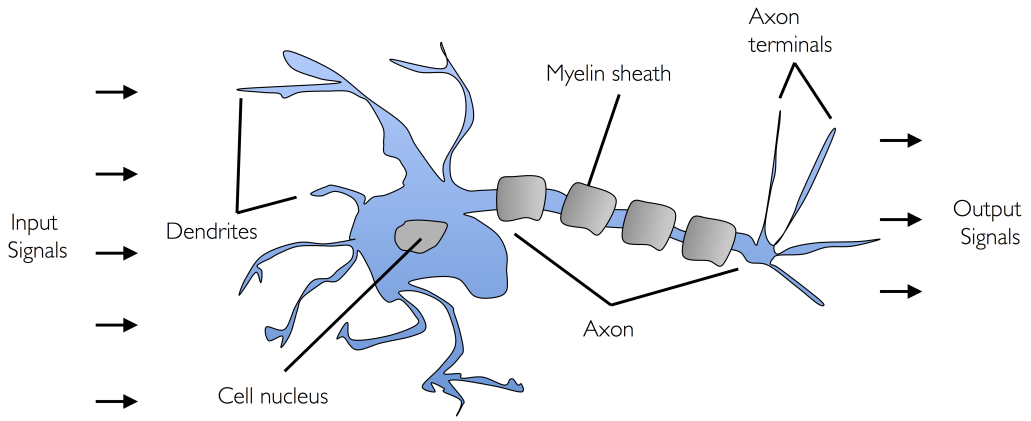


FIGURE 8 : Neurone biologique [27]

1.3.1 *Perceptron*

Le perceptron est un modèle simplifié d'un neurone biologique. Alors que la complexité des modèles de neurones biologiques est souvent nécessaire pour bien comprendre le comportement neuronal, la recherche suggère qu'un modèle linéaire de type perceptron peut produire certains comportements observés dans de vrais neurones.

Un perceptron est donc une unité de réseau neuronal, l'élément de traitement de base, qui effectue certains *calculs* pour détecter des caractéristiques ou une intelligence économique dans les données d'entrée.
Le perceptron, à l'instar du neurone artificiel classique, est conçu avec un algorithme d'apprentissage supervisé du même nom.

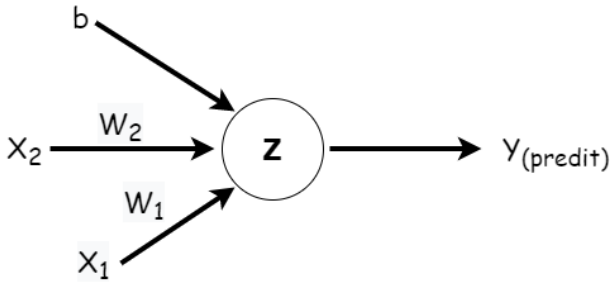


FIGURE 9 : Neurone logique avec deux entrées x_1 et x_2 et les paramètres w_1, w_2 (w_i dans un réseau de neurones il est nommé poids) et b aussi appelé biais. le fonction d'agrégation z (voir le point a, section 1.2.2 et la formule 21) sera : $z = w_1x_1 + w_2x_2 + b$.

Le perceptron a des entrées qui peuvent provenir de l'environnement ou peuvent être les sorties d'autres perceptrons. Associé à chaque entrée, $x_j \in \mathbb{R}$, avec $j = 1, 2, \ldots, n$, est un *poids de connexion, ou poids* synaptique $w_j \in \mathbb{R}$, et la sortie, $\hat{y}$. Dans le cas le plus simple $\hat{y}$ est une somme pondérée des entrées [2].

$$\hat{y} = \sum_{j=1}^{n} w_j x_j + w_0$$

w_0 est la valeur d'interception pour rendre le modèle plus général, il est généralement modélisé comme la pondération provenant d'une unité de biais supplémentaire, $b = w_0 x_0$, avec x_0 qui est toujours égale $+1$. Nous pouvons écrire la sortie du perceptron sous la forme d'un produit scalaire.

$$\hat{y} = x^T w$$

Pendant le test, avec des poids donnés, w, pour l'entrée x, nous calculons la sortie $\hat{y}$. Pour implémenter une tâche donnée, nous avons besoin d'apprendre les poids w, les paramètres du système, de sorte que des sorties correctes soient générées compte tenu des entrées.

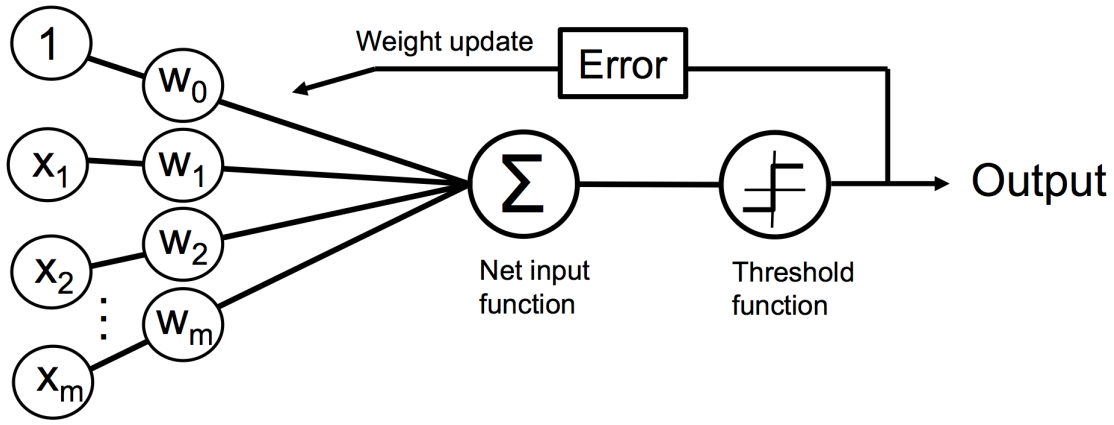


FIGURE 10 : Neurone artificiel modèle perceptron [27], cette figure est plus adapté pour représenter le modèle du perceptron.

A *Calcul avec l'algorithme du perceptron*

Le premier concept de règle d'apprentissage du perceptron, a été publié par Frank Rosenblatt [3], basé sur le modèle neuronal MP Neuron (McCulloch-Pitts Neuron).

Avec sa règle de perceptron, Il a proposé un algorithme qui apprendrait automatiquement les coefficients de poids optimaux qui sont ensuite multipliés par les caractéristiques d'entrée afin de décider si un neurone se déclenche ou non. Dans le cadre de l'apprentissage supervisé et de la classification, un tel algorithme pourrait alors être utilisé pour prédire si un échantillon appartient à une classe ou à l'autre [27].

Il est important de noter que la convergence du perceptron n'est garantie que si les deux classes sont linéairement séparables et que le taux d'apprentissage est suffisamment faible. Si les deux classes ne peuvent pas être séparées par une limite de décision linéaire, nous pouvons définir un nombre maximum de passages sur l'ensemble de données d'apprentissage (époques) et/ou un seuil pour le nombre d'erreurs de classification tolérées - le perceptron n'arrêterait jamais de mettre à jour les poids autrement [15, 27].

L'algorithme du perceptron travaille directement sur le vecteur a qui caractérise la surface discriminante cherchée. On n'a donc plus besoin ici de se placer dans l'espace de représentation de dimension n+1. Cet algorithme utilise un protocole d'apprentissage itératif : il prend les données d'apprentissage les unes après les autres, chacune étant choisie soit par un passage systématique dans l'ensemble d'apprentissage (version « *non stochastique* »), soit par un tirage au hasard dans celui-ci (version « *stochastique* »). Son nombre d'étapes effectives peut être important : un seul (exactement ou en moyenne) passage des données n'est en effet pas suffisant pour le faire converger (voir le chapitre **??**, section **??**). [3].

Ci-dessous un exemple d'un algorithme du perceptron implémenté en python.

```
        class Perceptron(object):
                # Parametres
                eta : float # taux d'apprentissage (between 0.0 and 1.0)
                n_iter : int # nombre d'iteration
                random_state : int # generateur de nombres aleatoires pour l'
                    initialisation de poids aleatoire.
                w_ : list # Poids apres fitting.
                errors_ : list # Nombre d'erreurs de classification (mises a jour)
                    a chaque epoque.

                def __init__(self, eta=0.01, n_iter=50, random_state=1):
                        self.eta = eta
                        self.n_iter = n_iter
                        self.random_state = random_state

                # Parameters
                X : list, # matrice d'echantillons de taille = (n \times m).
                y : list, # vecteur de taille = n, qui contient les valeurs cibles.

                def fit(self, X, y): # Ajuster les donnees d'entrainement (Fit
                    training data).
                        rgen = np.random.RandomState(self.random_state)
                        self.w_ = rgen.normal(loc=0.0, scale=0.01, size=1 + X.shape
                            [1])
                        self.errors_ = []

                        for _ in range(self.n_iter):
                                errors = 0
                                for xi, target in zip(X, y):
                                        update = self.eta * (target - self.predict(
                                            xi))
                                        self.w_[1:] += update * xi
                                        self.w_[0] += update
                                        errors += int(update != 0.0)
                                        self.errors_.append(errors)
                        return self

                def net_input(self, X): # Calculer la sortie net
                        return np.dot(X, self.w_[1:]) + self.w_[0]

                def predict(self, X):
                        return np.where(self.net_input(X) >= 0.0, 1, -1)
```

B *Le réseau de neurones artificiels (Perceptron Multicouche)*

Les réseaux de neurones ont été introduits pour la première fois comme méthode d'apprentissage par Frank Rosenblatt, bien que le modèle d'apprentissage appelé perceptron soit différent des réseaux de neurones modernes, nous pouvons toujours considérer le perceptron comme le premier réseau de neurones artificiels [25].

Le perceptron multicouche (en anglais : multilayer perceptron MLP) est un type de réseau neuronal artificiel organisé en plusieurs couches, où les informations ne circulent que de la couche d'entrée à la couche de sortie. Il s'agit donc d'un réseau à propagation directe (feed forward), autrement dit le réseau profond à action directe. Un perceptron multicouche est juste une fonction mathématique mappant un ensemble de valeurs d'entrée à des valeurs de sortie. La fonction est formée en composant de nombreuses fonctions plus simples. Nous pouvons considérer chaque application d'une fonction mathématique différente comme fournissant une nouvelle représentation de l'entrée [3, 17].

Les perceptrons à une seule couche ne sont capables d'apprendre que des motifs linéairement séparables. Pour une tâche de classification avec une fonction d'activation d'étape, un seul nœud aura une seule ligne divisant les points de données formant les motifs. Plus de nœuds peuvent créer plus de lignes de division, mais ces lignes doivent en quelque sorte être combinées pour former des classifications plus complexes. Une deuxième couche de perceptrons, voire de nœuds linéaires, suffit à résoudre de nombreux problèmes autrement non séparables [3].

Les réseaux de neurones artificiels (ANN) fonctionnent vaguement sur le principe de l'apprentissage d'une distribution distribuée de données. L'hypothèse sous-jacente est que les données générées sont le résultat d'une combinaison non linéaire d'un ensemble de facteurs latents et si nous sommes capables d'apprendre cette représentation distribuée, nous pouvons alors faire des prédictions précises sur un nouvel ensemble de données inconnues. Le réseau de neurones le plus simple aura une couche d'entrée, une couche cachée (résultat de l'application d'une transformation non linéaire aux données d'entrée) et une couche de sortie. Les paramètres du modèle ANN sont les poids de chaque connexion qui existent dans le réseau et parfois un paramètre de biais [25].

1.3.2 *Fonctions d'activation, poids et biais*

A *Fonctions d'activation tangente (Sigmoïde et hyperbolique)*

La fonction d'activation est responsable de la transformation de l'entrée pondérée sommée du nœud en activation du nœud ou de la sortie pour cette entrée. Pour un nœud donné, les entrées sont multipliées par les poids d'un nœud et additionnées. Cette valeur est appelée activation sommée du nœud. L'activation sommée est ensuite transformée via une fonction d'activation et définit la sortie spécifique ou « activation » du nœud [27].
La fonction sigmoïde (aussi appelé fonction logistique voir le point a, section 1.2.3) est utilisée ici comme une fonction d'activation.

La fonction d'activation la plus simple est appelée activation linéaire, où aucune transformation n'est appliquée. Un réseau composé uniquement de fonctions d'activation linéaires est très facile à former, mais ne peut pas apprendre des fonctions de cartographie

complexes. Les fonctions d'activation linéaires sont toujours utilisées dans la couche de sortie pour les réseaux qui prédisent une quantité (par exemple, les problèmes de régression, voir le point 1.2.2) [16, 20].

Les fonctions d'activation non linéaires sont préférées car elles permettent aux nœuds d'apprendre des structures plus complexes dans les données. Traditionnellement, deux fonctions d'activation non linéaires largement utilisées sont les fonctions d'activation tangente sigmoïde et hyperbolique [17].

La fonction **d'activation sigmoïde**, est traditionnellement une fonction d'activation très populaire pour les réseaux de neurones. L'entrée de la fonction est transformée en une valeur comprise entre 0,0 et 1,0. Les entrées qui sont beaucoup plus grandes que 1,0 sont transformées à la valeur 1,0, de même, les valeurs beaucoup plus petites que 0,0 sont alignées sur 0,0.

La forme de la fonction pour toutes les entrées possibles est une forme en S de zéro jusqu'à 0,5 à 1,0. Pendant longtemps, jusqu'au début des années 1990, c'était l'activation par défaut utilisée sur les réseaux de neurones [20].

Supposons que z est une fonction linéaire (comme la droite de la formule 21). Et la fonction sigmoïde (en partant de la fonction logistique générale, formule 23) est :

$$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-(w_1x_1+\cdots+w_nx_n+b)}} \tag{27}$$

avec $z = w_1x_1 + \cdots + w_nx_n + b$.

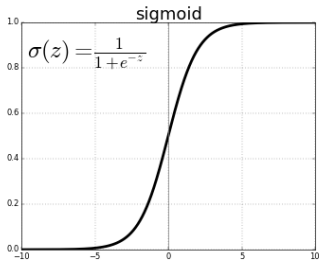


FIGURE 11 : Sigmoïde

La fonction **tangente hyperbolique**, ou **tanh** en abrégé, est une fonction d'activation non linéaire de forme similaire qui génère des valeurs comprises entre -1,0 et 1,0. À la fin des années 1990 et au cours des années 2000, la fonction tanh a été préférée à la fonction d'activation sigmoïde car les modèles qui l'utilisaient étaient plus faciles à entraîner et avaient souvent de meilleures performances prédictives [17]. La fonction d'activation tangente hyperbolique fonctionne généralement mieux que la sigmoïde logistique.

- *Limitations des fonctions d'activation sigmoïde et tanh*

Cela signifie que pour tanh et sigmoïde, les grandes valeurs sont alignées sur 1,0 et les petites valeurs sont alignées sur -1 ou 0. De plus, la fonction n'est vraiment sensible qu'aux changements proches du point médian de l'entrée. Par exemple, 0,5 pour les sigmoïdes et 0,0 pour la tanh.

Les unités sigmoïdales saturent sur la majeure partie de leur domaine - elles saturent à une valeur élevée lorsque z est très positif, saturent à une valeur faible lorsque z est très négatif et ne sont fortement sensibles à leur entrée que lorsque z est proche de 0 [27].

La sensibilité et la saturation limitées de la fonction se produisent indépendamment du fait que l'activation additionnée du nœud fourni en entrée contient des informations utiles ou non. Une fois saturé, il devient difficile pour l'algorithme d'apprentissage de continuer à adapter les poids pour améliorer les performances du modèle [17].
Les couches profondes des grands réseaux utilisant ces fonctions d'activation non linéaires ne reçoivent pas d'informations de gradient utiles. L'erreur est rétropropagée (voir le

chapitre **??**, section **??**) sur le réseau et utilisée pour mettre à jour les pondérations. La quantité d'erreur diminue considérablement avec chaque couche supplémentaire à travers laquelle elle se propage, compte tenu de la dérivée de la fonction d'activation choisie. C'est ce qu'on appelle le problème du gradient de fuite et empêche les réseaux profonds (multicouches) d'apprendre efficacement [16].

Bien que l'utilisation de fonctions d'activation non linéaires permette aux réseaux de neurones d'apprendre des fonctions de cartographie complexes, elles empêchent efficacement l'algorithme d'apprentissage de fonctionner avec des réseaux profonds.

B *Fonction d'activation ReLU*

Dans le domaine des réseaux de neurones artificiels, ReLU (Rectified Linear Unit) ou fonction d'activation d'unité linéaire rectifiée est une fonction d'activation définie comme la partie positive de son argument [17].

$$f(x) = x^+ = \max(0, x)$$

ReLU est une fonction linéaire par morceaux qui produira l'entrée directement si elle est positive, sinon, la sortie est nulle [30]. C'est devenu la fonction d'activation par défaut pour de nombreux types de réseaux de neurones, car un modèle qui l'utilise est plus facile à former et atteint souvent de meilleures performances [16].

La conception d'unités cachées est un domaine de recherche extrêmement actif et ne dispose pas encore de nombreux principes directeurs théoriques définitifs. Les fonctions d'activations ReLU sont un excellent choix par défaut d'unité cachée.
De nombreux autres types d'unités cachées sont disponibles. Il peut être difficile de déterminer quand utiliser quel type, bien que les unités linéaires rectifiées soient généralement un choix acceptable [17]. ReLU peut résoudre le problème des gradients de fuite, consultez le didacticiel [24].

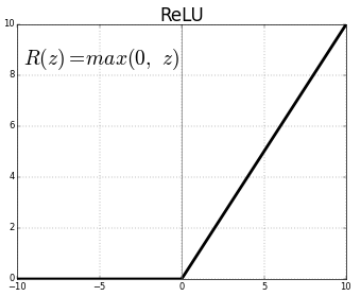


FIGURE 12 : ReLU

Autres fonctions d'activations

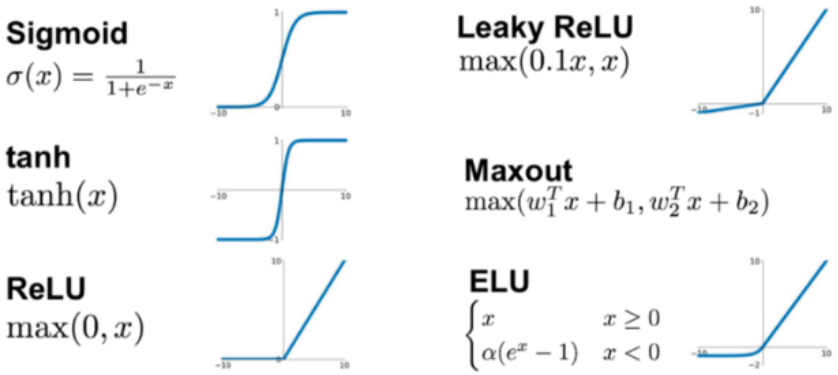


FIGURE 13 : Les différentes fonctions d'activation avec leurs graphes

1.3.3 *Réseau neuronal convolutif (CNN)*

Le réseau neuronal convolutif est un type de réseau neuronal artificiel (CNN, Convolutional Neural Network ou ConvNet) qui utilise plusieurs perceptrons qui analysent les entrées d'image et ont des poids et des bases apprenables sur plusieurs parties d'images et capables de se séparer les unes des autres [30].

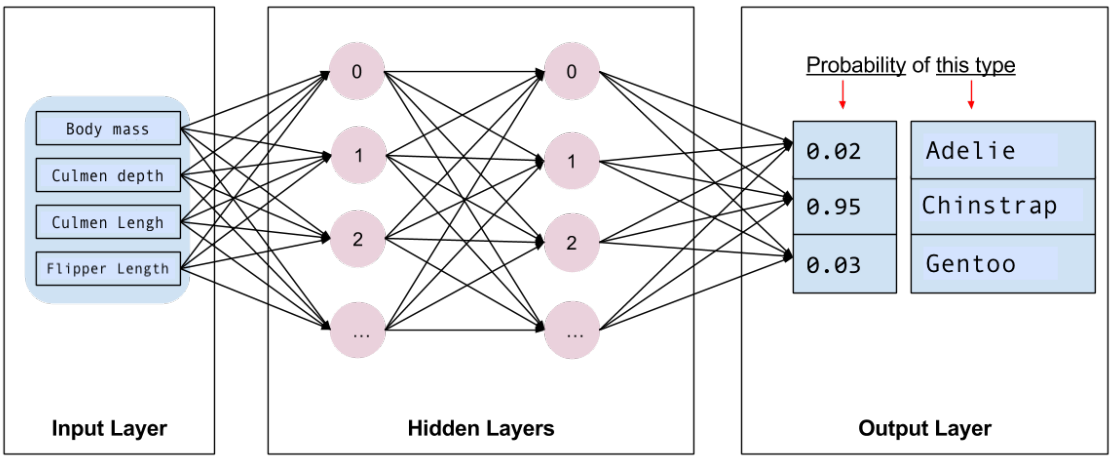


FIGURE 14 : Exemple de la classification avec un CNN à différentes couches : la couche d'entrée, les couches cachées et la couche de sortie (respectivement en anglais : Input Layer, Hidden Layers and Output Layer) [27]

.

Le CNN est un type de réseau de neurones acyclique à propagation avant, dans lequel le motif de connexion entre les neurones est inspiré par le cortex visuel des animaux. Les neurones de cette région du cerveau sont arrangés de sorte à ce qu'ils correspondent à des régions (appelés champs réceptifs) qui se chevauchent lors du pavage du champ visuel. Ils sont de plus organisés de manière hiérarchique, en couches (aire visuelle primaire V1, secondaire V2, puis aires V3, V4, V5 et V6, gyrus temporal inférieur), chacune des couches étant spécialisée dans une tâche, de plus en plus abstraite [3]. En simplifiant à l'extrême, une fois que les signaux lumineux sont reçus par la rétine et convertis en potentiels d'action :

- L'aire primaire V1 s'intéresse principalement à la détection de contours, ces contours étant définis comme des zones de fort contraste de signaux visuels reçus.
- L'aire V2 reçoit les informations de V1 et extrait des informations telles que la fréquence spatiale, l'orientation, ou encore la couleur.
- L'aire V4, qui reçoit des informations de V2, mais aussi de V1 directement, détecte des caractéristiques plus complexes et abstraites liées par exemple à la forme.
- Le gyrus temporal inférieur est chargé de la partie sémantique (reconnaissance des objets), à partir des informations reçues des aires précédentes et d'une mémoire des informations stockées sur des objets.

L'architecture et le fonctionnement des réseaux convolutifs sont inspirés par ces processus biologiques. Ces réseaux consistent en un empilage multicouche de perceptrons [30], dont le but est de pré-traiter de petites quantités d'informations.

Un réseau convolutif se compose de deux types de neurones, agencés en couches traitant successivement l'information. Dans le cas du traitement de données de type images [3], on a ainsi :

- des neurones de traitement, qui traitent une portion limitée de l'image (le champ réceptif) au travers d'une fonction de convolution[3, 30] ;
- des neurones de mise en commun des sorties dits d'agrégation totale ou partielle (pooling) [3, 30].

Un traitement correctif non linéaire est appliqué entre chaque couche pour améliorer la pertinence du résultat. L'ensemble des sorties d'une couche de traitement permet de reconstituer une image intermédiaire, dite carte de caractéristiques (feature map), qui sert de base à la couche suivante. Les couches et leurs connexions apprennent des niveaux d'abstraction croissants et extraient des caractéristiques de plus en plus haut niveau des données d'entrée [3, 28].

L'un des avantages de l'utilisation du réseau de neurones convolutifs est qu'il exploite l'utilisation de la cohérence spatiale locale dans les images d'entrée, ce qui leur permet d'avoir moins de poids car certains paramètres sont partagés [30].

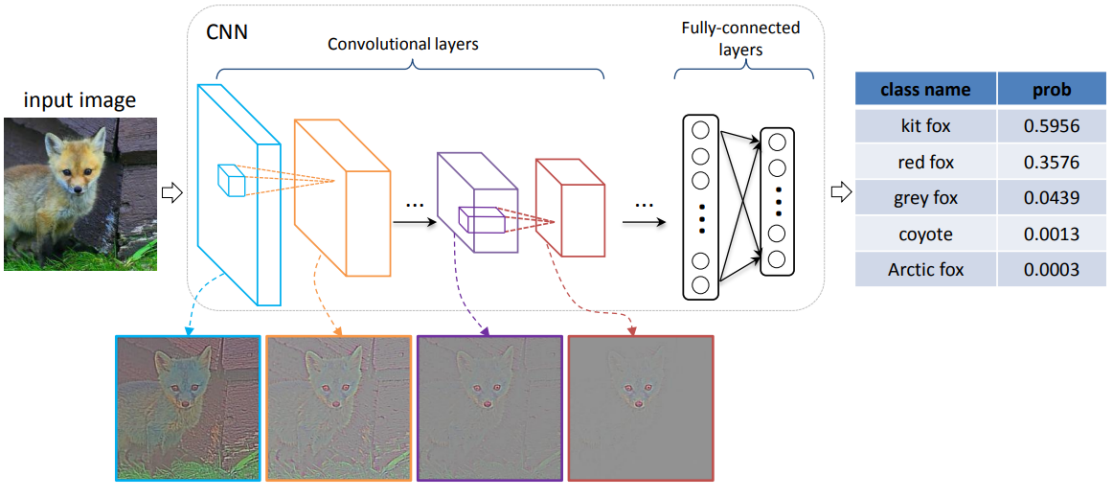


FIGURE 15 : L'illustration du comportement externe et interne d'un CNN. Le comportement externe correspond aux catégories de prédiction de sortie pour les images d'entrée. Le comportement interne est à sonder en visualisant les espaces de représentation construits par chaque couche et les informations visuelles conservées dans chaque couche. [32]

A *Les différents couches d'un CNN*

- *Couche de convolution*

La couche de convolution est la pierre angulaire du CNN. Il porte la majeure partie de la charge de calcul du réseau.

Cette couche effectue un produit scalaire entre deux matrices, où une matrice est l'ensemble de paramètres apprenables autrement connu sous le nom de noyau (en anglais kernel) K ou encore filtre de convolution, et l'autre matrice est la partie restreinte du

champ récepteur I. Le noyau est spatialement plus petit qu'une image mais il est plus en profondeur. Cela signifie que, si l'image est composée de trois canaux (RVB), la hauteur et la largeur du noyau seront spatialement petites, mais la profondeur s'étend jusqu'aux trois canaux [17].

DÉFINITION : *Soient* $h_1,\ h_2 \in \mathbb{N}, K \in \mathbb{R}^{(2h_1+1)\times(2h_2+1)}$. *La convolution de* I *par* K *est donnée par :*

$$(I * K)_{r,s} = \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} K_{u,v} I_{r+u,s+v} \tag{28}$$

où K est donnés par :

$$K = \begin{bmatrix} K_{-h_1,-h_2} & \cdots & K_{-h_1,h_2} \\ \vdots & K_{0,0} & \vdots \\ K_{h_1,-h_2} & \cdots & K_{h_1,h_2} \end{bmatrix}$$

La taille du filtre $(2h_1+1)\times(2h_2+1)$ précises le champ visuel capturé et traité par K. Lorsque K parcourt I, le déplacement du filtre est réglé par deux paramètres de *stride* (horizontal et vertical). Un stride de 1 horizontal (respectivement vertical) signifie que K se déplace d'une position horizontale (resp. verticale) à chaque application de la formule 28. Les valeurs de stride peuvent également être supérieures et ainsi sous-échantillonner I [3, 17].
Le comportement du filtre sur les bords de I doit également être précisé, par l'intermédiaire d'un paramètre de *padding*. Si l'image convoluée $(I * K)$ doit posséder la même taille que I, alors $2h_1$ lignes de 0 (h_1 à gauche et $2h_1$ à droite) et $2h_2$ colonnes de 0 (h_2 en haut et h_2 en bas) doivent être ajoutées. Dans le cas où la convolution est réalisée sans padding, l'image convoluée est de taille $n_1 - 2h_1 \times n_2 - 2h_2$ [3].

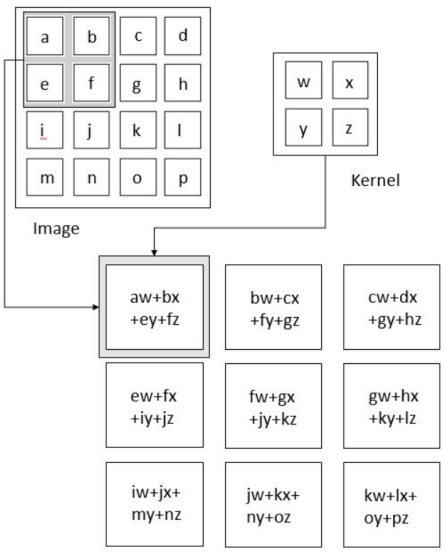


FIGURE 16 : Illustration de l'opération de convolution.

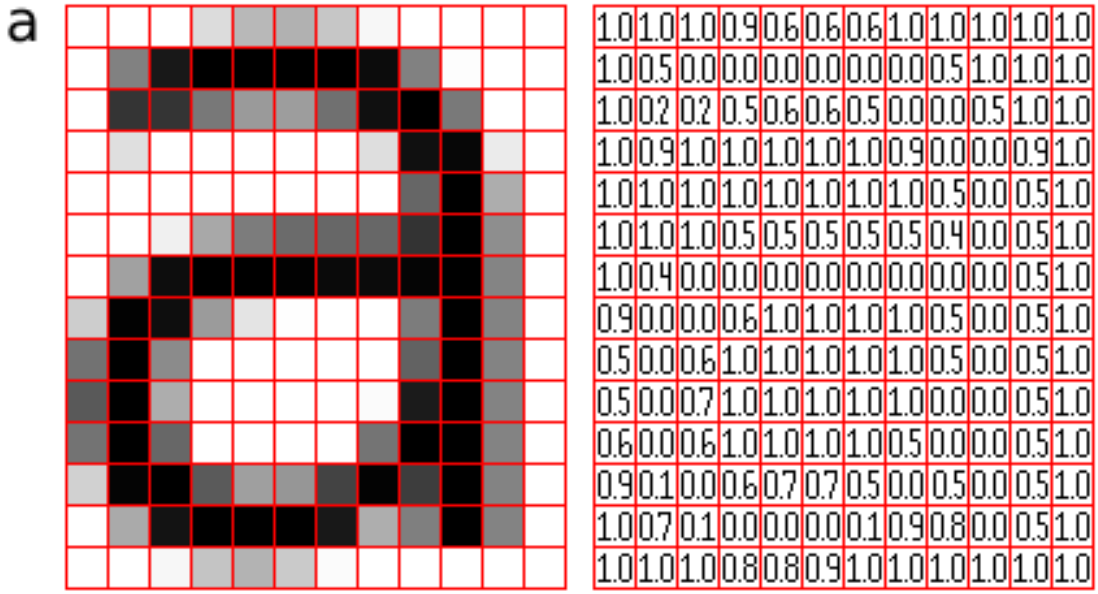


FIGURE 17 : Représentation de l'image sous forme de grille de pixels. Il contient une série de pixels disposés en forme de grille qui contient des valeurs de pixel pour indiquer la luminosité et la couleur de chaque pixel.

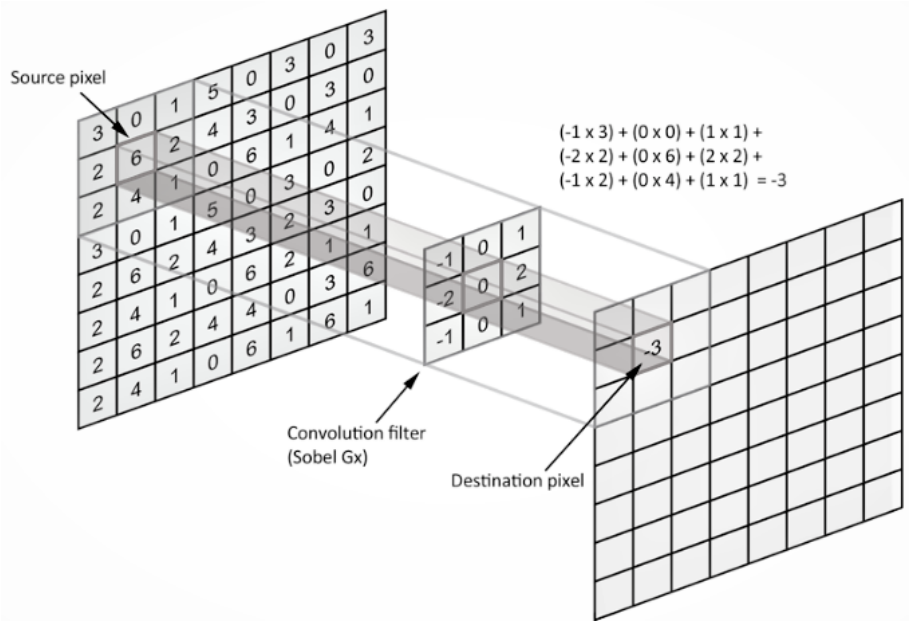


FIGURE 18 : Illustration des calculs effectués dans une opération de convolution [3].

Soit l une couche de convolution. L'entrée de la couche l est composée de $n^{(l-1)}$ cartes provenant de la couche précédente, de taille $n_1^{(l-1)} * n_2^{(l-1)}$. Dans le cas de la rétine $(l = 1)$ l'entrée est l'image I. La sortie de la couche l est formée de $n^{(l)}$ cartes de taille $n_1^{(l)} \times n_2^{(l)}$. La i^e cartes de la couche l notée $Y_i^{(l)}$ se calcule comme :

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{n^{(l-1)}} K_{i,j}^{(l)} * Y_j^{(l-1)} \tag{29}$$

où $B^{()}$ est une matrice de biais et $K_{i,j}^{(l)}$ est le filtre de taille $(2h_1^{(l)}+1) \times (2h_2^{(l)}+1)$ connectant la j^e carte de la couche $(l-1)$ à la i^e carte de la couche l.

$n_1^{(l)}$ et $n_2^{(l)}$ doivent prendre en compte les effets de bords : lors du calcul de la convolution, seuls les pixels dont la somme est définie avec des indices positifs doivent être traités. Dans le cas où le padding n'est pas utilisé, les cartes de sortie ont donc une taille de $n_1^{(l)} = n_1^{(l-1)} - 2h_1^{(l)}$ et $n_2^{(l)} = n_2^{(l-1)} - 2h_2^{(l)}$.

Par conséquent, la couche de convolution prend plusieurs images en entrée et utilise chaque filtre pour calculer la convolution de chaque image. Le filtre correspond exactement à la caractéristique que vous voulez trouver dans l'image [28]. Pour chaque paire (image, filtre), obtenez une carte d'activation ou une carte d'entités montrant où se trouvent les entités dans l'image. Plus la valeur est élevée, plus les points correspondants dans l'image sont similaires à l'entité [17].

- *Couche non linéaire*

Pour augmenter le pouvoir d'expression des réseaux profonds, on utilise couches non linéaires[3]. Étant donné que la convolution est une opération linéaire et que les images sont loin d'être linéaires, les couches de non-linéarité sont souvent placées directement après la couche de convolution pour introduire la non-linéarité dans la carte d'activation [17].

Trois grandes classes de fonctions d'activation f sont généralement utilisées : les fonctions de seuils (comme dans le perceptron linéaire à seuil), les fonctions linéaires par morceaux (ReLU : voir le la section 1.3.2, point b) et les fonctions de type sigmoïde [17]. Dans les deux premiers cas, de nombreux problèmes se présentent, notamment en raison de la non différentiabilité de ces fonctions (qui est nécessaire dans les algorithmes d'apprentissage du type descente de gradient), ou encore en raison de la faiblesse de leur pouvoir d'expression. Ainsi, il est préférable d'utiliser des fonctions de type sigmoïde [3].

- *Couche de regroupement (Pooling)*

Ce type de couche est souvent placé entre deux couches de convolution : elle reçoit en entrée plusieurs feature maps, et applique à chacune d'entre elles l'opération de pooling[4]. La couche de regroupement remplace la sortie du réseau à certains emplacements en dérivant une statistique récapitulative des sorties à proximité. Cela aide à réduire la taille spatiale de la représentation, ce qui diminue la quantité requise de calculs et de pondérations. L'opération de regroupement est traitée sur chaque tranche de la représentation individuellement [17].

Tout comme dans les couches conventionnelles, chaque neurone d'une couche de regroupement est connecté aux sorties d'un nombre limité de neurones de la couche

4 L'opération de pooling consiste à réduire la taille des images, tout en préservant leurs caractéristiques importantes.

précédente, situés dans un petit champ récepteur rectangulaire. Vous devez définir sa taille, la foulée et le type de rembourrage, comme avant. Cependant, un neurone de regroupement n'a pas de poids ; tout ce qu'il fait est d'agréger les entrées à l'aide d'une fonction d'agrégation telle que le max ou la moyenne [16].

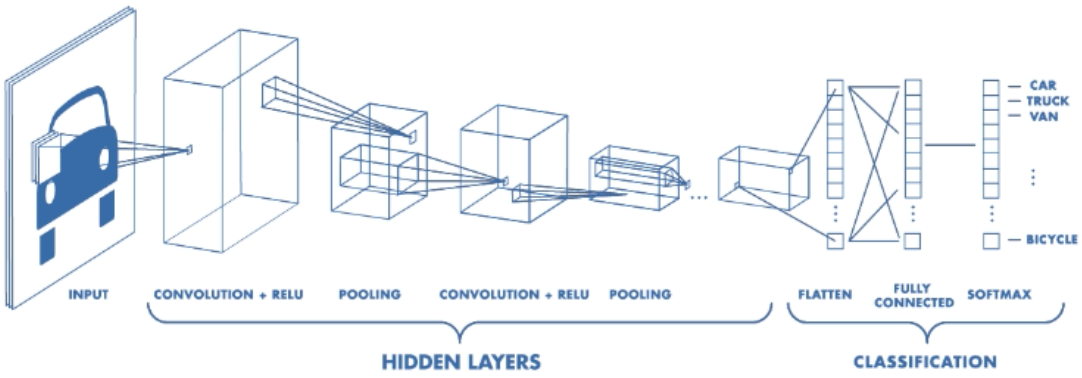


FIGURE 19 : Illustration de l'architecture d'un CNN.

Il existe plusieurs fonctions de regroupement telles que la moyenne du voisinage rectangulaire, la norme L2 du voisinage rectangulaire et une moyenne pondérée basée sur la distance par rapport au pixel central. Cependant, le processus le plus populaire est la mise en commun maximale, qui signale la sortie maximale du voisinage [17].

Le regroupement (pooling) des cartes obtenues par les couches précédentes a pour objectif d'assurer une robustesse au bruit et aux distorsions.
La sortie d'une couche d'agrégation est composée de $n^{(l)} = n^{(l-1)}$ cartes de taille réduite. En général, le pooling est effectuée en déplaçant dans les cartes d'entrée une fenêtre de taille $2p * 2p$ toutes les q positions, il y a recouvrement si $q < p$ et non recouvrement sinon, et en calculant, pour chaque position de la fenêtre, une seule valeur, affectée à la position centrale dans la carte de sortie [3]. On distingue généralement deux types de pooling :

- La moyenne : on utilise un filtre K_B de taille $(2h_1 + 1)(2h_2 + 1)$.
- Le maximum : la valeur maximum dans la fenêtre est retenue. Le maximum est souvent utilisé pour assurer une convergence rapide durant la phase d'entraînement. Le pooling avec recouvrement, il, semble assurer une réduction du phénomène de surapprentissage [3].

- *Couche entièrement connectée (fully-connected)*

La couche entièrement connectée constitue toujours la dernière couche d'un réseau de neurones, convolutif ou non, elle n'est donc pas caractéristique d'un CNN.

Les neurones de cette couche ont une connectivité complète avec tous les neurones de la couche précédente et suivante, comme on peut le voir dans un FCNN (Fully Convolutional Neural Network) normal. C'est pourquoi il peut être calculé comme

d'habitude par une multiplication matricielle suivie d'un effet de biais. La couche fully-connected aide à mapper la représentation entre l'entrée et la sortie [17].

Ce type de couche reçoit un vecteur en entrée et produit un nouveau vecteur en sortie. Pour cela, elle applique une combinaison linéaire puis éventuellement une fonction d'activation aux valeurs reçues en entrée.

La couche entièrement connectée détermine le lien entre la position des features dans l'image et une classe. En effet, le tableau en entrée étant le résultat de la couche précédente, il correspond à une carte d'activation pour une feature donnée : les valeurs élevées indiquent la localisation (plus ou moins précise selon le pooling) de cette feature dans l'image. Si la localisation d'une feature à un certain endroit de l'image est caractéristique d'une certaine classe, alors on accorde un poids important à la valeur correspondante dans le tableau [28].

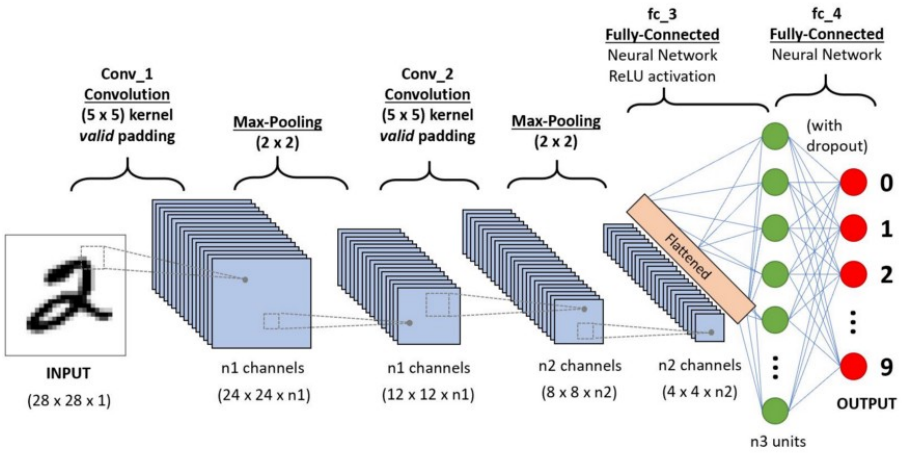


FIGURE 20 : Illustration d'un CNN montrant les différentes couches.

B *L'architecture VGGNet*

VGG signifie Visual Geometry Group, il s'agit d'une architecture standard de réseau de neurones à convolution profonde (CNN) à plusieurs couches [29].

Les réseaux VGG ont été les premiers à utiliser de petits filtres de convolution (3×3) et à les combiner pour décrire des séquences de convolution, l'idée étant d'émuler l'effet de larges champs réceptifs par cette séquence. Cette technique amène malheureusement à un nombre exponentiel de paramètres (le modèle entraîné qui peut être téléchargé a une taille de plus de 500 Mo). VGG a concouru à ILSVRC 2014, a obtenu un taux de bonne classification de 92.3% mais n'a pas remporté le concours [20]. Aujourd'hui, VGG est une famille de réseaux profonds (de A à E) qui varient par leur architecture (figure 22).

L'architecture VGG est la base d'un modèle innovant de reconnaissance d'objets. Développé en tant que réseau neuronal profond, VGGNet va au-delà d'ImageNet et dépasse la ligne de base de nombreuses tâches et ensembles de données. De plus, c'est toujours l'une des architectures de reconnaissance d'images les plus populaires [3, 30].

Les VGGNet sont basés sur les caractéristiques les plus essentielles des réseaux de neurones convolutifs (CNN). Le graphique suivant montre le concept de base du fonctionnement d'un CNN.

Un bref coup d'œil à l'architecture de VGG :

- **Entrée** : Le VGGNet prend une taille d'entrée d'image de 224×224. Pour le concours ImageNet, les créateurs du modèle ont recadré le patch central 224 × 224 dans chaque image pour conserver la cohérence de la taille d'entrée de l'image [29].
- **Couches convolutives** : Les couches convolutives de VGG tirent parti d'un champ de réception minimal, c'est-à-dire 3 × 3, la plus petite taille possible qui capture toujours haut/bas et gauche/droite. De plus, il existe également des filtres de convolution 1 × 1 agissant comme une transformation linéaire de l'entrée. Vient ensuite une unité ReLU, qui est une énorme innovation d'AlexNet qui réduit le temps de formation. La foulée de convolution est fixée à 1 pixel pour conserver la résolution spatiale préservée après la convolution (la foulée est le nombre de décalages de pixels sur la matrice d'entrée) [20, 30].
- **Couches cachées** : Toutes les couches cachées du réseau VGG utilisent ReLU. VGG n'utilise généralement pas la normalisation de la réponse locale (LRN) car elle augmente la consommation de mémoire et le temps de formation. De plus, il n'apporte aucune amélioration à la précision globale [30].
- **Couches entièrement connectées** : Le VGGNet a trois couches entièrement connectées. Sur les trois couches, les deux premières ont 4096 canaux chacune et la troisième a 1000 canaux, 1 pour chaque classe [30].

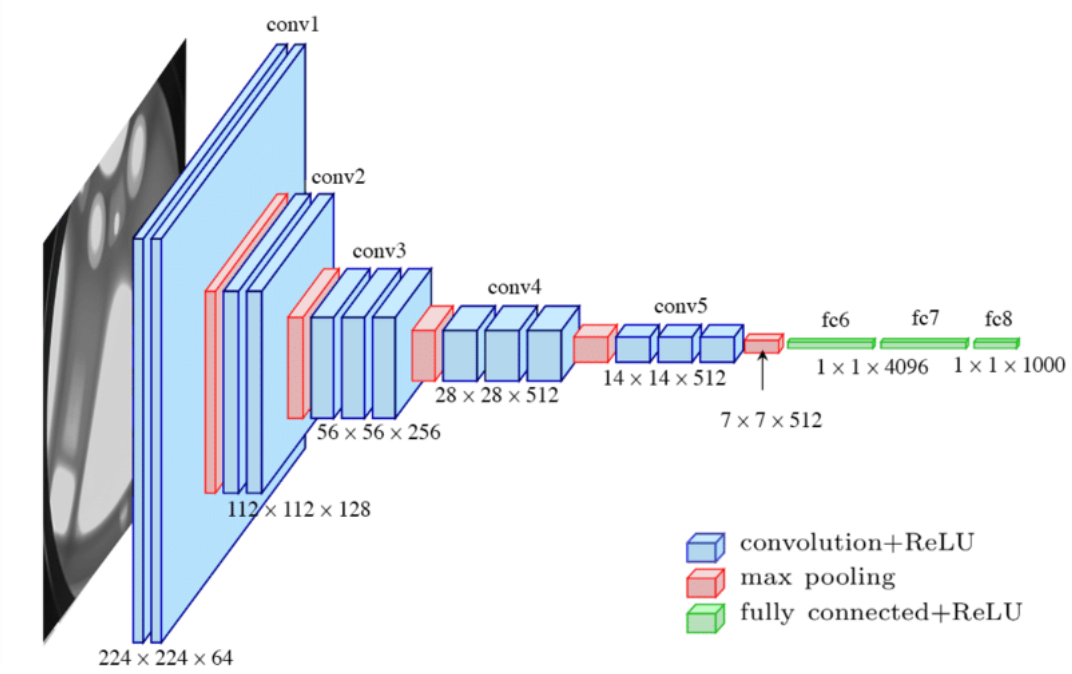


FIGURE 21 : CNN : architecture VGG [27]

.

LE MODÈLE VGG16 ET VGG19 : Visual Geometry Group et se compose de blocs, où chaque bloc est composé de couches 2D Convolution et Max Pooling. Il se décline en deux modèles - VGG16 et VGG19 - avec 16 et 19 couches [32].

VGG16 (également VGGNet-16) est modèle VGGNet, qui prend en charge 16 couches convolutives. Le nombre de filtres que nous pouvons utiliser double à chaque étape ou à travers chaque pile de la couche de convolution. C'est un principe majeur utilisé pour concevoir l'architecture du réseau VGG16. L'un des principaux inconvénients du réseau VGG16 est qu'il s'agit d'un réseau énorme, ce qui signifie qu'il faut plus de temps pour former ses paramètres [32].
En raison de sa profondeur et du nombre de couches entièrement connectées, le modèle VGG16 fait plus de 533 Mo. Cela rend la mise en œuvre d'un réseau VGG une tâche fastidieuse. Le modèle VGG16 est utilisé dans plusieurs problèmes de classification d'images d'apprentissage en profondeur, mais des architectures de réseau plus petites telles que GoogLeNet et SqueezeNet sont souvent préférables. Dans tous les cas, le VGGNet est un excellent élément de base à des fins d'apprentissage car il est simple à mettre en œuvre.

VGG16 surpasse largement les versions précédentes des modèles des compétitions ILSVRC-2012 et ILSVRC-2013. De plus, le résultat VGG16 est en compétition pour le vainqueur de la tâche de classification (GoogLeNet avec une erreur de 6,7%) et surpasse considérablement la soumission gagnante ILSVRC-2013 Clarifai. Il a obtenu 11,2% avec des données de formation externes et environ 11,7% sans elles. En termes de performances à réseau unique, le modèle VGGNet-16 obtient le meilleur résultat avec environ 7,0% d'erreur de test, dépassant ainsi un seul GoogLeNet d'environ 0,9% [29].

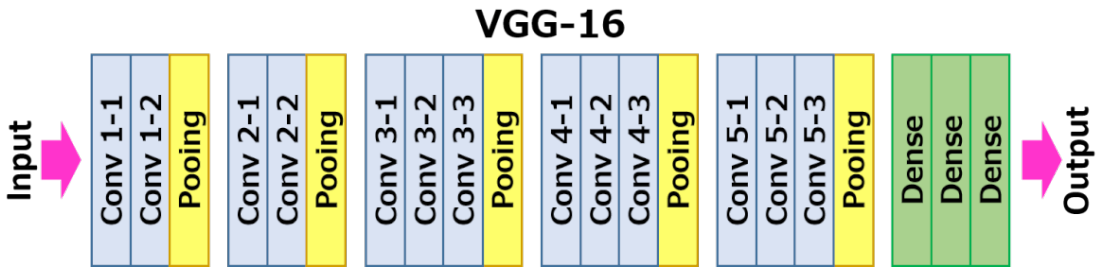


FIGURE 22 : Le modèle VGG-16 [30]

Le concept du modèle **VGG19** (également VGGNet-19) est le même que celui du VGG16, sauf qu'il prend en charge 19 couches. Le « 16 » et le « 19 » représentent le nombre de couches de poids dans le modèle (couches convolutives). Cela signifie que VGG19 a trois couches convolutionnelles de plus que VGG16. Nous aborderons plus en détail les caractéristiques des réseaux VGG16 et VGG19 dans la dernière partie de cet article [32].

ANNEXES ET BIBLIOGRAPHIES

BIBLIOGRAPHIE

[1] Yaovi Ahadjitse. "Reconnaissance d'objets en mouvement dans la vidéo par description géométrique et apprentissage supervisé". Thèse de doct. Université du Québec en Outaouais, 2013.

[2] E. Alpaydin. *Introduction to Machine Learning*. Adaptive computation and machine learning. MIT Press, 2010. isbn : 9780262012430.

[3] Vincent Barra Antoine Cornuéjols Laurent Michet. *Apprentissage artificiel : Deep leaning, concepts et algorithmes*. 3rd. Eyrolles, 2018, p. 239-263.

[4] P. Benner, M. Bollhöfer, D. Kressner, C. Mehl et T. Stykel. *Numerical Algebra, Matrix Theory, Differential-Algebraic Equations and Control Theory : Festschrift in Honor of Volker Mehrmann*. Springer International Publishing, 2015. isbn : 9783319152608. url : https://books.google.cd/books?id=MilACQAAQBAJ.

[5] M. Bierlaire. *Introduction à l'optimisation différentiable*. Enseignement des mathématiques. Presses polytechniques et universitaires romandes, 2006. isbn : 9782880746698. url : https://books.google.cd/books?id=HK55T5x2bygC.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. First. Springer-Verlag New York, 2006, p. 179-195.

[7] Léon Bottou. "Large-scale machine learning with stochastic gradient descent". In : *Proceedings of COMPSTAT'2010*. Springer, 2010, p. 177-186.

[8] Léon Bottou. "Stochastic gradient descent tricks". In : *Neural networks : Tricks of the trade*. Springer, 2012, p. 421-436.

[9] Léon Bottou, Frank E Curtis et Jorge Nocedal. "Optimization methods for large-scale machine learning". In : *Siam Review* 60.2 (2018), p. 223-311.

[10] F. Coulombeau, G. Debeaumarché, B. David, F. Dorra, S. Dupont et M. Hochart. *Mathématiques MPSI-PCSI : Programme 2013 avec algorithmique en Scilab*. Cap Prépa. Pearson, 2013. isbn : 9782744076527.

[11] Pádraig Cunningham, Matthieu Cord et Sarah Jane Delany. "Supervised learning". In : *Machine learning techniques for multimedia*. Springer, 2008, p. 21-49.

[12] R.B. Darlington et A.F. Hayes. *Regression Analysis and Linear Models : Concepts, Applications, and Implementation*. Methodology in the Social Sciences. Guilford Publications, 2016. isbn : 9781462521135.

[13] Natarajan Deepa, B Prabadevi, Praveen Kumar Maddikunta, Thippa Reddy Gadekallu, Thar Baker, M Ajmal Khan et Usman Tariq. "An AI-based intelligent system for healthcare analysis using Ridge-Adaline Stochastic Gradient Descent Classifier". In : *The Journal of Supercomputing* 77 (2021), p. 1998-2017.

[14] Kary Främling. "Scaled Gradient Descent Learning Rate". In : *Reinforcement Learning With Light-Seeking Robot, Proceedings of ICINCO* (2004), p. 1-8.

[15] Yoav Freund et Robert E Schapire. "Large margin classification using the perceptron algorithm". In : *Machine learning* 37.3 (1999), p. 277-296.

[16] A. Géron. *Hands-on Machine Learning with Scikit-Learn and TensorFlow : Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017. ISBN : 9781491962299.

[17] I. Goodfellow, Y. Bengio et A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016. ISBN : 9780262035613.

[18] F.E. Harrell et F.E.H. jrl. *Regression Modeling Strategies : With Applications to Linear Models, Logistic Regression, and Survival Analysis*. Graduate Texts in Mathematics. Springer, 2001. ISBN : 9780387952321. URL : https://books.google.cd/books?id=kfHrF-bVcvQC.

[19] Daniel Kirsch Judith Hurwitz. *Machine Learning For Dummies*. IBM Limited Edition. John Wiley et Sons, Inc., 2018.

[20] Alex Krizhevsky, Ilya Sutskever et Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In : *Advances in neural information processing systems* 25 (2012).

[21] N. Matloff. *Statistical Regression and Classification : From Linear Models to Machine Learning*. Chapman & Hall/CRC Texts in Statistical Science. CRC Press, 2017. ISBN : 9781351645898.

[22] Praneeth Netrapalli. "Stochastic gradient descent and its variants in machine learning". In : *Journal of the Indian Institute of Science* 99.2 (2019), p. 201-213.

[23] Jorge Nocedal et Stephen J Wright. *Numerical optimization*. T. 35. 1999.

[24] Arnu Pretorius, Elan Van Biljon, Steve Kroon et Herman Kamper. "Critical initialisation for deep signal propagation in noisy rectifier neural networks". In : *Advances in Neural Information Processing Systems* 31 (2018).

[25] D. Sarkar, R. Bali et T. Sharma. *Practical Machine Learning with Python : A Problem-Solver's Guide to Building Real-World Intelligent Systems*. Apress, 2017. ISBN : 9781484232071.

[26] Carl-Erik Särndal, Bengt Swensson et Jan Wretman. *Model assisted survey sampling*. Springer Science & Business Media, 2003.

[27] Vahid Mirjalili Sebastien Raschka. *Python Machine Learning and Deep Learning, with sckit-learn and Tensorflow*. 2nd. Packt, 2017, p. 17-139.

[28] Hoo-Chang Shin, Holger R Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura et Ronald M Summers. "Deep convolutional neural networks for computer-aided detection : CNN architectures, dataset characteristics and transfer learning". In : *IEEE transactions on medical imaging* 35.5 (2016), p. 1285-1298.

[29] Karen Simonyan et Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In : *arXiv preprint arXiv :1409.1556* (2014).

[30] Srikanth Tammina. "Transfer learning using VGG-16 with deep convolutional neural network for classifying images". In : *International Journal of Scientific and Research Publications (IJSRP)* 9.10 (2019), p. 143-150.

[31] Rob GJ Wijnhoven et PHN de With. "Fast training of object detection using stochastic gradient descent". In : *2010 20th International Conference on Pattern Recognition*. IEEE. 2010, p. 424-427.

[32] Wei Yu, Kuiyuan Yang, Yalong Bai, Tianjun Xiao, Hongxun Yao et Yong Rui. "Visualizing and comparing AlexNet and VGG using deconvolutional layers". In : *Proceedings of the 33 rd International Conference on Machine Learning*. 2016.