Les transformeurs et l'IA générative, sous le capot de ChatGPT

Culture Sciences de l'Ingénieur

Antoine CORDUANT

Édité le 21/09/2026

école normale supérieure paris-saclay

Cette ressource est issue d'un travail personnel d'Antoine Corduant, élève en quatrième année à l'ENS Paris-Saclay. Ce texte a été relu par Laurent Oudre, enseignant chercheur au Centre Borelli de l'ENS Paris-Saclay.

1 - Introduction à l'IA et démystification

1.1 - Qu'est-ce que l'IA ?

L'Intelligence Artificielle (IA) désigne à l'origine des machines ou des programmes cherchant à reproduire certains aspects de l'intelligence humaine, comme apprendre, résoudre des problèmes, prendre des décisions, créer ou innover. Aujourd'hui, on utilise le terme d'IA pour décrire des systèmes qui semblent agir avec intelligence, même si leur fonctionnement et leurs objectifs peuvent être très différents de celui du cerveau humain. Ainsi, certaines IA sont capables de nos jours d'obtenir des résultats que les humains ne pourraient en aucun cas trouver, comme analyser d'énormes quantités de données en quelques secondes ou résoudre des problèmes d'une complexité vertigineuse.

Cette définition très large explique pourquoi le terme IA peut être appliqué aujourd'hui à un si grand nombre de systèmes différents. Dans l'antiquité, des automates auraient pu être appelés IA, tout autant que les premiers **chatbot** dans les années 60-70, même s'ils feraient pâle figure comparés aux chatGPT (GPT pour Generative Pre-trained Transformer, sujet de la suite de ce document) et autres. Aujourd'hui le terme IA est souvent lié à une machine qui apprend par elle-même, même si cela n'est pas nécessaire : l'apprentissage automatique est arrivé bien après les premières IA. Il est incontestable que DeepBlue, l'ordinateur ayant battu Kasparov aux échecs en 1997, est une forme d'IA. Pourtant toute sa logique est humaine, sa force réside dans sa puissance de calcul. En 2016, le modèle AlphaGo devient indéniablement meilleur que l'homme au jeu de go. Cette fois-ci, il a appris à jouer par lui-même, sans que sa logique soit basée sur notre manière de comprendre le jeu. C'est aussi une IA, mais qui a utilisé le **machine learning** (apprentissage automatique, une méthode permettant à un modèle d'arriver à un résultat sans que sa logique soit dictée par un humain) pour arriver à cet exploit.

Dans la suite de cet article, le terme « modèle » va être utilisé pour décrire ces IA basées sur un programme informatique. Un modèle est donc un programme qui va produire des résultats complexes, même intelligents selon certains points de vue. Une IA peut être un assemblage de modèles.

1.2 - Arbres de décisions

Il peut être défendu que les IA les moins complexes sont celles conçues de A à Z par un opérateur humain (des IA symboliques). Elles sont souvent composées d'arbre de décision (voir figure 1) : si telle chose arrive, alors telle action est exécutée et ainsi de suite. Cette suite de questions/réponses crée une sorte d'arbre, avec à la racine l'état initial du problème. En parcourant les branches on arrive à avancer dans la situation, pour trouver la décision finale dans

les feuilles (le bout des branches). Ces modèles malgré leur apparente simplicité permettent d'arriver à des résultats largement suffisant pour leur domaine d'application, que ce soit en médecine[1], pour mettre en place des procédures d'urgence[2] ou pour gérer des investissements boursier[3]. Dans les deux premiers exemples, les arbres de décisions ne sont pas conçus pour être exploités par une machine, mais bien par des humains. Ces arbres permettent à l'humain de prendre des décisions, sans avoir besoin de consulter une bibliographie ou de prendre trop de temps à se décider. Dans le cas d'investissement boursier, on retrouve bien des modèles avec des arbres entièrement rédigés par la main de l'homme mais exploités par des machines. En effet aujourd'hui, tout le trafic boursier est géré par des ordinateurs. Même si les applications présentées sont un peu loin d'un comportement intelligent, ces divers exemples rappellent à quel point un modèle peut être simple. L'homme n'a pas attendu la révolution du **machine learning** des années 2010 pour mettre en place des modèles qui prennent des décisions avec l'humain, ou même à sa place.

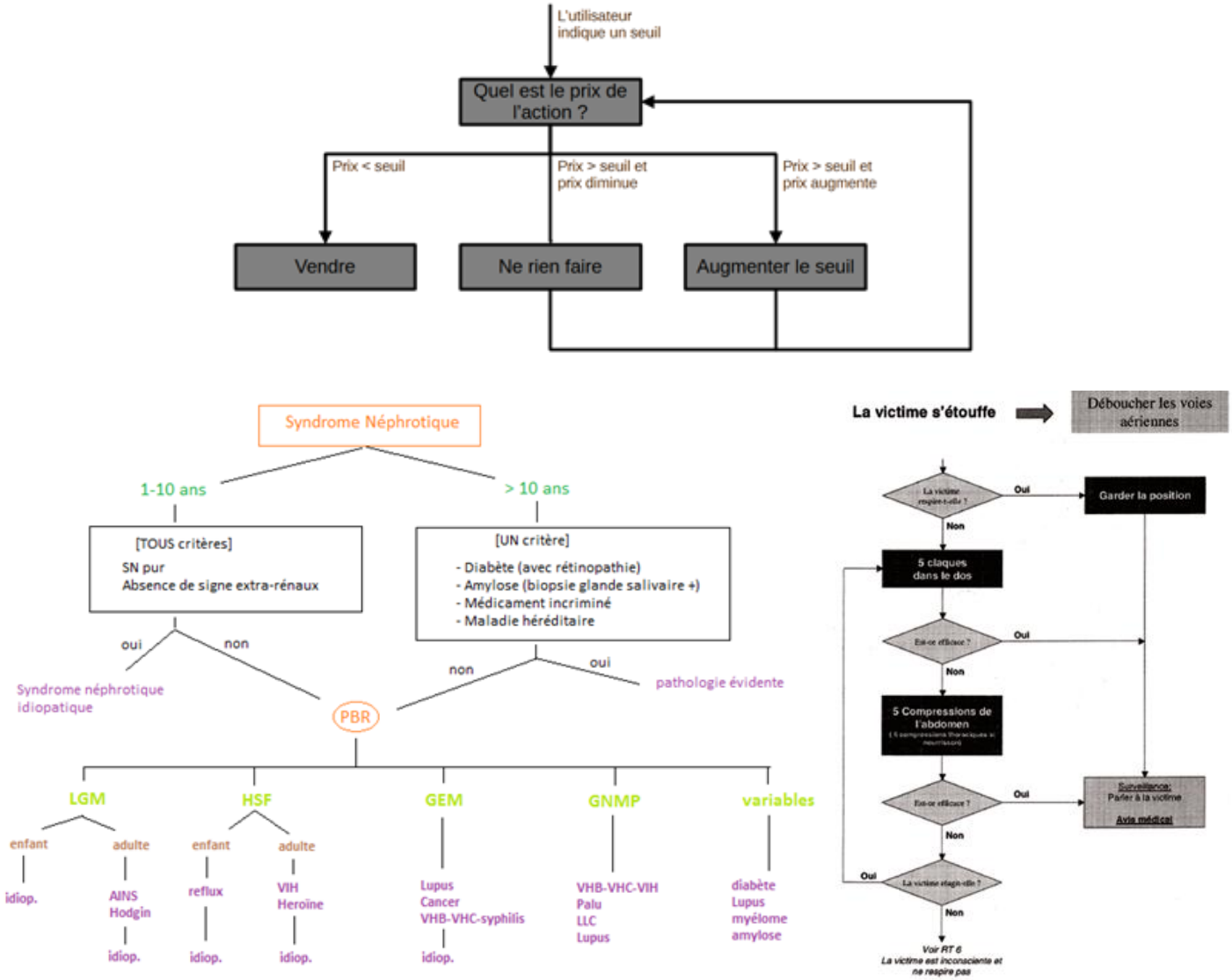


Figure 1 : Exemple d'arbre de décision sans intervention d'une machine, sources medg.fr et urgences-serveur.fr

Avec l'avènement de l'informatique dans les années 1970-1980, des chercheurs se sont vite aperçus que le processus de mise en place d'arbre de décisions pouvait être automatisé : c'est, en partie,

[1] https://www.medg.fr/informations/arbres-decisionnels-medicaux/
[2] https://www.urgences-serveur.fr/formation-aux-premiers-secours-858.html
[3] https://admiralmarkets.com/fr/formation/articles/base-du-forex/stop-loss-trading

le début du **machine learning**. Ainsi, dans cette période, des méthodes[4,5] ont été développées dans le but de créer automatiquement des arbres de décisions répondant à des problèmes pouvant être très complexes. Il suffit, entre autres, d'avoir un ensemble de données d'entraînement suffisamment exhaustif pour pouvoir mettre en place un arbre de décision très performant.

Par exemple, un réseau social pourrait avoir intérêt à savoir si une personne (M. X) aime les vidéos de chat avec pour but de lui recommander plus ou moins ce type de vidéos. Pour ce faire, le réseau social va utiliser l'immense base de données qu'il a mis en place au fil des années sur les utilisateurs de sa plateforme (des données d'entraînement) pour deviner la réaction qu'aura M. X lorsqu'une vidéo de chat lui sera présenté. À partir de la base de données, une méthode de création d'arbre de décision va utiliser des caractéristiques (**features**) connues des utilisateurs (âge, sexe, milieu social, zone d'habitation, etc.) pour construire un arbre. À chaque branche de cet arbre, les caractéristiques sont comparées à des valeurs (**paramètres**) trouvées automatiquement par la machine (d'où l'apprentissage automatique), et après avoir parcouru un chemin dans les branches de l'arbre (succession de comparaisons avec les paramètres) on arrive sur une feuille qui indique si oui ou non la personne étudiée aime les vidéos de chat.

Pour trouver les bons paramètres, la machine va piocher un utilisateur dans la base de données et observer la réponse donnée par l'arbre. Comme l'utilisateur est dans la base de données, sa réponse est connue à l'avance. Ainsi l'arbre va pouvoir être ajusté pour donner la même réponse. Si la prédiction est correcte rien ne se passe (l'arbre effectue bien son travail), mais si elle est mauvaise, les paramètres vont être modifiés pour avoir un arbre qui ne se trompe pas. Cette action est répétée sur un nombre suffisant d'utilisateurs d'entraînement. In fine, l'arbre a appris ce qui, dans les caractéristiques d'un utilisateur, permet de déterminer si oui ou non il aime les vidéos de chat. Par exemple : âge < 25, sexe masculin, classe moyenne supérieure, etc. Ainsi, un arbre pouvant prédire si M. X aime les vidéos de chat est créé, même si cette personne n'est pas dans la base de données d'entraînement. On dit que cet arbre de décision effectue une tâche de **classification** (c'est-à-dire mettre dans des cases) et qu'il a **généralisé** la réponse pour M. X à partir de la base de données.

Pour résumer, il est possible à partir de simples arbres de décisions de créer des modèles qui apprennent par eux-mêmes (**machine learning**) et capables de résoudre des tâches de classifications complexes[6], parfois même plus complexes que ce qui serait réalisable par un humain. À partir de cet exemple de départ, on comprend aussi la clé de voûte des IA actuelles : en général, plus on a de paramètres (et de **features**) plus le modèle est efficace. En effet, un arbre de décisions avec plus de comparaisons et plus de valeurs à comparer fera intuitivement un meilleur travail qu'un petit arbre. Cette idée s'applique aussi, et surtout, aux modèles s'appuyant sur des réseaux de neurones, fondation des IA génératives.

1.3 - Réseaux de neurones

a) Historique

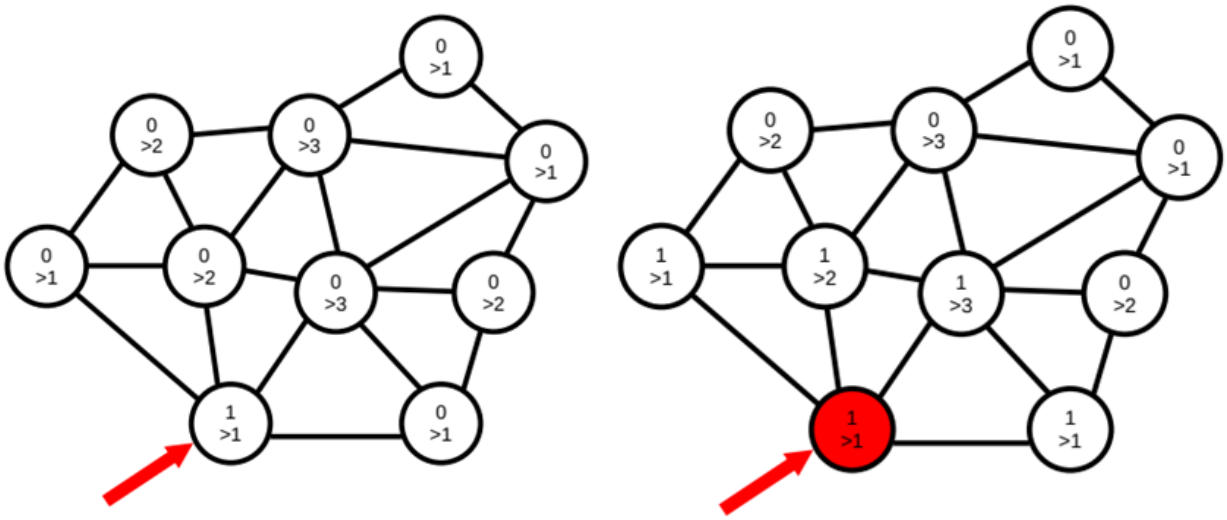
Le concept de réseau de neurones a été développé dans les années 1940, par McCulloch et Pitts[7]. L'idée originale était de modéliser le fonctionnement d'un cerveau : un neurone a un niveau d'excitation (de complètement éteint à très allumé, comme une lampe avec un variateur) et il est

[4] https://fr.wikipedia.org/wiki/CHAID
[5] https://fr.wikipedia.org/wiki/Algorithme_ID3
[6] Costa, V.G., Pedreira, C.E. Recent advances in decision trees: an updated survey. *Artif Intell Rev* **56**, 4765–4800 (2023). https://doi.org/10.1007/s10462-022-10275-5
[7] McCulloch, W.S., Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* **5**, 115–133 (1943). https://doi.org/10.1007/BF02478259

relié par des synapses à d'autres neurones. Les réseaux de neurones ont été rendus utilisables par Hopfield et Hinton dans les années 1980 en développant des méthodes pour permettre un entraînement automatique de ces réseaux[8,9]. Ces deux chercheurs ont reçu le prix Nobel de physique en 2024 pour leurs travaux sur ce sujet. C'est cette capacité d'apprendre à un réseau à donner des sorties très complexes pour des entrées tout autant complexes qui a révolutionné notre vision de l'IA. Par exemple, rien n'empêche en théorie d'avoir un réseau qui associe à une photo quelconque, la date et le lieu de la prise de vue. Ce comportement est extrêmement complexe, mais tant qu'il existe assez d'informations dans l'image, un réseau de neurone suffisamment grand et bien entraîné arrivera à réaliser cette tâche. Cela est globalement vrai d'après un théorème (théorème d'approximation universelle[10]) indiquant que n'importe quelle tâche peut être réalisée par un réseau de neurones, s'il existe assez d'informations pour trouver la réponse.

b) Fonctionnement

Le caractère intelligent émerge par le fait qu'un neurone excité peut envoyer son excitation à tous ses voisins, suivant une règle très simple : un neurone envoie une excitation lorsque la somme des excitations qu'il reçoit est supérieure à une valeur (un **paramètre**). Ce comportement est présenté sur les schémas suivants. En rouge les neurones excités, le premier nombre est la somme des excitations qu'il reçoit, le nombre en dessous est le paramètre. On constate sur cet exemple qu'une sorte de logique peut apparaître à partir de règles très simples. Par exemple, avec un si petit nombre de neurones et en choisissant les bons paramètres, on pourrait arriver à créer des machines faisant des additions. C'est l'idée de base des réseaux de neurones : utiliser une logique simple à très grande échelle pour faire émerger des résultats très complexes.



[8] Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. Nature 323, 533–536 (1986). https://doi.org/10.1038/323533a0

[9] J.J. Hopfield, Neurons with graded response have collective computational properties like those of two-state neurons., Proc. Natl. Acad. Sci. U.S.A. 81 (10) 3088-3092, https://doi.org/10.1073/pnas.81.10.3088 (1984)

[10] Cybenko, G. Approximation by superpositions of a sigmoidal function. Math. Control Signal Systems 2, 303–314 (1989). https://doi.org/10.1007/BF02551274

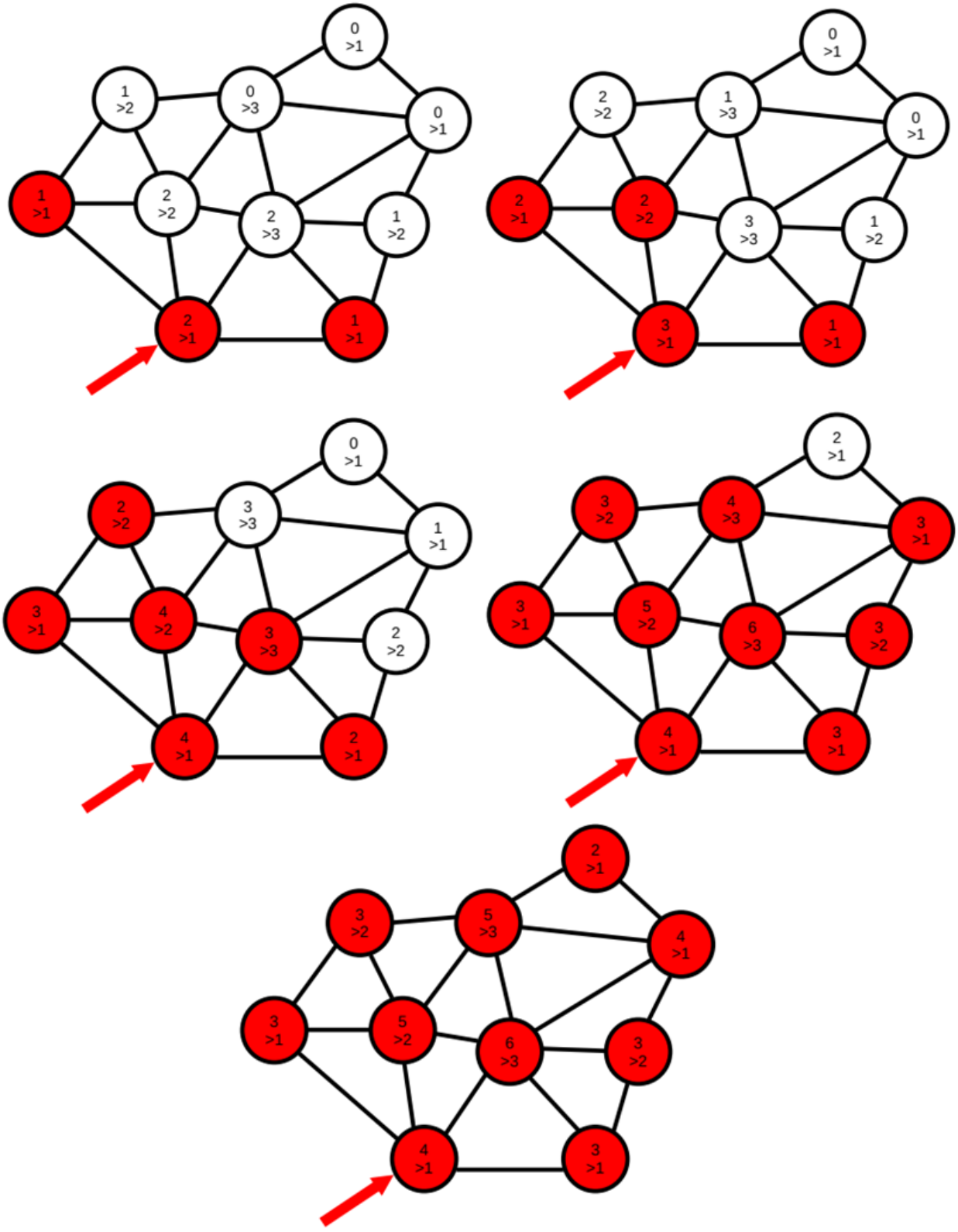


Figure 2 : Évolution pas à pas des variables d'un réseau de neurones simplifié

Les réseaux de neurones utilisés dans la majorité des IA actuelles se basent sur la structure présentée précédemment avec pour seules différences l'organisation des neurones et le fait que leur valeur d'excitation n'est pas obligatoirement un nombre entier, tout comme le seuil d'excitation. Le schéma de la figure 3 représente un réseau de neurones classique, dit **fully connected** car tous les neurones d'une couche sont connectés à tous ceux de la couche précédente et suivante. Malgré sa simplicité, cette architecture permet d'obtenir des résultats très complexes. C'est pourquoi les réseaux de neurone sont souvent la colonne vertébrale des modèles d'IA de pointe (voir partie 4.1.b).

L'exemple schématisé figure 3 dispose de 3 entrées, 2 couches cachées de 4 et 3 neurones respectivement, et 2 sorties. Ces dimensions peuvent être modifiées à volonté en fonction du problème à résoudre. Ce réseau de neurones comprend ainsi 3 entrées pour lesquelles l'utilisateur doit fournir 3 valeurs, par exemple la couleur d'un pixel en proportion de rouge, de vert et de bleu. À partir de ces entrées, le réseau est capable de donner 2 valeurs en sortie. Ces sorties dépendront des entrées et des paramètres des neurones : seuil d'activation, mais aussi un coefficient multiplicateur sur chaque synapse sortante, qui accentue l'excitation du neurone lors de la transmission d'information. Les paramètres sont choisis lors de la phase d'entraînement du réseau de neurones.

On appelle les deux couches centrales « cachées » car elles ne sont pas en interaction avec l'extérieur, se situant entre l'information brute qu'on donne à l'IA (comme la photo d'un animal) et la réponse finale (comme « c'est un chat »). Ces couches sont juste là pour ajouter des connexions et des paramètres au réseau, ainsi augmenter sa complexité et donc sa capacité de produire des résultats complexes. Même si on peut interpréter ce qu'il se passe dans ces couches, leur taille et leur nombre rendent une analyse globale impossible. Il est donc très difficile de suivre les évènements dans ces niveaux, c'est la fameuse « boîte noire » inhérente aux IA basées sur des réseaux de neurone. Ainsi, on sait que ça marche mais on ne sait pas vraiment pourquoi. En effet, dans le cas de modèle traitant des images par exemple, les toutes premières couches sont plutôt bien comprises : elles fonctionnent un peu comme des filtres photo capables de repérer des éléments basiques tels que des lignes, des contours ou des textures. Le véritable effet « boîte noire » apparaît au fur et à mesure que l'on s'enfonce dans les profondeurs du réseau, car à chaque niveau, le modèle applique des transformations mathématiques qui déforment et croisent les informations précédentes. Cette réaction en chaîne, répétée sur des millions de connexions, crée une complexité telle que, même si chaque calcul individuel reste parfaitement logique, l'accumulation de ces étapes rend le résultat final tellement entremêlé qu'il devient impossible pour un humain de retracer précisément le cheminement d'une décision au cœur du réseau. Evidemment cette suite d'évènement est logique pour le modèle, car c'est au travers de l'entraînement que ses paramètres se sont modifiés pour arriver à cette combinaison, ce n'est donc pas le fruit du hasard. Néanmoins, malgré toute cette logique, le pourquoi nous échappe quand même, tout ce qu'on peut comprendre ce sont les sorties, car le réseau a été entraîné à donner des sorties compréhensibles. C'est pourquoi ces couches sont « cachées ».

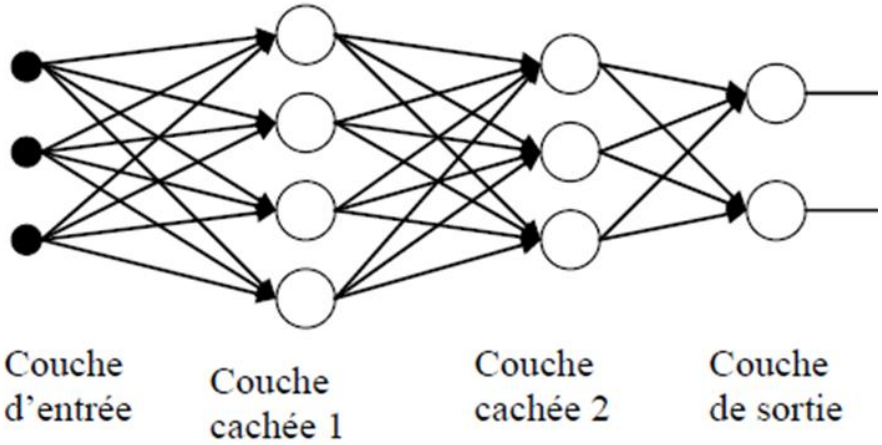


*Figure 3 : Réseau de neurones **fully connected** à 2 couches cachées, 3 entrées et 2 sorties. Chahmi, Abdelghani (2017), Identification paramétrique de la machine asynchrone dédiée au diagnostic*

1.4 - Utilisations

Contrairement à une idée largement répandue, l'intelligence artificielle ne s'est pas imposée récemment dans notre quotidien avec l'apparition de ChatGPT. En réalité, elle est utilisée depuis

de nombreuses années dans une multitude de domaines, souvent de manière discrète mais omniprésente. Les géants de la technologie n'ont pas attendu la révolution des modèles conversationnels pour exploiter ces avancées : les algorithmes de recommandations en sont un parfait exemple. Chaque fois que nous parcourons notre fil d'actualité sur des plateformes comme Facebook, Instagram, YouTube ou TikTok, ou même lorsque nous lisons des articles suggérés sur Google News, des modèles analysent nos préférences et comportements pour nous proposer du contenu jugé pertinent.

L'influence de l'IA ne s'arrête pas là. À chaque requête effectuée sur un moteur de recherche, comme Google ou Bing, un modèle se charge d'identifier les sites les plus pertinents afin de fournir une réponse optimisée aux besoins de l'utilisateur. Ce processus, bien que rapide et invisible, repose sur des calculs complexes exploitant des milliards de données en temps réel.

Même en dehors des usages liés au web, l'IA s'intègre de manière croissante dans notre vie quotidienne. Par exemple, lorsque nous déverrouillons notre téléphone grâce à la reconnaissance faciale ou nos empreintes digitales, un modèle compare instantanément nos caractéristiques biométriques pour authentifier notre identité. De même, les objets connectés, tels que les montres intelligentes, exploitent ces technologies pour mesurer notre rythme cardiaque, analyser les différentes phases de notre sommeil ou encore évaluer notre niveau de stress.

Enfin, dans le domaine des transports, l'intelligence artificielle joue un rôle clé dans le développement des véhicules autonomes. Depuis plus de quinze ans[11], des prototypes de voitures autonomes fonctionnelles circulent et évoluent grâce à des algorithmes sophistiqués capables de percevoir leur environnement, de prendre des décisions en temps réel et d'optimiser leur trajectoire en fonction des conditions de circulation. Ces avancées témoignent du fait que l'IA est loin d'être une innovation récente ; elle est, au contraire, un pilier technologique qui façonne depuis longtemps notre manière d'interagir avec le monde numérique et physique.

1.5 - IA générative

Le concept d'intelligence artificielle générative regroupe les modèles capables de créer du contenu inédit, qu'il s'agisse de textes, d'images, de musiques ou même de séquences vidéo. Plutôt que reconnaître une image ou classer des données comme des modèles plus classiques, ces algorithmes fabriquent de nouveaux éléments qui n'existaient pas auparavant, en imitant les styles, les structures et les rythmes qu'ils ont observés lors de leur entraînement.

Par exemple, un modèle de génération de texte peut prendre comme point de départ une proposition d'un utilisateur et rédiger un conte ou un article. En musique, un système entraîné sur un corpus de morceaux peut proposer de nouvelles mélodies en respectant le style original et les règles de base de la musique. Dans le domaine de l'image, un modèle dédié peut apprendre à créer une œuvre parfaitement originale, basée sur un artiste existant ou non, tout autant qu'apporter de très légères modifications à un visuel déjà existant.

Ces capacités ouvrent de nombreuses possibilités. Des artistes collaborent avec des modèles génératifs pour explorer des univers graphiques inédits ou détourner des créations existantes. Des professionnels de tout domaine sollicitent ces outils pour générer des propositions d'idées ou de textes pour étendre sa vision sur un sujet ou simplement pour gagner du temps. Dans les industries du divertissement, la génération automatique de visuels ou de dialogues permet d'accélérer la production et, dans le meilleur des cas, de libérer du temps pour la créativité humaine ou, dans le

[11] https://googleblog.blogspot.com/2010/10/what-were-driving-at.html

pire, se passer de l'humain pour réduire les coûts. Même les développeurs de logiciels utilisent aujourd'hui massivement des assistants génératifs pour suggérer du code ou expliquer des bouts de logique, facilitant la mise en œuvre de fonctionnalités. Les IA génératives sont aujourd'hui tellement performantes au niveau de l'écriture de code informatique qu'une nouvelle pratique s'est créée : le **vibe coding** (programmation au ressenti). Elle consiste à créer des programmes sans toucher à du code, en donnant les objectifs attendus à une IA qui aura la tâche de la rédaction. Cela permet de donner accès à la programmation à tous ceux qui n'ont pas les connaissances nécessaires.

Pour obtenir des résultats de qualité, il est essentiel de guider l'IA : on affine son comportement à l'aide de jeux de données ciblés (le **fine tuning**) et on soigne les consignes (les **prompts**) qu'on lui fournit, afin d'éviter les réponses hors sujet ou porteuses de biais. Ces étapes de réglage garantissent que l'IA générative conserve à la fois sa créativité et sa fiabilité vis-à-vis des objectifs qu'on lui assigne. Néanmoins, l'importance de celles-ci est trop souvent négligée par l'utilisateur, car un mauvais **prompt** peut mener un très bon modèle à donner de très mauvais résultats et un **fine tuning** biaisé peut tromper et mener dans des situations intolérables (sexisme, racisme, etc.). Une bonne connaissance du modèle est donc nécessaire pour n'importe quelle utilisation un tant soit peu sensible.

2 - Méthodes d'apprentissage

Il existe de nombreuses méthodes pour faire apprendre quelque chose à un modèle. Ces méthodes sont centrales car c'est ainsi qu'une IA va résoudre des problèmes ou réaliser des tâches sans intervention humaine. Contrairement à un programme classique qui est construit pour une application, un modèle est basé sur une structure générale (arbre de décision, réseau de neurone, etc.) qui va s'adapter à l'application via l'apprentissage. Les trois méthodes d'apprentissage présentées par la suite sont toutes des applications de **l'apprentissage automatique** et même si cette liste n'est pas exhaustive, elle présente les approches les plus courantes et permet de bien se représenter le monde de l'**apprentissage**.

2.1 - Supervisé

L'apprentissage supervisé est l'une des approches les plus courantes et les plus efficaces pour entraîner un modèle. Il repose sur un principe fondamental qui s'apparente à l'apprentissage humain : apprendre à partir d'exemples annotés. De la même manière qu'un professeur guide un élève en lui fournissant des exercices corrigés pour qu'il comprenne un concept et puisse l'appliquer à d'autres situations similaires, un modèle est exposé à un ensemble de données soigneusement préparées contenant à la fois une entrée et la sortie attendue. L'objectif est de permettre à l'algorithme de repérer des motifs et des relations entre les données afin qu'il puisse, à terme, **généraliser** ces connaissances et effectuer des **prédictions** sur de nouvelles données qu'il n'a encore jamais rencontrées.

a) Fonctionnement

La première étape dans ce processus consiste à constituer un **jeu de données d'entraînement**, c'est-à-dire un ensemble d'exemples où chaque donnée d'entrée est associée à une réponse correcte. Dans le cas d'un modèle conçu pour reconnaître des animaux sur des images, par exemple, il est nécessaire de rassembler un grand nombre de photographies où chaque image est annotée avec l'espèce qu'elle représente, comme « chien » ou « chat ». La richesse et la diversité du jeu de données sont déterminantes pour garantir l'efficacité du modèle, car un ensemble de données insuffisamment varié risquerait de limiter sa capacité à généraliser correctement.

Une fois ces données prêtes, le modèle est soumis à une phase d'entraînement au cours de laquelle il analyse chaque exemple, tente de prédire la réponse correcte et compare son résultat avec la véritable étiquette associée. Lors des premières itérations, les erreurs sont nombreuses, mais une formule mathématique appelée **fonction de coût** permet d'évaluer précisément l'écart entre la prédiction effectuée et la valeur correcte attendue. Grâce à un **algorithme d'optimisation** tel que la descente de gradient, le modèle ajuste progressivement ses paramètres en cherchant à minimiser ses erreurs. Ce processus est comparable à la manière d'ajuster la fréquence d'une radio pour capter la bonne station : on tourne la mollette de réglage, la sortie change et en fonction du changement on adapte notre réglage. Un modèle aura des millions, voire des milliards, de mollettes à ajuster pour donner la bonne sortie. Cette action, répétée des milliers ou des millions de fois selon la complexité du problème, permet à l'intelligence artificielle d'améliorer ses prédictions et d'affiner sa compréhension des relations sous-jacentes entre les données.

Toutefois, un bon modèle ne doit pas uniquement être performant sur les exemples qu'il a déjà vus, il doit également être capable de généraliser ses connaissances à de nouvelles données. C'est pourquoi une partie du jeu de données initial est généralement réservée à une phase de **validation** qui permet de détecter d'éventuels problèmes de **sur-apprentissage**. Ce phénomène, appelé aussi **overfitting**, survient lorsque le modèle a tellement bien mémorisé les données d'entraînement qu'il n'est plus capable de s'adapter à des cas nouveaux. Pour corriger ce type de défaut, le développeur mettant en place le modèle peut jouer sur des caractéristiques de l'apprentissage comme la vitesse de changement des paramètres ou la sensibilité aux exemples pris un à un plutôt qu'à une moyenne. Une fois ces ajustements effectués, une dernière évaluation est réalisée sur un **jeu de test** totalement indépendant afin de mesurer avec précision la capacité du modèle à effectuer des prédictions fiables sur des données inédites.

b) Applications

L'apprentissage supervisé est notamment omniprésent dans le domaine de la reconnaissance d'images, où il permet d'identifier des objets, des visages ou encore des caractères manuscrits avec une précision impressionnante. Dans le champ du traitement du langage naturel, il est utilisé pour la traduction automatique, la classification de textes ou encore la génération de texte. En médecine, il joue un rôle crucial dans le diagnostic assisté par ordinateur en permettant d'analyser des radiographies et de détecter la présence éventuelle d'une pathologie. Il est également un élément clé des systèmes de recommandation, comme ceux proposés par les plateformes de streaming vidéo et musical, qui analysent les préférences des utilisateurs pour leur suggérer des contenus pertinents. Il est facile de constater que tous ces exemples donnent un accès au préalable à des données d'entrainement annotées, point inévitable de l'apprentissage supervisé.

c) Limitations

Si l'apprentissage supervisé présente l'avantage de permettre d'obtenir des modèles extrêmement précis lorsqu'ils sont entraînés sur des données bien annotées, son efficacité dépend en grande partie de la qualité et de la quantité des données d'apprentissage. Cela peut poser des problèmes dans certains domaines où l'annotation manuelle des données est longue et coûteuse. De plus, il est particulièrement sensible aux biais présents dans les données d'entraînement, ce qui peut engendrer des prédictions erronées ou même discriminatoires si ces biais ne sont pas corrigés. Enfin, cette approche est parfois limitée lorsqu'il s'agit d'analyser des données non structurées ou de détecter des structures sous-jacentes sans intervention humaine, ce qui constitue l'une des raisons pour lesquelles d'autres formes d'apprentissage, comme l'apprentissage non supervisé ou l'apprentissage par renforcement, peuvent être plus adaptées à certaines problématiques.

d) Conclusion

L'apprentissage supervisé constitue ainsi l'un des piliers fondamentaux de l'intelligence artificielle moderne. Il permet de résoudre une grande variété de problèmes grâce à l'exploitation de vastes ensembles de données annotées et constitue une approche particulièrement efficace dans des domaines où la supervision humaine peut garantir la fiabilité des résultats. Cependant, ses limites soulignent la nécessité de se tourner vers d'autres stratégies d'entraînement pour surmonter les contraintes liées à la disponibilité des données et à la complexité des modèles. C'est précisément ce que nous verrons par la suite avec l'apprentissage non supervisé.

2.2 - Non-supervisé

L'apprentissage non supervisé constitue une approche différente et complémentaire de l'apprentissage supervisé. Là où cette dernière repose sur l'utilisation de données annotées, permettant au modèle d'apprendre en associant des entrées à des sorties prédéfinies, l'apprentissage non supervisé fonctionne sans aucune supervision humaine directe. Dans cette approche, l'algorithme est plongé dans un ensemble de données brutes, dépourvues d'étiquettes explicites, et doit en extraire lui-même des structures, des motifs ou des regroupements sous-jacents. Ce type d'apprentissage s'apparente davantage à une démarche exploratoire, où l'IA ne cherche pas à reproduire une réponse correcte définie à l'avance, mais plutôt à comprendre comment les données s'organisent et interagissent entre elles.

a) Fonctionnement

Le principe fondamental de l'apprentissage non supervisé repose sur l'identification de schéma au sein des données. Sans disposer d'exemples annotés pour lui indiquer la marche à suivre, l'algorithme analyse les similarités, les variations et les relations entre les données afin de les regrouper ou d'en extraire des structures sous-jacentes pertinentes. Ce mode de fonctionnement le rend particulièrement adapté aux situations où les données sont abondantes mais où il serait trop long ou trop coûteux de les annoter manuellement. Le fonctionnement précis des algorithmes utilisant de l'apprentissage non supervisé est très varié, donc donner une ici explication générale n'est pas réalisable. Néanmoins, nous allons étudier deux approches classiques pour avoir une vision générale de ces méthodes.

Parmi les principales approches de l'apprentissage non supervisé, on trouve la **classification automatique** (non supervisée) et la **réduction de dimensionnalité**. La classification automatique, souvent mise en œuvre par des algorithmes de **clustering (regroupement)**, consiste à regrouper les données en ensembles homogènes en fonction de leurs caractéristiques communes. L'objectif de ces techniques est de révéler des catégories dans des ensembles de données sans qu'un humain n'ait besoin d'indiquer au préalable les classes à reconnaître. Ce regroupement en catégories peut se visualiser comme sur la figure 4, on met dans des groupes (on classifie) les points de ce graphe sans avoir idée de la forme de la bonne solution.

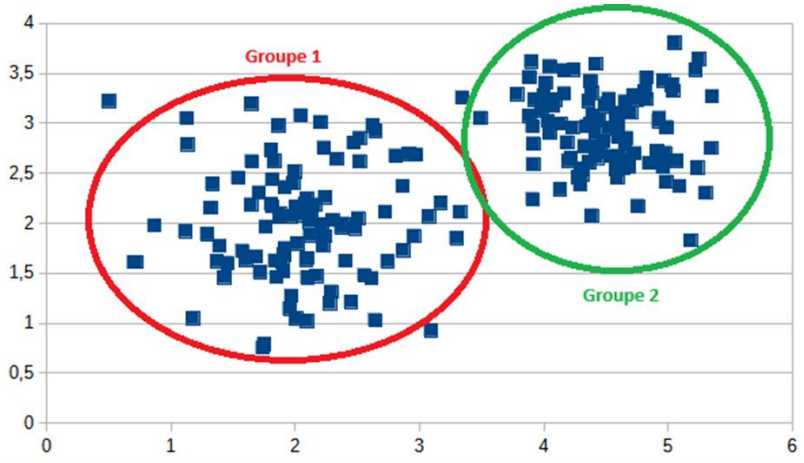


Figure 4 : Exemple de clustering sur un ensemble de point en 2 dimensions

La seconde grande application de l'apprentissage non supervisé est la réduction de dimensionnalité. Cette méthode est utilisée lorsque les données contiennent un très grand nombre de variables, rendant leur traitement complexe et coûteux en ressources. Des méthodes comme **l'analyse en composantes principales (ACP)** permettent de transformer un grand nombre de variables sur un jeu de données en un nombre de variables plus restreint qui permet pourtant de conserver la spécificité de chaque point de donnée. Par exemple pour décrire une foule, un jeu de données pourrait contenir pour chaque personne la taille, la couleur de cheveux, le salaire, l'adresse, etc. L'**ACP** pourrait permettre de s'apercevoir que cette foule habitant toute à Bouchier, un village des Hautes-Alpes, l'adresse n'ajoute pas d'information importante sur chaque personne. On peut donc restreindre les variables en enlevant l'adresse : cela rend le jeu de données plus facile à traiter et à stocker. Ce type d'approche est particulièrement utile dans l'analyse de données massives, comme celles issues d'expériences scientifiques, de bases médicales ou de grandes plateformes numériques, où il est nécessaire d'extraire l'information la plus pertinente sans être noyé sous une avalanche de détails superflus.

b) Applications

L'apprentissage non supervisé trouve des applications concrètes dans de nombreux domaines où l'identification de structures cachées constitue un enjeu central. Il est couramment utilisé dans la segmentation de clientèle, où les entreprises analysent les habitudes de consommation de leurs utilisateurs afin de détecter des profils types et d'adapter leurs stratégies marketing en conséquence. Dans le domaine de la cybersécurité, il permet de repérer des comportements inhabituels dans un réseau informatique et ainsi d'identifier des menaces potentielles, comme des attaques ou des fraudes. En bio-informatique, il est employé pour analyser des séquences génétiques et classer des espèces ou des mutations en fonction de leur proximité génétique. Dans le secteur de la finance, il est utilisé pour détecter des anomalies dans les transactions bancaires et anticiper d'éventuels risques. Enfin, dans le traitement automatique du langage, il permet d'identifier des thématiques récurrentes dans de grands corpus de textes, facilitant ainsi l'organisation de vastes bases de données documentaires.

c) Limitations

Si l'apprentissage non supervisé offre un potentiel considérable pour découvrir des structures invisibles à l'œil humain, il présente néanmoins des défis spécifiques. L'absence d'annotations signifie que les résultats obtenus doivent être interprétés avec prudence, car il n'existe pas de vérité absolue permettant d'évaluer immédiatement la pertinence des regroupements ou des relations détectées. De plus, la qualité des résultats dépend fortement du choix des algorithmes et

des paramètres utilisés, ce qui peut rendre leur mise en œuvre plus délicate que dans un cadre supervisé où les performances sont plus directement mesurables.

d) Conclusion

L'apprentissage non supervisé constitue donc une approche essentielle dans le champ de l'intelligence artificielle, permettant d'exploiter des données non étiquetées pour en extraire des connaissances précieuses. Il se distingue par sa capacité à explorer et organiser de grandes quantités d'informations sans intervention humaine directe, ce qui en fait un outil puissant dans des contextes où l'annotation manuelle des données est impraticable.

2.3 - Par renforcement

L'apprentissage par renforcement est un mélange entre les deux méthodes précédentes : on ne connait pas directement la bonne réponse, mais on sait dans quelle direction aller pour la trouver. En effet, cette approche est inspirée du comportement humain et animal, où l'acquisition de connaissances se fait par essais et erreurs, guidés par des récompenses ou des punitions. À l'instar d'un enfant qui apprend à faire du vélo en ressentant la satisfaction de rester en équilibre ou la chute désagréable lorsqu'il perd le contrôle, un agent (le modèle) interagit avec un environnement et ajuste progressivement son comportement pour maximiser la somme des retours positifs qu'il reçoit. Concrètement, l'agent prend une action dans un état donné (par exemple, tourner le guidon ou pédaler plus fort), puis observe la réaction de l'environnement (rester en équilibre ou tomber) et perçoit une récompense (plaisir de réussir ou douleur de l'échec).

L'objectif de l'apprentissage par renforcement est de découvrir une stratégie, appelée « politique », qui indique à l'agent quelle action entreprendre dans chaque situation pour obtenir le meilleur bénéfice à long terme. Cette méthode se distingue de l'apprentissage supervisé : ici, contrairement à un jeu d'exemples annotés, il n'existe pas de « bonne réponse » fournie en amont, mais seulement un signal de récompense qui guide l'exploration. Au fil de nombreux essais, l'agent apprend à préférer les actions les plus rémunératrices, tout en explorant parfois des choix moins évidents pour ne pas passer à côté de stratégies plus efficaces sur le long terme.

Grâce à ce mécanisme, l'apprentissage par renforcement permet de résoudre des problèmes où l'on ne connaît pas à l'avance la séquence optimale d'actions, cas classique pour des jeux comme le go ou les échecs, mais aussi pour des systèmes de pilotage autonome.

Une différence clé entre l'apprentissage par renforcement et l'apprentissage supervisé réside dans la nature du signal de récompense. Dans l'apprentissage supervisé, le modèle reçoit en une seule fois l'erreur (ou la « récompense négative ») correspondant à une prédiction incorrecte, et une étiquette correcte pour guider directement sa mise à jour. Il n'existe pas de notion de séquence d'actions ni de récompense différée : chaque exemple est traité comme une paire entrée–sortie indépendante. En revanche, en apprentissage par renforcement, le signal de récompense est souvent retardé et lié à une série d'actions successives. L'agent doit apprendre non seulement à prédire la bonne réponse, mais aussi à prendre des décisions cohérentes dans le temps pour maximiser une somme de récompenses. Cette dépendance à la trajectoire d'actions rend l'apprentissage par renforcement particulièrement adapté aux problèmes où chaque choix influence les états ultérieurs et où il est nécessaire de planifier sur le long terme.

a) Fonctionnement

Dans l'apprentissage par renforcement, l'agent évolue au sein d'un environnement en passant d'une situation à une autre, un peu comme un joueur de plateau avance case après case. À chaque

instant, l'agent observe sa situation actuelle, par exemple la position d'un robot sur une grille ou l'état d'un jeu vidéo, et choisit une action à effectuer. L'environnement réagit alors à cette action en se transformant et en envoyant à l'agent un retour, que l'on appelle récompense : si l'agent se rapproche de son objectif, il reçoit un retour positif ; dans le cas contraire il reçoit un retour négatif, comme un signal d'erreur.

Au fil des essais, l'agent cherche à mémoriser quelles actions, dans quelles situations, ont conduit aux meilleurs retours à long terme. Pour cela, il associe à chaque couple (situation, action) une estimation de la valeur attendue : plus cette valeur est élevée, plus l'action est jugée prometteuse. Lorsqu'une action menée dans une situation génère une récompense différente de celle qui était attendue, l'agent ajuste légèrement cette estimation pour la rapprocher de la réalité. Concrètement, si une action a rapporté plus qu'espéré, on augmente l'estimation ; si elle a rapporté moins, on la diminue. L'estimation est évidemment produite par un ensemble de paramètres (formant un réseau de neurones par exemple), et de la même manière que pour l'apprentissage supervisé, ces paramètres sont optimisés pour donner la meilleure estimation possible (comme on cherche la fréquence d'une station radio).

Pour trouver un bon équilibre entre exploitation des actions connues pour être rémunératrices et exploration d'actions encore peu testées, l'agent utilise des stratégies comme l'**epsilon-greedy** : la plupart du temps, il choisit l'action qu'il estime la meilleure immédiatement, sans se soucier du long terme (**approche gloutonne**, bien connue des algorithmes devant effectuer une suite de choix), mais de temps en temps, il en tente une autre pour découvrir de nouvelles possibilités, peut-être plus intéressantes sur le long terme. Après des milliers d'interactions, qu'on appelle épisodes, l'estimation des valeurs se stabilise. Ainsi l'agent a appris la politique qu'on espère optimale : c'est-à-dire la meilleure façon de choisir ses actions selon chaque situation pour maximiser les récompenses cumulées sur le long terme.

b) Applications

Les méthodes d'apprentissage par renforcement trouvent des applications variées dans des domaines où les décisions s'enchaînent en séquence et où chaque décision peut influencer le résultat final. Par exemple, dans le monde des jeux vidéo et des jeux de plateau, ces algorithmes ont appris à battre des joueurs humains en simulant des millions de parties et en affinant leur stratégie par essai et erreur. Les échecs et le jeu de go sont des exemples classiques des performances des modèles d'apprentissage par renforcement.

En robotique, l'apprentissage par renforcement permet à un bras mécanique d'apprendre à saisir un objet ou à un robot mobile de se frayer un chemin dans un espace complexe sans être programmé pour chaque situation précise. C'est en reproduisant virtuellement de nombreuses interactions que le robot découvre les gestes efficaces pour atteindre son objectif.

Dans le monde industriel et énergétique, on l'utilise pour optimiser la consommation d'énergie d'un bâtiment ou la gestion d'une chaîne de production : l'algorithme teste différentes configurations de chauffage ou de planning de machines et retient celles qui réduisent les coûts tout en respectant les contraintes.

Les systèmes de recommandation dynamiques peuvent aussi tirer profit de l'apprentissage par renforcement : au lieu de proposer le même contenu à tous, ils apprennent quelles suggestions retenir pour maximiser l'engagement des utilisateurs au fil de leurs interactions avec la plateforme. Enfin, dans la finance, ces méthodes servent à élaborer des stratégies de trading automatisé, où chaque décision d'achat ou de vente est évaluée en fonction des gains cumulés sur une série de transactions, tout en s'adaptant aux fluctuations du marché.

c) Limitations

L'apprentissage par renforcement nécessite souvent un grand nombre d'interactions pour apprendre correctement : simuler des dizaines de milliers d'épisodes peut prendre beaucoup de temps et d'énergie, surtout si chaque simulation est coûteuse en calcul ou en ressources physiques, comme dans la robotique réelle.

Ensuite, la conception de la fonction de récompense est délicate : si ce signal n'est pas bien calibré, l'agent peut adopter des comportements inattendus, voire dangereux, pour maximiser la récompense. Par exemple, un robot devant servir des verres peut apprendre à renverser l'entièreté d'une bouteille par verre, s'assurant que le verre soit bien rempli, s'il n'est pas pénalisé pour le gâchis. De plus, l'approche gloutonne limite inévitablement les modèles entraînés par renforcement. En effet, la politique de décision est principalement basée sur le fait de maximiser la récompense sur le court terme, cela peut être incompatible avec certaines tâches. Par exemple, sacrifier des pièces lors d'une partie d'échec ou accepter d'arrêter une machine pour éviter une grosse panne plus tard.

D'un point de vue pratique, il peut également être compliqué de transférer ce qui a été appris dans une simulation vers le monde réel, en raison de petites différences entre l'environnement virtuel et la réalité physique. Ces limitations motivent la recherche de solutions hybrides ou de méthodes plus efficaces pour réduire le coût d'apprentissage tout en garantissant la sécurité et la robustesse du système.

d) Conclusion

L'apprentissage par renforcement se révèle être une approche puissante dès lors que l'on doit prendre des décisions séquentielles et optimiser un objectif à long terme sans disposer d'exemples annotés, tout en ayant une bonne idée de l'objectif à atteindre. En tirant parti d'un signal de récompense et de mécanismes d'exploration, un agent peut apprendre des stratégies sophistiquées, comme on l'a vu dans les jeux, la robotique, l'industrie ou la finance. Toutefois, cette méthode exige de nombreux essais, un réglage minutieux de la récompense, et peut être limitée par la complexité des environnements et au transfert vers le monde réel.

Ainsi, l'apprentissage par renforcement complète l'apprentissage supervisé et non supervisé en offrant une voie d'apprentissage adaptée aux problèmes dynamiques où chaque action influence la suite du processus. Son développement continu, notamment grâce à l'amélioration des méthodes de modélisation de l'environnement, ouvre la voie à des applications toujours plus ambitieuses, tout en invitant à rester vigilant sur les coûts, la sécurité et la robustesse des solutions déployées.

3 - Modèles de langage

3.1 - Introduction

Alan Turing, père de l'informatique et de l'intelligence artificielle a cherché à concrétiser en 1950 le concept d'intelligence d'une machine grâce à un test, appelé depuis **test de Turing**[12]. L'idée est de faire converser par le biais d'un écran un humain avec, dans un cas, un autre humain et dans l'autre cas une machine. Si la personne testée n'arrive pas à discerner l'homme de la machine au cours de la conversation, la machine a réussi le **test de Turing**. Dans l'idée de Turing, communiquer

[12] Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, *59*(236), 433–460. http://www.jstor.org/stable/2251299

comme un humain est en lien direct avec la capacité de penser. Même si cette affirmation est critiquable, Turing lui-même donne ses limites, ce test a été un des objectifs visés dans le développement des IA dès leur genèse.

En effet, le thème central de la révolution actuelle à propos des IA est le langage écrit. C'est autour de cette fondation que tout s'est construit. En novembre 2022, OpenAI donne accès à ChatGPT, le monde découvre un **chatbot** (un robot qui répond à nos messages comme lors d'un échange par SMS avec une connaissance, en français **agent conversationnel**) tellement performant qu'on pourrait le confondre avec un humain au premier abord (passant ainsi le test de Turing). Néanmoins, cela ne va pas beaucoup plus loin : le modèle permet d'avoir une discussion sur n'importe quel sujet, tant qu'il ne faut pas trop s'étendre dessus et qu'il ne demande aucune certitude. En effet, GPT3.5 (le premier modèle derrière ChatGPT) a toujours du mal à avoir un discours construit sur la longueur et est incapable techniquement d'affirmer quoi que ce soit. Il ne fait qu'imaginer la réponse la plus logique sans avoir de processus de vérification, ce qui lui permet d'halluciner des auteurs, des personnages historiques, des noms de livres, etc., lorsque la réponse n'est pas évidente. Ce point, entre autres, a été la source de multiples améliorations permettant aux IA tels **ChatGPT**, **Gemini**, **Claude**, **Le Chat** (maintenant **Vibe**) de ne plus être que de simples agents conversationnels, mais de véritables petites usines composées de multiples modèles effectuant diverses tâches comme faire des recherches internet, générer des images, exécuter du code, etc.

Un modèle de langage (**LLM** en anglais, pour Large Language Model) est donc un modèle qui permet d'imiter un humain au niveau de son expression écrite. Par exemple, à sa sortie, **ChatGPT** utilisait le modèle de langage GPT3.5. Un agent conversationnel est donc juste une manière d'utiliser un modèle de langage, mais ce n'est pas la seule : il est possible d'utiliser un **LLM** comme traducteur, comme concepteur de résumé, comme correcteur, etc., et ce sans l'aspect discussion, juste comme un outil. C'est **OpenAI**, avec **ChatGPT**, qui a démontré que l'approche d'un **LLM** via un agent conversationnel permettait de rassembler toutes les capacités du modèle dans une seule application. Plus besoin d'un site de traduction, d'un moteur de recherche, d'un correcteur automatique, tout est compris dans une unique interface : un **chatbot**.

3.2 - Fonctionnement général

Le fonctionnement d'un modèle de langage repose avant tout sur la capacité à assimiler d'immenses volumes de texte pour en extraire des tendances statistiques. Dès le départ, on confronte le **LLM** à un corpus très varié : livres, articles de presse, forums, pages web, et même dialogues. L'objectif de cette phase, dite d'entraînement, est de « lire » ces milliards de mots et d'apprendre, pour chaque séquence de quelques mots, quelle est la suite la plus vraisemblable. Concrètement, on présente au modèle une phrase tronquée, par exemple « Le chat dort sur le », et il doit deviner que « tapis » ou « canapé » est une suite cohérente. Cette tâche de prédiction de mot suivant, répétée sans cesse sur différents contextes, permet au modèle de construire une compréhension statistique des relations entre mots, expressions et idées, sans pour autant disposer d'une base de connaissances explicite : il « sait » ce qui est fréquemment associé, mais ne vérifie pas la véracité factuelle de chaque affirmation.

La partie suivante (4 – Transformeur, ou la brique élémentaire d'un LLM) présentant le fonctionnement des **LLM**, explique plus longuement et en détail leur architecture, en particulier les transformeurs. Cependant, pour comprendre le fonctionnement des LLM, voici une courte présentation de ce qui se passe sous le capot : pour stocker et exploiter la connaissance apprise en étant entraîné sur des corpus de texte, le modèle attribue à chaque mot une suite de nombres, mathématiquement des vecteurs, qui peuvent être vus comme des définitions numériques de ces mots. Ces vecteurs (ou **embeddings**) sont affinés lors de l'entraînement pour que des termes

proches dans le langage (par exemple « chien » et « canidé ») soient également proches dans le monde des vecteurs, qu'ils pointent dans la même direction. À chaque nouvel exemple de texte, le **LLM** transforme successivement les mots de cet exemple en vecteurs et les fait passer à travers plusieurs couches de calcul. Chaque couche ajuste ces vecteurs, ces définitions, en modifiant la suite de nombres attribuée à l'origine, pour capturer les particularités propres au contexte entourant chaque mot : un verbe et son sujet, un personnage évoqué plusieurs phrases plus tôt, sans qu'il soit nécessaire d'indiquer manuellement au modèle des règles linguistiques. C'est cette transformation progressive et automatique qui confère au modèle sa flexibilité et sa puissance : il peut s'adapter à des tournures de phrase complexes et à des sujets variés, du résumé de document à la traduction.

Lorsque l'on souhaite générer du texte, on commence par un ou plusieurs mots de départ (le **prompt**) que l'on soumet au modèle. Celui-ci renvoie alors une probabilité sur l'ensemble de son vocabulaire. Le mot choisi pour être le suivant sera un de ceux ayant la plus grande probabilité. Une fois le mot choisi, le processus se répète, créant mot après mot un texte cohérent. Par exemple, si l'on écrit « La météo aujourd'hui est », le modèle assignera une forte probabilité à « ensoleillée » ou « pluvieuse » selon les contextes appris et une très faible probabilité à « chambre ». Ainsi, un même **LLM** peut produire un poème lyrique, un code informatique ou un résumé précis, cela dépend seulement du texte fourni en entrée.

Enfin, pour orienter le modèle vers des usages concrets, on peut lui appliquer un **fine-tuning**, où l'on corrige ses sorties et on le récompense lorsqu'il fournit une réponse satisfaisante. Cette étape, réalisée après l'entraînement, permet de spécialiser le **LLM** : on lui apprend à privilégier l'exactitude factuelle, à adopter un ton formel ou à respecter des règles éthiques précises. Mais, fondamentalement, tout part de cette même mécanique de lecture massive, de conversion en vecteurs et de prédiction de la suite la plus plausible. C'est ce mariage de volume de données, de puissance de calcul et de mathématiques des vecteurs qui rend les **LLM**s si polyvalents et révolutionnaires dans le traitement automatisé du langage.

3.3 - Exemples

Pour mieux comprendre le fonctionnement général d'un **LLM**, nous allons étudier quelques exemples. Le premier présente une phrase très simple :

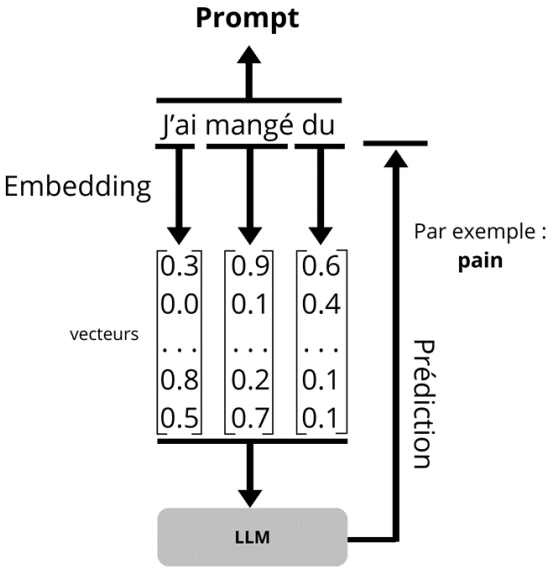


Figure 5 : Exemple de prédiction d'un LLM pour une phrase simple

Pour cet exemple, le **prompt** est « J'ai mangé du ». Chaque mot est associé à des suites de nombres (des vecteurs) qui sont fournis au LLM. Celui-ci donne une prédiction sous la forme d'un vecteur qui est traduit en langage courant. La prédiction « pain » a été faite seulement à partir du **prompt**, elle aurait donc pu être sans problème un autre plat ou quelque chose de comestible en général (poulet, yaourt, gâteau, poisson, …), par contre elle n'aurait pas pu être autre chose car le modèle cherche le mot le plus probable pour compléter le prompt. Le choix particulier de « pain » par rapport à d'autres plats est totalement dû au hasard, contrairement au fait que ce soit comestible. Voici une phrase donnant plus de contexte :

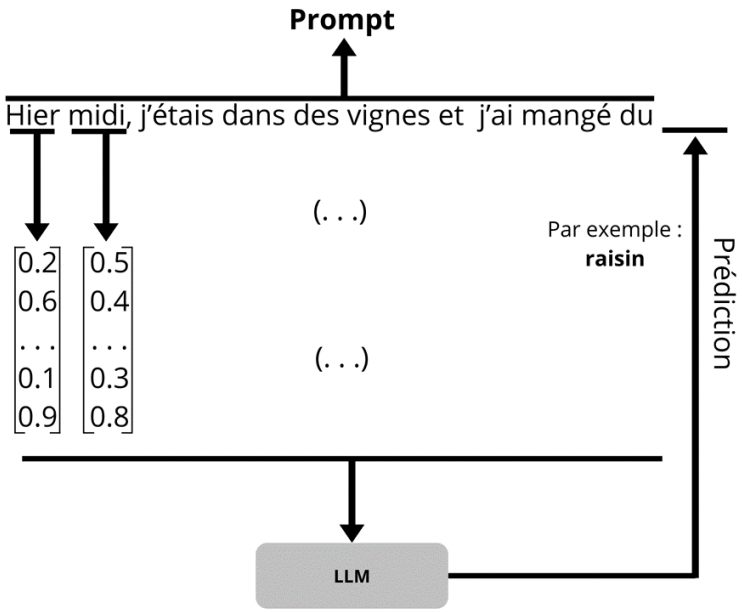


Figure 6 : Exemple de prédiction d'un LLM pour une phrase complexe

À l'instar de l'exemple précédent, le prompt parle de manger quelque chose. La différence est sur la précision supplémentaire dans la situation : « Hier midi, j'étais dans des vignes ». Le **LLM** va donc proposer une prédiction en accord avec cette situation, la plus probable étant raisin. Ici le choix n'est quasiment plus aléatoire étant donné que la situation restreint énormément les possibilités. Si on suppose que le modèle a été bien entraîné et qu'il a une architecture bien choisie, il sera capable de comprendre la situation et donner une des seules prédictions acceptables pour ce prompt (ici raisin).

Il est important de comprendre que pour un **LLM** classique, la prédiction ne dépend que du texte d'entrée et de l'entraînement (si tous les textes étudiés sont en français, le modèle aura du mal à parler anglais). À aucun moment des informations extérieures viennent impacter la réponse (internet, les échanges de la veille, etc.). Cependant :

1. Des modèles plus avancés proposent de faire des recherches pour appuyer leur propos.

2. Les modèles actuels ont très souvent une mémoire des discussions passées, des informations sur l'utilisateur, etc.

2. Le texte d'entrée peut être très long (des milliers de mots, ce qui est appelé **fenêtre de contexte**), ainsi lors d'un échange avec un **chatbot** celui-ci prend comme entrée l'entièreté de la discussion actuelle et pas seulement le dernier prompt (la dernière question ou instruction).

Prédire un mot, c'est bien, générer un texte c'est mieux. C'est pourquoi le **LLM** n'est pas appelé qu'une seule fois mais plusieurs fois à la suite en utilisant le prompt précédent, additionné de la prédiction. Ainsi un modèle génère du texte de manière incrémentale, avançant de plus en plus dans le propos. Ce comportement est visible lors de l'utilisation d'un **chatbot**, on a l'impression qu'il écrit mot par mot sa réponse au lieu de tout afficher d'un coup, c'est bien parce que le texte

affiché est construit petit à petit, prédictions après prédictions. La **fenêtre de contexte** est la taille maximale du texte qui peut être fourni en entrée du modèle pour prédire le prochain mot, celle-ci est de dizaines de milliers de mots[13] pour les **LLM** actuels (**chatGPT**, **Llama**, **Claude**, **Mistral**, **Le Chat -** maintenant **Vibe**, etc.).

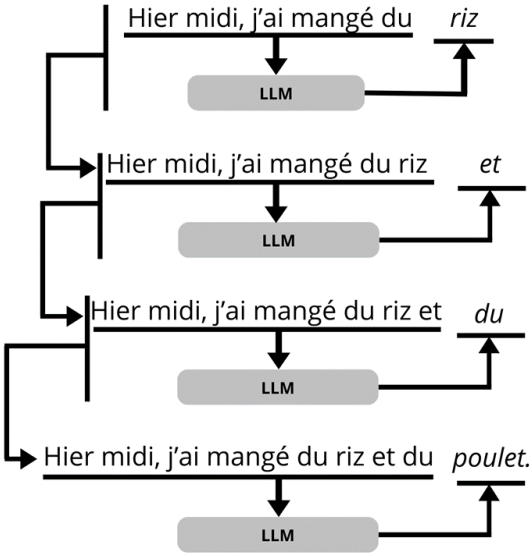


Figure 7 : Exemple de génération d'un texte par un LLM

3.4 - Entraînement

Pour obtenir un **LLM** capable des prouesses présentées dans la partie précédente, il faut l'entraîner. On distingue trois étapes : a) le pré-entraînement, b) le **fine tuning** et c) le RLHF (apprentissage par renforcement suivant un retour humain).

a) Le pré-entraînement

C'est ici où l'on cherche à amasser le plus de données textuelles possible dans le but d'avoir la plus grande exhaustivité et variété sur notre manière d'écrire une langue. Pour les **LLM** les plus performants, on parle ici de la quasi-entièreté des textes disponibles sur internet, en partant de discussions sur un forum jusqu'aux textes de Victor Hugo en passant par des milliers de pages Wikipédia (etc.). Ces textes sont récupérés par des *crawler*, des robots qui parcourent le web et qui enregistrent tout ce qu'ils découvrent (par exemple, le projet *Common Crawl*[14] donne accès gratuitement à une base de données énorme contenant la majorité du web accessible). Cela représente des milliards de textes, dans toutes les langues, dans tous les contextes. On retrouve donc d'authentiques discussions, des textes d'auteurs, ou des articles mais aussi des lignes de codes, des recettes de cuisines, etc. Toutes ces informations sont aussi bien d'excellente qualité que de très mauvaise, en passant par toutes les nuances, on suppose donc que la base de données utilisée représente suffisamment l'expression écrite dans son ensemble pour que le **LLM** apprenne « la bonne manière » de s'exprimer, et non un sous-groupe non représentatif ; cet écueil est arrivé aux premiers modèles textuels (comme le modèle Tay[15] de Microsoft introduit en 2016 sur un réseau social, qui a, en une journée, « appris » à émettre des commentaires inappropriés à cause d'internautes un peu trop joueurs).

[13] https://www.llmreference.com/model-family/gpt-5, https://www.llmreference.com/model-family/claude-4.5, https://www.llmreference.com/model-family/llama-4, …
[14] https://fr.wikipedia.org/wiki/Common_Crawl
[15] https://fr.wikipedia.org/wiki/Tay_(intelligence_artificielle)

Les textes rassemblés sont utilisés pour entraîner le modèle de manière supervisée (partie 2.1 sur l'apprentissage supervisé) : on coupe le texte aléatoirement et le modèle doit prédire le mot suivant. Comme ce mot est connu, on corrige les paramètres du modèle pour arriver à la bonne prédiction. En répétant ce processus quelques milliers de milliards de fois avec l'entièreté des données, on obtient un modèle capable de créer des textes plus vrais que nature.

Cet entraînement prend évidemment des mois de calculs sur des milliers d'unités de calculs[16] extrêmement coûteux[17] (à l'achat comme à l'utilisation). Ces unités sont, pour la majorité, fabriquées par NVIDIA devenue en l'espace de quelques années la plus grande entreprise au monde en termes de capitalisation boursière, devant Apple, Microsoft ou Google, grâce à l'écrasante majorité de son activité basée sur la production de ces unités de calculs, vendues à d'autres géants dans le but d'entraîner des modèles. Cela donne une idée de l'importance qu'a pris l'IA dans notre économie actuelle, au-delà du simple **chatbot**.

b) Le fine tuning

Le pré-entraînement permet d'avoir une boîte noire qui, lorsqu'on lui donne une suite de mot, prédit un mot qui compléterait bien cette suite. C'est insuffisant pour avoir un équivalent de chatGPT. En effet on veut s'assurer que la prédiction aille bien dans le sens d'une réponse d'un **chatbot** et pas autre chose. C'est pourquoi on accorde, on **fine tune**, le **LLM** à la tâche qu'on lui demande. Pour un **chatbot**, on va légèrement modifier les milliards de paramètres trouvés lors de la phase de pré-entraînement pour orienter toutes les prédictions afin d'avoir, justement, un comportement de chatbot. Cela se fait en ré-entraînant le **LLM** sur des corpus de discussions style **chatbot**. On doit donc encore se doter d'une grande quantité d'échanges représentant le comportement attendu. On cherche ici seulement à modifier très légèrement les paramètres comme on accorde une guitare un peu désaccordée : on n'est pas loin d'un bon résultat, il suffit d'un peu jouer autour de l'état initial pour trouver un état satisfaisant. Ainsi le pré-entraînement apprend au **LLM** à parler et le **fine tuning** lui apprend à être un chatbot.

c) Le RLHF

Cette étape utilisée pour la première fois par OpenAI est assez nouvelle[18], les premières itérations de **LLM** s'arrêtaient au **fine tuning**. Le **RLHF**, ou apprentissage par renforcement suivant un retour humain, consiste à accorder le LLM en fonction de retours humains sur ses réponses. On a donc un **chatbot** quasiment fonctionnel qui va échanger avec des opérateurs, ces derniers notant la qualité de la conversation. On parle ici d'apprentissage par renforcement (voir partie 2.3) car le modèle va modifier ses paramètres dans le but de maximiser la note de l'opérateur. Contrairement aux deux étapes précédentes, où les paramètres étaient modifiés pour atteindre un résultat connu (les corpus de textes), ici le résultat est inconnu : il n'y a que la note donnant une idée de la direction à suivre pour atteindre un bon résultat. On arrive donc à aller au-delà de la possible médiocrité des corpus d'entraînement étant donné que le modèle apprend par lui-même sans suivre d'exemples. Il faut néanmoins que la notation des opérateurs soit de bonne qualité et en quantité suffisante (des milliers de discussions). Cette approche a deux intérêts, le premier est d'améliorer la qualité des réponses du modèle de manière générale, le second est de donner un style au **LLM** et de cadrer ses réponses.

---

[16] https://milvus.io/ai-quick-reference/how-long-does-it-take-to-train-an-llm
[17] https://tech-insider.org/nvidia-blackwell-gpu-pricing/
[18] Ouyang et al., Training language models to follow instructions with human feedback, NeurIPS 2022, https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf

Ainsi la qualité des réponses arrive avec la diminution des phases de « délire » où le modèle va inventer sans prévenir et créer des phrases sans aucun sens, ce qui est lourdement pénalisé par les opérateurs. Le second point est plus politique : c'est grâce au **RLHF** que les **LLM** acceptent de dire certaines choses et d'en censurer d'autres. Ainsi certains **LLM** sont plus souples que d'autres sur des sujets sensibles. Cette différence est évidente en comparant **ChatGPT** (d'OpenAI) et **Grok** (de xAI, entreprise d'Elon Musk). **Grok** a été développée en réponse à la trop grande neutralité de **ChatGPT**. C'est pourquoi **Grok**, contrairement à **ChatGPT**, peut exposer des points de vue politiques, faire de l'humour noir ou donner des informations sur des pratiques illégales. **Grok** a eu de nombreuses fois des dérapages complotistes, climato-sceptiques ou même antisémites. Il est très probable que ces deux modèles soient entraînés sur les mêmes corpus de documents, la différence de comportement se jouerait donc sur le **RLHF** et la manière de noter de la part des opérateurs. Le **RLHF** sert donc à aligner un **LLM** avec les comportements humains, que ce soit dans la forme (qualité des réponses) ou le fond (contenu des réponses).

4 - Transformeurs, ou la brique élémentaire d'un LLM

4.1 - Présentation

a) Motivations

Un LLM est basé sur la prédiction du mot suivant. Il faut donc une fonction réalisant cette prédiction, c'est le rôle du **transformeur**. Les **transformeurs** ont été inventés dans le but de repousser les limites des IA sur le terrain du langage. C'est la principale motivation pour le développement de cette approche. À l'origine utilisés pour la traduction, ce sont aujourd'hui la base des **LLM**. Même si leur fonctionnement est toujours dans la lignée des modèles utilisés précédemment (une entrée passe au travers de couches appliquant un grand nombre de calculs simples, donnant une sortie très complexe), les **transformeurs** arrivent à traiter formidablement la structure de l'entrée, ce qui est nécessaire pour le traitement du langage. Tout en étant très rapide à exécuter (via des calculs en parallèle), cette architecture s'est montrée bien plus performante que ce qui était utilisé auparavant. C'est pourquoi son utilisation est maintenant incontournable dans tout modèle où la structure de l'entrée joue une part importante de l'information totale (le texte ou les images sont très concernés par cela).

b) Historique

Avant 2017, l'approche considérée comme la plus efficace pour le traitement automatique des langues était basée sur des réseaux de neurones récurrents[19,20] (**RNN** ou Recurrent Neural Networks). Dans le cas d'une traduction, un **RNN** va transformer une phrase mot après mot en adaptant son comportement à chaque nouveau mot : examen du premier mot de la phrase, traduction, mot suivant, etc. L'intérêt se trouve dans le fait que la traduction du mot suivant dépendra des mots précédents. C'est comme si le sens de la phrase à traduire était compris au fur et à mesure. Cette approche (dans des formes plus complexes) a été la reine des années 2010. Néanmoins elle comprend un problème fondamental, elle est séquentielle : chaque phrase doit être traitée mot après mot. Cela pose un gros problème de mise à l'échelle car ces mêmes années ont vu naître dans le monde de l'IA la parallélisation massive des calculs (lancé par la création du

[19] https://fr.wikipedia.org/wiki/R%C3%A9seau_de_neurones_r%C3%A9currents
[20] Lipton, Z. C., Berkowitz, J., & Elkan, C. (2015). A Critical Review of Recurrent Neural Networks for Sequence Learning. *arXiv [Cs.LG]*. http://arxiv.org/abs/1506.00019

modèle de reconnaissance d'image AlexNet[21], une des avancées les plus importantes dans le monde du **deep learning** et développé par un doctorant de Hinton, un des pères des réseaux de neurones). Paralléliser des calculs signifie qu'au lieu de faire les calculs les uns à la suite des autres, ils sont tous exécutés en même temps (généralement sur des **GPU**, souvent développés et construits par NVIDIA). L'avantage est un gain de temps massif et l'augmentation importante de la taille des modèles. Or les **RNN** peuvent difficilement être parallélisés, il semblait donc nécessaire de trouver une architecture de remplacement. En 2017, une équipe de recherche chez Google, propose une méthode qui se passe complètement de l'aspect récurrent des **RNN** tout en gardant un des concepts développés pour ces réseaux : **l'attention**.

Dans la publication, devenue célèbre « Attention is all you need »[22], cette équipe démontre que même sans récurrence, une architecture utilisant seulement le concept d'attention arrive à des performances bien supérieures aux **RNN** de pointe pour des tâches de traduction : « L'attention est tout ce dont vous avez besoin » (et la récurrence est superflue). Ils ont nommé cette architecture de modèle un **transformeur**. Aujourd'hui les **transformeurs** sont massivement utilisés aussi bien dans des traducteurs que dans des modèles de génération de vidéos ou des moteurs d'échecs. Une décennie plus tard, le monde de l'IA est en train de montrer que l'attention est vraiment tout ce dont on a besoin, dans toutes les applications possibles.

c) Concept général

Un **transformeur** est une grosse pièce à imbriquer dans un modèle plus large, de la même manière qu'un réseau de neurones. Il a pour but de comprendre la logique fondamentale derrière une suite d'information, souvent tirée d'un langage, par exemple comment les mots interagissent entre eux, quel est le sens d'une phrase, quelle est sa forme, etc. L'intérêt final est d'utiliser cette compréhension pour extraire les caractéristiques d'une entrée, souvent un texte et les utiliser pour donner lieu à des sorties très complexes.

Pour arriver à de tels résultats, un **transformeur** est construit en un mille-feuille de couches qui affine sa compréhension au fur et à mesure. Dans le cas d'un texte (par exemple « Hier j'ai mangé du riz et »), un modèle à quatre couches entrainé comme **LLM** pourrait agir ainsi : la première couche identifie la nature des mots (« Hier » : adverbe, « je » : sujet, « ai mangé » : verbe, « du riz » : nom et son déterminant, « et » : conjonction), la seconde leurs relations (« Hier » : précise la temporalité de l'action, « je » : source de l'action, « ai mangé » : action, « du riz » : sur quoi on agit, « et » : on agit sur quelque chose d'autre), la troisième le sens du texte (l'auteur décrit le repas qu'il a fait la veille, il n'a pas seulement mangé du riz) et la dernière le mot complétant au mieux ce texte(une possibilité probable étant « du poulet »).

Ces couches sont basées sur des **têtes d'attentions**. Ce sont elles qui vont « lire » le texte. Le **transformeur** est donc un assemblage de **têtes d'attentions** et de réseaux de neurones **fully connected** (voir partie 1.3).

---

[21] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems* (Vol. 25). https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[22] Vaswani, A. et al. (2017). Attention is All you Need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds), *Advances in Neural Information Processing Systems* (Vol. 30). Retrieved from https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Pour qu'un modèle informatique puisse traiter du texte, celui-ci doit être converti en données numériques. Alors que les lettres de l'alphabet ont un sens pour nous, les machines ne savent interpréter et manipuler que des suites de nombres. Au lieu de donner simplement une équivalence lettre/nombre, on va chercher à associer chaque mot (chaque **token** plus précisément, des parties de mots permettant de simplifier le vocabulaire) à une longue suite de nombre. Cette suite, nommée vecteur, va représenter fidèlement son mot, comme si on en avait extrait le sens pour le stocker sous forme numérique : une sorte de définition numérique du mot. Cette association est le **word embedding** (plongement lexical). Evidemment la représentation numérique de chaque mot ne suffit pas à comprendre le sens d'un texte. Une phrase n'est pas une suite de mot, mais un assemblage. On peut perdre le sens en enlevant un seul mot. Le rôle du **transformeur** est de comprendre l'assemblage.

4.2 - Word embedding

a) Présentation

Pour pouvoir traiter du texte, les ordinateurs ne lisant pas directement les mots, un modèle a besoin de recevoir en entrée une information formatée dans sa langue : des valeurs numériques associées à chaque mot. On cherche donc à trouver la meilleure méthode pour lier un mot à une valeur ou une suite de valeur. Le **word embedding**, c'est un dictionnaire de traduction langue/machine.

b) Approche naïve

Prenons l'exemple du français, une idée naïve pourrait être de prendre le dictionnaire, et associer à chaque mot pris dans l'ordre alphabétique un nombre entier croissant. On créerait donc un dictionnaire de traduction comprenant les entrées suivantes :

- a : 1
- à : 2
- aalénien : 3

…

- zymotique : 112378
- zythum : 112379

Cette approche présente l'intérêt d'être très facile à mettre en place. Nombreux modèles l'utilisent comme première marche vers des dictionnaires de traductions plus complexes. Elle s'appelle le **one-hot encoding**[23]. Néanmoins, elle porte un gros désavantage : les nombres proches indiquent une proximité alphabétique (sur la forme), et pas sémantique (sur le sens). On n'aide pas du tout la machine à traiter notre langage avec cette méthode. Pour un humain, « Papa » et « Maman » sont très proches sémantiquement, alors que, dans le dictionnaire de traductions, ils seront séparés par des milliers de mots. On perd donc avec cette approche des informations très importantes, que la machine devra retrouver avant même de commencer son travail de traitement du texte.

c) Approche sémantique

Pour dépasser le problème de perte d'information par l'approche naïve, on cherche à faire des associations basées sur la sémantique plutôt que sur l'ordre alphabétique. Les mots sont listés selon

[23] https://en.wikipedia.org/wiki/One-hot#Natural_language_processing

leur proximité sémantique (à quel point ils ont le même sens) et non par ordre croissant. À cette liste, on associe un entier croissant de la même manière que précédemment, et nous avons un nouveau dictionnaire de traduction langage/machine :

- papa : 1
- maman : 2
- grand-parent : 3

…

- branche : 55423
- arbre : 55424
- chêne : 55425

…

Ainsi, lorsque le modèle rencontre le nombre 55425, il sait « facilement » que 55423 ou 55424 peuvent le remplacer ou le compléter. L'association entre ces trois mots aurait demandé beaucoup d'apprentissage avec l'approche naïve, elle n'en demande quasiment aucun avec cette approche sémantique. Il reste néanmoins des défauts à ce dictionnaire de traduction : le choix de la liste est le plus important. Effectivement, cette liste semble être un casse-tête à créer même pour une machine : pourquoi placer « grand-parent » après « maman » et pas avant « papa » ? Si on décide de couper la poire en deux et de le placer au milieu, où placer « arrière-grand-parent » ? Le fait que les mots doivent être arrangés en liste est la source de ce problème, il faut donc arriver à s'en séparer.

d) Approche vectorielle

En mécanique, si l'on veut décrire le déplacement d'une voiture, sa vitesse en km/h ne suffit pas. Il faut savoir dans quelle direction la voiture se déplace. Or, contrairement à la valeur de la vitesse, cette direction ne peut pas être résumée en un nombre, on utilise donc un vecteur. Ce vecteur sépare en trois valeurs la direction : par exemple 20% vers le nord, 70% vers l'est et 10% vers le haut. On arrive donc, en assemblant trois nombres (20,70,10), à décrire parfaitement le déplacement de la voiture à condition de donner la base du vecteur : dans notre exemple (nord, est, haut). Le vecteur (-100, 0, 0) décrit donc une situation où la voiture se dirige uniquement vers le sud et ainsi de suite. Il est intéressant de noter que deux vecteurs exprimés dans la même base sont comparables. Par exemple, si un passager d'une voiture se dirigeant suivant (60,-40, 0) jette une balle dans la direction (0,0,100), alors la balle se déplacera suivant (30,-20,50) soit la somme des deux vecteurs (remise entre -100 et 100 pour rester en pourcentage). On va utiliser ce concept de vecteurs pour créer un dictionnaire de traduction.

Pour mettre en place ce dictionnaire de traduction langage/machine, on va cette fois évaluer les mots sur de multiples critères. Ces critères forment la base de l'évaluation, par exemple : « agréable », « coûteux », « froid », « rapide », etc. Dans cette base, « voiture » se verra attribuer un haut score sur le critère « coûteux » et « rapide », mais sûrement une petite valeur sur « froid ». Un « trombone » n'est pas très « agréable » ni « rapide » et est peu « coûteux », ces informations lui donneraient donc un bas score sur ces trois critères. À la fin de l'évaluation on obtient une suite de scores associés aux critères pour chaque mot : un vecteur dont la base n'est pas de seulement trois caractéristiques (nord, est, haut) mais peut être bien plus (« agréable », « coûteux », « froid », « rapide », etc.). Si on arrive à trouver une base suffisamment détaillée, il est possible d'arriver à associer un vecteur unique à chaque mot du dictionnaire. En effet, « maman » et « papa » seront très proches sur la plupart des critères, mais se différencieront sur le critère « genre ». De même, « papi » sera très proche de « papa » sauf sur le critère « âge ». Cela peut

donner le dictionnaire suivant, avec la base (« agréable », « genre », « âge », « commun ») et une notation sur 100 :

- papa : (100, 0, 50, 0)
- maman : (100, 100, 50, 0)
- papi : (100, 0, 90, 0)
- grand-parent :(100, 50, 90, 0)
- arbre : (65, 50, 100, 100)
- chêne : (70, 50, 100, 80)

…

Il semble donc qu'on ait ici un dictionnaire de traduction qui passe du langage à des informations traitables par une machine en ne perdant pas le sens des mots. Effectivement, « papa » reste proche de « maman » qui reste proche de « grand-parent ». Malgré cette proximité, les vecteurs associés restent différents. Ce dictionnaire de traduction est le **word embedding**, utilisé aujourd'hui par tous les modèles de traitement automatiques des langages. Il reste deux problèmes : comment trouver la base et les scores de manière automatique ?

e) Construction

Créer cet **embedding** ne peut se faire « à la main » : il y a des centaines de milliers d'entrées à évaluer et il est peu probable que la base choisie soit la meilleure. On appelle ici « meilleure base » celle qui va donner les plus grandes variations de score sur chaque critère. Par exemple, « âge » est un bon critère pour différencier « papa » de « papi », au contraire « agréable » n'en est pas un. On conservera donc plus « âge » que « agréable » dans cette base. On cherche donc une base qui englobe toutes les subtilités sémantiques de tous les mots de la langue étudiée.

Etant donné que cet **embedding** (ou dictionnaire) n'est pas réalisable à la main, les chercheurs travaillant sur le **NLP** (traitement automatique des langages) ont mis en place une méthode : **word2vec**[24] (mot vers vecteur). Pour pouvoir faire un dictionnaire automatiquement, il faut un critère compréhensible par une machine, et pas seulement une idée telle que le « meilleur dictionnaire ». L'approche **word2vec** consiste à fabriquer le dictionnaire de manière à ce qu'il permette de retrouver un mot à partir de ses voisins dans une phrase. En effet, cette méthode considère qu'un « bon dictionnaire » est un dictionnaire qui permet de retrouver un mot en faisant la moyenne des traductions de ses voisins (sous forme de vecteur). Par exemple, on va étudier la phrase « Un petit chat mignon chasse une souris ». Un bon dictionnaire va faire en sorte que la moyenne de « un », « petit », « mignon » et « chasse » permette de retrouver « chat ». De la même manière, « chat », « mignon », « une » et « souris » doivent permettre de retrouver « chasse ». Même si ce critère peut sembler étonnant, il s'avère qu'il fait partie de ceux qui donnent les meilleurs dictionnaires.

Maintenant que nous avons un critère, il ne reste plus qu'à créer ce dictionnaire. Pour cela, on choisit arbitrairement la dimension de la base. Dans l'exemple de la présentation de l'approche vectorielle, la base (« agréable », « genre », « âge », « commun ») a une dimension de 4. Lorsque la dimension est choisie (et seulement la dimension), on laisse la machine faire. En partant d'un dictionnaire complètement aléatoire, elle va petit à petit préciser ces traductions, en essayant de valider le plus possible le critère pour être un « bon dictionnaire » sur un très grand nombre de phrases. Si la machine n'arrive pas à deviner « chat » avec les voisins fournis, elle modifie légèrement son dictionnaire jusqu'à ce que la combinaison mathématique de « petit », « mignon »

[24] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *arXiv [Cs.CL]*. http://arxiv.org/abs/1301.3781

et « chasse » se rapproche de la définition numérique de « chat ». Répétez cela sur des milliards d'exemples, et nous obtenons à la fin un « bon dictionnaire ».

f) Propriétés

Word2vec permet de créer un dictionnaire de traduction langage/machine très intéressant. En effet, à aucun moment le contenu de la base n'est précisé. Tout se passe comme si on demandait à la machine un « bon dictionnaire » sur quatre critères, sans les préciser. Pourtant à la fin de l'entraînement « papa » sera proche de « maman », alors que rien ne l'imposait. C'est en fait une répercussion de notre définition de « bon dictionnaire » : on va souvent trouver « papa » et « maman » dans des phrases similaires, avec des voisins similaires. Le dictionnaire va donc donner à ces mots une définition numérique proche. L'autre propriété de ce dictionnaire est que si l'on ajoute (ou soustrait) deux mots, on trouve le point commun ou la différence entre ces deux mots. Par exemple, « papa » + « maman » = « parent », « papa » - « maman » = « homme » et « maman » - « papa » = « femme ». C'est comme si on pouvait ajouter ou soustraire des mots numériquement pour en créer d'autre. On pourrait imaginer « maman » + « couronne » = « reine », « course » - « vitesse » = « marche » ou même « content » = - « triste ».

Ces deux propriétés facilitent grandement le travail des **LLM**. Effectivement, ils peuvent ainsi jouer avec les mots en faisant des opérations numériques, et obtenir des résultats qui restent logiques dans notre langage.

Le site https://cemantix.certitudes.org/ permet de jouer avec cette propriété de similarité entre les définitions numériques de deux mots proches. Le but du jeu est de trouver un mot en ayant comme indice la similarité entre le mot secret et tout autre mot de notre choix. Cette proximité prend la forme d'un score de -100 à 100. Si un mot peut facilement prendre la place d'un autre dans une phrase, leur score commun sera proche de 100 car notre « bon dictionnaire » les a placés à côté. Au contraire, deux mots difficilement interchangeables auront un score commun proche de -100. Ce score est donc une métrique de la similarité sémantique entre deux mots et il découle d'un dictionnaire de traduction langage/machine créé à la manière **word2vec**. Ce type de jeu permet de comprendre comment les mots sont perçus par les **LLM**.

g) Limitations

On pourrait penser que ces **word embeddings** suffisent à prédire le mot suivant (la tâche des **LLM**). En effet, il suffirait de combiner les définitions numériques des mots précédents pour trouver le suivant. Hélas, ce dictionnaire de traductions langage/machine est fixé, il ne prend donc pas en compte le contexte autour des mots. Or, on sait bien qu'un mot peut avoir plusieurs significations, ou que le contexte de la phrase change l'idée générale de ce mot. Par exemple, un « lion » va souvent être lié à la force, le charisme ou l'impitoyabilité, alors qu'un « lion en peluche » est tout de suite moins impressionnant. Ces subtilités ne sont pas prises en compte par ce dictionnaire. Traduire une phrase mot à mot n'a jamais été une bonne idée, on a donc besoin d'un mécanisme qui ajuste le sens des mots en fonction du contexte, pour pouvoir trouver une prédiction acceptable : c'est le rôle de l'attention.

4.3 - Attention

a) Concept

Les **LLM** fonctionnent grâce à un mécanisme simple mais très puissant. Les parties suivantes, sur l'attention et les transformeurs, sont basées sur l'approche de G. Sanderson dans « L'attention

dans les transformers, expliquée visuellement | Chapitre 6, Apprentissage profond »[25] (explication claire et visuelle sur le mécanisme d'attention dans un LLM), ainsi que celle de B. Ghojogh et A. Ghodsi dans leur papier « Attention Mechanism, Transformers, BERT, and GPT: Tutorial and Survey »[26] (plus mathématique mais aussi plus complète).

L'attention est un mécanisme servant à réévaluer les définitions données par le **word embedding** pour coller plus justement au sens réel du mot. Au fur et à mesure de la progression dans le transformeur, les définitions vont être tellement précisées qu'elles vont même contenir le sens de la phrase au lieu du sens des mots. Pour rappel, à chaque mot est attribuée une définition numérique, sous la forme d'un vecteur (l'**embedding**). Cette définition numérique est donc un vecteur associé à un mot, initialement déterminé par le **word embedding**, elles sont vouées à évoluer. L'attention va modifier ces vecteurs pour améliorer (par rapport à la situation) la définition d'un mot. On se retrouve après le passage d'une phrase dans une **tête d'attention** (le nom du bloc appliquant l'attention), avec la même phrase composée de définitions plus « adaptées » à l'ensemble. Plus ces définitions seront « adaptées », plus le sens de la phrase se reflètera dans les définitions.

Les **LLM** utilisent donc l'attention pour extraire le sens précis de chaque mot d'une phrase (en prenant en compte le contexte) et prédire au mieux le mot qui pourrait suivre.

b) Fonctionnement général

Pour arriver à réévaluer le sens de chaque mot d'une phrase, une tête d'attention va faire « parler les mots entre eux ». Chaque mot d'une phrase va communiquer sa définition à tous les autres, et au travers de cette communication, la mise à jour des définitions s'effectue. Plus précisément, chaque mot de la phrase va proposer une modification aux autres, et les mots vont plus ou moins la prendre en compte selon leur affinité avec le mot à l'origine de la modification. C'est de cette affinité que vient le terme attention : si un mot a une bonne affinité avec un autre, on dit qu'il lui prête attention. Par exemple, le fait que le « lion » soit une « peluche » semble très important, « lion » devrait donc prêter beaucoup d'attention à « peluche ». Ainsi, lors de la mise à jour des définitions, « peluche » va proposer une modification qui sera prise en compte par « lion » et cette modification imprégnera la définition de « lion » du côté sympathique et mignon de « peluche » tout en enlevant la force et le charisme de la définition originale de « lion ».

c) Fonctionnement détaillé

Même si ce processus semble « magique », tout n'est qu'histoire d'additions, de multiplications, de vecteurs et de matrices. Ainsi, le mécanisme d'attention se sépare en quatre calculs : les **requêtes** (queries), les **clés** (keys), le **schéma d'attention** (attention pattern) et les **valeurs** (values). Les résultats de ces calculs ne sont pas uniques pour chaque mot, cela dépend des paramètres appris par le modèle lors de l'entraînement. Ainsi, un mot peut avoir différentes **requêtes**, **clés** et **valeurs**, le but de l'entraînement est de trouver celles qui donnent les meilleurs résultats :

- Une **requête** est un vecteur calculé à partir de la définition actuelle d'un mot. Elle permet d'encapsuler ce dont le mot a besoin pour préciser sa définition, ce que le mot demande pour pouvoir être mis à jour le plus efficacement possible (sa requête). Cette requête est

[25] 3Blue1Brown. (2024). https://youtu.be/eMlx5fFNoYc?si=A6MnFSpzXjQlK5xf
[26] Benyamin Ghojogh, Ali Ghodsi. (2020). https://hal.science/hal-04637647v1

donc un vecteur qui, de la même manière que les **word embeddings**, reflète un sens ou une idée dans notre langage ;

- Une **clé** est aussi un vecteur calculé à partir de la définition actuelle d'un mot ainsi que de sa position dans la phrase. Elle encapsule ce que le mot représente dans le contexte, il peut s'agir de son genre, de son nombre, de sa nature et bien plus. Il peut être intéressant de voir une clé comme le reflet d'une de ces caractéristiques et pas un concentré de toutes les informations. Cela permet à la tête d'attention de vraiment décomposer la complexité du langage en de petites briques bien plus faciles à traiter ;
- Le **schéma d'attention** est composé des correspondances entre les requêtes et les clés pour chaque paire de mots. C'est-à-dire, il évalue à quel point la clé répond à la requête (techniquement, en faisant un produit scalaire entre les deux vecteurs). Ainsi, dans la phrase « une belle maison s'est construite », le nom « maison » va peut-être demander dans sa requête s'il y a des adjectifs proches. Supposons que « belle » va indiquer sa nature d'adjectif dans sa clé, la correspondance de la clé de « belle » avec la requête de « maison » va être forte. **« Maison » prête attention à « belle ».** Au contraire, l'association de « construite » et « belle » sera moins forte, car la requête de « construite » ne correspond sûrement pas à la clé de « belle ». Un tableau à double entrée est donc créé, avec pour chaque mot la correspondance entre sa requête et toutes les autres clés. C'est le schéma d'attention, il permet de savoir quels mots pourraient influencer positivement la définition d'autres mots. Un exemple de ce schéma est présenté sur la figure 8. Verticalement les mots sont associés à leur clé. Horizontalement les mots sont associés à leur requête. Si la clé correspond à la requête, la case à l'intersection aura une couleur vive ;

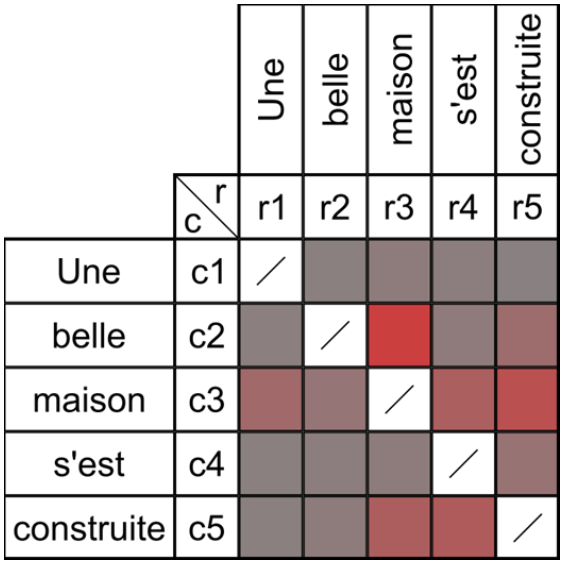


Figure 8 : Exemple d'un schéma d'attention. Plus la couleur est vive, plus la clé (c) correspond à la requête (r). On constate la réponse très positive de « belle » à la requête de « maison ».

- Une **valeur** est un vecteur calculé à partir de la définition actuelle d'un mot. Elle va contenir ce que le mot pourrait ajouter à la définition d'autres mots. Cet ajout est en rapport avec la clé. Dans l'exemple « une belle maison s'est construite », la valeur de « belle » va contenir ce que l'adjectif pourrait ajouter à la définition d'un nom. Ici la « maison » ne serait plus définie numériquement comme un lieu d'habitation banal, mais plutôt comme un endroit particulièrement beau. Si la clé de « belle » ne portait pas sur la nature d'adjectif mais plutôt le caractère féminin du mot, sa valeur apporterait une modification tournée vers le genre et plus vers la qualité. Il est important de noter que la valeur est constante et ne change pas en fonction du mot sur lequel elle est appliquée. Cela permet des calculs très

rapides : une seule modification par mot est calculée et cette modification est ajoutée à tous les mots (dans des proportions variables, déterminées avec le schéma d'attention).

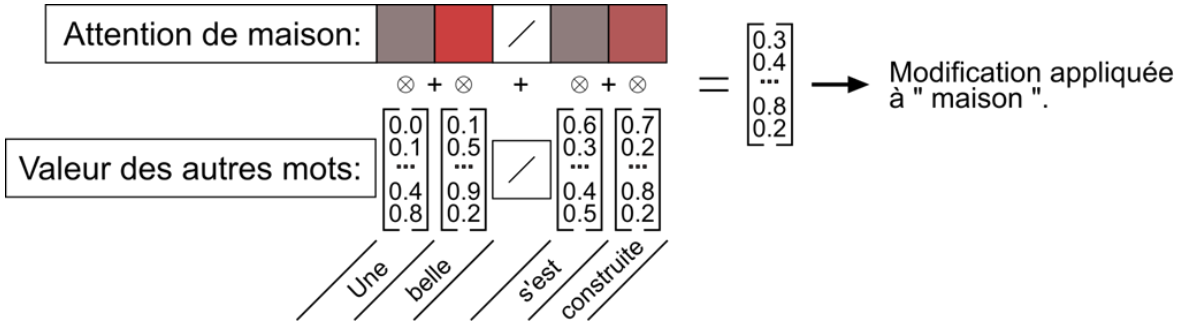


Figure 9 : Calcul de la modification appliquée à maison à partir de l'exemple précédent. Les nombres composant les vecteurs de valeurs ne représentent rien de spécial dans cet exemple. On peut constater que la modification finale est proche de la valeur de « belle », car c'est le mot sur lequel « maison » portait le plus d'attention.

Ainsi, le **schéma d'attention** résume les correspondances entre **requêtes** et **clés** et chaque **valeur** indique les modifications à faire. Si une clé correspond bien à une requête, cela veut dire que le mot émetteur de la requête a intérêt à se faire modifier par l'émetteur de la clé. Le mécanisme d'attention est donc le suivant : pour chaque mot d'un texte, on applique la valeur de tous les autres mots à la hauteur de la correspondance entre la requête du mot et les clés des autres.

Mathématiquement, pour calculer la modification sur la définition d'un mot, on procède de la manière suivante. Pour chaque autre mot du texte, la valeur de ce dernier est multipliée avec le produit scalaire de la requête du mot initial par la clé de ce dernier. Il suffit de faire la somme de ces valeurs pondérées et on obtient la modification. Dans un souci de stabilité et d'efficacité, différentes fonctions sont appliquées à la modification pour la normaliser.

4.4 - Transformeurs

a) Attention à tête multiple

Un transformeur est la mise en pratique de l'attention. Seule, une tête d'attention est peu efficace : elle ne modifie les définitions des mots que selon une requête et les mots ne proposent qu'une valeur. La modification n'est donc portée que sur un aspect précis du sens du mot, loin de changer drastiquement la compréhension de celui-ci. C'est pourquoi un transformeur va appliquer en parallèle un grand nombre de têtes d'attention avec dans chaque tête des requêtes, clés et valeurs différentes et combiner les résultats en une grosse modification. Cette modification aura un impact marqué sur les définitions. C'est l'**attention à tête multiple** (multi-head attention).

b) Feed-forward network

Grâce à l'attention, un transformeur peut modifier la définition de chaque mot d'un texte pour qu'elle reflète mieux le contexte. Néanmoins, en plus du contexte, il faudrait que notre modèle soit capable d'intégrer des connaissances et les appliquer au-delà du contexte, par exemple dans la phrase « je me trouve proche de la tour Eiffel », il est intéressant que la définition de « Paris » apparaisse. Cela ne va pas provenir des têtes d'attentions, car elles ne traitent que le contexte, il faudrait donc ajouter des paramètres dans notre transformeur qui stockent des connaissances générales. Ces connaissances vont être apportées par le biais d'un réseau de neurones « **feed-forward** » (ou **fully connected**, voir partie 1.3).

De manière simplifiée, le réseau de neurones va prendre en entrée la définition d'un mot et sortir une modification tirée de sa connaissance. Ainsi le mot « Eiffel » donnera sûrement une sortie

comprenant l'idée de « Paris », mais aussi « ingénieur » ou « acier ». Le transformeur applique cette modification au mot fourni en entrée pour préciser sa définition.

Après avoir vu ses définitions précisées selon le contexte par l'attention à tête multiple, un texte est passé dans le réseau de neurones pour que chaque mot soit imprégné de ce que le modèle sait à son propos.

c) Architecture

Pour construire un transformeur, il suffit d'empiler l'association **attention à tête multiples/réseau de neurones** les unes à la suite des autres. Ainsi chaque bloc va avancer de plus en plus profondément dans la compréhension du texte, affinant à un niveau extrême les définitions numériques de chaque mot.

Pour récapituler, un transformeur prend en entrée un texte dont chaque mot est traduit en une suite de nombre qui représente le sens de ce mot, formant un vecteur. C'est le **word embedding** qui fournit ces définitions. Ces définitions numériques sont passées en parallèle à un ensemble de blocs **: les têtes d'attention**. Ces têtes vont modifier la définition de chaque mot pour la rendre plus compatible avec le contexte. Après cela, les définitions sont encore précisées avec des **réseaux de neurones**, indépendamment des autres et en fonction de la connaissance du transformeur. Ces opérations sont répétées autant de fois que nécessaire sur la sortie du bloc précédent. Ce comportement est représenté graphiquement sur la figure 10.

Même si tous les exemples ont été basés sur du texte, les transformeurs permettent de traiter n'importe quelles données, tant qu'elles sont sous la forme d'une série de vecteurs. Ainsi, un signal audio, une image, une vidéo, un cours de bourse, des données météos, etc. peuvent être traitées par un transformeur. Cette architecture permet, à la différence de modèles plus classiques, de travailler avec des ensembles de données qui portent de l'information dans leur agencement. En effet, un texte serait incompréhensible si les mots étaient lus dans le désordre, tout autant qu'une image si tous ses pixels sont mélangés. Un transformeur permet de traiter des données en utilisant le sens de leur agencement en plus de leur sens intrinsèque. Un visage est composé de pixels majoritairement de la couleur de la peau et des cheveux (sens intrinsèque), ce qui permet de le reconnaître est bien l'agencement de ces pixels. C'est ici la puissance du transformeur et la raison de la révolution que cette architecture a créée.

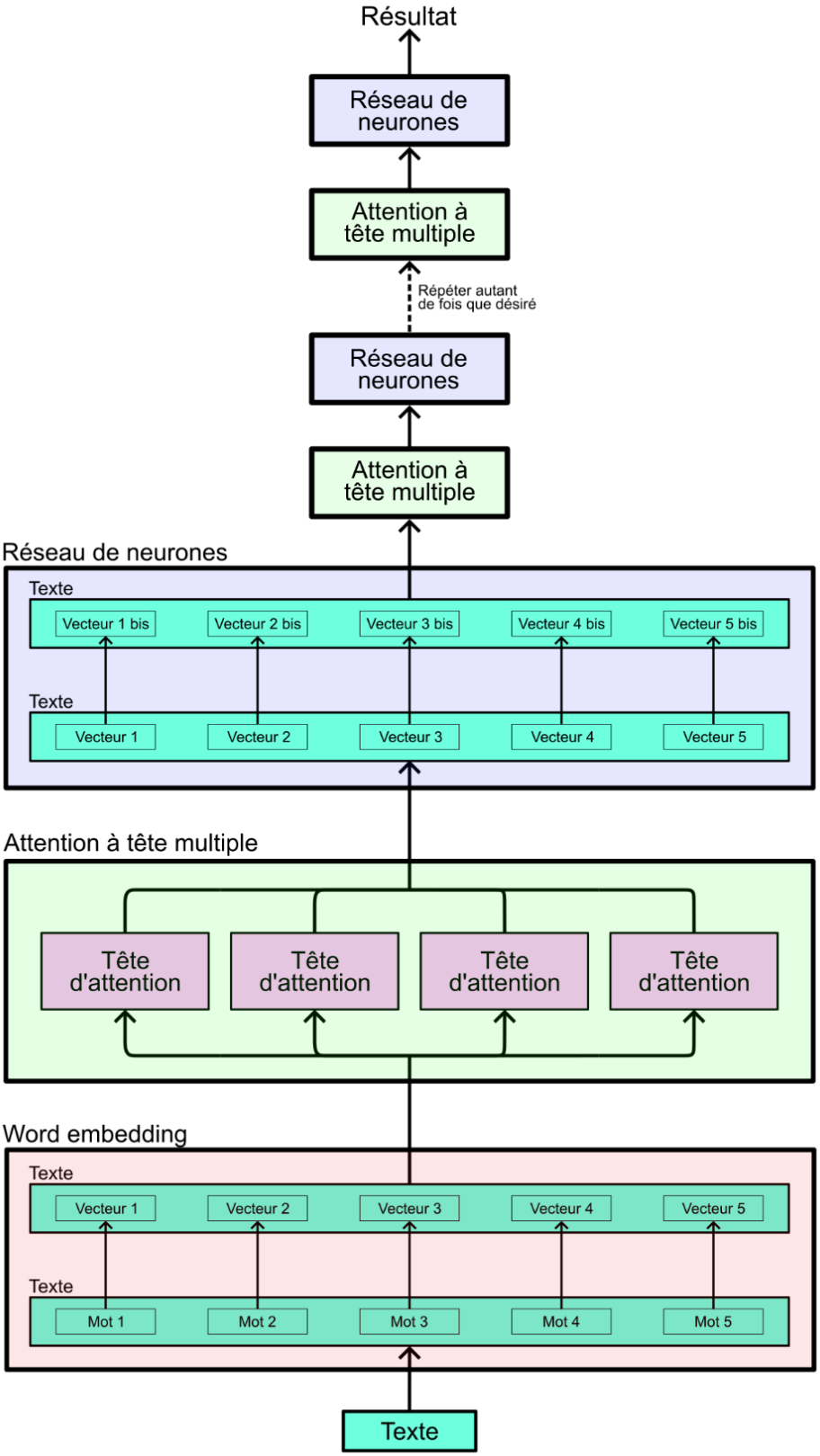


Figure 10 : Schéma de l'architecture simplifiée d'un transformeur appliqué à du texte

5 - Conclusion

Après avoir présenté le fonctionnement de modèles primitifs, avec les arbres de décisions, pour aller jusqu'à une méthode avancée et très actuelle, les transformeurs, je souhaitais souligner le but de cet article : désopacifier le monde de l'IA actuel.

En effet depuis 2022, quand ces méthodes d'IA générative ont pris une part non négligeable de notre quotidien, une partie d'entre nous s'est mise à massivement utiliser ces outils, que ce soit dans le cadre personnel, professionnel[27,28] et aussi dans le cadre des études[29]. Hélas bien souvent l'utilisateur ne sait pas ce qu'il se trame entre son **prompt** et la réponse du modèle. Connaître dans les grandes lignes le fonctionnement de ce type de technologie est crucial tant on leur délègue des tâches de plus en plus critiques. Par exemple, un sondage IPSOS[30] indique que 25% d'un panel de français sondés en décembre 2025 a déjà utilisé une IA générative (sûrement un **LLM** basé sur des transformeurs) pour se renseigner sur des personnalités ou partis politiques.

Cette utilisation n'est pas nécessairement une erreur, mais il est important de comprendre les mécanismes utilisés pour obtenir une réponse. Un **LLM** est essentiellement une machine créée pour reproduire au mieux l'expression d'un langage, et les exemples utilisés pour apprendre ce mode d'expression sont tirés de corpus rédigés par des humains. Il est donc normal d'attendre de ces modèles d'IA des biais humains et des méthodes de réflexion ou déduction humaines. Même si les développeurs de ces IA (OpenAI, Anthropic, Google, Meta, Mistral, …) cherchaient à éliminer ces caractéristiques humaines, les modèles resteront basés sur l'humain. Ainsi il est crucial de se souvenir que ces modèles ne sont pas des robots sans âme ni biais, capables de tout sans erreurs, mais bien des machines ayant comme seul but d'imiter l'humain (cela explique beaucoup de comportement indésirable de ces machines). Les LLM sont donc des modèles basés sur le comportement humain, et comme il y a des humains qui sortent de l'ordinaire, Victor Hugo, Albert Einstein, Marie Curie ou Sophie Germain, rien n'oblige un LLM à se limiter à un niveau standard. Des **LLM** de pointe arrivent même à de meilleures performances que la majorité d'entre nous en mathématique, programmation, expression écrite, etc.[31]. Ces modèles ultra performants sont souvent accessibles avec un abonnement payant, mais les versions gratuites ne sont pas pour autant incapables.

Pour finir, j'espère que cet article permet aussi d'enlever le voile magique qui tourne autour de cette technologie. Il est important de se souvenir que n'importe quel modèle est juste une grosse calculette qui calcule très vite. L'information prend la forme de nombre et est traitée par le modèle en lui appliquant des transformations mathématiques basiques (composées majoritairement d'additions et de multiplications) apprises lors de son entraînement. La complexité émerge avec la taille du modèle, certains ayant des centaines de milliards de paramètres. La même architecture avec seulement une poignée de paramètres serait comparable à une formule dans un tableur, capable de seulement calculer une opération simple comme une moyenne ou une somme. Rien de magique donc, seulement beaucoup d'opérations simples qui donnent des résultats bluffants à utiliser en connaissance de cause.

Références complémentaires :

[1]: LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015). https://doi.org/10.1038/nature14539

[2]: Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems* (Vol. 25).

[27] https://www.ipsos.com/fr-fr/intelligence-artificielle-quels-sont-les-usages-des-francais
[28] https://www.ipsos.com/fr-fr/ia-en-entreprise-etat-des-lieux-et-leviers-dacceleration
[29] https://www.ipsos.com/fr-fr/les-etudiants-face-lia-un-usage-generalise-devenu-difficile-contourner
[30] https://www.ipsos.com/fr-fr/pres-dun-francais-sur-deux-pret-utiliser-lia-generative-pour-sinformer-sur-la-politique
[31] https://artificialanalysis.ai/evaluations/artificial-analysis-intelligence-index

https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[3]: Vaswani, A. et al. (2017). Attention is All you Need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds), *Advances in Neural Information Processing Systems* (Vol. 30). Retrieved from https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[4]: wikipedia.org (Toutes les pages sur le machine learning sont plutôt justes et très complètes).

[5]: Deep Learning with PyTorch: A 60 Minute Blitz, https://docs.pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html. (Un exemple pratique d'implémentation de modèles, pour ceux qui ont des bases en programmation).

[6]: 3Blue1Brown, https://youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&si=g2tpGWw2EahCyqtm . (La meilleure ressource pour comprendre le monde des réseaux de neurones, très accessible, avec des visuels très percutants, clairs et en plus un doublage français de qualité).