

L'IA : intelligente ou artificielle ?

Hugo Pompougnac, *Espaces Marx*

Résumé

L'essor des techniques d'Intelligence Artificielle (IA) pose des questions implacables : qu'est-ce que l'intelligence, et *a fortiori*, qu'est-ce que l'intelligence de l'IA ? Pour répondre, il faut nécessairement partir de ce que ces logiciels calculent *vraiment*. Cet article prend donc le parti de présenter les algorithmes fondamentaux du domaine et de les mettre en perspective sous l'angle du matérialisme dialectique.

Introduction

Dès *L'idéologie allemande*¹, Marx et Engels remarquaient que l'humanité fragmentée par la société de classes ne parvient plus à se reconnaître dans ce qu'elle crée ; alors, ses propres forces productives lui semblent étrangères, surnaturelles. . . magiques. Plus d'un siècle et demi plus tard, les logiciels dits d'Intelligence Artificielle (IA) sont particulièrement concernés par le phénomène. Les mots qui les désignent sont porteurs d'un imaginaire de science-fiction, sinon même de sorcellerie, décrivant une intelligence sans sujet, purement machinique.

La réalité est plus prosaïque. Les logiciels d'IA – comme les logiciels en général – cristallisent des savoirs collectifs sous une forme instrumentale. Ils ne constituent pas de nouvelles formes de vie. Ils ne reproduisent pas non plus le cerveau d'une personne humaine, pas davantage qu'un bras hydraulique ne reproduit le bras d'une personne humaine. À tout le plus, ils en imitent certaines caractéristiques limitées afin de pouvoir les utiliser dans des contextes précis.

D'où vient, alors, cette idéologie de la machine intelligente ? Comme toutes les idéologies, elle reflète les conditions matérielles de ceux qui l'émettent². En l'occurrence, elle s'enracine dans la vie concrète des ingénieurs, des entrepreneurs et des chercheurs de la Silicon Valley, dans les livres et les films qu'ils aiment, dans les formules ultra-futuristes qu'ils utilisent pour prendre le dessus dans les batailles de communication entre laboratoires, dans l'anthropologie libérale qui règne sans partage parmi ces vainqueurs de la mondialisation capitaliste.

Démystifions.

1. Karl Marx et Friedrich Engels, *L'idéologie allemande*, Paris, Éditions Sociales-Messidor, 1990, pp.92-96

2. Karl Marx, *Critique de l'économie politique*, Paris, Les éditions sociales, 2014, p.63

Intelligence Artificielle et Apprentissage Machine

L'encyclopédie Britannica³ nous indique que l'Intelligence Artificielle est la capacité, pour un ordinateur, d'accomplir des tâches associées à des êtres intelligents. Le terme est généralement appliqué, poursuit le texte, au projet de développer des systèmes embarquant des mécanismes intellectuels analogues à ceux des humains.

La première difficulté consiste à donner une représentation numérique, compatible avec les unités de calcul d'un ordinateur, des données que l'on veut traiter. Par exemple, une image va être représentée sous la forme d'un tableau de pixels (une *matrice*), dont chaque élément décrit la couleur du pixel représenté ; un texte va être représenté sous la forme d'une séquence de nombres (un *vecteur*) dont chacun représente un caractère de l'alphabet.

Pour mieux comprendre, prenons l'exemple de la norme ASCII, qui est la norme d'encodage de caractères la plus courante aujourd'hui. Dans ce format, chaque caractère est représenté par un nombre entier compris entre 0 et 256 ; par exemple, 'b' est représenté par 98, 'o' par 111, 'n' par 110, 'j' par 106, 'u' par 117, 'r' par 114. Le vecteur `98 111 110 106 111 117 114` peut donc être utilisé pour représenter le mot "bonjour". Pour écrire le mot "brun", on utilisera le vecteur `98 114 117 110`. Et ainsi de suite...

Il faut aussi donner une représentation numérique du résultat attendu ; par exemple, si on développe une application qui doit deviner la langue dans laquelle un texte est écrit, il faut prévoir que cette dernière réponde 1 pour le français, 2 pour l'anglais, 3 pour le russe, etc.

Une fois que l'on a caractérisé numériquement les données consommées et produites par l'application, le plus dur reste à faire : il faut programmer la séquence de calculs qui permet de produire le résultat attendu. Une telle séquence de calculs est ce qu'on appelle un algorithme. Il s'agit en quelque sorte de la recette de cuisine pour un programme informatique : prenez telles données, transformez-les de telle manière, si elles présentent telles caractéristiques faites cela, sinon faites ceci, répétez jusqu'à obtenir tel résultat, etc.

Il existe une grande variété d'algorithmes pour l'IA. Par exemple, dans les années 1980, on parlait surtout d'Intelligence Artificielle pour désigner les systèmes experts, logiciels intégrant des connaissances spécialisées comme le diagnostic médical ou l'identification des constituants chimiques d'un matériau. Ces connaissances étaient directement programmées dans le système sous la forme de règles logiques⁴.

Au fond, Marx décrivant ce processus dès 1857-58, en indiquant que "le développement du capital fixe indique jusqu'à quel degré le savoir social général, la

3. B.J. Copeland, "Artificial Intelligence", *Britannica*, 2025

4. Randall Davis et al., "Production rules as a representation for a knowledge-based consultation program", *Artificial Intelligence*, vol. 8, n° 1, 1977

connaissance, est devenue force productive immédiate⁵”. De fait, les premières machines industrielles *programmables* – comme le métier à tisser Jacquard, mis au point en 1801 – étaient passées par là. Nous verrons que cette idée s’applique tout aussi bien (mais à un niveau supérieur) à l’IA moderne.

Sa préhistoire remonte à 1950. Alan Turing, alors se demandait comment simuler l’esprit d’un humain adulte. La meilleure stratégie consisterait, disait-il, à imiter un cerveau d’enfant ; il suffirait ensuite de l’éduquer pour arriver au fonctionnement, plus élaboré, d’un cerveau d’adulte⁶. Cette formule a lancé les travaux sur l’*apprentissage machine* (dit aussi ML, pour *Machine Learning*). Initialement périphérique dans le monde de l’IA, ce sous-domaine est devenu, depuis les années 2000, hégémonique ou presque.

Concrètement, un logiciel d’apprentissage machine s’exécute dans deux modes différents : 1. Un mode dit d’apprentissage, ou d’entraînement. Dans ce mode, le logiciel est alimenté par des données étiquetées – ou *annotées* – dont il essaie d’extraire les caractéristiques principales, sous la forme de lois statistiques. Par exemple, dans le cas de la classification d’images, il reçoit un flot d’images de chiens portant l’étiquette “chien”, de chats portant l’étiquette “chat”, et de souris portant l’étiquette “souris”. Il s’efforce alors de retenir ce qui les distingue les uns des autres : taille, forme, couleur, etc. Les étiquettes lui permettent d’évaluer ses propres résultats et de se corriger lorsqu’il se trompe (par exemple, lorsqu’il identifie un chat comme un chien). 2. Un mode dit de prédiction, ou d’inférence. Dans ce mode, le logiciel utilise le modèle statistique acquis lors de l’apprentissage pour étiqueter lui-même de nouvelles données, non annotées ; vue telle caractéristique, il est probable que telle image représente un chat ; telle autre, une souris, etc. C’est ce mode que l’utilisateur final va manipuler lorsqu’il interroge une application d’IA.

Avec l’apprentissage machine, l’expertise humaine n’est plus programmée “à la main”, comme dans les systèmes experts, mais distillée automatiquement dans un modèle statistique à partir des données d’entraînement. Ces approches s’appuient donc sur la souplesse et la relative simplicité conceptuelle des méthodes statistiques... et souffrent des mêmes limites : réponses possiblement approximatives, prédictions probabilistes, et dans le cas des modèles génératifs, “hallucinations”⁷.

Ainsi, l’intelligence de l’IA n’a rien d’artificiel ; ce sont bien nos idées, nos mots, nos décisions que l’on incorpore à la machine. Sous cet angle, elle est bien une technologie de stockage de l’information, comme le livre imprimé – qui stocke l’information sous la forme de symboles alphabétiques –, la photographie – sous la forme de couleurs – ou le microsillon – sous la forme du relief d’un disque en PVC. L’IA, elle, stocke l’information sous la forme d’un modèle statistique.

5. Karl Marx, *Manuscrits de 1857-1858 dits “Grundrisse”*, Paris, Les éditions sociales, 2011, p.662

6. Alan Turing, “Computing Machinery and Intelligence”, *Mind*, vol. 59, n° 236, 1950

7. Adam Tauman Kalai et al., “Why Language Models Hallucinate”, 2025

Les données d’entraînement

La production des données d’entraînement est ainsi un enjeu décisif : l’ “intelligence” du robot réside en grande partie dans la pertinence des données qui l’alimentent. Différentes techniques existent, qui dépendent du type d’application que l’on est en train de développer. Seront présentés ici l’apprentissage supervisé, l’apprentissage non-supervisé, l’apprentissage par renforcement et l’apprentissage auto-supervisé.

Dans l’apprentissage supervisé, des opérateurs humains sont chargés d’étiqueter manuellement les données qui servent à l’entraînement de l’application. La reconnaissance d’images appartient à ce domaine : il faut bien qu’un opérateur humain indique à la machine ce qu’elle est censée reconnaître dans une image. À l’ère du *big data*, ce processus implique souvent des travailleurs précaires répartis sur l’ensemble de la planète et coordonnés au moyen de plateformes d’externalisation supranationales⁸.

Dans l’apprentissage non-supervisé, les données d’entraînement ne sont pas étiquetées. L’algorithme doit trouver des structures ou des motifs cachés dans les données par lui-même. Par exemple, imaginons un ensemble de données contenant des informations sur les habitudes d’achat des clients dans un supermarché. Un algorithme d’apprentissage non-supervisé va regrouper les clients en différents segments en fonction de leurs achats, sans avoir besoin de catégories prédéfinies. Les clients répartis de cette manière feront ensuite l’objet de campagnes de *marketing* ciblées.

Dans l’apprentissage par renforcement, les données d’entrée ne sont pas littéralement annotées, mais le logiciel interagit en direct avec son environnement, duquel il reçoit un signal de récompense. Ce signal lui permet d’évaluer ses propres choix et constitue une annotation indirecte. C’est par exemple le cas d’une application qui joue à un jeu vidéo⁹ (ou qui fait du *trading*) : l’environnement est le jeu lui-même, et le nombre de points (ou de dollars) obtenus permet au logiciel de savoir s’il s’est trompé ou non. Par définition, l’apprentissage par renforcement ne requiert pas une phase séparée de pré-traitement des données. Pour autant, le signal de récompense qui guide l’entraînement est parfois directement produit par des humains, comme lorsqu’on envoie un “j’aime” sur un réseau social.

À mi-chemin des techniques précédentes, l’apprentissage auto-supervisé consiste, pour l’application, à raffiner elle-même ses données d’entraînement. Par exemple, un logiciel qui est conçu pour essayer de deviner la fin d’une phrase peut être alimenté avec un texte complet. L’application va supprimer des mots au hasard dans le texte, puis essayer de deviner quels sont les mots manquants à l’aide de son modèle statistique. Elle n’a plus qu’à comparer son résultat avec le mot effectivement supprimé.

8. Clément Le Ludec et al., “The problem with annotation. Human labour and outsourcing between France and Madagascar”, *Big Data & Society*, vol. 10, n° 2, 2023

9. Volodymyr Mnih et al., “Human-level control through deep reinforcement learning”, *Nature*, vol. 518, 2015

Bref : qu'il s'agisse de créer les données manuellement, d'écrire des programmes informatiques pour collecter celles qui sont générées par nos activités quotidiennes – loisirs, travail, etc. – ou pour les dériver de documents existants, elles sont le fruit d'un travail social, qui est invisibilisé dans le procès de la marchandise. De là vient l'illusion qui, dans le débat public, donne l'impression que les données échappent à l'humanité, qu'elles apparaissent ou circulent sans contrôle, et même qu'elles pourraient prendre conscience d'elles-mêmes ou se soulever contre leurs utilisateurs. Marx montre que ce *fétichisme de la marchandise* est la norme de la production marchande¹⁰.

Les neurones artificiels

La supervision des données n'est qu'un aspect du problème. Les calculs utilisés pour extraire l'information sont un autre aspect. Alors même que le premier "neurone artificiel" (le perceptron de Rosenblatt¹¹) a été formalisé en 1957/1958, il a fallu attendre les années 1990 et 2000, en fait l'abondance de données et la puissance de calcul permises par l'informatique moderne, pour que la technique de l'apprentissage profond (*deep learning*) puisse prendre son envol¹².

Malgré son lexique issu des neurosciences, le procédé n'a rien d'organique. Il s'agit essentiellement d'appliquer différentes "couches" de calcul, dont chacune contient une grille d'opérateurs mathématiques que l'on désigne comme des *neurones artificiels*. Le résultat est successivement transformé par ces différentes couches, jusqu'au moment où il est livré à l'utilisateur.

Nous entrons dans le détail algorithmique de l'opération. Les paragraphes suivants ne sont pas indispensables pour comprendre la logique d'ensemble. Nous conseillons de les lire, mais celles et ceux qui sont vraiment allergiques aux notations mathématiques peuvent aller directement à la sous-partie "L'arrivée du perroquet stochastique".

Concentrons-nous sur l'exemple fondateur mentionné plus haut : le perceptron de Rosenblatt. Il consiste simplement en une fonction affine composée avec une fonction d'activation.

On note la fonction affine $ax + b$: étant donné un échantillon x en entrée du neurone et deux paramètres a et b qui sont acquis par l'apprentissage, on va obtenir le résultat voulu en pondérant x (en calculant a multiplié par x et en ajoutant b au total). C'est par cette pondération que les valeurs apprises a et b – aussi appelées les *poïds* du neurone – passent dans l'échantillon x . Voilà le mécanisme fondamental des neurones artificiels, une multiplication et une addition !

10. Karl Marx, *Le Capital*, Paris, PUF, 1993, Livre premier, pp.84-86

11. Frank Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain", *Psychological Review*, vol. 65, n° 6, 1958

12. Yann Le Cun, *Quand la machine apprend*, Paris, Odile Jacob, 2019, pp.55-57

Le résultat de cette fonction affine est ensuite transmis à une fonction d'activation qui décide si le neurone est actif ou non. Différentes fonctions d'activation existent, et c'est au programmeur d'en choisir une ; par exemple, la fonction d'activation ReLU (unité linéaire rectifiée), très courante, désactive le neurone si son résultat est inférieur à zéro.

Un tel modèle est limité, mais il est suffisant pour accomplir des tâches simples ; par exemple, apprendre à décider si un son est *acceptable* (0) ou *trop fort* (1) en fonction de son volume en décibels. Formellement, on le décrira au moyen de l'équation $x' = \text{act}(ax + b)$, où x est le volume en décibels, et x' l' "acceptabilité" du son.

L'algorithme du gradient

Notre problème, désormais, est de trouver un moyen pour que ce neurone apprenne quelque chose, c'est-à-dire pour que ses paramètres a et b évoluent en fonction des données qu'il rencontre. C'est le rôle de l'algorithme dit "du gradient". Voici comment il fonctionne.

En toute généralité, l'entraînement d'un modèle d'IA consiste à minimiser une fonction d'erreur. Une fonction d'erreur est une fonction qui, étant donné le résultat calculé par le modèle, renvoie son erreur, c'est-à-dire un nombre indiquant à quel point elle s'est trompée. Typiquement, un modèle a une erreur de zéro s'il a trouvé le bon résultat, proche de zéro s'il s'est légèrement trompé, bien plus grande que zéro s'il s'est complètement trompé. Minimiser la fonction d'erreur consiste à modifier le modèle pour que son erreur soit toujours très proche de zéro.

Dans le cas de l'apprentissage supervisé, la fonction d'erreur est donnée par la différence entre l'étiquette d'une donnée (le résultat attendu) et le résultat effectivement obtenu : plus la différence est petite, plus le modèle se comporte bien. Lors de l'apprentissage, on fournit une valeur supplémentaire au réseau : l'étiquette de x , notée l (pour label), et qui décrit la réponse souhaitée, de laquelle le modèle doit essayer de s'approcher. Une fonction d'erreur simple peut ainsi être la différence entre l'étiquette et le résultat du calcul : $e = l - y$ (en pratique, les fonctions d'erreur modernes sont plus complexes). De fait, si l et y sont proches l'une de l'autre, alors l'erreur sera proche de zéro.

À partir du résultat de cette fonction d'erreur, on essaie d'améliorer le neurone. On cherche donc de nouvelles valeurs pour a et pour b , produisant *in fine* un résultat moins éloigné du résultat souhaité. Pour ce faire, on calcule les dérivées partielles de la fonction d'erreur par rapport à a et b . En analyse, la dérivée est la fonction qui décrit les variations d'une fonction donnée (en l'occurrence, notre fonction d'erreur). L'expression peut sembler intimidante, mais il s'agit simplement d'une formule à appliquer, comme les tables de multiplication.

Notons donc ces dérivées partielles ∂a et ∂b (il y en a une pour chaque paramètre du neurone). La dérivée partielle est un objet mathématique qui a une vertu très utile dans notre cas : elle indique quelle valeur il faut soustraire à a et b pour minimiser la fonction d'erreur. On obtient ainsi des paramètres grâce auxquels le neurone aurait donné un meilleur résultat. Cela signifie que si je construis $a' = a - \Delta a$ et $b' = b - \Delta b$ et que je remplace a par a' et b par b' dans mon neurone artificiel, j'obtiens normalement une erreur plus petite. Le comportement de notre neurone artificiel a changé : il a "appris" quelque chose. On peut passer à la suite et traiter un nouvel échantillon \bar{x} .

L'ensemble des dérivées partielles d'une fonction est appelé son gradient. Le processus décrit ci-dessus est donc appelé la descente du gradient (car on descend par des soustractions, au lieu de monter par des additions).

En pratique, on introduit aussi un multiplicateur α , choisi par le programmeur, de sorte qu'on calcule $a' = a - \alpha \times \Delta a$ et $b' = b - \alpha \times \Delta b$. Ce multiplicateur – appelé "taux d'apprentissage" – permet de contrôler la vitesse à laquelle le neurone évolue (plus α est grand, plus le neurone évolue vite), et par là la capacité de l'algorithme à converger, à se stabiliser une fois que ses résultats sont suffisamment bons.

La même recette est répétée autant de fois que nécessaire pour que l'erreur moyenne soit suffisamment petite. De cette manière, le réseau apprend petit à petit à donner de bons résultats.

Cet algorithme s'appelle l'algorithme du gradient.

Les réseaux de neurones profonds

Le principe de l'algorithme du gradient est connu depuis 1847. Il appartient donc à la famille d'innovations qui, comme la machine volante à propulsion mécanique, l'ordinateur programmable ou le grand moulin hydraulique, ont longtemps végété à l'état de théories abstraites ou de prototypes, avant que les conditions matérielles de leur déploiement n'arrivent finalement à maturité. En l'occurrence, il a fallu que l'humanité dispose de grands volumes de données, et de la puissance de calcul requise pour les traiter en un temps raisonnable. Ces prérequis n'ont été remplis qu'à la fin du vingtième siècle, et ont nécessité, en retour, de raffiner l'algorithme.

Pour commencer, les réseaux de neurones du monde réel ne se limitent pas à un seul neurone. Ils sont à la fois plus *larges* et plus *profonds*. Que signifient ces expressions ?

L'algorithme que nous avons décrit jusqu'ici manipule des données très simples, qu'un seul nombre suffit à représenter, comme un volume sonore. Mais, comme nous l'avons déjà vu, d'autres types de données, plus complexes, requièrent plusieurs nombres pour être décrites ; c'est notamment le cas des textes ou

des images. Ces nombres ne sont pas fournis au logiciel en vrac, mais sous la forme d'objets structurés ; citons les *vecteurs* (de simples séquences de nombres), utilisés pour la représentation numérique des textes ; les *matrices* (des tableaux de nombres, ou autrement dit des séquences de séquences de nombres), pour la représentation numérique des images en noir et blanc ; ou les tenseurs plus généraux (des séquences de séquences de séquences... de nombres), pour la représentation numérique des données les plus riches, comme les images en couleur ou les vidéos. Pour traiter toute cette complexité, le logiciel a besoin de plusieurs neurones artificiels, chacun avec ses propres paramètres **a** et **b**.

Ces neurones sont organisés en *couches*. Plus une couche contient de neurones, plus on dit qu'elle est large ; plus un réseau contient de couches, plus on dit qu'il est profond. Il y a une fonction d'activation par couche, dont la responsabilité est d'indiquer quels neurones sont actifs ; et une seule fonction d'erreur pour tout le réseau, dont la responsabilité est d'indiquer si le résultat final est correct. Un neurone apprend à répondre à un motif simple (identifier une couleur), une couche de neurones apprend une structure (identifier une silhouette), un réseau de neurones complet apprend un concept (identifier un chat).

Appliquer une couche de neurones consiste à pondérer toutes les valeurs des données d'entrée avec les paramètres des neurones. On multiplie et/ou on additionne les uns avec les autres de manière structurée, au moyen d'opérations comme les *convolutions* et les *multiplications de matrices*. Le résultat est lui aussi, le plus souvent, multi-dimensionnel (et la fonction d'activation est appliquée à chacun de ses nombres constitutifs) : un vecteur, une matrice, ou un tenseur plus général. On le transmet alors à la couche de neurones suivante ou, s'il s'agit de la dernière couche de neurones, à la fonction d'erreur du réseau. Le réseau peut donc être vu comme une *fonction composée* de fonctions simples qui sont les différentes couches de neurones.

Problème : la méthode classique pour calculer les dérivées partielles d'une fonction composée, celle que l'on enseigne au lycée – le théorème de dérivation des fonctions composées, ou *chain rule* en anglais –, est trop coûteuse en calcul et en mémoire pour être utilisée en production. Il a donc fallu trouver une autre manière de calculer les dérivées partielles, appelée *retropropagation du gradient* et décrite par Seppo Linnainmaa dans son mémoire de *master*¹³. Cette technique consiste à descendre le gradient à travers plusieurs couches de neurones “à l'envers”, en propageant les dérivées partielles à partir de la dernière couche du réseau. Elle a été appliquée aux réseaux de neurones pour la première fois en 1986, par Rumelhart¹⁴, mais c'est seulement à la fin des années 90, avec les travaux de Yann Le Cun sur les réseaux de neurones dits *convolutifs* (basés sur

13. Seppo Linnainmaa, “The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors”, 1970

14. David E. Rumelhart et al., “Learning representations by back-propagating errors”, *Nature*, vol. 323, 1986

des convolutions, moins coûteuses en calcul que les multiplications de matrices), qu'elle est passée à l'échelle ¹⁵.

La révolution de l'apprentissage machine pouvait commencer.

L'arrivée du perroquet stochastique

Bien que l'apprentissage profond ait donné lieu à une grande diversité d'applications, une en particulier lui a permis d'atteindre directement le grand public : il s'agit des robots conversationnels (*chatbots*) alimentés par de grands modèles de langage (*Large Language Models*, LLMs). Ces derniers sont des réseaux de neurones consommant du texte et générant du texte en retour (c'est pourquoi on les dit *génératifs*). L'aller-retour entre le texte fourni par l'utilisateur et le texte créé par la machine donne à l'ensemble la forme d'une *conversation*.

On qualifie parfois ces logiciels de “perroquets stochastiques” ¹⁶, pour souligner leur caractère machinique, non-humain. Par définition, les LLMs font abstraction de nombreuses dimensions de la vie sociale, qu'on ne sait pas vraiment représenter sous forme textuelle et qui passent par le son, le geste, l'espace, la durée, les affects, les habitudes, et bien d'autres phénomènes encore. Bien conscient de ces limites, Yann Le Cun propose que la recherche se tourne vers la conception de “modèles du monde” permettant aux logiciels de reproduire la compréhension multi-dimensionnelle que l'on trouve chez les humains ¹⁷.

Il n'en reste pas moins vrai que, depuis la publication de ChatGPT en novembre 2022, les LLMs ont fait une entrée fracassante sur la scène de l'histoire. Pourquoi ? Parce que pour la première fois, il devient possible d'interagir avec l'ordinateur et d'explorer de vastes jeux de données sans compétences spécialisées, simplement en s'exprimant dans sa langue maternelle.

Le texte écrit est devenu, au fil des millénaires, une interface de communication universelle – ou presque – entre les êtres humains. La grande majorité des membres de notre espèce apprend à lire et à écrire dès l'enfance. Le cas échéant, la plupart des locuteurs parlent des langues qui ont été transcrites, donc que l'on sait ramener (y compris automatiquement) à du texte. Dans un pays comme la France, le texte écrit joue un rôle tellement central – échanges interpersonnels, paperasse administrative, listes de courses, éducation, journalisme, roman, code informatique, science, politique – qu'il est aujourd'hui une “seconde peau” du langage. Ce fait a permis l'adoption massive des LLMs et leur pénétration rapide dans la vie sociale.

15. Yann Le Cun et al., “Gradient-Based Learning Applied to Document Recognition”, *Proceedings of the IEEE*, vol. 86, n° 11, 1998

16. Emily M. Bender et al., “On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?”, *Proceedings of the 2021 ACM Conference on Fairness, Accountability and Transparency*, 2021, pp.610-623

17. Yann Le Cun, “A Path Towards Autonomous Machine Intelligence”, 2022

Pour les mêmes raisons, les “nappes phréatiques” de l’IA sont pleines, regorgent de données de qualité. De fait, l’humanité a déjà cristallisé une partie significative de ses pratiques sociales et de ses savoirs, depuis des millénaires. Elle l’a fait, et continue de le faire sous la forme de texte : textes religieux, philosophie, supports d’éducation, littérature, archives juridiques ou parlementaires, etc.

L’expropriation de ces volumes gigantesques de texte par les multinationales du domaine est en même temps une *socialisation* d’ampleur inédite. Le processus bouleverse les principes même de la propriété privée (ici, intellectuelle) et suscite les réactions furieuses des capitalistes du livre, de l’industrie cinématographique ou musicale, de l’édition scientifique. Cette contradiction entre le caractère social des forces productives et le caractère privé de l’échange est accentuée par les acteurs principaux du domaine qui, par calcul commercial, diffusent presque tous des modèles gratuits et partiellement *open-source* (dits *open-weight*).

On est ainsi tenté de paraphraser la démonstration du manifeste communiste : la dynamique même du capitalisme vient saper les fondations de la domination bourgeoise¹⁸.

Sommes-nous arrivés au bout de l’innovation permise par ces modèles ? Rien n’est moins sûr. De fait, l’interface textuelle facilite non seulement l’interaction des LLMs avec les opérateurs humains et avec les données qu’ils produisent, mais aussi avec d’autres logiciels. Ce fait est à la base du paradigme *Reason and Act*¹⁹, qui sert de cadre à l’IA dite *agentique*. Dans cette approche, un LLM ou un système de LLMs – vus comme des *agents* – interagissent de manière autonome entre eux et avec d’autres logiciels : boîte mail, service commercial, gestionnaire de code source, etc.

Attention is all you need

Les LLMs qui ont percé reposent sur une architecture très simple, dite Transformer²⁰ (GPT est l’acronyme de *Generative Pre-trained Transformer*, ou *Transformer* Génératif Pré-entraîné en français). Cette dernière se caractérise par un mécanisme dit d’*attention*, qui permet à l’application d’apprendre à différencier le sens d’un mot suivant son contexte. Par exemple, le mot “mine” ne signifie pas la même chose si on parle d’une *mine de crayon*, d’une *triste mine* ou d’une *mine de charbon*.

Le logiciel utilise ce contexte pour essayer de deviner – statistiquement – quel est le mot qui vient ensuite. Typiquement, si le contexte contient le membre de phrase “Le chat dort sur le”, il est probable que les mots “canapé”, “lit” ou “tapis” viennent ensuite.

18. Karl Marx, *Le manifeste du parti communiste*, Pékin, Éditions en langues étrangères, 1970, pp.39-41

19. Shunyu Yao et al., “ReAct: Synergizing Reasoning and Acting in Language Models”, 2022

20. Ashish Vaswani et al., “Attention Is All You Need”, *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp.6000-6010

Le nouveau mot est ajouté au contexte, puis on utilise la même méthode pour calculer le suivant ; et encore le suivant ; et ainsi de suite. Il suffit de répéter l'opération plusieurs fois, jusqu'à former des phrases, des paragraphes, des textes entiers.

En pratique, l'entraînement se déroule en deux temps.

D'abord, les modèles ingèrent de manière auto-supervisée de vastes segments du web²¹ : articles d'encyclopédie sur Wikipedia, dépôts de code source sur GitHub, thèses de doctorat et articles de recherche sur les portails académiques, romans, journaux, etc. Cette première phase est appelée le *pre-training*, le pré-entraînement. Les modèles résultants sont dits "fondationnels"²², parce qu'ils servent de "fondation" à différents produits : entraînés sur de larges volumes de données, ils peuvent être reconfigurés pour servir à une grande variété de tâches.

Cette approche s'appuie sur l'hypothèse dite de "passage à l'échelle" (*scaling hypothesis*), d'après laquelle des modèles mathématiques simples suffisent pour réaliser des fonctions de plus en plus complexes, pour peu qu'ils soient suffisamment grands, et qu'ils soient alimentés par des données de qualité. Des études récentes ont permis de mieux comprendre ce mécanisme. Au départ, les données ingérées par le modèle lui permettent d'apprendre la langue. Au bout d'un certain volume de données, il atteint un plateau, durant lequel la qualité de ses réponses ne progresse plus. Si on continue à l'approvisionner de données, il finit par sortir de ce plateau et par acquérir des connaissances structurées²³ ; c'est le fameux "passage à l'échelle", qui fait écho à la transformation de la quantité en qualité dans la dialectique hégélienne²⁴.

À ce stade, le modèle contient beaucoup de connaissances, mais il manque des finitions requises pour être distribué au grand public. Il n'est pas difficile, par exemple, d'obtenir qu'il fournisse la recette d'une bombe artisanale ou qu'il écrive un manifeste néonazi.

Une seconde phase est donc prévue, que l'on appelle *l'alignement*. Il s'agit d'*aligner* le modèle avec les intentions des humains qui l'utilisent²⁵, de restreindre certains usages et d'en raffiner d'autres. Le logiciel met ainsi ses réponses en cohérence avec l'idéologie de ses concepteurs. Cette phase d'alignement associe généralement des méthodes d'apprentissage supervisé et des méthodes d'apprentissage par renforcement.

L'ensemble du processus est très coûteux en calcul, et donc aussi en ressources. Il utilise des processeurs spécialisés (GPU, TPU) que l'on regroupe dans des fermes de calcul consommatrices de grandes quantités d'électricité²⁶.

21. Aaron Grattafiori et al., "The Llama3 Herd of Models", 2024

22. Rishi Bommasani et al., "On the Opportunities and Risks of Foundation Models", *Center for Research on Foundation Model*, 2022

23. Nicolas Zucchet et al., "How do language models learn fact ? Dynamics, curricula and hallucinations", 2025

24. Friedrich Hegel, *La Science de la Logique*, Paris, Aubier Montaigne, 1972, t.1, pp.334-342

25. Tong Xiao et al., "Foundations of Large Language Models", 2025

26. David Patterson et al., "Carbon Emissions and Large Neural Network Training", 2021

Conclusion

Sous ses dehors magiques, l'IA n'ajoute pas un sujet à l'histoire. Les modèles ne pensent pas : ils compriment, corrélient et restituent des régularités extraites de nos savoirs collectifs, ce qui explique à la fois leurs forces et leurs limites. Le fétichisme consiste à prendre cet amas statistique pour une âme.

Au plan théorique, la leçon est simple : avec ces logiciels, le capital constant incorpore toujours plus de science, mais cette science reste une force productive aliénée tant qu'elle n'est pas appropriée par les producteurs associés. L'IA, en ce sens, n'est pas l'horizon de la pensée. Elle est l'un des terrains où se joue la lutte pour l'usage social du patrimoine intellectuel de l'humanité.

Ces techniques sont aujourd'hui mises au service d'un discours idéologique néolibéral, volontiers transhumaniste et réactionnaire. Mais elles réduisent aussi significativement le coût d'accès aux savoirs collectifs de l'humanité – une ressource précieuse pour les peuples, y compris dans la lutte des classes²⁷. Bien des usages des *chatbots* vont d'ores et déjà dans ce sens, que ce soit dans le domaine de l'éducation et de l'auto-formation, des langues, de la documentation technique, de l'aide à la création.

Sous cet angle, la dénonciation du nivellement supposément engendré par l'IA rappelle celle, de nature aristocratique, que Platon prête à Socrate au sujet de la diffusion des textes écrits : “Une fois écrit, un discours roule de tous côtés, dans les mains de ceux qui le comprennent comme de ceux pour qui il n'est pas fait, et il ne sait pas même à qui il doit parler, avec qui il doit se taire”²⁸. Est-il vraiment si grave que les mots et les idées roulent de tous côtés ?

27. Hugo Pompougnac, “De l'intelligence artificielle, de Lénine et des cuisinières”, *Silomag*, 2023

28. Platon, *Œuvres de Platon*, P.-J. Rey Libraire-Éditeur, 1849, t.6, p.124