



Introduction à l'intelligence artificielle et aux modèles génératifs

Pierre-Alexandre Mattei, Serena Villata

► To cite this version:

Pierre-Alexandre Mattei, Serena Villata. Introduction à l'intelligence artificielle et aux modèles génératifs. Bruno Martin; Sara Riva. Informatique Mathématique: Une photographie en 2022, CNRS Editions, 2022. hal-03849387

HAL Id: hal-03849387

<https://hal.science/hal-03849387v1>

Submitted on 13 Dec 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Chapitre 1

Introduction à l'intelligence artificielle et aux modèles génératifs

Pierre-Alexandre Mattei, Serena Villata
Université Côte d'Azur, Inria

L'intelligence artificielle est un champ polymorphe, caractérisé par son interdisciplinarité et sa situation particulière, au carrefour de plusieurs branches des mathématiques et de l'informatique. On offrira ici un panorama des différentes familles d'intelligence artificielle, ainsi que leur articulation historique. Puis, nous traiterons un exemple d'approche récente en détail : celui des modèles statistiques génératifs.

1.1 Introduction

Le terme "intelligence artificielle" (ci-après "IA") a été inventé par John McCarthy en 1956, à l'occasion d'un séminaire de deux mois (qu'il a organisé au Dartmouth College à Hanover, New Hampshire, États-Unis) qui a eu le mérite de faire se rencontrer dix chercheurs américains (sur la théorie des automates, les réseaux de neurones et l'intelligence) et de donner l'imprimatur au terme "intelligence artificielle" comme nom officiel d'un nouveau domaine de recherche.

Depuis lors, l'IA s'est établie et a évolué ; aujourd'hui, elle est reconnue comme une branche autonome, bien qu'elle soit liée à l'informatique, aux mathématiques, aux sciences cognitives, à la neurobiologie et à la philosophie.

De nombreuses définitions ont été données à ce sujet : elles diffèrent par les tâches effectuées par les machines que l'IA cherche à construire. Ces tâches peuvent être classées selon deux dimensions orthogonales [20] : les machines qui pensent ou agissent, et les machines qui simulent les humains (ou se comportent de manière rationnelle). Quatre classes au total, selon que les machines : pensent comme des humains, agissent comme des humains, pensent rationnellement, agissent rationnellement. Quatre approches distinctes de la recherche sur l'IA, donc, qui sont toutes activement poursuivies.

L'objectif de l'approche "les machines pensent comme les humains" est de reproduire le raisonnement humain dans les machines. Elle peut se faire à deux niveaux : en imitant les méthodes de raisonnement ou en reproduisant le fonctionnement du cerveau. Dans le premier cas, les sciences cognitives nous fournissent un point de départ important, obtenu par l'introspection et les expériences psychologiques. Dans le second cas, c'est la neurobiologie qui nous fournit un modèle approprié. Ce premier critère vise donc à produire des automates qui, non seulement se comportent comme des humains, mais aussi "fonctionnent" comme des humains.

L'objectif de l'approche des "machines qui se comportent comme des humains" est de produire des machines qu'il est impossible de distinguer des humains. Cette propriété a été mieux définie par Alan Turing qui, dans un article de 1950 [26], a proposé le test qui porte son nom : un "juge" est autorisé à poser des questions écrites à un "sujet" et, sur la base des réponses, doit décider s'il s'agit d'un humain ou d'une machine. Pour réussir le test de Turing, une machine doit présenter les capacités suivantes :

- le traitement du langage naturel, afin de communiquer efficacement dans la langue du juge ;
- la représentation des connaissances, afin de mémoriser ce que l'on sait ou ce que l'on importe ;
- le raisonnement automatique, pour déduire (produire), à partir de ses propres connaissances, les réponses au juge ;
- l'apprentissage automatique, pour augmenter sa base de connaissances.

Le test de Turing n'implique pas d'interaction physique entre le juge et la machine, car cela n'est pas nécessaire. Si l'on veut, on peut penser à un test de Turing total dans lequel, au lieu de réponses écrites, le juge reçoit un signal audio-vidéo, et a la possibilité de faire passer des objets à la machine par une fente. Dans ce cas, la machine doit également présenter les capacités suivantes : la vision artificielle, pour reconnaître les objets reçus ; la robotique, pour les manipuler ; le traitement de la parole, pour comprendre les questions du juge et y répondre. Le test de Turing total rappelle le test

Voight-Kampf du film Blade Runner, grâce auquel les policiers distinguent les androïdes des humains.

L'approche de la "machines pensant rationnellement" ne s'intéresse pas aux machines qui fonctionnent comme des humains, mais uniquement au raisonnement rationnel, le terme "rationnel" étant précisément défini par les mathématiques, y compris les techniques que les humains n'utilisent pas naturellement. La logique, par exemple, est l'étude de la manière de mener un raisonnement imparable. La logique joue un rôle important dans l'IA, même si l'on s'attend initialement à ce qu'elle ne soit pas utilisée par les humains ou très peu.

De même, la dernière des quatre classes alternatives d'IA se concentre sur l'approche "machines agissant rationnellement" qui utilise la définition de "l'action rationnelle" fournie en économie, à savoir : la sélection d'actions menant au meilleur résultat, ou au meilleur résultat attendu s'il existe des éléments d'imprévisibilité. L'objectif de cette approche est de créer un agent, une entité capable d'agir dans un environnement, afin d'atteindre un ou plusieurs objectifs. L'agent utilisera un raisonnement rationnel pour choisir les actions à effectuer, mais dans certains cas, il devra réagir à des stimuli environnementaux si rapidement qu'il "outrepassera" son choix (par exemple, lorsqu'une inaction menace son existence). Si l'on touche quelque chose de chaud, on réagit en retirant immédiatement la main, sans raisonnement conscient ; de même, l'agent, dans certaines situations, doit être capable d'agir sans raisonner. Les agents peuvent être de deux types : uniquement logiciels, auquel cas ils sont appelés *softbots*, ou à la fois matériels et logiciels, alors appelés *robots*. Dans le cas des softbots, l'environnement externe dans lequel ils opèrent est le Web, où ils interagissent avec des humains et d'autres softbots. C'est actuellement l'approche la plus poursuivie, car elle promet les résultats pratiques les plus utiles.

Afin de présenter les différentes techniques proposées par l'IA, nous les diviserons en deux grandes classes : symbolique et subsymbolique. La première propose d'automatiser le raisonnement et l'action, en représentant les situations objet de l'analyse par des symboles compréhensibles par l'être humain, et en les traitant par des algorithmes. La dernière, en revanche, ne représente pas explicitement les connaissances de manière directement compréhensible, et est basées sur la reproduction de phénomènes naturels à l'aide de méthodes statistiques. Nous détaillerons ensuite un exemple de cette approche statistique, à travers l'étude de modèles à variables latentes.

Les techniques symboliques ont été le paradigme dominant de l'IA du milieu des années 1950 à la fin des années 1980 et ont été appelées GOFAI ("Good Old-Fashioned Artificial Intelligence") dans [5]. Les techniques subsymboliques, et en particulier les réseaux de neurones, ont pris

leur essor depuis les années 1990 et ont remporté d'importants succès dans divers domaines tels que la vision par ordinateur. Récemment, la recherche s'est orientée vers la combinaison de techniques symboliques et subsymboliques.

1.2 IA symbolique

Parmi les techniques symboliques, nous décrirons la recherche dans l'espace des états, le raisonnement automatique et l'apprentissage automatique.

1.2.1 Recherche dans l'espace des états

Cette technique est utilisée lorsque l'on veut choisir une série d'actions menant d'un état initial à un ou plusieurs états finaux souhaités. Les conditions, pour qu'elle soit utilisée, sont que l'état du monde extérieur puisse être représenté de manière concise (sous forme symbolique), que les actions disponibles puissent être exprimées sous forme de règles de passage d'un état à l'autre, et qu'il existe un test pour établir si un état est final.

Examinons un exemple de problème qui peut être traité de cette manière. Dans le jeu de huit (ou puzzle de huit), nous avons un échiquier de trois cases sur trois, dans lequel huit cases sont occupées par huit tuiles numérotées de 1 à 8 et une case est vide. Les mouvements possibles consistent à déplacer une tuile numérotée adjacente vers la tuile vide. Pour ce problème, "l'état" consiste en la position des huit tuiles numérotées. Le but est de trier les tuiles de 1 à 8, en essayant de faire le moins de déplacements possible.

C'est un problème qui semble exiger de l'intelligence : un être humain le résoudrait en essayant différents mouvements et en tentant de prédire le résultat. La méthode de résolution proposée par l'IA consiste à effectuer une recherche dans l'espace des états possibles. À cette fin, on peut représenter "l'espace" comme un arbre dans lequel chaque nœud correspond à un "état". La racine de l'arbre est l'état initial, les enfants d'un nœud sont les états qui peuvent être atteints à partir de l'état associé au nœud en appliquant un seul déplacement.

Le problème est résolu lorsqu'un chemin de l'état initial à un état final a été trouvé. En général, il ne suffit pas de trouver une solution, mais celle qui a le coût minimum. Il est donc nécessaire de définir un "coût" : normalement, un coût est attribué aux différents coups, et le coût d'un chemin est mesuré comme la somme des coûts des coups qui le composent.

Dans le cas du jeu de huit, chaque coup coûte 1, et on cherche la solution nécessitant le nombre minimal de coups.

Un autre problème similaire est le suivant. Le problème "missionnaires et cannibales" consiste à faire traverser une rivière à 3 missionnaires et 3 cannibales, en utilisant un bateau et en empêchant les cannibales de manger les missionnaires (lorsqu'ils sont plus nombreux sur les deux rives). Le bateau ne peut contenir que deux personnes à la fois et doit être dirigé par au moins une personne pour se déplacer d'un côté à l'autre de la rivière. L'état de ce problème peut être représenté à l'aide d'un triplet de nombres, dans lequel les deux premiers quantifient les missionnaires et les cannibales sur la banque initiale, et le troisième est égal à 1 si le bateau est sur la rive initiale, et 0 s'il est sur l'autre rive. L'état initial est (3,3,1), l'état final est (0,0,0) et un état possible (2,2,1) indique qu'il y a deux missionnaires et deux cannibales sur la rive initiale et que le bateau est sur la rive initiale. Cinq opérations sont possibles : traverser la rivière avec deux missionnaires, avec deux cannibales, avec un missionnaire et un cannibale, avec un seul cannibale ou avec un seul missionnaire. Toutes les opérations ne sont pas autorisées dans tous les états : par exemple, à partir de l'état (2,2,1), il n'est pas possible d'appliquer l'opération "traverser la rivière avec un missionnaire" car sur la rive initiale il resterait un missionnaire et deux cannibales, et alors les cannibales mangeraient le missionnaire. Le coût, dans ce cas, est unitaire pour toutes les opérations, on cherche donc des solutions avec le minimum de traversées de rivière.

Passons à des problèmes réels, comme le "calcul d'itinéraire", qui consiste à aller d'un endroit à un autre au coût minimum, en passant par des endroits intermédiaires, avec un coût associé à chaque lien. Un exemple est le voyage en voiture d'une ville à une autre : les liens sont les routes et le coût peut être la distance ou le temps nécessaire pour parcourir ce lien. Si vous voyagez en avion, les liens correspondent aux vols disponibles, et le coût peut correspondre à la durée ou au prix du vol. Les "états" sont les lieux, et les mouvements disponibles consistent à utiliser l'un des liens du lieu actuel pour se déplacer vers un autre lieu.

Comment résoudre les problèmes de recherche ? L'espace de recherche doit être généré et parcouru à partir du nœud initial jusqu'à ce que l'on trouve un état qui passe le test d'état final par vérification ; lorsque le test échoue, le nœud doit être "étendu", ses successeurs générés à l'aide de déplacements ou d'opérateurs possibles, et examinés jusqu'à ce que l'un d'entre eux passe le test, ou jusqu'à ce que tout l'espace des états soit exploré. L'arbre de recherche est ainsi généré progressivement.

Les algorithmes de recherche dans l'espace d'état diffèrent par le choix

du nœud à développer (stratégie de recherche). Les deux stratégies les plus courantes sont la recherche en profondeur et la recherche en largeur.

Dans la recherche en profondeur, le nœud ayant la plus grande profondeur qui a été généré mais pas encore développé est toujours développé. Cela signifie que l'on procède d'abord en profondeur jusqu'à ce que l'on arrive à un nœud qui ne peut plus être développé ou pour lequel on ne peut pas trouver de solution. Dans le premier cas, on part des nœuds du niveau de profondeur précédent et ainsi de suite. Dans la recherche en largeur, par contre, les nœuds de moindre profondeur qui ont été générés mais pas encore développés sont toujours développés. Dans ce cas, la racine de l'arbre est développée en premier, puis tous ses enfants, puis tous les enfants des enfants et ainsi de suite.

1.2.2 Raisonnement automatique

Le raisonnement automatique est l'utilisation de connaissances afin d'en déduire de nouvelles. À cette fin, il est nécessaire de représenter les connaissances dans un format qui peut être stocké par un ordinateur et utilisé pour faire des inférences. Ces exigences limitent le format de représentation aux langages formels, c'est-à-dire aux langages dont la syntaxe et la sémantique sont précisément définies.

L'un des langages formels les plus étudiés est la logique. Elle trouve son origine dans la philosophie et les mathématiques de la Grèce antique. Le père fondateur de la logique en tant que discipline autonome peut être considéré comme Aristote (vers 384-321 av. J.-C.), tandis que Chrysippe de Soli (vers 280-205 av. J.-C.), de l'école stoïcienne, a défini les connecteurs logiques, les axiomes et les règles fondamentales de la logique propositionnelle.

La naissance de la logique mathématique moderne remonte à George Boole (1815-1864), qui a publié en 1847 une méthode permettant de décrire la théorie des syllogismes aristotéliciens et de la logique propositionnelle sous la forme d'équations algébriques, et a proposé une procédure mécanique pour leur résolution. Gottlob Frege (1848-1925) a été le premier à développer un système d'axiomes et de règles pour la logique du premier ordre, dépassant ainsi les limites imposées par les syllogismes et la logique propositionnelle.

En 1965, John Robinson a publié la méthode de résolution, qui permet une automatisation efficace de l'inférence déductive en logique du premier ordre. La programmation logique s'appuie sur cette méthode, et notamment sur le langage Prolog (PROgramming in LOGic), dont les bases ont été posées par des chercheurs des universités d'Édimbourg et de Marseille au

début des années 1970. En particulier, Robert Kowalski, à Édimbourg, a travaillé à la définition des fondements théoriques de la programmation logique, et a proposé une interprétation procédurale des formules logiques qui permet de réduire le processus de preuve d'un théorème à un processus de calcul sur un ordinateur traditionnel. Alain Colmerauer, à Marseille, a été le premier à créer un interpréte pour le langage Prolog en 1972.

En logique propositionnelle, les unités élémentaires sont des propositions atomiques, c'est-à-dire des énoncés qui ne peuvent être décomposés davantage et qui peuvent être vrais ou faux. Les propositions arbitraires ou formules propositionnelles sont obtenues à partir de formules atomiques en les combinant à l'aide de connecteurs logiques : négation, conjonction, disjonction et implication.

En logique du premier ordre, les formules atomiques se distinguent de celles la logique propositionnelle car elles peuvent avoir un ou plusieurs arguments. Les arguments sont des termes, c'est-à-dire des représentations d'un individu dans le domaine du discours. Dans les cas les plus simples, les termes sont variables s'ils indiquent un individu non spécifié, ou constants s'ils déterminent un individu spécifique. Dans ce qui suit, nous utiliserons la convention Prolog qui consiste à utiliser des mots commençant par des lettres minuscules pour désigner les constantes, et des mots commençant par des lettres majuscules pour désigner les variables. Les propositions deviennent des prédicats et expriment les propriétés de leurs arguments. Par exemple, $p(a)$ exprime le fait que "l'individu a est p " ou "l'individu a a la propriété p " et $q(a, b)$ exprime le fait que "le couple d'individus a et b est q " ou "le couple a et b a la propriété q ", c'est-à-dire que " a et b sont liés par la relation q ". Des exemples de formules atomiques en logique propositionnelle sont $man(socrates)$, qui signifie que "Socrate est un homme", et $father(paul, peter)$, qui indique que Paul et Pierre sont liés par la relation $father$, c'est-à-dire que Paul est le père de Pierre. Les formules atomiques en logique du premier ordre peuvent être combinées avec les mêmes connecteurs logiques qu'en logique propositionnelle. En outre, de nouvelles formules peuvent être obtenues en utilisant les quantificateurs (\exists et \forall).

Les systèmes experts ont été développés dans de nombreux domaines. Le premier et peut-être le plus connu est Mycin, créé dans les années 1970 à l'université de Stanford par Edward Shortliffe. Son objectif était de diagnostiquer les maladies infectieuses du sang et de recommander des antibiotiques, dont la posologie était adaptée au poids du patient. Le système a bien fonctionné, mais n'a jamais été utilisé en raison de problèmes juridiques.

Voici quelques domaines dans lesquels les systèmes experts peuvent être utilisés :

- le diagnostic, qui consiste à tenter de détecter une maladie chez un être humain ou un dysfonctionnement d'une machine sur la base de symptômes, c'est-à-dire de manifestations observables de la maladie ou du dysfonctionnement;
- la surveillance, dont l'objectif est de garder un processus sous contrôle en recueillant des informations et en faisant des estimations sur son déroulement;
- la planification, qui vise à atteindre un certain objectif avec les ressources disponibles;
- l'interprétation des informations et des signaux, dont le but est d'identifier l'occurrence de situations particulières d'intérêt dans les données d'entrée.

Le développement d'un système expert nécessite l'écriture de règles générales sur le domaine, qui doivent être recueillies en interrogeant un expert du domaine. Ce processus, connu sous le nom d'extraction de connaissances, s'est avéré extrêmement long et difficile. Afin de l'automatiser, il est possible d'utiliser l'apprentissage automatique, abordé dans la section suivante.

1.2.3 Apprentissage automatique

En 1984, Simon a donné la définition suivante de l'apprentissage [22] : "L'apprentissage consiste en des changements dans le système qui sont adaptatifs, dans le sens où ils permettent au système d'exécuter la même tâche ou des tâches tirées de la même population de manière plus efficace et efficiente la prochaine fois". Il est certain que pour fabriquer des machines que l'on peut qualifier d'intelligentes, il est nécessaire de leur donner la capacité d'étendre leurs connaissances et leurs compétences de manière autonome. Les deux principales utilisations de l'apprentissage automatique sont l'extraction de connaissances et l'amélioration des performances d'une machine.

Les connaissances extraites peuvent ensuite être utilisées par une machine comme base de connaissances d'un système expert, ou par des humains, par exemple dans le cas de la découverte de nouvelles théories scientifiques. L'amélioration des performances d'une machine passe, par exemple, par l'augmentation des capacités perceptives et motrices d'un robot.

Comme les techniques d'IA en général, les techniques d'apprentissage peuvent être divisées en techniques symboliques et sous-symboliques.

La technique la plus intéressante de l'apprentissage symbolique est l'apprentissage inductif : le système part de faits et d'observations provenant d'un instructeur ou de l'environnement, et les généralise pour obtenir des connaissances qui, espérons-le, sont également valables pour des cas non encore observés (induction).

Dans l'apprentissage inductif par les exemples, l'enseignant fournit un ensemble d'exemples et de contre-exemples d'un concept, et le but est de déduire une description du concept lui-même. Un exemple consiste en la description d'une instance du domaine du discours et d'une étiquette ; cette dernière peut être + si l'instance appartient au concept à apprendre, ou – si l'instance n'y appartient pas (contre-exemple). Dans le premier cas, on parle d'une instance appartenant à la classe positive et dans le second d'une instance appartenant à la classe négative. Un concept n'est donc rien d'autre qu'un sous-ensemble de l'ensemble de toutes les instances possibles du domaine de discours, ou univers. L'ensemble d'exemples et de contre-exemples fournis par l'enseignant est appelé jeu d'apprentissage. La description du concept à apprendre doit être telle qu'elle puisse être utilisée pour décider si une nouvelle instance, n'appartenant pas au jeu d'apprentissage, appartient ou non au concept.

Les systèmes d'apprentissage, qu'ils soient issus de langages attributs-valeurs ou de programmes logiques, ont eu un large éventail d'applications, allant du diagnostic des maladies à la prédiction des relations structure-activité dans la conception des médicaments, en passant par la prédiction de la cancérogénicité des substances chimiques. Avec la quantité croissante de données qui sont stockées chaque jour par les entreprises et les organisations en général, les algorithmes d'apprentissage deviennent de plus en plus importants car ils permettent d'extraire de cette masse de données des informations cachées, nouvelles et potentiellement utiles. C'est ce qu'on appelle l'exploration de données, ou l'extraction de connaissances à partir de données brutes.

1.3 IA sub-symbolique

Nous allons maintenant nous intéresser à deux techniques subsymboliques : les réseaux de neurones et les algorithmes génétiques.

1.3.1 Les réseaux de neurones

L'idée de simuler le fonctionnement du cerveau humain et animal afin d'obtenir un comportement intelligent précède le développement de

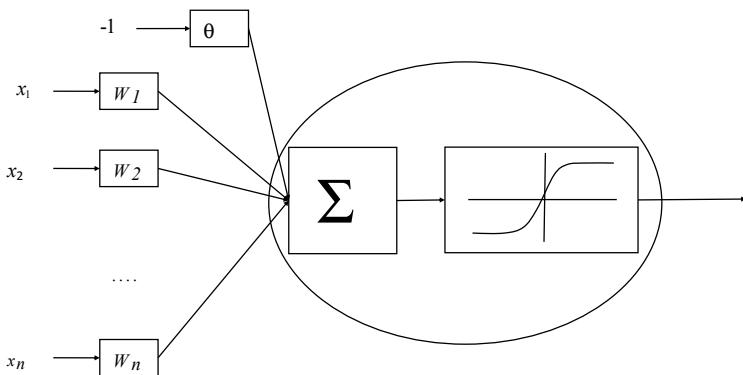


FIGURE 1.1 – Un modèle de neurone artificiel avec une fonction sigmoïde.

l'ordinateur, en particulier à l'article de 1943 de McCulloch et Pitts [15] dans lequel un modèle mathématique du neurone humain a été proposé et il a été montré comment les réseaux composés de tels neurones artificiels étaient capables de représenter des fonctions booléennes complexes.

Le modèle de neurone le plus populaire actuellement est appelé neurone sigmoïde et consiste en une unité avec n entrées numériques et une sortie numérique. La sortie est calculée en fonction des entrées de la manière suivante : chaque entrée x_i est multipliée par un poids W_i , les produits de ces multiplications sont additionnés et le résultat est donné en entrée d'une fonction sigmoïde. Un modèle de ce type de neurone est présenté en Figure 1.1, ainsi que l'aspect de la fonction sigmoïde. Notons que dans la somme, il y a un terme constant qui est assimilé à une entrée commune en supposant que cette entrée est toujours à -1 et que le poids de cette entrée vaut i . Ce modèle se comporte de la même manière qu'un neurone naturel : il est "activé" lorsqu'il reçoit les "bonnes" entrées et "désactivé" lorsqu'il reçoit les "mauvaises" entrées. Un neurone est actif lorsque sa sortie est proche de $+1$ et inactif lorsque sa sortie est proche de -1 . Les valeurs des poids d'entrée déterminent quelles entrées sont bonnes ou mauvaises : des valeurs positives des poids font que les entrées relatives tendent à conduire le neurone vers l'activation et des valeurs négatives vers la désactivation, et vice versa dans le cas de poids négatifs. Les neurones sont ensuite connectés les uns aux autres dans des réseaux, de sorte que la sortie d'un neurone peut être l'entrée d'autres neurones et que son activation affecte l'activation des neurones en aval. Le neurone sigmoïde dérive du perceptron proposé en 1962 par Rosen-Blatt : il en diffère car au lieu d'une fonction sigmoïde, le perceptron a une fonction à pas, c'est-à-dire une fonction qui vaut 0 pour les valeurs inférieures à 0 et 1 pour les valeurs supérieures ou égales

à 0. Un seul neurone sigmoïde est capable de représenter une certaine classe de concepts en fonction de ses poids : en particulier, il est capable de représenter les concepts dans lesquels les exemples sont séparés des contre-exemples par un hyperplan (en imaginant de considérer l'espace des entrées comme un espace euclidien). Puis ils isolent dans l'espace des entrées un demi-espace, c'est-à-dire une région délimitée par un hyperplan (dans le cas de deux entrées, il s'agit d'une droite). Afin de représenter des concepts plus complexes, il est nécessaire de composer les neurones en réseaux.

Les réseaux les plus simples sont appelés réseaux feedforward et sont constitués de couches de neurones : les entrées sont connectées à la première couche de neurones, les sorties de la première couche de neurones sont connectées aux entrées de la deuxième couche, et ainsi de suite, jusqu'à atteindre la dernière couche dont les sorties deviennent les sorties du réseau. Les couches de neurones de la première à l'avant-dernière couche sont dites "cachées". Ainsi, un réseau sigmoidal à une couche cachée pourra correspondre par exemple à la fonction $f_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^q$ définie par

$$f_\theta(x) = W_1 \text{sigmoïde}(W_0 x + b_0) + b_1, \quad (1.1)$$

où les poids du réseau sont les coefficients des matrices $W_1 \in \mathbb{R}^{h \times q}$, $W_1 \in \mathbb{R}^{d \times h}$ et des vecteurs (dits "biais") $b_0 \in \mathbb{R}^h$, $b_1 \in \mathbb{R}^q$. Le vecteur $\theta = (W_1, W_0, b_1, b_0)$ contient tous les poids du réseau. L'entier h correspond au nombre d'unités cachées. Plus il sera grand, et plus le réseau pourra modéliser des fonctions complexes.

Mais comment obtenir un réseau qui identifie un concept ? Contrairement aux systèmes basés sur la connaissance, dans les réseaux neuronaux, le choix des poids à la main serait trop complexe. Pour cela, des algorithmes d'apprentissage sont utilisés. Dans ce cas également, nous disposons d'un ensemble d'apprentissage, qui contient un ensemble de paires (entrées, sorties), à la différence que les entrées sont toutes continues, que les sorties peuvent être multiples et qu'elles sont également continues, c'est-à-dire qu'elles ne sont pas des + ou des - mais des nombres réels (comme par exemple dans l'équation (1.1)). Dans ce cas, on recherche la valeur des poids pour laquelle une certaine fonction de l'erreur sur l'ensemble d'apprentissage est minimale, c'est-à-dire une fonction des différences entre les sorties de l'ensemble d'apprentissage et celles du réseau lorsque les valeurs de la paire sont fournies en entrée. La fonction la plus utilisée est la somme des erreurs quadratiques. Comme on le verra dans la section suivante, quand on dispose d'un modèle probabiliste, la fonction d'erreur choisie est l'opposé de la vraisemblance du modèle.

L'algorithme le plus largement utilisé pour l'apprentissage dans les réseaux neuronaux multicouches est appelé rétropropagation (*backpropagation*). Le nom de "backpropagation" et a été proposé par Rumelhart, Hinton et Williams en 1986. Elle consiste à calculer l'erreur de la couche de sortie du réseau sur chaque exemple et à la propager en arrière vers les neurones des couches cachées. Sur la base de l'erreur propagée, les poids des neurones sont ensuite mis à jour. Si, après avoir considéré tous les exemples de cette manière, l'erreur est tombée en dessous d'un seuil prédéfini, on s'arrête, sinon tous les exemples de l'ensemble d'apprentissage sont considérés à nouveau.

Les réseaux multicouches profonds ont connu un grand succès et ont été appliqués dans de nombreux domaines, notamment la vision par ordinateur, la reconnaissance vocale, la traduction automatique, la bioinformatique, la conception de médicaments et l'analyse d'images médicales. Dans le domaine de la vision par ordinateur, par exemple, ils ont été capables de reconnaître des caractères manuscrits, de reconnaître des objets dans des images et des vidéos et de classer des images. Ces résultats ont été obtenus à l'aide d'un type particulier de réseau neuronal appelé "convolutif". Dans ce réseau, certaines couches appliquent une opération de convolution à l'entrée : un filtre est appliqué à l'entrée bidimensionnelle pour produire une image filtrée. Ces filtres, qui sont entraînés conjointement avec les paramètres des couches traditionnelles, identifient des caractéristiques de l'image de complexité croissante : par exemple, une première couche convulsive pourrait identifier des bords rectilignes approximatifs dans l'image d'entrée, une deuxième couche des combinaisons de bords rectilignes (angles) et ainsi de suite, jusqu'à identifier les caractéristiques de l'image qui servent à la classer.

1.3.2 Les algorithmes génétiques

Alors que les réseaux neuronaux s'inspirent du cerveau humain pour produire un comportement intelligent, les algorithmes génétiques s'inspirent de la théorie de l'évolution. Il s'agit d'algorithmes de recherche dans l'espace des états, dans lequel un état est considéré comme un individu, au sein d'une population d'individus qui est amené à évoluer selon les lois de l'évolutionnisme afin d'obtenir des états qui sont de bonnes solutions au problème.

Pour appliquer un algorithme génétique, il est nécessaire de représenter l'état comme une séquence de symboles (dans le cas le plus fréquent une séquence de bits), qui représente le patrimoine génétique (ou génotype) d'un individu et le caractérise complètement. Il est alors nécessaire

de disposer d'une fonction d'aptitude qui, compte tenu d'une séquence de symboles, indique le degré d'aptitude de l'individu, c'est-à-dire sa capacité à survivre dans son environnement. Dans le cas d'un algorithme génétique, la fonction de *fitness* représente la proximité de l'état par rapport à une solution ou sa qualité en tant que solution.

Un algorithme génétique commence avec une population initiale d'individus générés aléatoirement. Il exécute ensuite un cycle qui se termine lorsque la fitness du meilleur individu dépasse un certain seuil, c'est-à-dire lorsque la population contient une solution suffisamment bonne. À chaque étape du cycle, une nouvelle population est générée à l'aide des opérateurs de sélection, de croisement et de mutation. En pratique, chaque itération du cycle correspond à une génération. La nouvelle population est générée en sélectionnant des paires d'individus au hasard, mais avec une probabilité qui dépend de leur aptitude : les individus ayant une meilleure aptitude ont plus de chances d'être sélectionnés. Ensuite, on applique l'opérateur de croisement qui, étant donné le couple d'individus, en produit un autre, obtenu en "mélangeant" le patrimoine génétique de différentes manières : dans le cas de génotypes présentés comme des séquences de bits de longueur fixe n , l'opérateur de croisement le plus simple choisit un nombre entier aléatoire i inférieur à n et copie dans le premier descendant les premiers bits du premier parent et les derniers $n - 1$ bits du second parent, tandis que dans le second descendant il copie l'inverse.

Les descendants ainsi obtenus sont ensuite soumis à une mutation, dans laquelle de petites modifications du génotype sont apportées au hasard. Le processus de sélection, de croisement et de mutation est répété jusqu'à l'obtention d'une nouvelle population de taille fixe. La fitness de tous les individus de la nouvelle population est ensuite calculée. Il existe de nombreuses variantes de ce type d'algorithme. Par exemple, dans certains, une partie de l'ancienne population est transférée directement dans la nouvelle, en utilisant la sélection. Les algorithmes génétiques peuvent également être utilisés pour effectuer des tâches d'apprentissage automatique. Dans ce cas, il est nécessaire de représenter les descriptions du concept à apprendre comme une séquence de symboles : l'aptitude sera donnée par la précision avec laquelle une description du concept classe les exemples de l'ensemble d'apprentissage.

Les algorithmes génétiques ont eu de nombreuses applications en biologie, en ingénierie et dans les sciences physiques et sociales. L'une des plus intéressantes est la programmation automatique, c'est-à-dire la génération automatique de programmes informatiques pour résoudre un certain problème. Dans ce cas, le génotype des individus est constitué d'arbres représentant un seul programme dans un langage de programmation donné.

L'opérateur de croisement consiste à remplacer un sous-arbre d'un parent par un sous-arbre de l'autre parent. La fonction de fitness est calculée en exécutant le programme sur un ensemble de données d'entrée.

1.4 Modèles génératifs et apprentissage statistique

La tâche générale de l'apprentissage statistique est d'apprendre un objet mathématique (par exemple une fonction permettant de réaliser des prédictions) à partir d'une base de données dite d'entraînement ou d'apprentissage. Voici quelques exemples :

- en apprentissage supervisé, il s'agit d'apprendre une fonction permettant de prédire la valeur d'une réponse y (par exemple la sévérité d'une maladie) à partir de données x (par exemple une image médicale);
- en classification non-supervisée (aussi appelée *clustering*), il s'agit d'apprendre une partition de l'espace dans lequel vivent les données, afin de pouvoir les grouper en classes homogènes;
- en apprentissage par renforcement, il s'agit d'apprendre une fonction qui à chaque situation préconise une action à entreprendre (par exemple dans le cas d'un algorithme de jeu d'échecs).

Dans tous ces cadres, les fonctions à apprendre dépendent de la loi de probabilité (inconnue) des données observées. Une approche générale serait donc d'apprendre dans un premier temps cette loi de probabilité, puis de l'utiliser afin de résoudre le problème spécifique qui nous intéresse. La tâche générale d'apprentissage d'une loi de probabilités à partir de données est un des problèmes fondamentaux des statistiques, souvent appelé *estimation de densité*. L'un des avantages de cette approche est qu'elle permet souvent d'être capable de générer de nouvelles "fausses" données après la phase d'apprentissage, c'est pourquoi l'on parle de *modèles génératifs*. La génération de nouvelles données synthétiques est en effet utile dans de nombreux cadres :

- elle peut nous permettre de juger si le modèle que l'on a appris est bon : en effet, si les nouvelles données sont très différentes de la base de données originales, cela falsifiera le modèle et nous encouragera à l'améliorer ;
- dans certains cas applicatifs, la génération de nouvelles données est un objectif en soi (par exemple, la génération de nouvelles molécules en médecine [10]);
- si certaines données sont incomplètes, un modèle génératif peut permettre de compléter (voir par exemple [13]).

En guise d'exemple fil rouge, nous nous baserons sur une base de données très classique appelée MNIST¹. MNIST est constituée de 60000 images de chiffres calligraphiés de 28×28 pixels. Pour simplifier, on considérera une version binarisée de ces images, où chaque pixel ne peut prendre que les valeurs 0 ou 1. Les données vivent donc dans $\{0, 1\}^{28 \times 28}$, et suivront donc une loi inconnue sur cet espace discret. Quelques numéros de cette base de données sont montrés sur la Figure 1.2.

1.4.1 Apprentissage par maximum de vraisemblance

On suppose avoir affaire à des données $x_1, \dots, x_n \in \mathcal{X}$ où \mathcal{X} est un espace équipé d'une mesure de référence (par exemple la mesure de Lebesgue si les données sont continues, ou la mesure de comptage si elles sont discrètes, comme dans notre exemple fil rouge). On suppose que ces x_1, \dots, x_n sont autant de réalisations indépendantes et identiquement distribuées (i.i.d.) d'une variable aléatoire X , admettant une densité $p_{\text{données}}$ vis-à-vis de notre mesure de référence. La densité $p_{\text{données}}$ est inconnue, et nous souhaitons l'approximer à l'aide d'un *modèle paramétrique*, à savoir une famille de densités $(p_\theta)_{\theta \in \Theta}$, indexée par un ensemble appelé ensemble des paramètres (généralement inclus dans un espace vectoriel de dimension finie). L'idée serait donc d'utiliser nos données afin de trouver un $\hat{\theta} \in \Theta$ tel que $p_{\hat{\theta}} \approx p_{\text{données}}$. Cette tâche, généralement appelée *estimation statistique*, peut être effectuée de bien des manières, selon le choix du sens que l'on souhaite donner à l'assertion " $p_{\hat{\theta}} \approx p_{\text{données}}$ ". Nous allons nous concentrer ici sur l'exemple le plus connu de technique d'estimation, à savoir *l'estimation par maximum de vraisemblance*, introduite par Fisher au début du siècle dernier. Parmi les méthodes concurrentes, les méthodes bayésiennes (détaillées par exemple dans le livre de Robert [19]) sont historiquement les plus importantes.

La question principale permettant d'établir une technique d'inférence est donc : "quel sens donner à $p_{\hat{\theta}} \approx p_{\text{données}}$?". Une idée naturelle serait de se donner une distance d sur l'ensemble des densités, puis de choisir

$$\hat{\theta} \in \operatorname{argmin}_{\theta \in \Theta} d(p_\theta, p_{\text{données}}). \quad (1.2)$$

Cette idée se heurte à un écueil important : nous ne connaissons pas $p_{\text{données}}$, donc nous ne pourrons certainement pas minimiser la fonction $\theta \mapsto d(p_{\hat{\theta}}, p_{\text{données}})$. En revanche, nous avons accès à n tirages i.i.d. de $p_{\text{données}}$: notre jeu de données x_1, \dots, x_n . Tout le jeu va consister à trouver

1. <http://yann.lecun.com/exdb/mnist/>

un moyen d'utiliser ces données pour résoudre approximativement le problème (1.2). C'est ici que le choix de la distance peut faire toute la différence : pour certaines distances, les choses seront en effet plus simples que pour d'autres. Le maximum de vraisemblance correspond à une notion distance issue de la théorie de l'information : la *divergence de Kullback-Leibler*.

Définition 1.4.1. Soient p_1 et p_2 deux densités sur \mathcal{X} telles que le support de p_1 est inclus dans le support de p_2 . La divergence de Kullback-Leibler entre p_1 et p_2 est la quantité

$$\text{KL}(p_1, p_2) = \int_{\mathcal{X}} \log \left(\frac{p_1(x)}{p_2(x)} \right) p_1(x) dx = \mathbb{E}_{X \sim p_1} \left[\log \left(\frac{p_1(X)}{p_2(X)} \right) \right]. \quad (1.3)$$

Malheureusement, la divergence de Kullback-Leibler n'est pas une véritable distance sur l'espace des densités. En effet, l'axiome de symétrie est violé car on n'a pas, en général, $\text{KL}(p_1, p_2) = \text{KL}(p_2, p_1)$. le cas gaussien, pour lequel la divergence a une forme explicite, produit déjà un exemple de cette asymétrie.

Proposition 1.4.2. Pour tous $\mu_1, \mu_2 \in \mathbb{R}$, et σ_1, σ_2 , on a

$$\text{KL}(\mathcal{N}(\mu_1, \sigma_1), \mathcal{N}(\mu_2, \sigma_2)) = \log \frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}. \quad (1.4)$$

Si elle n'est pas une distance, la divergence de Kullback-Leibler possède tout de même un certain nombre de bonnes propriétés, résumées dans la proposition suivante.

Proposition 1.4.3. Soient p_1 et p_2 deux densités sur \mathcal{X} telles que le support de p_1 est inclus dans le support de p_2 . On a

1. $\text{KL}(p_1, p_2) \geq 0$,
2. $\text{KL}(p_1, p_2) = 0 \iff p_1 = p_2$.

Ces bonnes propriétés étant essentiellement des conséquences de l'inégalité de Jensen, il paraît naturel de se dire que l'on pourrait remplacer la fonction logarithme par une autre fonction convexe. Ce type de généralisation est à la base de la notion de f -divergences. Cependant, le logarithme dispose d'une autre qualité qui sera (comme on va bientôt le voir) fondamentale : sa propriété de morphisme vis-à-vis de l'addition et de la multiplication. Cette propriété explique la place centrale tenue par la divergence de Kullback-Leibler au sein des (pseudo)distances entre lois de probabilités.

Revenons à notre problème d'estimation statistique. On s'intéresse à minimiser $\text{KL}(p_{\text{données}}, p_\theta)$. En utilisant la propriété de morphisme du logarithme, on peut réécrire

$$\text{KL}(p_{\text{données}}, p_\theta) = \mathbb{E}_{X \sim p_{\text{données}}} \left[\log \left(\frac{p_{\text{données}}(X)}{p_\theta(X)} \right) \right] \quad (1.5)$$

$$= \mathbb{E}_{X \sim p_{\text{données}}} [\log p_{\text{données}}(X)] - \mathbb{E}_{X \sim p_{\text{données}}} [\log p_\theta(X)]. \quad (1.6)$$

On remarque alors que le premier terme de cette somme ne dépend pas de θ , par conséquent

$$\operatorname{argmin}_{\theta \in \Theta} \text{KL}(p_{\text{données}}, p_\theta) = \operatorname{argmax}_{\theta \in \Theta} \mathbb{E}_{X \sim p_{\text{données}}} [\log p_\theta(X)]. \quad (1.7)$$

Les choses ont alors été considérablement simplifiées. En effet, on peut utiliser notre jeu de données pour approcher l'espérance $\mathbb{E}_{X \sim p_{\text{données}}} [\log p_\theta(X)]$ par Monte Carlo :

$$\mathbb{E}_{X \sim p_{\text{données}}} [\log p_\theta(X)] \approx \frac{1}{n} \sum_{i=1}^n \log p_\theta(x_i). \quad (1.8)$$

Les données étant supposées être des réalisations i.i.d. de densité $p_{\text{données}}$, la loi des grands nombres garantira la justesse asymptotique de cette approximation. Ce raisonnement motive la définition de la *fonction de vraisemblance* des données :

$$\ell : \theta \mapsto \sum_{i=1}^n \log p_\theta(x_i). \quad (1.9)$$

L'estimateur du *maximum de vraisemblance* sera alors

$$\hat{\theta}_{\text{MV}} \in \operatorname{argmax}_{\theta \in \Theta} \ell(\theta). \quad (1.10)$$

On aimerait idéalement que $\hat{\theta}_{\text{MV}}$ approche la solution de (1.2) quand n est grand. La question principale quant à la qualité de l'estimateur du maximum de vraisemblance est alors : qu'a-t-on perdu lors de l'approximation de Monte Carlo (1.8) ? C'est l'objet de la statistique asymptotique, exposée par exemple dans le livre classique de Van der Vaart [27].

Au delà de son interprétation à l'aide de la divergence de Kullback-Leibler, la méthode du maximum de vraisemblance peut être justifiée par le fait qu'on choisit le paramètre qui maximise la probabilité des données. Cette justification est en fait la motivation historique principale de cette méthode. Une analyse historique intéressante du développement du maximum de vraisemblance a été réalisée par Stiegler [23].

1.5 Modèles profonds à variables latentes

La contrainte principale posée par la méthode du maximum de vraisemblance est la nécessité d'être capable d'évaluer et d'optimiser la densité des données selon notre modèle. Cela restreint considérablement le choix des modèles probabilistes pouvant être utilisés. Nous allons présenter ici une famille de modèles n'obéissant pas à cette contrainte (les *modèles profonds à variables latentes*). Bien que leur vraisemblance ne soit pas calculable aisément, ces modèles peuvent être entraînés par maximum de vraisemblance approché, via une méthode très générale appelée *inférence variationnelle*. On s'intéressera en particulier à une version récente de l'inférence variationnelle, appelée inférence variationnelle par échantillonnage préférentiel (*importance weighted variational inference*), introduite par Burda, Grosse et Salakhutdinov [2].

1.5.1 Modèles linéaires à variables latentes

L'idée générale des modèles à variables latentes est la suivante : bien que les données vivent généralement dans un espace de grande dimension (dans le cas d'images en niveaux de gris, la dimension est le nombre de pixels), on suppose qu'un faible nombre de "facteurs" cachés expliquent raisonnablement bien la diversité des données. Avant de formaliser ce postulat d'existence de facteurs cachés, donnons quelques exemples concrets :

- si les données sont des images de grains de beauté ou potentiel mélanomes, savoir une poignée d'informations clés (la taille de la tache, sa rotundité, la couleur de peau, ...) sera suffisant pour résumer l'image sans perte d'information ;
- il en est de même si les données sont des images de visages : dans ce cas, les facteurs pourraient être la pilosité, la couleur de peau, la présence de lunettes ;
- si les données sont des textes, connaître le thème, le ton et le style du texte peut permettre d'en avoir une idée assez précise.

La variable latente sera généralement vue comme une *représentation synthétique des données*, et vivra à ce titre souvent dans un espace (ou une variété) de dimension plus faible que celui des données. On désignera parfois la variable latente sous le nom de *code* ou de *facteurs*.

Afin de mettre cette idée générale en pratique, les modèles à variables latentes supposent l'existence de variables aléatoires latentes $z \in \mathcal{Z}$, qui permettent d'expliquer nos données $x \in \mathcal{X}$. On aimerait donc formaliser l'idée que " z explique x ". Pour ce faire, les modèles à variables latentes postuleront que la loi de x sachant z est une loi "simple", par exemple une

loi gaussienne (dans le cas de données continues), ou un produit de lois de Bernoulli (dans le cas de notre exemple fil-rouge).

Analyse factorielle Il s'agit du plus ancien exemple de modèle à variable latente (voir par exemple Jöreskog [8]), pour lequel les données sont continues ($\mathcal{X} = \mathbb{R}^d$), et expliquées par un code de faible dimension ($\mathcal{Z} = \mathbb{R}^q$ avec $q \ll d$). Ici, la variable latente et les données sont gaussiennes, et une fonction affine $z \mapsto Wz + b$ permet de les lier :

$$\begin{cases} z \sim \mathcal{N}(0, I_d) \\ x \sim \mathcal{N}(Wz + b, \Sigma). \end{cases} \quad (1.11)$$

Les paramètres inconnus du modèle sont $W \in \mathbb{R}^{d \times q}$, $\mu \in \mathbb{R}^q$, et $\Sigma \in \mathcal{S}_d^{++}$, où \mathcal{S}_d^{++} désigne le cône des matrices définies positives de taille $q \times q$. Ces derniers peuvent être estimés par maximum de vraisemblance. En effet, la vraisemblance est aisée à calculer, et peut être maximisée en utilisant par exemple une méthode de gradient. Le modèle d'analyse factorielle a une interprétation géométrique simple : les données, bien que de grande dimension d , sont proches d'un sous-espace affine de faible dimension $q \ll d$, à savoir l'image de \mathbb{R}^d par la fonction $z \mapsto Wz + b$. Le cas particulier où Σ est proportionnelle à l'identité a été étudié en détail par Tipping et Bishop [25], qui ont appelé ce modèle particulier *analyse en composantes principales probabiliste* (ACPP). Pour l'ACPP, le maximum de vraisemblance de tous les paramètres peut être obtenu directement en effectuant une décomposition en valeurs singulières de la matrice des données, sans avoir recours à un algorithme d'optimisation.

Données non-euclidiennes Si l'analyse factorielle et ses variations sont adaptées au cas de données vivant dans un espace euclidien \mathbb{R}^d , il n'est pas conceptuellement difficile de les étendre à des cas plus complexes, en remplaçant simplement la gaussienne $\mathcal{N}(Wz + b, \Sigma)$ par une loi adaptée à l'espace qui nous intéresse. Par exemple, si les données sont des vecteurs d'entiers, vivant donc dans \mathbb{N}^d , on pourrait remplacer la gaussienne multivariée par un produit de lois de Poisson \mathcal{P} , et considérer ainsi le modèle

$$\begin{cases} z \sim \mathcal{N}(0, I_d) \\ x \sim \prod_{j=1}^d \mathcal{P}(\exp([Wz + b]_j)). \end{cases} \quad (1.12)$$

La présence de la fonction exponentielle permet de s'assurer que le paramètre de la loi de Poisson sera bien strictement positif. Ce type de modèle a par exemple été employé par Chiquet, Mariadassou et Robin [3] dans le cadre de modèles d'écologie microbienne. Revenons un instant à notre

exemple fil-rouge des images binarisées. Ici, les données vivent dans un ensemble discret $\{0, 1\}^d$. Un choix naturel est alors d'utiliser un produit de lois de Bernoulli :

$$\begin{cases} z \sim \mathcal{N}(0, I_d) \\ x \sim \prod_{j=1}^d \mathcal{B}(\text{sigmoïde}([Wz + b]_j)). \end{cases} \quad (1.13)$$

Ici, la fonction sigmoïde : $t \mapsto 1/(1 - e^{-t})$ se charge de contraindre le paramètre de la loi de Bernoulli à être bien entre 0 et 1. En effet, on a $\text{sigmoïde}(\mathbb{R}) =]0, 1[$.

(Indé)pendance des variables observées Dans le cas du modèle (1.12) adapté aux vecteurs d'entiers comme dans celui des images binaires (1.13), le fait que la loi de x sachant z soit modélisée par un produit de lois simples implique que, conditionnellement à z , les coordonnées de x sont indépendantes :

$$x_1 \perp\!\!\!\perp x_2 \perp\!\!\!\perp \dots \perp\!\!\!\perp x_d | z. \quad (1.14)$$

En revanche, si l'on ne conditionne pas à z , les x_1, \dots, x_d ne sont plus indépendants ! Cela illustre le fait général que la loi de $x|z$ peut être très simple, tandis que celle de x demeure complexe.

1.5.2 Modèles profonds à variables latentes

Les modèles précédents sont limités par le fait que la fonction envoyant le code z vers la loi de $x|z$ est essentiellement linéaire. Une généralisation naturelle serait alors de remplacer cette fonction linéaire par une fonction plus générale, par exemple paramétrée par un réseau de neurones. Ainsi, notre modèle d'analyse factorielle

$$\begin{cases} z \sim \mathcal{N}(0, I_d) \\ x \sim \mathcal{N}(Wz + b, \Sigma). \end{cases} \quad (1.15)$$

sera remplacé par un modèle du type

$$\begin{cases} z \sim \mathcal{N}(0, I_d) \\ x \sim \mathcal{N}(\mu_\theta(z), \Sigma_\theta(z)), \end{cases} \quad (1.16)$$

où $\mu_\theta : \mathcal{Z} \rightarrow \mathbb{R}^d$ et $\Sigma_\theta : \mathcal{Z} \rightarrow \mathcal{S}_d^{++}$ sont des réseaux de neurones, dont les paramètres sont stockés dans le vecteur $\theta \in \Theta$.

Plus généralement, on appellera *modèle profond à variables latentes* un modèle du type

$$\begin{cases} z \sim \pi \\ x \sim \Phi(f_\theta(z)), \end{cases} \quad (1.17)$$

où π est une loi de probabilités sur \mathcal{Z} appelée *loi a priori*, $(\Phi(\eta))_{\eta \in E}$ est une famille paramétrée de lois de probabilités sur \mathcal{X} , et $f_\theta : \mathcal{Z} \rightarrow E$ est un réseau de neurones paramétré par $\theta \in \Theta$. La famille $(\Phi(\eta))_{\eta \in E}$ est appelée *modèle d'observation* et le réseau f_θ est appelé *décodeur* ou *réseau génératif*. En effet, son rôle est de transformer un code z en paramètres du modèle d'observation, permettant par là même de générer des échantillons selon le modèle. Le fonctionnement général de ce type de modèle est présenté figure 1.2. Ces modèles sont également connus sous le nom d'*autoencodeur variationnel*, et ont été introduits indépendamment par Rezende, Mohamed et Wiestra [18] et par Kingma et Welling [9].

Vérifions rapidement que cette construction générale contient bien les modèles linéaires présentés précédemment. On retrouve bien l'analyse factorielle classique en prenant un a priori gaussien $\pi = \mathcal{N}(0, I_q)$, un modèle d'observation gaussien $(\Phi(\eta))_\eta = (\mathcal{N}(\mu, \Sigma))_{\mu, \Sigma}$ ainsi qu'un décodeur linéaire $f_\theta = (\mu_\theta, \Sigma_\theta)$, avec $\mu_\theta(z) = Wz + b$ et $\Sigma_\theta(z)$ constante égale à Σ . Dans ce cas les paramètres du décodeur sont donc $\theta = (W, b, \Sigma)$. En choisissant un modèle d'observation de produits de lois de Bernoulli ainsi qu'un décodeur linéaire, on retrouve également le modèle (1.13).

Tout l'indérêt est cependant d'aller au delà des modèles linéaires, en utilisant plutôt des réseaux profonds tels que décrits plus haut. On pourrait par exemple choisir un réseau tel que celui de l'équation (1.1) :

$$\mu_\theta(z) = W_1 \text{sigmoïde}(W_0 z + b_0) + b_1, \quad (1.18)$$

dans le cas du modèle d'observation gaussien. On pourrait également utiliser un réseau adapté à l'architecture de nos données. Par exemple, si les données sont des images, on pourrait choisir un réseau convolutif; si les données sont des textes, on pourrait choisir un réseau récurrent.

On supposera dans le reste de ce chapitre que π a une densité $p(z)$ vis-à-vis d'une certaine mesure de référence sur \mathcal{Z} , et que, pour tout η , $\Phi(\eta)$ a pour densité $\Phi(x|\eta)$ vis-à-vis d'une mesure de référence sur \mathcal{X} . La densité conditionnelle de z sachant x sera donc $p_\theta(z|x) = \Phi(x|f_\theta(z))$, et la loi marginale des données sera

$$p_\theta(x) = \int_{\mathcal{X}} p_\theta(x|z)p(z)dx = \int_{\mathcal{X}} \Phi(x|f_\theta(z))p(z)dz. \quad (1.19)$$

On sera également amené à regarder la densité de z sachant x , généralement appelée *densité a posteriori*, et qui vaut, d'après le théorème de Bayes

$$p_\theta(z|x) = \frac{p_\theta(x|z)p(z)}{p_\theta(x)}. \quad (1.20)$$

Cette densité a posteriori est fondamentale car elle permet d'encoder une donnée x . En effet, la loi de $z|x$ pourra être vue comme une représentation d'une certaine donnée de grande dimension x . Notons cependant que notre représentation de x par sa loi a posteriori n'est pas vraiment une réduction de dimension pour l'instant. En effet, la représentation $z|x$ est une loi de probabilités sur un espace de petite dimension \mathcal{Z} , et non pas directement un vecteur de petite dimension. On peut cependant obtenir une représentation véritablement de petite dimension en considérant une statistique descriptive de $z|x$, par exemple sa moyenne $\mathbb{E}[z|x]$. Bien qu'on utilise le vocabulaire "a priori" et "a posteriori", il convient d'insister sur le fait que les modèles considérés ici ne sont pas bayésiens. En inférence bayésienne, on place une loi a priori sur des paramètres inconnus, afin de modéliser notre incertitude (voir par exemple [19]). Ici, l'a priori est sur des variables latentes, et les paramètres sont traités comme des quantités déterministes. Toutefois, les recettes présentées dans ce chapitre sont applicables également à des modèles bayésiens (voir par exemple [4]).

Les notations introduites dans le paragraphe précédent sont quelque peu abusives, car on appelle p à la fois les densités de x et z , conditionnelles ou non. Ce genre d'abus (tout comme la confusion habituelle entre variable aléatoire et réalisation) permet toutefois d'alléger considérablement les notations. Remarquons également que l'on a choisi d'ajouter un indice θ à toutes les densités qui dépendent du décodeur f_θ . Cela permettra de repérer de manière commode ce qui dépend ou non des paramètres à optimiser par maximum de vraisemblance.

1.5.3 La vraisemblance des modèles profonds à variables latentes

Une approche naturelle pour estimer les paramètres inconnus θ d'un modèle à variables latentes serait d'en maximiser la vraisemblance. Celle-ci est égale à

$$\ell(\theta) = \sum_{i=1}^n \log p_\theta(x_i) = \sum_{i=1}^n \log \left(\int_{\mathcal{X}} p_\theta(x_i|z) p(z) dz \right) \quad (1.21)$$

$$= \sum_{i=1}^n \log \left(\int_{\mathcal{X}} \Phi(x_i|f_\theta(z)) p(z) dz \right). \quad (1.22)$$

Dans le cas de certains modèles linéaires évoqués précédemment (par exemple l'analyse en composantes principales probabiliste ou l'analyse factorielle), les n intégrales impliquées dans l'expression de la vraisemblance peuvent être explicitement calculées, et $\ell(\theta)$ peut être optimisée à l'aide d'algorithmes d'optimisation plus ou moins standards. En revanche, dans

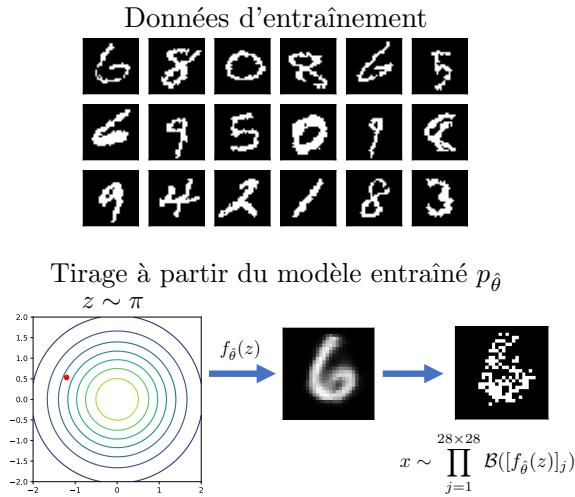


FIGURE 1.2 – Exemple d’un modèle profond à variables latentes utilisant un espace latent de deux dimensions pour modéliser MNIST.

les cas qui nous intéressent impliquant des réseaux de neurones profonds en guide de f_θ , ces intégrales n’ont pas généralement de forme simple. Comme ces intégrales sont des espérances, une approche naturelle serait d’estimer celles-ci à l’aide d’une méthode de Monte Carlo. C’est l’idée derrière l’inférence variationnelle par échantillonnage préférentiel.

1.5.4 Inférence variationnelle par échantillonnage préférentiel

Notre objectif est d’approcher

$$\log p_\theta(x_i) = \log \left(\int_{\mathcal{X}} \Phi(x_i | f_\theta(z)) p(z) dz \right), \quad (1.23)$$

pour un certain $i \in \{1, \dots, n\}$, à l’aide d’une méthode de Monte Carlo. Une approche simple serait de tirer quelques $z_1, \dots, z_K \sim \pi$ indépendants, puis de regarder l’approximation

$$\int_{\mathcal{X}} \Phi(x_i | f_\theta(z)) p(z) dz \approx I_K(x_i) = \frac{1}{K} \sum_{k=1}^K \Phi(x_i | f_\theta(z_k)). \quad (1.24)$$

La loi des grands nombres et la linéarité de l’espérance assurent que $I_K(x_i)$ est un estimateur sans biais et consistant de l’intégrale. Lorsqu’un estima-

teur est sans biais, une manière commode de quantifier sa précision est simplement de regarder sa variance. Dans ce cas,

$$\text{Var}(I_K(x_i)) = \frac{\text{Var}(\Phi(x_i|f_\theta(z_1)))}{K}. \quad (1.25)$$

La variance décroîtra donc en $O(1/K)$, mais risque de demeurer grande si $\text{Var}(\Phi(x_i|f_\theta(z_1)))$ l'est. Une manière classique de modifier l'estimateur de Monte Carlo afin de réduire sa variance est de faire de l'*échantillonage préférentiel*. L'idée est d'introduire une nouvelle densité de probabilité $q(z)$, dont le support contient celui de $p(z)$, et d'écrire

$$\int_{\mathcal{X}} \Phi(x_i|f_\theta(z)) p(z) dz = \int_{\mathcal{X}} \frac{\Phi(x_i|f_\theta(z)) p(z)}{q(z)} q(z) dz \quad (1.26)$$

$$\approx I_K^q(x_i) = \frac{1}{K} \sum_{k=1}^K \frac{\Phi(x_i|f_\theta(z_k)) p(z_k)}{q(z_k)}, \quad (1.27)$$

où les z_1, \dots, z_K sont désormais tirés selon la loi de q plutôt que π . La condition de support assure que le dénominateur ne sera jamais nul.

Est-ce que ce nouvel estimateur $I_K^q(x_i)$ est meilleur que l'estimateur simple $I_K(x_i)$? Pour les mêmes raisons, $I_K^q(x_i)$ est également consistant et sans biais, mais qu'en est-il de sa variance? La réponse dépend du choix de la densité de proposition q . Étonnamment, il est possible de choisir une certaine q_i^* telle que $\text{Var}(I_K^{q_i^*}(x_i)) = 0$, c'est à dire qu'un seul tirage de Monte Carlo suffit pour estimer parfaitement notre intégrale! En effet, en choisissant

$$q_i^*(z) = p_\theta(z|x_i) = \frac{\Phi(x_i|f_\theta(z)) p(z)}{p_\theta(x_i)}, \quad (1.28)$$

on voit que $I_K^{q_i^*}(x_i) = p(x_i)$ pour tout K .

Malheureusement, tirer selon la loi de densité $q_i^*(z) = p_\theta(z|x_i)$ n'est pas chose aisée. Des méthodes de Monte Carlo par chaînes de Markov pourraient être utilisées à cette fin, mais à un coût calculatoire très important. Ici, nous préférons voir cette loi idéale non pas comme un objectif en soi, mais comme une raison d'espérer qu'en choisissant "raisonnablement" une loi de proposition, l'échantillonage préférentiel peut fonctionner beaucoup mieux qu'un estimateur de Monte Carlo simple. Notre objectif sera ainsi *de choisir des densités de propositions q_1, \dots, q_n afin d'approcher au mieux $\log p(x_1), \dots, \log p(x_n)$* .

Une idée naturelle serait de choisir ces q_1, \dots, q_n dans une famille paramétrée de densités $(\Psi(z|\kappa))_{\kappa \in \mathcal{K}}$ sur \mathcal{Z} (par exemple si \mathcal{Z} est un espace euclidien, on pourrait choisir une famille de gaussiennes multivariées).

Cela veut dire que choisir q_1, \dots, q_n reviendra à choisir des paramètres $\kappa_1, \dots, \kappa_n \in \mathcal{K}$ pour chacune des lois de proposition. Dans le cas de propositions gaussiennes, cela impliquerait de choisir n moyennes et n matrices de covariances. Cependant, dans les situations qui nous intéressent, le nombre d'observations n est généralement très grand (avec peu d'observations, entraîner des réseaux de neurones profonds est peu indiqué). Cela veut dire que choisir ces $\kappa_1, \dots, \kappa_n$ sera difficile et coûteux en mémoire. On va montrer ici une autre solution, appelée *inférence variationnelle amortie*. L'idée de base est assez simple : plutôt que de choisir une valeur de κ pour chaque x_i , on va apprendre une fonction envoyant x vers le κ correspondant. Cette fonction, qu'on appellera $g : \mathcal{X} \rightarrow \mathcal{K}$, sera appelée *encodeur*, car elle envoie des données x vers une distribution $\Psi(g(x))$ sur l'espace des codes. Nous allons à nouveau modéliser cette fonction à l'aide d'un réseau de neurones, dont les paramètres sont stockés dans un vecteur $\gamma \in \Gamma$, on notera donc g_γ . En définitive, notre proposition sera donc, pour chaque point x_i

$$q_i(z) = \Psi(z|g_\gamma(x_i)). \quad (1.29)$$

Il est commode de voir ces distributions non pas comme une suite de n distributions, mais comme des lois conditionnelles toutes paramétrées par un même γ :

$$q_i(z) = q_\gamma(z|x_i), \quad (1.30)$$

où, pour tout $x \in \mathcal{X}$,

$$q_\gamma(z|x) = \Psi(z|g_\gamma(x)). \quad (1.31)$$

L'intérêt de définir cette loi conditionnelle $q_\gamma(z|x)$ est multiple :

- nous n'aurons pas à apprendre une loi q_i pour chaque point de données mais une seule loi conditionnelle ;
- si nous sommes en présence d'un nouveau point \tilde{x} qui n'était pas dans la base de données initiale, nous pouvons aisément calculer une proposition adéquate, en encodant x et calculant $q_\gamma(z|x)$;
- nous pouvons choisir un type d'architecture pertinent pour g_γ en fonction de nos données. Par exemple, dans le cas d'images, nous pourrions considérer un réseau convolutif.

En définitive, notre estimateur de la log-vraisemblance sera

$$\ell(\theta) \approx \mathcal{L}_K(\theta, \gamma) = \sum_{i=1}^n \mathbb{E}_{z_{i1}, \dots, z_{iK} \sim q_\gamma(z|x_i)} \left[\log \left(\frac{1}{K} \sum_{k=1}^K \frac{\Phi(x_i|f_\theta(z_{ik})) p(z_{ik})}{q_\gamma(z_{ik}|x)} \right) \right]. \quad (1.32)$$

L'idée sera de maximiser $\mathcal{L}_K(\theta, \gamma)$ plutôt que la vraisemblance $\ell(\theta)$. Cela pose tout de suite quelques questions :

- *Est-il vraiment plus facile de maximiser $\mathcal{L}_K(\theta, \gamma)$ que $\ell(\theta)$?* Après tout, $\mathcal{L}_K(\theta, \gamma)$ est également composé de n espérance à priori difficiles à calculer. Cependant, en tirant à l'aide de $q_\gamma(z|x)$ il est possible de calculer un estimateur sans biais du gradient de $\mathcal{L}_K(\theta, \gamma)$, comme expliqué par exemple par Mohamed et ses coauteurs [16]. Cela permet ainsi de maximiser $\mathcal{L}_K(\theta, \gamma)$ à l'aide de l'algorithme du gradient stochastique (voir par exemple la revue de littérature de Bottou, Curtis et Nocedal [1]).
- *On a introduit un paramètre en plus : γ . Que fait-on de lui ?* En fait, on va tout simplement maximiser $\mathcal{L}_K(\theta, \gamma)$ à la fois en γ et en θ ! Pourquoi est-ce pertinent ? Pour tous θ, γ , l'inégalité de Jensen appliquée au logarithme entraîne $\ell(\theta) \geq \mathcal{L}_K(\theta, \gamma)$. Ansi, à θ fixé, la meilleure approximation de $\ell(\theta)$ qui puisse s'écrire comme un $\mathcal{L}_K(\theta, \gamma)$ sera

$$\ell(\theta) \approx \max_{\gamma \in \Gamma} \mathcal{L}_K(\theta, \gamma). \quad (1.33)$$

Le fait que notre approximation est une borne inférieure de la vraisemblance justifie le nom "*evidence lower bound*" pour désigner $\mathcal{L}_K(\theta, \gamma)$.

- *Quelle est l'influence du nombre de tirages K ?* Intuitivement, il faudrait choisir K aussi grand que possible. On peut en effet prouver (voir par exemple [14]) que

$$\mathcal{L}_{K+1}(\theta, \gamma) \geq \mathcal{L}_K(\theta, \gamma), \quad (1.34)$$

c'est à dire que l'approximation devient de plus en plus précise quand K grandit. Domke et Sheldon [4] on également prouvé, au prix d'hypothèse sur les moments des poids de l'échantillonnage préférentiel, que $\mathcal{L}_K(\theta, \gamma)$ converge vers $\ell(\theta)$ à vitesse $O(1/K)$. Il s'agirait donc idéalement de chosir K très grand. Cependant, plus K sera grand, et plus il sera computationnellement coûteux d'optimiser $\mathcal{L}_K(\theta, \gamma)$.

En définitive, on a donc un moyen relativement simple d'apprendre les paramètres inconnus de notre modèle à variables latentes, applicable dans de nombreuses situations. En effet, l'inférence par échantillange préférentiel telle que rapidement décrite ici, a été appliquée dans de nombreux cadres : si elle a été introduite en premier lieu pour des modèles profonds à variables latentes [2], cette technique a par exemple été appliquée à l'apprentissage de processus gaussiens [21], de modèles séquentiels [12, 17, 11], de modèles de graphes aléatoires [24], à l'apprentissage avec données manquantes [13, 6, 7], ou à l'inférence bayésienne en général [4].

1.6 Conclusion

Allant du général au particulier, nous nous sommes attachés à offrir un panorama de l'IA dans sa diversité (des modèles symboliques à l'apprentissage), suivi d'un zoom sur certains modèles statistiques récents. Cette progression sera également l'objectif de la série de cours basés sur ce chapitre : situer l'IA comme champ interdisciplinaire, et détailler un exemple (les modèles profonds à variables latentes), des mathématiques à l'implémentation.

Bibliographie

- [1] L. BOTTOU, F. E. CURTIS et J. NOCEDAL : Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.
- [2] Y. BURDA, R. GROSSE et R. SALAKHUTDINOV : Importance weighted autoencoders. In *International Conference on Learning Representations*, 2016.
- [3] J. CHIQUET, M. MARIADASSOU et S. ROBIN : Variational inference for probabilistic poisson PCA. *The Annals of Applied Statistics*, 12(4):2674–2698, 2018.
- [4] J. DOMKE et D. R. SHELDON : Importance weighting and variational inference. In *Advances in neural information processing systems*, p. 4470–4479, 2018.
- [5] J. HAUGELAND : *Artificial Intelligence : The Very Idea*. Massachusetts Institute of Technology, USA, 1985.
- [6] N. B. IPSEN, P.-A. MATTEI et J. FRELLSEN : not-MIWAE : Deep generative modelling with missing not at random data. In *International Conference on Learning Representations*, 2021.
- [7] N. B. IPSEN, P.-A. MATTEI et J. FRELLSEN : How to deal with missing data in supervised deep learning? In *International Conference on Learning Representations*, 2022.
- [8] K. G. JÖRESKOG : A general approach to confirmatory maximum likelihood factor analysis. *Psychometrika*, 34(2):183–202, 1969.
- [9] D. P. KINGMA et M. WELLING : Auto-encoding variational Bayes. In *International Conference on Learning Representations*, 2014.
- [10] M. J. KUSNER, B. PAIGE et J. M. HERNÁNDEZ-LOBATO : Grammar variational autoencoder. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, p. 1945–1954. JMLR. org, 2017.

- [11] T. A. LE, M. IGL, T. RAINFORTH, T. JIN et F. WOOD : Auto-encoding sequential Monte Carlo. In *International Conference on Learning Representations*, 2018.
- [12] C. J. MADDISON, J. LAWSON, G. TUCKER, N. HEESS, M. NOROUZI, A. MNIIH, A. DOUCET et Y. TEH : Filtering variational objectives. In *Advances in Neural Information Processing Systems*, p. 6573–6583, 2017.
- [13] P.-A. MATTEI et J. FRELLSEN : MIWAE : Deep generative modelling and imputation of incomplete data sets. In *International Conference on Machine Learning*, p. 4413–4423, 2019.
- [14] P.-A. MATTEI et J. FRELLSEN : Uphill roads to variational tightness : Monotonicity and Monte Carlo objectives. *arXiv preprint arXiv:2201.10989*, 2022.
- [15] W. MCCULLOCH et W. PITTS : A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.
- [16] S. MOHAMED, M. ROSCA, M. FIGURNOV et A. MNIIH : Monte Carlo gradient estimation in machine learning. *Journal of Machine Learning Research*, 21(132):1–62, 2020.
- [17] C. NAESSETH, S. LINDERMAN, R. RANGANATH et D. BLEI : Variational sequential Monte Carlo. In *International Conference on Artificial Intelligence and Statistics*, p. 968–977, 2018.
- [18] D. REZENDE, S. MOHAMED et D. WIERSTRA : Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning*, p. 1278–1286, 2014.
- [19] C. ROBERT : *Le choix bayésien : Principes et pratique*. Springer Science & Business Media, 2005.
- [20] S. RUSSELL et P. NORVIG : *Artificial Intelligence : A Modern Approach*. Prentice Hall, 3 édn, 2010.
- [21] H. SALIMBENI, V. DUTORDIR, J. HENSMAN et M. DEISENROTH : Deep Gaussian processes with importance-weighted variational inference. In K. CHAUDHURI et R. SALAKHUTDINOV, éds : *Proceedings of the 36th International Conference on Machine Learning*, vol. 97 de *Proceedings of Machine Learning Research*, p. 5589–5598. PMLR, 09–15 Jun 2019.
- [22] H. A. SIMON : Why should machines learn? In R. MICHALSKI, J. CARBONNEL et T. MITCHELL, éds : *Machine Learning : An Artificial Intelligence Approach*, p. 25–37. Tioga, Palo Alto, CA, 1983.
- [23] S. M. STIGLER : The epic story of maximum likelihood. *Statistical Science*, p. 598–620, 2007.

- [24] L. S. L. TAN et N. FRIEL : Bayesian variational inference for exponential random graph models. *Journal of Computational and Graphical Statistics*, 29(4):910–928, 2020.
- [25] M. E. TIPPING et C. M. BISHOP : Probabilistic principal component analysis. *Journal of the Royal Statistical Society : Series B (Statistical Methodology)*, 61(3):611–622, 1999.
- [26] A. M. TURING : Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.
- [27] A. W. Van der VAART : *Asymptotic statistics*. Cambridge university press, 2000.