UNIVERSITÉ DE LA MANNOUBA
École Nationale des Sciences de l'Informatique
Laboratoire TSIRS, ENIT

UNIVERSITÉ DE STRASBOURG
École Doctorale MSII
Laboratoire ICUBE, CSTB

THÈSE en cotutelle

présentée par :

Yesmina JAAFRA

soutenue le : **11/09/2020**

pour obtenir le grade de : **Docteur**

Spécialité : **Informatique**

Méta-Apprentissage par Renforcement pour le Contrôle Adaptatif

Thèse dirigée par :

Mme Aline Deruyver	Maître de conférences, CSTB-ICUBE Unistra
M. Mohamed Saber Naceur	Professeur, LTSIRS ENIT

Rapporteurs :

M. David Filliat	Professeur, U2IS-ENSTA ParisTech
M. Slim Yacoub	Professeur, SITI-ENIT INSAT

Autres membres du jury :

Mme Elisa Fromont	Professeur, IRISA/INRIA Université de Rennes
M. Lhassane Idoumghar	Professeur, IRIMAS-MAGE ENSISA
M. Pierre Collet	Professeur, CSTB-ICUBE Unistra

À mes parents pour leurs sacrifices,
à ma sœur pour sa générosité,
à mon frère pour sa vivacité,
à mes amis et à tous ceux qui m'aiment
et me souhaitent le bien et le bonheur.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Contexte général

Avec l'avènement de l'apprentissage profond, la recherche en intelligence artificielle (AI) s'est munie d'un outil fondamental qui lui a permis de basculer vers le traitement de problèmes larges et à grande échelle dont la performance et le succès sont soutenus, à une majeure partie, par une croissance exponentielle de puissance de calcul et de données d'entraînement. Cette tendance de l'AI à évoluer par la maximisation des ressources est incompatible avec la réalité de la cognition humaine dont elle est sensée automatiser la rationalité de l'action (Russell & Norvig, 2009). En effet, l'esprit humain ne dispose que d'une quantité de ressources limitée dans l'espace (le corps) et dans le temps (la vie) qu'il doit allouer de manière adaptative pour résoudre efficacement des problèmes complexes.

Le fait d'avoir de telles contraintes pour apprendre signifie que les humains doivent être en mesure de tirer le meilleur parti de chaque élément de données à leur disposition, en construisant puis en s'appuyant sur un riche modèle du monde dans lequel leur apprentissage est en train d'évoluer. En conséquence du développement de ces capacités, les humains deviennent des apprenants efficaces et polyvalents, un fort contraste avec les systèmes AI actuels qui nécessitent d'énormes quantités de données d'entraînement et sont hautement spécialisés pour des tâches particulières. Ces deux composantes de l'intelligence humaine relatives à l'optimisation des ressources de calcul et l'utilisation efficace des données sont liées, respectivement, à deux problèmes étudiés dans la littérature sur l'AI : le méta-raisonnement et le méta-apprentissage (Griffiths et al., 2019). C'est dans le cadre de ce dernier concept que se déroule notre recherche, dont l'objectif principal est de contribuer à l'introduction des principes du méta-apprentissage dans le fonctionnement des systèmes de contrôle complexes du monde réel.

Le point de départ de notre démarche est la constatation faite par Widmer (1997) qui stipule que « les concepts dans le monde réel ne sont pas des entités ou des structures éternellement figées, mais peuvent avoir une apparence, une définition ou une signification différentes dans des contextes différents ». Plusieurs tâches nécessitent une adaptation au contexte, telles que les prévisions météorologiques en fonction de la saison ou de la géographie, la reconnaissance de la parole avec l'origine du locuteur, les processus de contrôle des installations industrielles avec les facteurs climatiques et la vision par ordinateur avec les conditions d'éclairage et les couleurs de fond. Tel qu'indiqué dans le début de cette section, une solution pour faire face à cette variabilité consiste à imiter le comportement humain qui est plus à l'aise à apprendre avec peu d'expérience et à s'adapter à des perturbations inattendues. Ces différences naturelles façonnent les recherches actuelles dans l'AI visant à éviter le problème de l'inefficacité des données et à améliorer les capacités de généralisation des agents artificiels (Lake et al., 2017). Traitant cette problématique en tant qu'un exercice d'apprentissage multitâche (Caruana, 1997), le méta-apprentissage a

donné des résultats prometteurs et constitue l'un des cadres privilégiés pour concevoir des stratégies d'adaptation rapide (Santoro et al., 2016; Ravi & Larochelle, 2017).

Le méta-apprentissage fait référence à des approches d'apprendre à apprendre qui visent à produire un modèle sur un ensemble de tâches différentes mais liées, puis à généraliser à de nouveaux cas en se basant sur quelques exemples supplémentaires (Finn et al., 2017). En effet, pour acquérir de nouvelles compétences, il est plus utile de s'appuyer sur une expérience antérieure que de partir de zéro. Ainsi, nous apprenons à travers les tâches nécessitant, à chaque itération, moins de données et d'effort pour conquérir de nouvelles compétences (Lake et al., 2017). Le terme méta-apprentissage faisant référence à l'acquisition de l'apprentissage sur la base d'une expérience antérieure a été cité pour la première fois par Biggs (1985) dans le domaine de la psychologie de l'éducation. Il consiste à prendre le contrôle d'un processus d'apprentissage et à le guider en fonction du contexte d'une tâche spécifique.

Dans la recherche sur l'apprentissage automatique (ML), le méta-apprentissage n'est pas un nouveau concept et présente de nombreux points communs avec la définition ci-dessus (Thrun & Pratt, 1998; Schmidhuber & Huber, 1991; Naik & Mammone, 1992). Il suppose que plutôt que de construire une stratégie d'apprentissage sur la base d'une tâche unique, il serait plus efficace de s'entraîner sur une série de tâches partageant un ensemble de similitudes, puis de généraliser à de nouvelles situations. En acquérant des biais antérieurs, le méta-apprentissage s'adresse aux imperfections des modèles explorés réalisant une adaptation rapide à partir de quelques données supplémentaires (Clavera et al., 2018). Il se distingue ainsi de l'apprentissage conventionnel de base (base-learning) par l'étendue de son niveau d'adaptation. Tandis que ce dernier possède un biais fixe a priori et accumule de l'expérience par rapport à une tâche d'apprentissage spécifique, le méta-apprentissage, quant à lui, choisit dynamiquement son biais selon le contexte d'étude et accumule de l'expérience relativement à plusieurs applications du système d'apprentissage (Vilalta & Drissi, 2002).

1.2 Problématique

La performance de nombreuses méthodes ML dépendait fortement du choix de la représentation des données auxquelles elles sont appliquées. Pour cette raison, une grande partie de l'effort de déploiement d'algorithmes ML a été dédiée à la conception de solutions de prétraitement de données capables de prendre en charge un apprentissage efficace. Cette inévitable étape d'ingénierie de caractéristiques nécessitait une intervention manuelle conséquente et a mis en évidence une limite des algorithmes classiques dans l'extraction des informations discriminantes à partir des données (Bengio, 2013). En effet, les extracteurs de caractéristiques laborieusement réalisés à la main constituaient un obstacle à l'extension des systèmes traditionnels d'apprentissage vers des ensembles de données de grande taille. Afin d'élargir la portée de l'applicabilité du ML et promouvoir la commodité de son utilisation, il fallait produire des algorithmes d'apprentissage moins dépendants de l'ingénierie manuelle de caractéristiques.

La réponse à ce besoin a été apportée par l'apprentissage profond basé sur les réseaux de neurones artificiels (ANN), une approche relativement récente au sein de la communauté de recherche en ML. Les réseaux de neurones profonds (DNN) sont des cas particuliers

d'apprentissage de la représentation qui automatisent l'extraction de caractéristiques en procédant d'une manière hiérarchique à de multiples niveaux. Le concept fondamental derrière toute méthodologie d'apprentissage profond est la découverte automatisée de l'abstraction, avec l'hypothèse que des représentations plus abstraites de données telles que des images, des signaux vidéo et audio ont tendance à être plus utiles (Shrestha & Mahmood, 2019). Les architectures profondes aboutissent à des structures hiérarchiques car des concepts plus abstraits sont construits en fonction d'éléments moins abstraits en dissociant graduellement le contenu sémantique des données de leurs caractéristiques brutes de bas niveau (par exemple pixels et formes d'onde).

Le passage de l'apprentissage superficiel (shallow) à l'apprentissage profond favorisé par la prolifération d'unités de traitement puissantes et moins coûteuses ainsi qu'un grand volume de données (big data) structurées a permis d'approximer des fonctions plus complexes et non linéaires. Ce domaine de recherche instigué par le travail pionnier de Hinton et al. (2006) connaît désormais une croissance et un succès rapides dans plusieurs domaines d'activité dont l'éducation, l'économie, la médecine et l'industrie. Techniquement, l'apprentissage profond est largement utilisé dans les trois principaux paradigmes du ML, supervisé, non supervisé et par renforcement. Dans cette recherche, nous nous intéressons à l'apprentissage par renforcement (RL) dont l'objectif principal est l'apprentissage de systèmes de contrôle spécialement en l'absence de modèle précis définissant l'environnement. C'est un cadre de modélisation général pouvant englober toute l'AI qui permet de déplacer l'attention des méthodes ML de la reconnaissance de patterns vers la prise de décision séquentielle fondée sur l'expérience (Russell & Norvig, 2009).

Inspiré de la psychologie comportementaliste, le RL est une approche fondamentale de l'optimisation orientée objectifs (Sutton & Barto, 2018). L'ossature du RL est un agent qui apprend par interaction avec son environnement, guidé par un signal d'impact (récompense). Le retour d'environnement oblige l'agent à sélectionner de nouvelles actions améliorant le processus d'apprentissage, d'où le nom d'apprentissage par renforcement. Les premières approches RL manquaient d'extensibilité et se limitaient par nature à des problèmes relativement peu dimensionnels, ce qui réduisait le nombre de cas d'application pouvant être couverts par ce paradigme. Ces limitations existaient parce que, comme les autres méthodes du ML, les algorithmes RL rencontraient des problèmes de représentation et de dimensionnement, un phénomène qualifié dans la littérature par « Malédiction de la dimensionnalité » (Bellman, 1957).

Les avancées récentes de l'apprentissage profond mentionnées plus tôt dans cette section ont ouvert la voie à de nouveaux horizons pour le RL. La combinaison de l'universalité d'approximation du premier et la puissance de modélisation du dernier dans le cadre d'un RL profond (DRL), s'est avérée particulièrement efficace dans les tâches avec des espaces d'état larges. Le DRL consiste généralement à entraîner des DNNs pour approximer une des composantes de RL, principalement la politique, les fonctions de valeur et la fonction de récompense. Il parvient à s'adapter à des entrées sensorielles de grande dimension (des milliers de pixels dans le cas de perception visuelle) et à une logique de contrôle complexe, sans nécessiter l'intervention manuelle d'ingénierie ni se limiter à des modèles simples et insuffisamment expressifs. Ainsi, les algorithmes DRL ont obtenu des résultats remarquables dans des environnements complexes tels que les jeux (Mnih et al., 2015; Silver et al., 2016) et les manipulations robotiques avancées (Levine et al., 2016; Lillicrap et al., 2016) dépassant parfois la performance humaine.

Bien que les méthodes DRL puissent obtenir des résultats impressionnants dans des conditions expérimentales, elles peinent encore à récolter le même succès sur les tâches du monde réel qui présentent beaucoup plus de défis et de contraintes. Les travaux réalisés pour les jeux vidéos et la robotique sont généralement caractérisées par l'utilisation d'environnements simulés stationnaires, des règles déterministes, des espaces d'actions réduits et des manipulations répétitives. Cependant, les tâches de contrôle du monde réel telles que la prédiction financière, les réseaux intelligents et la conduite autonome sont non-stationnaires et incertains en raison de l'observabilité partielle de l'environnement et impliquent souvent des actions évoluées et des objectifs composites à modéliser.

Les perturbations et les situations invisibles du monde réel entraînent l'échec de politiques RL qui héritent des limites de l'AI et de l'apprentissage profond (plus spécifiquement) mentionnées dans la section précédente. En effet, le DRL nécessite considérablement d'exploration et de données d'entraînement avant d'atteindre une stratégie de contrôle efficace, reflétant un problème d'inefficacité de données qui peut rendre intraitable des tâches associées à des environnements instables (Wahlström et al., 2015). D'un autre côté, les méthodes DRL se spécialisent généralement dans un contexte étroit empêchant leur adaptation à la variabilité des situations au sein d'un même problème. Ainsi, même si l'apprentissage profond fournit une architecture de bout en bout pour le développement de stratégies de contrôle optimales à partir des données, des progressions notables doivent être réalisées pour réussir l'application du RL à des tâches du monde réel. Les méthodes proposées doivent parvenir à généraliser un comportement approprié de l'agent dans un contexte non-stationnaire.

1.3 Contributions

Motivés par l'extension du ML aux applications de contrôle dans le monde réel, nous développons un algorithme de méta-apprentissage par renforcement (MRL) pour pallier aux limites des stratégies RL décrites dans la section précédente. Notre objectif est de produire des politiques de contrôle qui se comportent de manière efficace et flexible dans des conditions variables en employant des méthodes de transfert de connaissances et de généralisation d'apprentissage à travers les tâches. Plus précisément, le méta-apprentissage est utilisé pour induire de nouvelles perspectives à l'agent RL qui, à partir de l'expérience sur des tâches antérieures, apprend comment orienter les règles d'exploration en fonction des mises à jour de l'environnement. Une grande majorité des travaux de méta-apprentissage a été assignée à des applications de régression et de classification. Dans ce travail, notre contribution consiste à étudier l'applicabilité du méta-apprentissage au paradigme RL. Ainsi, nous développons et évaluons une approche MRL sur un environnement dynamique impliquant les aspects réalistes et complexes des tâches du monde réel, qui est le simulateur CARLA pour la conduite autonome en milieu urbain (Dosovitskiy et al., 2017).

Nous adoptons une démarche d'implémentation en 2 étapes. Tout d'abord, nous proposons une approche RL robuste en introduisant un critique avec retours multi-step réduisant la variance et stabilisant l'apprentissage de l'agent de conduite. Concrètement, nous essayons de résoudre, à travers l'apprentissage par différence temporelle (TD) issu de la programmation dynamique et des méthodes Monte Carlo, l'inadaptabilité entre les spécificités du DRL et la non-stationnarité des tâches aboutissant à des gradients de

politique de haute variance et un apprentissage instable et non convergent (Haarnoja et al., 2018). En effet, l'estimation de la fonction de valeur par le bootstrapping de l'apprentissage TD offre une meilleure évaluation de la politique de l'agent en arbitrant entre la variance des retours empiriques et le biais des estimations réalisées par les DNNs d'approximation. En procédant de la sorte, il est possible d'examiner à quels niveaux du système RL se situent les éventuelles limites induites par la variabilité des environnements. Les résultats de cette étape permettent de justifier le recours au méta-apprentissage pour des politiques plus généralisables répondant aux limites mises en exergue.

Dans la continuité de la première contribution, nous proposons au cours de la deuxième étape une approche MRL combinant le RL robuste et le méta-apprentissage par optimiseur (Finn et al., 2017). Ce dernier présente l'avantage de l'universalité et l'adaptabilité à toute méthode ML entraînée par descente de gradient tel que le DRL. Après avoir effectué une vaste revue de littérature soulignant le passage du méta-apprentissage classique issu des théories de sélection d'algorithme et « No Free Lunch » vers le méta-apprentissage profond dédié à des tâches plus complexes et de grande dimensionnalité, nous implémentons une couche de méta-apprentissage à la méthode RL utilisée dans l'étape précédente. Le noyau de l'architecture développée se compose d'une inner loop dédiée à l'apprentissage spécifique à chaque tâche à travers un contrôleur DNN assurant une adaptation robuste et continue. Le méta-niveau consiste en une outer loop apprenant graduellement à travers les tâches grâce à un méta-apprenant basé sur le gradient capable de produire des initialisations de modèles généralisables.

La tâche choisie pour l'évaluation de nos approches fait partie d'un secteur d'activité qui va révolutionner le développement du transport et qui promet également une nouvelle conception du mode de vie urbain. Il s'agit de la conduite autonome, un moyen de mobilité qui du fait de sa structure séquentielle, de l'incertitude causée par des caractéristiques associées à l'être humain, l'environnement et les technologies utilisées, constitue un domaine du monde réel naturellement compatible à l'application des approches RL (Fridman et al., 2019). Incluant plusieurs problématiques non résolues telles que la perception de la scène, le contrôle du véhicule et l'optimisation de la trajectoire, la conduite autonome reste encore une activité trop complexe pour être formalisée en tant qu'un système robotique complètement autonome. Techniquement, un véhicule autonome est un système de contrôle qui traite des flux d'observations provenant de différents capteurs embarquées pour prendre des décisions de conduite. Il doit être testé sur des millions de kilomètres pour valider sa fiabilité, ce qui pourrait nécessiter plusieurs années d'essais (Kalra & Paddock, 2016).

Les systèmes de simulation permettent, dans ce cadre, de reproduire une infinité de scénarios dans le but d'atteindre un haut degré d'optimisation et de précision à moindre coût et dans les plus courts délais. Ainsi nous déroulons nos scénarios de conduite dans le simulateur CARLA offrant des environnements virtuels réalistes dans lesquels évolue un véhicule autonome en interaction avec plusieurs acteurs dynamiques. CARLA est un système serveur-client échangeant des commandes de simulation et des lectures de scène interprétées en tant que des trajectoires d'actions, statuts et récompenses par l'algorithme RL. Cet outil est bien adapté à l'apprentissage de bout en bout que nous développons en codant le processus hiérarchique de conduite en une seule architecture d'apprentissage profond traduisant les observations sensorielles en commandes de contrôle (Xu et al., 2017).

1.4 Plan de la thèse

Le présent manuscrit s'articule autour de trois grandes parties reflétant l'évolution chronologique et scientifique de la recherche ainsi que les spécificités et les interactions entre trois champs piliers de l'apprentissage automatique qui ont servi à la réalisation de notre objectif :

- Chapitre 2 : apprentissage profond.

  Après un passage en revue du contexte et des paradigmes classiques de l'apprentissage automatique, nous présentons les spécificités des réseaux de neurones artificiels et les circonstances qui ont mené à l'essor de l'apprentissage profond dont l'apparition de nouveaux algorithmes d'entraînement, la construction de larges bases de données benchmark et la migration des systèmes informatiques vers des puissances de calcul exceptionnelles. Nous nous intéressons spécialement aux réseaux de neurones à convolution (CNNs) qui ont révolutionné le traitement de la vision automatique et l'approximation des fonctions dont les politiques RL. Ce chapitre permettra ainsi de mieux comprendre le contexte technique général utilisé tout au long de cette thèse. Nous consacrons la dernière partie du chapitre à un cas d'application introductif à l'apprentissage profond (classification d'images satellitaires) et aux différentes technologies dédiées à son utilisation telles que les librairies et les environnements de développement intégrés (IDE).

- Chapitre 3 : apprentissage par renforcement pour le contrôle.

  Nous exposons dans ce chapitre les principales composantes du RL et ses origines remontant à la programmation dynamique classique, l'apprentissage par TD et les approximations stochastiques. Nous présentons par la suite notre première contribution de recherche qui consiste à développer une approche RL actor-critic robuste et l'évaluer sur une tâche du monde réel beaucoup plus complexe que les benchmark de tests actuels (essentiellement les jeux et les manipulations robotiques), qui est la conduite autonome. Notre objectif est de répondre aux questions suivantes : comment consolider la robustesse d'une approche RL face à la tâche complexe de conduite autonome en milieu urbain ? A quels niveaux de ce système RL se situent les éventuelles limites induites par la non-stationnarité ? Les outputs de l'expérimentation serviront de base à l'implémentation du méta-apprentissage discutée dans le quatrième chapitre.

- Chapitre 4 : contrôle adaptatif par méta-apprentissage.

  Après le RL, c'est au tour du méta-apprentissage d'être révolutionné par les apports de l'apprentissage profond. Dans ce chapitre nous expliquons comment nous sommes passés du méta-apprentissage classique de recommandation d'algorithmes et de paramétrage au méta-apprentissage par réseaux de neurones orienté vers le few shot learning (FSL). Nous présentons par la suite notre deuxième contribution de recherche qui consiste à développer et évaluer une approche de MRL dans l'objectif de combler la faiblesse de généralisation des méthodes RL révélée dans notre expérimentation au chapitre 3. Nous maintenons la même application de conduite autonome tout en changeant les configurations du simulateurs CARLA afin d'accentuer la non-stationnarité des environnements d'expérimentation.

Le dernier chapitre conclura cette thèse en interprétant les résultats des travaux présentés et en proposant les perspectives pour des éventuelles améliorations de l'approche, notamment l'utilisation de la recherche d'architecture neuronale (NAS) pour ajuster automatiquement la conception des DNNs d'approximation à la nature des tâches étudiées et le recours aux méthodes RL hybrides pour une meilleure maîtrise de l'incertitude de l'environnement de conduite.

Chapitre 2

Apprentissage profond

2.1 Introduction

L'AI dont l'expression a été utilisée pour la première fois en 1956 par les chercheurs John McCarthy et Marvin Minsky est un domaine très vaste qui englobe l'ensemble des théories et des pratiques dédiées à la simulation de l'intelligence par les machines. Plusieurs définitions ont été présentées dans la littérature reflétant l'idée principale d'automatiser la rationalité de l'action et de la pensée humaines (Russell & Norvig, 2009). Dans ce chapitre, nous nous intéressons aux techniques de ML, un champ d'étude de l'AI fondé sur les approches statistiques. Il consiste à fournir des méthodes automatisées capables de détecter des modèles dans les données et de les exploiter pour réaliser certaines tâches de prévision et de prise de décision.

Après avoir rappelé quelques généralités sur le ML dans la première section, nous présentons les spécificités de l'apprentissage profond en le situant par rapport aux autres méthodes de ML et en soulignant ses spécificités qui en ont fait le principal outil actuel de l'AI, dont notamment sa propriété d'approximation universelle. Nous nous focalisons par la suite sur les CNNs qui constituent un composant de base des contributions de cette thèse en servant à approximer les politiques RL. A la fin de ce chapitre, nous évoquons l'aspect technologique inhérent aux applications de l'apprentissage profond très demandeur en capacités de calcul. Nous présentons dans le même cadre un cas d'application de CNN à la classification d'images satellitaires qui a constitué notre point d'entrée pour la maîtrise de la partie technologique de la recherche.

2.2 Vue d'ensemble de l'apprentissage automatique

Contrairement aux programmes informatiques classiques dans lesquels la tâche est formalisée en une séquence prédéfinie d'instructions précises, les algorithmes ML construisent leur fonction de décision sur la base des informations relatives aux données. Ceci est particulièrement important dans les cas où il n'est pas possible de spécifier explicitement la tâche ou lorsque les spécifications statiques ne sont pas suffisamment robustes. L'approximation de ces fonctions de décision se situe au cœur du ML et dépend principalement de la nature du signal traité. Nous présentons dans cette section quelques préliminaires sur le ML qui seront utiles par la suite pour comprendre le rôle d'approximation universelle des DNNs.

2.2.1 Contexte

Le ML est le domaine consacré au développement d'algorithmes permettant l'extraction de connaissance à partir d'un ensemble de données en définissant les concepts et patterns

régissant ces données. C'est une sous discipline de l'AI joignant à la fois l'informatique et les sciences statistiques. Elle étudie le développement des systèmes informatiques qui s'améliorent par l'expérience (Alpaydin, 2010). Devenant un ingrédient primordial de l'AI, le ML est actuellement la base de plusieurs disciplines et champs de recherche : big data, data science, modélisation prédictive, data mining, vision par ordinateur, traitement automatique du langage naturel, etc. Mitchell (1997) propose une définition d'ordre général de cette discipline : « une machine fait de l'apprentissage par rapport à une tâche T, une performance métrique P et une sorte d'expérience E quand elle améliore sa performance P sur la tâche T selon l'expérience acquise E ». La connaissance acquise permet ainsi de guider les prédictions de nouveaux cas qui n'ont pas été rencontrés auparavant.

Plus précisément, un algorithme d'apprentissage prend en entrée un ensemble de données d'entraînement D et retourne une fonction de prédiction f qui pour chaque nouvel élément x_i fera correspondre une valeur de sortie $f(x_i)$. La modélisation et la résolution des tâches d'apprentissage sont basées sur l'idée d'approximation de cette fonction qui se situe au cœur de ML. Il existe de nombreux types de méthodes d'approximation : modèles linéaires de régression (Bishop, 2006), machine à vecteurs de support (Boser et al., 1992), arbre de décision (Liaw & Wiener, 2002), réseaux de neurones et plus récemment l'apprentissage profond (Lecun et al., 1998). Tel qu'illustré dans la figure 2.1, les méthodes ML peuvent être catégorisées suivant 2 axes. Tout d'abord, la nature des inputs aboutit à la distinction de trois paradigmes : l'apprentissage supervisé (données labellisées), non-supervisé (données non labellisées) et par renforcement (données séquentielles). Le deuxième critère considère l'utilisation (ou non) de l'apprentissage profond pour l'approximation d'une composante du système d'apprentissage.

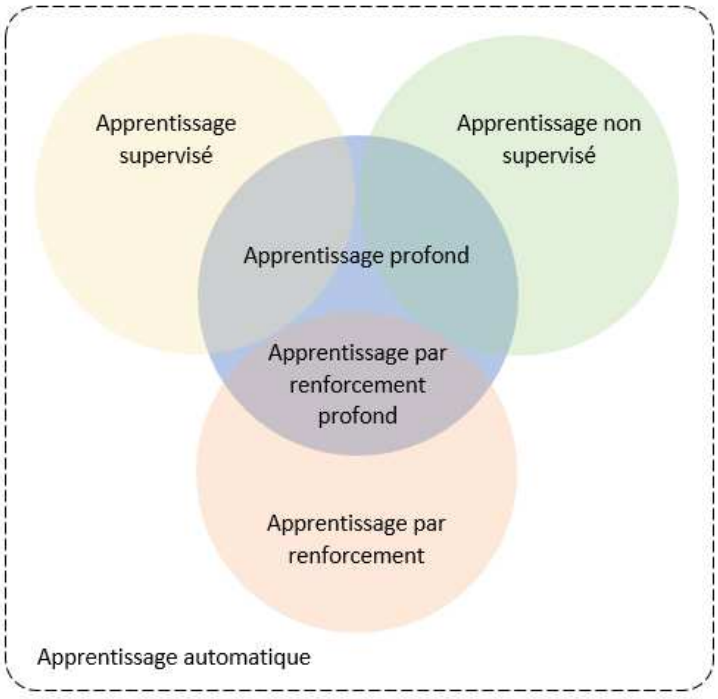

FIGURE 2.1 – Les paradigmes de l'apprentissage automatique.

2.2.2 Paradigmes

En fonction du type de signal reçu avec les données d'entrée, les algorithmes ML peuvent être divisés en 3 catégories : apprentissage supervisé, non supervisé, et par renforcement. Ces paradigmes d'apprentissage sont décrits ci-dessous :

- Apprentissage supervisé : ce type d'apprentissage utilise des données étiquetés a priori par un expert d'où son qualificatif de « supervisé ». Dans ce cas, D correspond à un ensemble de n paires $\{(x_i, y_i) \mid i = 1..n\}$ où $x_i \in X$ domaine d'exemples et $y_i \in Y$ domaine de classes. La tâche d'apprentissage consiste à déterminer une fonction de correspondance $f : X \rightarrow Y$ à partir de l'entraînement de l'algorithme d'apprentissage sur D.
- Apprentissage non supervisé : il vise à caractériser la distribution et les relations entre les entités étudiées en recherchant une structure cachée dans des collections de données non étiquetées. Dans ce cas, D correspond l'ensemble d'éléments $x_i \in X$. La tâche d'apprentissage consiste à déduire des structures formées par x_i groupées selon des régularités ou propriétés communes à partir de l'entraînement de l'algorithme d'apprentissage sur D.
- Apprentissage par renforcement : son objectif est de résoudre des problèmes de prise de décision séquentielle. En fonction de l'état de l'environnement, un agent essaye de prédire une action maximisant une mesure des avantages cumulés. Cela consiste à effectuer une transposition de ses observations x_i en des actions générant des récompenses $f(x_i)$, où f est qualifiée de politique.

Parfois, le RL est considéré comme une version de l'apprentissage non supervisé vu qu'il n'utilise pas des données étiquetées. Toutefois, les algorithmes RL reçoivent des informations d'apprentissage sous la forme de signaux de récompense, ce qui constitue une similarité avec le paradigme supervisé. Découvrant la structure dans l'expérience d'un agent et améliorant le comportement évalué par un signal de l'environnement, le RL est ainsi considéré comme un troisième paradigme à part.

2.2.3 Modèles d'apprentissage

L'une des plus importantes dichotomies du ML est celle qui existe entre les modèles paramétriques et non paramétriques. La principale distinction entre ces familles de modèles réside dans les différentes manières d'approcher le traitement de données. Dans les modèles paramétriques, le nombre de paramètres spécifiant un modèle est fixe, tandis que dans les modèles non paramétriques, le nombre de paramètres augmente avec la quantité de données d'apprentissage. Les modèles paramétriques sont généralement plus rapides à entraîner et faciles à interpréter, mais ils font des hypothèses contraignantes et parfois inutiles sur la nature des distributions de données. Les modèles non paramétriques sont flexibles, mais difficiles à résoudre pour les larges bases de données (Murphy, 2012). Le tableau 2.1 présente les spécificités de chacun de ces types d'algorithmes.

Caractéristiques	Modèles Paramétriques	Modèles non paramétriques
Principe	Considérer un ensemble fini de paramètres θ. Inclure 2 étapes de traitement : (i) sélectionner une forme pour la fonction f, (ii) apprendre les coefficients de la fonction à partir des données d'entraînement. Les prédictions futures sont indépendantes des données déjà observées : $\mathcal{P}(f(x_i)\|\theta, D) = \mathcal{P}(f(x_i)\|\theta)$	Supposer que la distribution des données ne peut pas être définie en termes d'un ensemble fini de paramètres θ. L'entraînement consiste à (i) stocker les instances dans une table de recherche, (ii) apprendre à connaître les « modèles similaires ». Utiliser une mesure de distance pour effectuer la similarité et l'interpolation.
Avantage	La forme fonctionnelle fixe simplifie et accélère le processus d'apprentissage. Il suffit d'estimer les coefficients de f pour disposer d'un modèle prédictif.	La quantité d'information que θ peut capturer sur les données D peut augmenter à mesure que la quantité de données augmente. Cela les rend plus flexibles.
Faiblesse	La complexité du modèle est limitée même si le volume de données tend à augmenter, ce qui sanctionne la flexibilité de ces méthodes et leur pouvoir d'approximer la réalité des données.	L'augmentation du nombre de paramètre avec le volume de données ralentit l'apprentissage et engendre un coût de calcul plus lourd.
Exemple d'algorithmes	– Réseau de neurones – Régression logistique	– k plus proche voisins – Arbres de décision

Table 2.1 – Tableau comparatif des modèles paramétriques et non paramétriques.

2.3 Réseau de neurones artificiels

Les ANNs ont été utilisés avec succès dans diverses applications telles que la reconnaissance d'images et de la parole, le traitement du langage naturel, etc. De nombreux travaux ont tenté d'apporter un soutien théorique au succès de ces réseaux dont une grande partie a été consacrée à l'expressivité des ANNs, c'est-à-dire leur capacité d'approximer efficacement une variété de classes de fonctions. Le résultat classique bien connu sur ce sujet est le théorème d'approximation universelle, qui stipule que toute fonction continue peut être représentée par un ANN. Dans cette section nous présentons les spécificités des ANNs et les circonstances qui ont mené à l'essor de l'apprentissage profond. Un accent particulier est mis sur les CNNs qui constituent un composant de base des contributions de cette thèse en servant à approximer les politiques RL.

2.3.1 Propagation vers l'avant

Les ANNs constituent un important domaine de l'AI qui tente de reproduire le fonctionnement du cerveau humain. Ces réseaux sont également appelés réseaux feedforward car il n'y a pas de cycles : les signaux d'entrée traversent le réseau directement de la première couche (entrée) à la dernière (sortie). S'il existe d'autres couches entre celles d'entrée et de sortie, elles sont appelées couches cachées. La figure 2.2 montre un exemple d'ANN impliquant des couches classiques entièrement connectées où chaque neurone est connecté à toutes les couches précédentes.

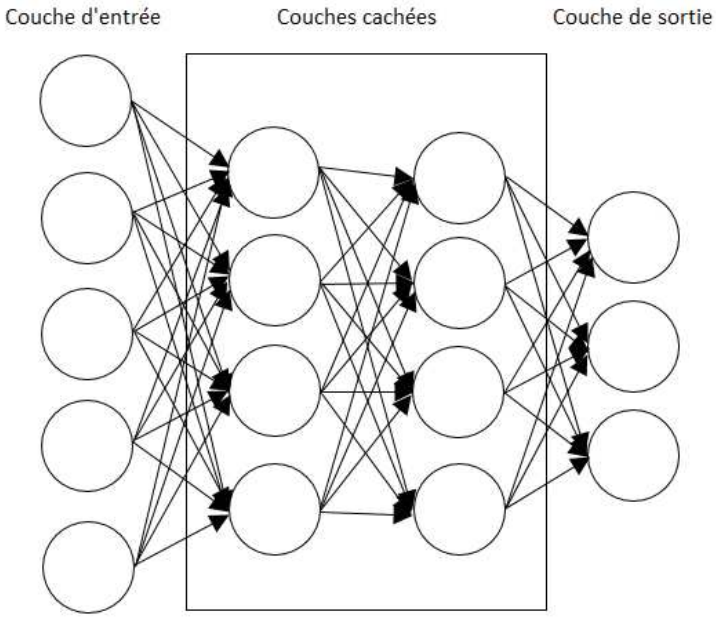

FIGURE 2.2 – Exemple d'architecture d'un ANN incluant une couche d'entrée, une de sortie et deux couches cachées.

L'objectif d'un ANN est d'approximer une fonction f^* en définissant une fonction $f_\theta(x)$ paramétrée par θ, appelée aussi modèle d'apprentissage, puis essayer de trouver les paramètres appropriés pour cette fonction en utilisant les données d'entraînement. Il existe deux types de paramètres, w et $b \in \theta$ qui conditionnent les performances prédictives de $f_\theta(x)$. Les pondérations de connexion w mesurent à quel degré la sortie d'un neurone impactera celle d'un neurone de niveau supérieur. Le biais b estime globalement la présence d'une caractéristique à travers l'ensemble des entrées. Par conséquent, le résultat d'un neurone i dans une couche j peut être formulé comme une combinaison linéaire d'entrées pondérées et de biais associé :

$$f_j(x_i) = \sum_{k=1}^{n} w_{ik}^j \; f_{j-1}(x_k) + b_i^j \tag{2.1}$$

Avec n correspond au nombre d'inputs provenant de la couche antérieure. Afin de permettre au réseau d'opérer des transformations non linéaires, une fonction d'activation ϕ est appliquée à l'équation précédente. Le résultat du neurone devient :

$$f_j(x_i) = \phi(\sum_{k=1}^{n} w_{ik}^j \ f_{j-1}(x_k) + b_i^j) \tag{2.2}$$

Nous remarquons que la fonction d'activation occupe une place centrale dans l'équation précédente et permet de différencier les ANNs des modèles linéaires. Le choix de cette fonction influence considérablement les performances et l'efficacité des calculs. Ainsi, de nombreuses fonctions d'activation ont été proposées pour améliorer l'apprentissage des ANNs (Ramachandran et al., 2018) dont les plus utilisées sont :

- Unité de rectification linéaire : $ReLU(x) = max(0, x)$
- Fonction sigmoïde : $\sigma(x) = \frac{1}{1+e^{-x}}$
- Fonction tangente hyperbolique : $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

En présence de l couches dans l'architecture du réseau, la fonction globale d'approximation se compose des différentes transformations effectuées au niveau de chaque couche de neurones :

$$f_\theta(x) = f_l(f_{l-1}(\dots f_1(x))) \tag{2.3}$$

Selon le théorème d'approximation universelle (Hornik et al., 1989), les ANNs peuvent approximer n'importe quelle fonction avec un nombre de neurones suffisant. Cette propriété d'approximation universelle a été démontrée sur différentes classes de fonctions permettant le passage d'une représentation tabulaire adaptée à des problèmes de petite taille vers une application du ML à grande échelle (Barron, 1993).

2.3.2 Optimisation par rétropropagation

L'optimisation de f_θ consiste à déterminer les paramètres θ pour approximer f^*. Cette procédure repose sur l'évaluation d'une fonction de perte $\mathcal{L}(f_\theta, D)$ qui quantifie l'erreur d'approximation de f_θ sur les données D avec les paramètres θ. Par exemple, dans un scénario d'apprentissage supervisé, la perte est assimilée à l'imprécision d'appariement entre les prévisions et les valeurs « vérité-terrain ». L'objectif consiste alors à minimiser cette erreur par rapport aux paramètres du réseau :

$$\theta^* = \arg\min_\theta \mathcal{L}(f_\theta, D) \tag{2.4}$$

Pour résoudre ce problème d'optimisation, la méthode de descente de gradient est généralement préconisée dans le cas des ANNs. Elle consiste à calculer le gradient de la fonction de perte par rapport à θ :

$$\nabla_\theta \mathcal{L}(f_\theta, D) = \frac{\partial \mathcal{L}(f_\theta, D)}{\partial \theta} \tag{2.5}$$

Puis effectuer la descente suivante pour mettre à jour les paramètres :

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(f_\theta, D) \tag{2.6}$$

Où α est un taux d'apprentissage. Il est à noter que selon que l'objectif de l'évaluation soit

de minimiser (taux d'erreur) ou de maximiser (retour de récompense) un critère, il s'agira d'effectuer une descente ou une montée de gradient. Dans le dernier cas, le signe moins sera remplacé par un signe plus dans l'équation précédente. Un standard d'entraînement pour les ANNs est l'algorithme de rétropropagation (Rumelhart et al., 1986) qui exécute la descente de gradient en un seul passage rétrograde. Il consiste à calculer la perte pour les neurones de sortie puis à appliquer des mises à jour par gradient de la couche de sortie vers la couche d'entrée. La rétropropagation applique la règle de calcul en chaîne en utilisant la programmation dynamique. Considérons l'entrée et la sortie de neurone i au niveau de la couche j, notées respectivement z_i^j et a_i^j et définies comme suit :

$$z_i^j = \sum_{k=1}^{n} w_{ik}^j \; a_k^{j-1} + b_i^j \tag{2.7}$$

$$a_i^j = \phi(\sum_{k=1}^{n} w_{ik}^j \; a_k^{j-1} + b_i^j) \tag{2.8}$$

En représentant chaque entrée de neurone z_i^j par un nœud et les gradients de la perte par les connexions entre ces nœuds, la règle de calcul de chaîne démontre par récurrence que pour trouver le gradient par rapport à un nœud, il suffit de connaître le gradient par rapport à son prédécesseur et le gradient de ses prédécesseurs par rapport au nœud lui-même. Cela se traduit par la relation suivante :

$$\frac{\partial \mathcal{L}(f_\theta, D)}{\partial z_i^j} = \sum_{k=1}^{n} \frac{\partial \mathcal{L}(f_\theta, D)}{\partial z_k^{j+1}} \; \frac{\partial z_k^{j+1}}{\partial z_i^j} \tag{2.9}$$

Après avoir calculé la valeur du gradient au niveau de chaque entrée de neurone, il suffit d'en déduire le gradient au niveau de chaque paramètre sur la base des dérivées partielles de l'équation 2.7 :

$$\frac{\partial \mathcal{L}(f_\theta, D)}{\partial w_{ik}^j} = \frac{\partial z_i^j}{\partial w_{ik}^j} \; \frac{\partial \mathcal{L}(f_\theta, D)}{\partial z_i^j} = a_k^{j-1} \; \frac{\partial \mathcal{L}(f_\theta, D)}{\partial z_i^j} \tag{2.10}$$

$$\frac{\partial \mathcal{L}(f_\theta, D)}{\partial b_i^j} = \frac{\partial z_i^j}{\partial b_i^j} \; \frac{\partial \mathcal{L}(f_\theta, D)}{\partial z_i^j} = \frac{\partial \mathcal{L}(f_\theta, D)}{\partial z_i^j} \tag{2.11}$$

Deux modes d'apprentissage d'ANN sont possibles. Le premier est l'entraînement en ligne où les paramètres du réseau sont constamment mis à jour par rétropropagation pour chaque entrée traitée. Dans le deuxième mode appelé descente de gradient stochastique (SGD), des sous-ensembles de données d'entraînement sont échantillonnés aléatoirement. Le calcul du gradient et la descente associée sont effectués pour l'intégralité du sous-ensemble sélectionné. Ainsi, la différence avec l'entraînement en ligne est que les paramètres sont constants pour tous les éléments de l'échantillon.

2.3.3 Approche profonde

L'apprentissage profond fait référence au traitement de ML au sein d'un ANN multicouche avec au moins une couche cachée (Jarrett et al., 2009), d'où la qualification

de réseaux de neurones profonds. Ce concept se définit par opposition à l'apprentissage superficiel, tel que la régression linéaire, les machines à vecteur support (SVM) et les arbres de décision, où il y a seulement une couche d'entrée et une couche de sortie. Ces méthodes classiques requièrent également des extracteurs de caractéristiques conçus par les experts, ce qui s'avère parfois être une tâche difficile ou infaisable. Les DNNs apprennent automatiquement les représentations à partir d'entrées brutes en récupérant hiérarchiquement la composition des signaux traités. Les couches cachées réalisent des représentations transitoires des données d'entrée évoluant de caractéristiques de bas niveau (lignes et bords dans le cas d'une classification d'image) vers des motifs plus complexes au fur et à mesure que des couches plus profondes sont atteintes. Cette représentation distribuée est l'idée centrale de l'apprentissage profond, ce qui implique que de nombreuses caractéristiques peuvent spécifier chaque entrée et que chaque caractéristique peut représenter de nombreuses entrées. Les représentations profondes et distribuées permettent de faire face aux défis de la grande dimensionnalité et de favoriser l'entraînement de bout en bout qui évite l'ingénierie manuelle d'extracteurs de caractéristiques.

L'origine d'ANN remonte aux années 1950 avec l'avènement du « perceptron », le premier réseau de neurones mis au point par Frank Rosenblatt. Cependant, les modèles de neurones n'ont été largement utilisés que récemment. En effet, le travail de Hinton et al. (2006) a marqué le début d'un regain d'intérêt pour les ANNs en montrant qu'il était possible d'entraîner leur forme profonde de manière efficace. Au cours des années suivantes, le domaine d'apprentissage profond a connu un progrès exponentiel soutenu par les avancées majeures en matière d'algorithmes d'apprentissage, de matériel informatique et, de façon moins intuitive, de la disponibilité de bases de données d'entraînement de haute qualité :

- Amélioration des algorithmes d'entraînement : les chercheurs ont pu surmonter certaines limites « classiques » des ANNs facilitant la convergence rapide et la fiabilité de l'étape d'entraînement. Les avancées les plus importantes sont l'émergence des algorithmes de rétropropagation, l'introduction de la fonction d'activation ReLU qui a permis de surmonter le problème de la disparition du gradient et l'implémentation d'une étape de dropout (Srivastava et al., 2014) pour diminuer les paramètres du réseau et limiter l'impact du sur-apprentissage.
- Construction de larges bases de données d'entraînement : les jeux de données, dont plusieurs ont été publiés dans des revues scientifiques, font partie intégrante du domaine de ML. En effet, la conception et l'étiquetage (dans le cas d'apprentissage supervisé) de milliers, voir parfois de millions d'instances constituent un exercice difficile et coûteux en termes de moyens et de temps, permettant de tester et d'exploiter les capacités des DNNs à grande échelle. Parmi les benchmarks d'entraînement on peut citer ImageNet (Deng et al., 2009) (plus de 14 million d'instances) et CIFAR (Krizhevsky et al., 2012) (60 mille instances) pour la reconnaissance d'objets, MNIST (60 mille instances) pour la détection de caractères manuscrits et SAT-4 Airborne (Basu et al., 2015) pour les images satellitaires.
- La transformation des systèmes informatiques a accompagné pendant les dernières années le développement et l'intégration de l'apprentissage profond dans plusieurs domaines de recherche et de la vie réelle. Ainsi, l'essor de l'apprentissage profond depuis 2006 est également le résultat de l'émergence d'une puissance de calcul en

évolution continue. En particulier, l'utilisation des GPUs (Graphical Processing Unit) a permis de réaliser des opérations mathématiques complexes de façon parallèle et la conception de réseaux beaucoup plus profonds. Des outils technologiques récents tels que TensorFlow, Pytorch et Keras ont quant à eux facilité l'accès aux GPUs pour les entités de recherche académiques et industrielles.

2.4 Réseau de neurones à convolution

Les ANNs classiques tels que les perceptrons multicouches ne sont pas bien adaptés à certains types de données, en particulier les images. En effet, ils sont supposés prendre des vecteurs en tant que données d'entrée. Cependant, transformer les images en vecteurs cause la perte des informations spatiales et de voisinage entre pixels. D'un autre côté, l'apprentissage superficiel basé sur l'extraction de variables d'intérêt nécessite un grand volume d'expérience sur des tâches de vision par ordinateur. Les CNNs introduits par LeCun et al. (1990) ont révolutionné le traitement d'images en supprimant l'extraction manuelle de caractéristiques et en agissant directement sur les matrices. De nos jours, ils sont largement utilisés dans un grand nombre de problèmes de reconnaissance de formes. Dans le reste de cette section, les spécificités des architectures CNN sont détaillées à travers les trois traitements essentiels qu'elles englobent : convolution, pooling et couche entièrement connectée.

2.4.1 Convolution

La couche de convolution est le composant de base d'un CNN qui a été inspiré par la recherche sur le traitement hiérarchique du signal dans le cortex visuel des mammifères (Hubel & Wiesel, 1962). Les cellules simples détectent les attributs primitifs, tandis que les structures évoluées sont extraites subséquemment par des cellules plus complexes. Ainsi, la couche de convolution consiste en un ensemble de cartes de caractéristiques issues de la convolution de différents filtres (noyaux ou kernels) avec une image en entrée ou une sortie de la couche précédente. Les cartes bidimensionnelles sont empilées pour produire le volume résultant de la couche de convolution. Ce processus réduit considérablement la complexité du réseau puisque les neurones d'une même carte de caractéristiques partagent les mêmes pondérations et biais, en maintenant un nombre faible de paramètres à apprendre (Wu & Gu, 2015). La convolution discrète entre deux fonctions f et g est définie par :

$$(f * g)(x) = \sum_t f(t)\, g(x+t) \tag{2.12}$$

Pour les signaux bidimensionnels tels que les images, on considère les convolutions $2D$:

$$(K * I)(i,j) = \sum_{m,n} K(m,n)\, I(i+m, j+n) \tag{2.13}$$

où K est un noyau de convolution appliqué à un signal $2D$ (ou une image) I. Cette définition bidimensionnelle peut facilement être étendue aux matrices à plus grandes dimensions. Comme le montre la figure 2.3, le principe d'une convolution $2D$ consiste à faire glisser un noyau de convolution sur l'image. À chaque position, une convolution est calculée entre le

noyau et la partie de l'image traitée (champ récepteur). Le noyau se déplace d'un nombre de pixels appelé foulée.

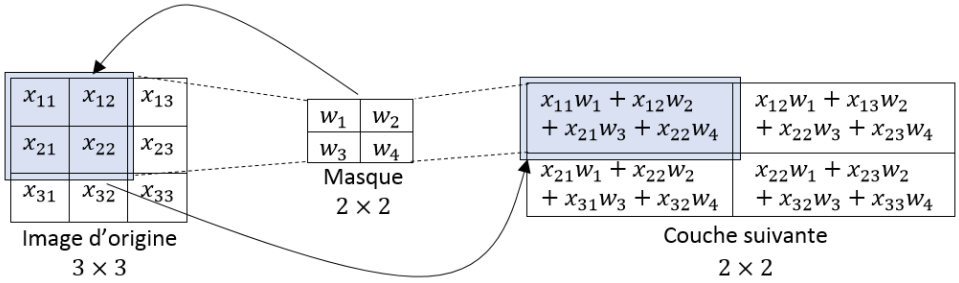

FIGURE 2.3 – Opération de convolution à 2D avec une foulée de 1 pas.

Les hyperparamètres caractérisant une couche de convolution sont la profondeur F (nombre de filtres), la foulée S (stride, mouvement de filtre d'un champ récepteur au suivant) et le remplissage zéro (padding) P pour contrôler la taille d'entrée (Chollet, 2017b). En supposant que la taille du filtre $(\boldsymbol{hauteur, largeur, profondeur}) = (h, l, p)$, les dimensions des cartes de caractéristiques générées peuvent être obtenues selon la formule suivante :

$$(H_1, L_1, F) = \left(\frac{H + 2P - h}{S} + 1,\ \frac{L + 2P - l}{S} + 1,\ F\right) \tag{2.14}$$

où H et L correspondent respectivement à la hauteur et la largeur de l'image d'entrée.

2.4.2 Pooling

Les architectures CNN alternent généralement des couches de convolution et de pooling. Ces dernières ont pour objectif de réduire la complexité du réseau et d'éviter le problème de sur-apprentissage. Au niveau biologique, le pooling est assimilé au comportement de cellules corticales complexes qui révèlent un certain degré d'invariance de position. Un neurone de la couche de pooling est connecté à une région de la couche précédente en effectuant une fonction non paramétrée. Ainsi, il diffère de la convolution puisqu'il n'a pas de pondération ni de biais pouvant être appris et, en outre, il conserve la même profondeur que la couche précédente. Le max pooling (Zeiler & Fergus, 2013) est l'un des types de pooling les plus courants consistant à conserver la valeur maximale d'un cluster de neurones (voir figure 2.4). Cela signifie que le max pooling détecte si une caractéristique donnée a été identifiée dans un champ récepteur sans enregistrer l'emplacement exact (Nielsen, 2018).

2.4.3 Couche entièrement connectée

Les couches de convolution identifient les caractéristiques locales dans les données d'entrée, telles que les lignes et les formes. La couche entièrement connectée réalise le raisonnement de haut niveau (classification pour le cas d'image) en combinant les informations de toutes les couches précédentes. Comme dans un ANN ordinaire (voir figure 2.2), les neurones de ce niveau sont entièrement connectés à tous ceux de la couche

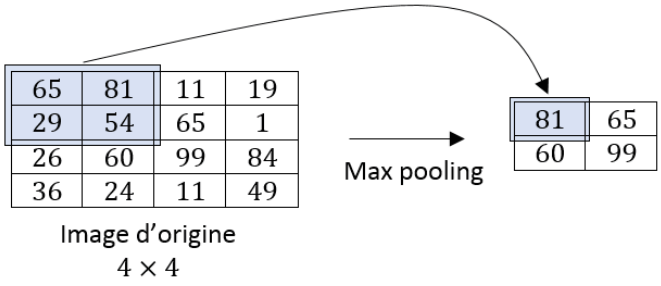

FIGURE 2.4 – Exemple d'une opération de max pooling de taille 2x2

précédente. Le CNN est complété par une autre couche entièrement connectée utilisant la fonction d'activation softmax pour lui donner la capacité de faire des prédictions. L'objectif principal de la fonction softmax est de transformer la sortie (non normalisée) des unités d'une couche antérieure en une distribution de probabilité :

$$softmax(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad avec \quad \sum_i softmax(x)_i = 1 \tag{2.15}$$

2.5 Cas d'application

L'aspect technologique est primordial dans notre étude étant donné l'existence de prérequis en termes de ressources informatiques pour dérouler correctement des applications en apprentissage profond très demandeur en capacités de calcul. Par conséquent, une partie de la recherche a été affectée à l'exploration des outils dédiés à ce domaine. Nous présentons dans cette section les langages et les environnements de développement spécifiques au ML, et plus particulièrement à l'apprentissage profond. Nous présentons également un cas d'application de CNN à la classification d'images satellitaires qui a constitué notre point d'entrée pour la maîtrise de la partie technologique de la recherche. Cet exemple d'application ne rentre pas dans le périmètre des cas de la vie réelle qui nécessitent le recours au traitement lourd et coûteux par méta-apprentissage, objet de notre thèse, vu la stationnarité et l'absence de variabilité dans ce type de données. Toutefois, nous avons choisi les images satellitaires vu qu'elles constituent un cas classique de classification pour lequel il existe plusieurs bases de données benchmark disposant de cartes vérité-terrain de haute qualité. Cela a facilité la mise au point de l'expérimentation et la focalisation sur les logiciels et les architectures utilisés.

2.5.1 Outils informatiques

La plupart des innovations récentes au niveau des logiciels et de l'infrastructure informatiques tournent autour de l'apprentissage profond et sont en majorité open source. Nous allons énumérer dans ce paragraphe, sous forme de tableaux comparatifs, les outils les plus utilisés pour cette branche de ML, parmi les langages, les librairies et les IDEs.

2.5.1.1 Langages

Parmi la variété de langages disponibles, quelques uns sont dominants dans la sphère ML. Python est considéré comme le langage de référence absolue offrant une large gamme d'outils d'apprentissage profond. Historiquement, R et Lisp sont parmi les pionniers et sont plutôt dédiés aux statistiques et aux techniques superficiels du ML. Java se présente comme le langage le plus stable mais le plus compliqué pour le développement et l'apprentissage (voir tableau 2.2).

Langage	Spécificités	Utilisateurs de référence
Python	Le plus présent, avec une large communauté de support. Facile à apprendre, une syntaxe simple et de nombreuses librairies sont disponibles.	Amazon, Google.
Java	Programmation standard et performance d'exécution. Codage relativement complexe.	YouTube, Amazon, eBay et LinkedIn.
R	Orienté statistiques et analyse prédictive. Très utilisée dans la data science et la big data.	Google, Uber, New York times.
Lisp	Un des plus anciens, plutôt orienté vers le ML classique. Parmi les plus difficiles à coder, il n'est pas recommandé aux débutants.	N.A

Table 2.2 – Langages largement utilisés dans le développement de méthodes ML.

2.5.1.2 Librairies

Plus complètes que les librairies AI classiques, les librairies orientées apprentissage profond sont généralement désignées sous la catégorie de framework. Malgré le niveau avancé requis pour son utilisation, Tensorflow est de loin le framework le plus sollicité par la communauté de recherche grâce à une documentation et un support intensifs. Toutefois, Keras et Pytorch constituent encore une bonne option, spécialement pour les débutants et les travaux d'initiation, vu le codage et les modèles simplifiés qu'ils offrent. Caffe est le framework le plus ancien dans le domaine d'apprentissage profond, mais n'est plus d'actualité et connais moins de mises à jour que les autres concurrents (tableau 2.3).

Librairie	Langage supporté	Éditeur	Spécificité
Caffe	Python, C++, Matlab.	Berkeley AI Research (BAIR).	Orientée traitement d'images, peu supportée.
Keras	Python	François Chollet (Chollet, 2017b).	Codage simplifié et expérimentation rapide. Appropriée aux débutants.
Pytorch	Python	Facebook AI Research (FAIR).	Définition de modèles transparente et dynamique. Adaptée aux besoins de recherche.
TensorFlow	C++, Python.	Google Brain.	La plus utilisée, documentée et supportée. Efficace pour des projets à un niveau avancé.

Table 2.3 – Librairies adaptées à l'apprentissage profond.

2.5.1.3 Environnements de développement intégrés

La majorité des IDEs existants permettent de coder en Python, langage préféré des travaux en apprentissage profond et disposent naturellement des trois composants de base : éditeur, débogueur, et compilateur. Quelques un sortent du lot dont les plus simples sont Spyder (adapté à la recherche) et JuPyter Notebook (dédié à la solution cloud de Google). Pycharm est cependant considéré parmi les meilleurs avec ses deux versions dédiées aux industriels et à la recherche (voir tableau 2.4).

IDE	Éditeur	Spécificité
Spyder.	Pierre Raybaut.	Simple et léger avec une documentation détaillée. Approprié pour la recherche scientifique.
Pycharm.	Jetbrains.	Beaucoup d'options relatives au codage. Largement adopté par les grandes entreprises dans sa version Professional Edition. La Community Edition dispose également de ressources et tutoriaux avancés.

JuPyter Notebook.	Community project.	Application web simple avec plusieurs préférences utilisateur de partage et de simulation. Permet notamment d'utiliser Google Colaboratory, entièrement dans le cloud.

TABLE 2.4 – Exemples d'IDEs supportant le codage en Python.

2.5.2 Classification d'image hyperspectrale

Au niveau de ce paragraphe, nous exposons les détails de l'implémentation d'un algorithme d'apprentissage profond qui consiste à traiter un cas de classification d'image hyperspectrale par un CNN en utilisant la librairie TensorFlow.

2.5.2.1 Problématique

Les données hyperspectrales présentent une information riche et volumineuse offrant une signature plus détaillée des objets observés. Une image hyperspectrale est représentée par un tenseur $3D$ de dimensions $H \times L \times C$, où H et L correspondent à la hauteur et à la largeur de l'image et C à ses canaux (bandes spectrales), tel qu'illustré dans la figure 2.5. La richesse de ces données a encouragé la caractérisation de la couverture du sol et a permis d'envisager un élargissement des possibilités de détection. Cette grande dimensionnalité rend la tâche de classification encore plus complexe vu que le nombre de caractéristiques (des centaines de bandes spectrales) est relativement élevé par rapport aux échantillons d'apprentissage disponibles (pixels d'entraînement) engendrant la dégradation de la performance des méthodes de classification « shallow ». En effet, devant l'augmentation du nombre de bandes hyperspectrales, le nombre d'échantillons nécessaires pour maintenir un minimum de fiabilité statistique pour la tâche de classification croît de manière exponentielle. Ce phénomène est appelé dans la littérature « malédiction de dimensionnalité » ou « Hughes effect » (Hughes, 1968).

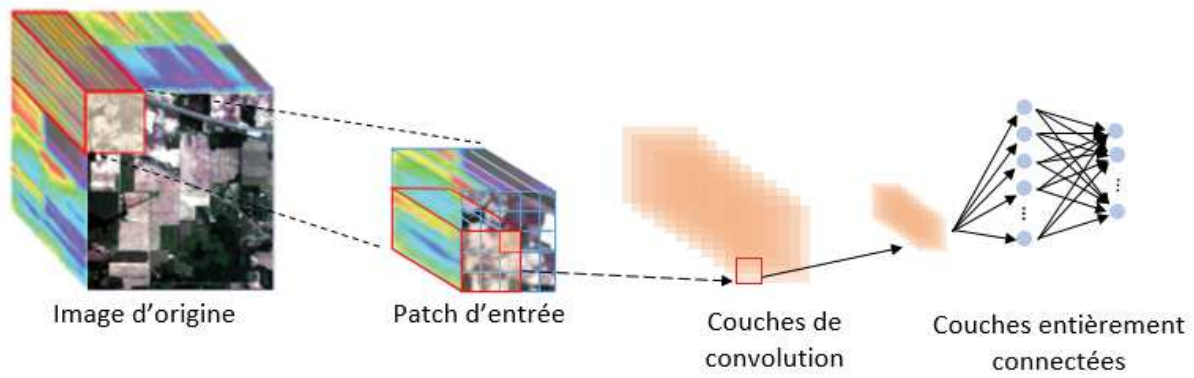

FIGURE 2.5 – Cube hyperspectral traité par un CNN.

2.5.2.2 Modèle de classification

Comme vu dans les sections précédente, les CNNs réduisent systématiquement les paramètres du modèle d'apprentissage et par conséquent, la dimensionnalité du problème. Cette caractéristique est due à leurs fonctions de partage de poids et de connexion sélective où la connectivité d'un neurone de couche cachée est limitée à un sous-ensemble de neurones de la couche précédente. Nous considérons l'exploitation d'un CNN pour la classification des données hyperspectrales, c'est-à-dire la classification de chaque pixel en un nombre prédéfini de classes en fonction de ses propriétés spectrales et spatiales. Les caractéristiques spectrales sont associées aux propriétés de réflexion de chaque pixel pour chaque bande spectrale, tandis que les informations spatiales sont calculées en tenant compte de son voisinage. Plus précisément, afin de classer les pixels $x_{i,j}$ de coordonnées $(i = 1..H,\ j = 1..L)$ sur le plan de l'image et de fusionner les informations spectrales et spatiales, nous utilisons un patch carré de taille paramétrable $N \times N$ centré sur le pixel $x_{i,j}$. Notons $y_{i,j}$ l'étiquette de classe du pixel $x_{i,j}$ et $m_{i,j}$ le tenseur $3D$ centré sur le pixel $x_{i,j}$ de taille $N \times N \times C$ et contenant l'information spectrale et spatiale de ce pixel. Nous constituons alors la nouvelle base de données $D = (m_{i,j},\ y_{i,j})$ pour tous les pixels de l'image qui servira pour l'entraînement du CNN de classification. Ainsi, le tenseur $m_{i,j}$ composé de C matrices de dimension $N \times N$ constitue l'input du CNN. Ce dernier procédera à la codification des données spectrales et spatiales en caractéristiques hiérarchiques de haut niveau à travers ses couches de convolution avant de prédire les classes en se basant sur ses dernières couches entièrement connectées. Le schéma d'exécution du processus d'entraînement est repris dans l'algorithme 1.

2.5.2.3 Expérimentation et résultats

Au cours de l'expérimentation, le paramètre N a été fixé à 5 afin de prendre en compte les 24 pixels voisins les plus proches de chaque pixel traité. Concernant l'architecture du CNN, elle comprend 2 couches de convolutions et 2 couches entièrement connectées. Nous n'utilisons pas de couche de pooling car les patchs de pixels ne posent pas de problématique d'invariance comme pour le cas d'autres types de tâches de classification. La première couche du réseau CNN proposé est une couche de convolution $Conv_1$ constituée de 3 filtres de taille $3 \times 3 \times C$. Cette couche produit 3 cartes de caractéristiques de dimensions 3×3 qui constitueront l'entrée d'une deuxième couche de convolution $Conv_2$. Cette dernière comprend elle aussi 3 filtres de tailles 3×3. Le dernier output est traité par 2 couches entièrement connectées pour produire le résultat de la classification. L'environnement technologique inclut le langage Python pour le développement, la bibliothèque TensorFlow et l'IDE Pycharm. L'expérimentation s'est déroulée avec un serveur NVIDIA P5000 (1GPU, 16 GB RAM).

Le site étudié est celui d'Indian Pines capturé par le satellite Aviris. L'image du site est de taille 145×145 pixels et de résolution 20 m/p. Elle comporte 220 bandes spectrales avec une couverture de $0{,}4\ \mu m$ à $2{,}5\ \mu m$. L'expérimentation a produit une précision de classification de 91%. L'image ainsi que la vérité-terrain et le résultat de la classification sont exposés dans la figure 2.6.

Algorithme 1 : CNN pour la classification d'images satellitaires hyperspectrales

Entrée : I = Image hyperspectrale, Y= Image vérité terrain, θ =Initialisation aléatoire du modèle.

Sortie : θ^* `// Modèle optimisé.`

e = Nombre d'époque
b = Taille de batch
N = Taille de patch
C = Profondeur (I)
I_{norm} = Normalisation (I)
P = Creation_patchs (I_{norm},N,C) `/* Partitionner` I_{norm} `en patchs de taille` $N \times N \times C$ `*/`
D_{Tr} = Echantillon_entrainement (P)
D_{Ts} = Echantillon_test (P)
K = Creation_batches (D_{Tr}, b)
k = Nombre de batches

pour $i = 1 : e$ **faire**
  pour $j = 1 : k$ **faire**
    `// Parcourir les batches`
    $Y^{'}$ ← Prediction_classe (K_j, θ) `/* Extraction de caractéristiques par couches de convolutions et classification par couches entièrement connectées pour les patchs du batch` K_j `*/`
    Er ← Calcul_erreur ($Y^{'}$,Y) `/* Calcul de l'erreur en fonction de la prédiction et de la vérité terrain */`
    $\theta^{'}$ ← Optimisation (θ, Er) `/* Effectuer une descente de gradient stochastique */`
  fin
fin

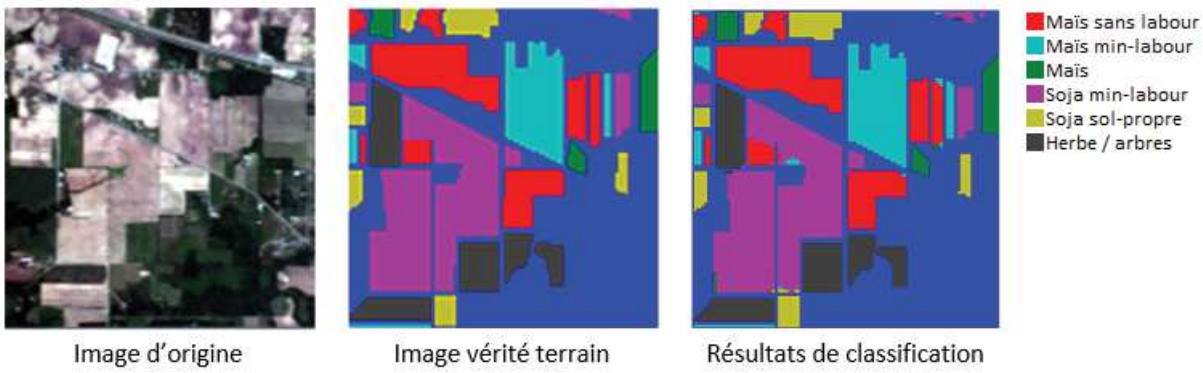

FIGURE 2.6 – Résultats de classification de l'image Indian Pines.

2.6 Conclusion

A travers ses nombreux paradigmes, notamment l'apprentissage supervisé, non supervisé, et le RL, le ML se base sur des connaissances préalables pour résoudre des problèmes de différentes natures. Les DNNs ont ouvert la voie à une application à plus grande échelle du ML grâce à leurs représentations profondes et distribuées des données surmontant l'obstacle de la grande dimensionnalité et instaurant la notion d'entraînement de bout en bout qui ne nécessite pas l'ingénierie manuelle d'extracteurs de caractéristiques. Plus spécifiquement, les CNNs ont révolutionné le traitement d'images et l'approximation de fonctions en général suite à l'adoption d'un traitement hiérarchique du signal inspiré par le fonctionnement du cortex visuel de mammifères.

Trois caractéristiques principales rendent cette technique d'apprentissage profond efficace et adaptée aux données visuelles. Premièrement, les champs récepteurs reflètent parfaitement les données de l'image, corrélées localement et indépendantes au niveau des segments globaux. Deuxièmement, les pondérations partagées permettent une réduction substantielle des paramètres du réseau sans pour autant altérer le résultat de traitement étant donné que la convolution est applicable à l'image toute entière. Enfin, les images structurées en grille permettent des opérations de pooling qui simplifient les données sans perdre les informations utiles (Nielsen, 2018). Tirant profit de ces avantages, nous implémentons des politiques d'apprentissage basées sur les CNNs dans nos approches RL et MRL de conduite autonome détaillées dans les chapitres 3 et 4.

Chapitre 3

Apprentissage par renforcement pour le contrôle

3.1 Introduction

Les systèmes de contrôle revêtent une importance particulière dans quasiment toutes les activités humaines telles que le transport, l'industrie, le bâtiment et l'exploitation de ressources naturelles. Un dénominateur commun dans les applications du contrôle est le besoin d'influencer ou de modifier le comportement de systèmes dynamiques pour atteindre des objectifs prédéfinis. Une approche pour y parvenir consiste à attribuer une mesure de performance à chaque trajectoire d'état du système. Le problème de contrôle est ensuite résolu en recherchant une politique de contrôle qui mène le système le long de trajectoires correspondant à la meilleure valeur de l'indice de performance. Cette approche réduit le problème de la recherche de bonnes politiques de contrôle à la recherche de solutions à un problème d'optimisation mathématique.

Les premiers travaux dans le domaine du contrôle optimal remontent aux années 1950 avec les recherches pionnières de la programmation dynamique introduite par Bellman. La programmation dynamique fait toujours partie des outils de pointe couramment utilisés pour résoudre les problèmes de contrôle lorsqu'un modèle représentant le système est disponible. L'idée alternative de trouver une solution en l'absence de modèle a été explorée ultérieurement et a conduit au développement du paradigme de RL. Le thème central de la recherche en RL est la conception d'algorithmes qui apprennent les politiques de contrôle uniquement à partir de la connaissance des échantillons ou des trajectoires de transitions collectées par interaction avec le système.

Malgré quelques succès du RL par le passé sur des tâches de faible dimensionnalité, les approches précédentes ont montré leurs limites dans le traitement de problèmes avec de larges espaces état-action à cause de l'utilisation de représentations exactes des fonctions de valeurs et des politiques. Ce défi était déjà connu lors du développement des premières techniques de la programmation dynamique et a été qualifié de « malédiction de la dimensionnalité » par Bellman (1957). Ce n'est qu'au cours des dernières années que le RL a connu un regain de popularité et est devenu capable de résoudre des problèmes décisionnels séquentiels relativement compliqués avec l'avènement de l'apprentissage profond qui a fourni des solutions effectives pour l'approximation de fonctions adaptée à la grande dimensionnalité.

Implémentés dans de nombreux algorithmes RL, les DNNs ont permis d'obtenir des résultats remarquables dans certaines applications telles que les jeux (Mnih et al., 2015; Silver et al., 2016) et les manipulations robotiques avancées (Levine et al., 2016; Lillicrap et al., 2016), dépassant parfois la performance humaine. Malgré ces progrès significatifs, le DRL rencontre encore des difficultés à être étendu à grande échelle en dehors de la sphère de la recherche à cause d'un entraînement « inefficace » aux données et de sa tendance à

se spécialiser aux tâches traitées. Nous nous proposons dans ce chapitre de développer une approche RL actor-critic robuste et de l'évaluer sur une tâche complexe du monde réel qui est la conduite autonome.

L'objectif de cette contribution qui a été publiée dans (Jaafra et al., 2019a) est de répondre aux questions suivantes : comment consolider la robustesse d'une approche RL face à la tâche complexe de conduite autonome en milieu urbain ? A quels niveaux de ce système RL se situent les éventuelles limites induites par la variabilité des environnements ? Les réponses apportées à ces questions permettront, dans le chapitre 4, de motiver le recours au méta-apprentissage pour des politiques plus généralisables et l'optimisation de son implémentation afin de mieux répondre aux limites mises en exergue dans ce chapitre.

3.2 Éléments d'un problème RL

Nous avons vu dans le chapitre précédent que le RL est un paradigme particulier du ML qui améliore le comportement d'un agent en explorant son expérience évaluée grâce à un signal émis par l'environnement. Cette interaction agent-environnement effectuée à travers les espaces d'action et d'état conditionne le fonctionnement du RL et plus spécifiquement les éléments déterminants qui le composent dont principalement la politique, le signal de récompense et la fonction de valeur (Sutton & Barto, 2018). Dans cette section, nous rappelons brièvement les origines du RL et le contexte de son évolution. Nous formalisons également un problème RL en détaillant ses éléments de base dans le cadre d'une modélisation Markovienne pour la prise de décision séquentielle.

3.2.1 Contexte

Le ML a, par essence, exécuté de nombreux concepts psychologiques sous forme numérique dont principalement la capacité d'apprendre et de s'améliorer à partir des essais antérieurs sur la même tâche. Le RL ne déroge pas à ce principe en adoptant et appliquant de manière unique le concept de conditionnement en psychologie dans le but de faciliter un apprentissage fiable. En effet, l'idée de renforcement est empruntée aux travaux de Pavlov sur le comportement animal (Pavlov & Anrep, 1927). Dans le conditionnement Pavlovien, modélisé mathématiquement par la suite dans (Rescorla & Wagner, 1972), le renforcement est décrit comme l'effet d'excitation ou d'inhibition d'un comportement en réponse à un stimulus spécifique.

D'un point de vue mathématique, le point de départ du RL consiste à résoudre un processus de décision de Markov (MDP). Les MDPs sont une formalisation classique de la prise de décision séquentielle, dans laquelle les actions influencent non seulement les récompenses immédiates, mais aussi les situations ou les états ultérieurs à travers des récompenses futures. Ce processus répandu dans la théorie du contrôle stochastique trouve ses origines dans les travaux de Richard Bellman (Bellman, 1957) depuis les années 50 . Il existe dans la littérature trois classes fondamentales de méthodes permettant de résoudre des problèmes MDP : la programmation dynamique, les méthodes de Monte Carlo et l'apprentissage TD. Chaque classe de méthodes a ses avantages et ses inconvénients. Les méthodes de la programmation dynamique sont bien développées mathématiquement, mais nécessitent un modèle complet et précis de l'environnement. Les méthodes Monte Carlo ne requièrent pas de modèle et sont simples sur le plan conceptuel, mais ne sont

pas bien adaptées au calcul incrémental pas à pas. Enfin, les méthodes TD ne nécessitent aucun modèle et sont totalement incrémentielles, mais sont plus complexes à analyser. Ces méthodes diffèrent également de plusieurs manières en ce qui concerne leur efficacité et leur rapidité de convergence (Sutton & Barto, 2018).

En informatique, l'objectif d'un agent RL est d'apprendre à optimiser une mesure de la récompense cumulée reçue sur le long terme tout en interagissant avec son environnement et en mettant à jour sa politique en fonction des conséquences de ses actions. L'agent connaît généralement les actions qu'il peut entreprendre et l'état actuel de l'environnement. Cependant, il ne sait pas initialement comment se comporter de manière optimale qui maximise la récompense espérée pour tout état possible. En interagissant avec son environnement et en observant les conséquences de ses actions, l'agent peut apprendre à se rapprocher d'une politique optimale. Ainsi, l'un des aspects les plus passionnants du RL moderne réside dans ses interactions substantielles et fructueuses avec d'autres disciplines scientifiques et techniques. Le RL fait partie d'une évolution de plusieurs décennies dans le domaine du ML qui tend à une plus grande intégration avec les statistiques, l'optimisation et d'autres filiales mathématiques.

3.2.2 Structure agent-environnement

Avant de commencer à détailler la modélisation d'un système RL, nous présentons dans ce paragraphe la structure agent-environnement utilisée pour décrire de nombreux problèmes traités dans le cadre d'une configuration RL. Ses principaux composants, notamment l'agent, l'environnement, les espaces d'état et d'action sont décrits en soulignant leurs caractéristiques intrinsèques qui permettent l'adaptation de cette structure agent-environnement à plusieurs cas de figures.

3.2.2.1 Agent

Au niveau structurel, un agent est composé d'un programme implémentant ses fonctions élémentaires et d'une architecture gérant ses capteurs et actionneurs physiques. Quatre types de programmes d'agent peuvent être recensés dans la littérature et ont été catégorisés selon leur base de génération d'actions (Russell & Norvig, 2009) : réflexe simple, modèle, objectif ou utilité. Ces différents programmes sont présentés dans tableau 3.1 avec plus de détails, en allant du plus simple vers le plus développé.

Base de l'agent	Description
Réflexe simple	L'agent agit selon une règle dont la condition formulée explicitement correspond à l'état actuel, tel que reflété par l'observation perçue. Son fonctionnement est simple mais dispose d'une intelligence rigide et limitée.
Modèle	L'agent a la capacité de garder une trace de l'historique de sa perception de l'environnement au niveau d'un modèle interne qui reflète des aspects non observés de l'état actuel. Cette connaissance lui permet de prévoir comment le monde évolue et quelle est l'impact de ses actions sur ce monde.

Objectif	En plus d'une description de l'état actuel, l'agent exploite une sorte d'information sur l'objectif décrivant les situations souhaitables. Les actions choisies doivent mener à la réalisation de certains objectifs. Ce type de prise de décision implique une réflexion sur l'avenir et plus de flexibilité dans le comportement.
Utilité	Les objectifs seuls ne suffisent pas à générer un comportement optimal dans la plupart des environnements. Une mesure de performance plus générale devrait permettre de comparer différents états de l'environnement en fonction du degré de satisfaction de l'agent. Ce dernier utilise alors une fonction d'utilité qui consiste essentiellement en une internalisation de la mesure de performance. Son objectif s'étend désormais à entreprendre rationnellement des actions qui maximisent son utilité.

Table 3.1 – Les caractéristiques d'un agent de contrôle.

D'un point de vue comportement, l'une des définitions de l'agent les plus adaptées au cadre RL est celle d'agent rationnel (Russell & Norvig, 2009). C'est un programme qui fonctionne d'une manière autonome, perçoit son environnement, persiste pendant une période de temps prolongée, s'adapte au changement et est capable d'atteindre des objectifs. Sa rationalité se manifeste en agissant de manière à atteindre la meilleure utilité ou, en cas d'incertitude, la meilleure utilité espérée. Tel qu'illustré dans la figure 3.1, un agent RL effectue une transposition de son état interne et de ses perceptions en des actions sur l'environnement. L'état interne de l'agent (un modèle d'apprentissage par exemple) change au fil du temps sous l'influence des perceptions.

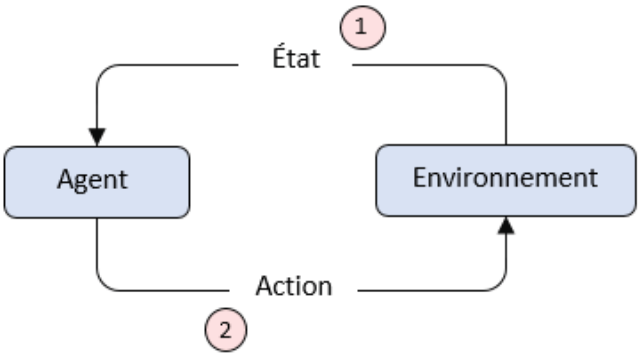

Figure 3.1 – Interaction agent / environnement.

3.2.2.2 Environnement

L'environnement est la représentation du système qui se situe en dehors du composant agent. A chaque pas de temps, l'agent occupe un état de l'environnement qui lui retourne le signal de récompense correspondant. La conception de l'environnement dépend de

la modélisation effectuée pour structurer le problème. Ainsi l'environnement peut être catégorisé selon l'une des dimensions décrites dans le tableau 3.2 déterminant à un large degré la conception et l'implémentation de l'agent (Russell & Norvig, 2009).

Dimension	Description
Complètement observable (vs partiellement observable)	Ce critère correspond au degré d'accès de l'agent à l'état de l'environnement et les informations qui lui sont associées. Un environnement est complètement observable si les capteurs de l'agent détectent tous les aspects pertinents pour le choix de l'action.
Déterministe (vs stochastique)	Si l'état futur de l'environnement est complètement déterminé par son état actuel et par l'action entreprise par l'agent, alors il s'agit d'un environnement déterministe.
Épisodique (vs séquentiel)	Dans un environnement épisodique, la décision courante n'est pas affectée par la décision précédente impliquant une indépendance des actions de l'agent au cours du temps.
Statique (vs dynamique)	L'état dans un environnement statique ne change pas au cours de la prise de décision par l'agent.
Discret (vs continu)	Dans un environnement discret, le périmètre des actions de l'agent, des états de l'environnement et des observations constitue un ensemble fini.
Connu (vs inconnu)	Les règles « physiques » gérant l'interaction avec l'environnement sont maîtrisées. L'agent n'aura pas à apprendre son fonctionnement technique.

Table 3.2 – Les types d'environnements.

3.2.2.3 État

En général, l'état reflète une relation entre l'agent et l'environnement, une situation propre à l'environnement ou un état interne à l'agent. Étant l'élément principal utilisé pour évaluer la situation du système à un instant donné, l'état est traduit en une observation percevable par l'agent. Dans une configuration RL, un état empirique est l'observation de l'environnement qui comprend toutes les informations dont l'agent dispose pour choisir une action. Toutefois, l'agent pourrait ne pas être capable d'apercevoir tous les états possibles dans l'environnement, donc il est important de faire une distinction entre les états et les observations. En effet, les états que l'agent peut réellement expérimenter existent dans l'espace d'observation, qui est parfois limité à seulement un sous-ensemble de l'espace d'état réel de l'environnement.

3.2.2.4 Action

Les actions effectuées par l'agent ont un impact direct sur son évolution au sein de l'environnement. L'objectif de l'agent est de savoir quelle action sélectionner parmi un périmètre autorisé en fonction de l'état, afin de maximiser la récompense totale reçue. D'un

autre côté, les actions peuvent être discrètes ou continues selon la nature du problème traité. Ce choix dépend également d'un compromis entre la complexité du problème et la faisabilité d'apprentissage (Lillicrap et al., 2016). Le type discret est le plus élémentaire et contient un nombre fini et dénombrable d'actions. Cependant, dans les scénarios réels, un problème peut reposer sur un espace d'action continue plus complexe. Bien que plusieurs de ces cas puissent être résolues en discrétisant les variables d'action (Santamaría et al., 1997), une solution différente s'impose pour les problèmes nécessitant un contrôle de haute précision. D'un côté, des petits écarts par rapport à l'optimum conduisent à des valeurs d'utilité très faibles, de l'autre, l'utilisation de discrétisations très fines engendre un coût de calcul énorme pour le processus d'apprentissage. Ainsi, certaines méthodes traitent ce problème en déduisant des valeurs par interpolation des actions discrètes disponibles sur la base de leur fonction d'utilité (Hasselt & Wiering, 2007).

3.2.2.5 Modèle de l'environnement

Certains systèmes d'apprentissage RL utilisent un modèle externe pour simuler le comportement de l'environnement. Le rôle de cette composante optionnelle est de résoudre des problèmes de planification pour prédire la prochaine action. Il s'agit de mettre en place un plan d'actions en considérant les situations futures possibles avant qu'elles ne soient réellement vécues. Les méthodes RL qui utilisent les modèles et la planification sont appelés model-based par opposition aux méthodes sans modèle ou model-free qui utilisent des apprenants par essai-erreur. Ces dernières apprennent directement à partir de l'expérience générant des politiques comportementales qui indiquent à l'agent ce qu'il doit faire dans chaque situation.

L'algorithme model-based utilise quant à lui un nombre réduit d'interactions avec l'environnement réel pendant la phase d'apprentissage afin de mettre à jour son modèle. Celui-ci est utilisé par la suite pour simuler les épisodes suivants et retourner les résultats de la simulation vers l'agent. Ce fonctionnement a l'avantage d'accélérer substantiellement l'apprentissage, puisqu'il n'est pas nécessaire d'attendre la réponse de l'environnement. En revanche, si le modèle est imprécis l'agent risque d'apprendre des stratégies incompatibles avec la réalité. D'un autre côté, un modèle externe implique plus d'hypothèses et d'approximations limitant l'application du RL à des types de tâches spécifiques avec un moindre degré de complexité.

3.2.3 Propriété de Markov

Le fonctionnement d'un agent RL peut formellement être décrit à travers un processus de décision de Markov (MDP). Considérant une interaction de l'agent avec l'environnement par pas de temps discret $t \in \mathbb{N}$, nous définissons alors le n-uplet (S, A, r, Tr) :

- S est l'ensemble d'états, où $s_t \in S$ désigne l'état de l'agent à l'instant t.
- A est l'ensemble d'actions où $a_t \in A$ désigne l'action entreprise par l'agent à l'instant t.
- $r : S \times A \to \mathbb{R}$ est la fonction de renforcement calculant la récompense immédiate $r_t(s_t, a_t) \in \mathcal{R}$ suite à une action a_t entreprise par l'agent à partir d'un état s_t.

- $Tr : S \times \mathcal{R} \times S \times A \rightarrow [0,1]$ est la fonction de transition calculant la probabilité $\mathcal{P}(s_{t+1}, r_{t+1}|s_t, a_t)$ de transition vers un état s_{t+1} générant une récompense r_{t+1} suite à une action a_t entreprise par l'agent à partir d'un état s_t.

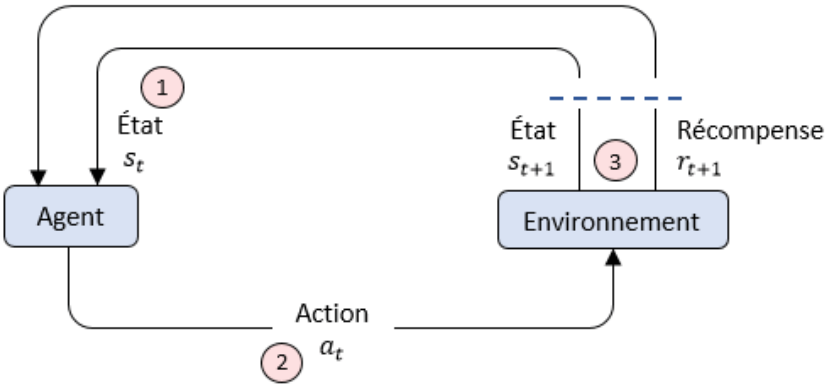

FIGURE 3.2 – Dynamiques de l'environnement dans le cadre d'un MDP.

Supposons que l'agent se situe à l'état de l'environnement s_t et choisit une action a_t (voir figure 3.2). La propriété de Markov est respectée au niveau de ce processus stochastique si et seulement si la prédiction du prochain état s_{t+1} ne dépend que de l'état courant et de l'action exécutée. Ainsi l'agent dispose de toute l'information nécessaire à travers l'état s_t pour prendre une décision maximisant sa fonction de récompense. Mathématiquement, cette propriété se traduit par :

$$\mathcal{P}(s_{t+1}, r_{t+1}|s_t, a_t) = \mathcal{P}(s_{t+1}, r_{t+1}|s_t, a_t, s_{t-1}, a_{t-1} \ldots, s_0, a_0) \tag{3.1}$$

3.2.4 Politique

Appelée aussi stratégie, la politique π_t est le composant de l'agent qui définit son comportement. Elle constitue la séquence des règles de décision utilisées par l'agent à chaque instant t de sélection d'une action.

$$\pi = (\pi_t)_{t \geq 0} \tag{3.2}$$

Cette transposition des états vers les actions $\pi : S \rightarrow A$ correspond en psychologie à l'ensemble de règles de réflexes conditionnés. Dans certains cas, la politique peut être une simple table de correspondance, mais elle peut également impliquer un processus complexe de recherche. Une politique peut être déterministe ou stochastique. Une stratégie déterministe associe des actions aux états tandis qu'une stratégie stochastique (avec incertitude) produit la distribution de probabilité sur l'ensemble des éléments d'actions A. Cette différence est formalisée mathématiquement par :

- Politique déterministe : $\pi(s_t) = a_t$
- Politique stochastique : $\pi(s_t) = [\mathcal{P}(a_1|s_t), \mathcal{P}(a_2|s_t), \ldots]$

Dans une politique, les règles de décision sont généralement différentes à chaque instant. Dans le cas où la règle de décision est invariable par rapport au temps, la politique est dite stationnaire où $\forall t \geq 0, \pi_t = \pi_0$.

3.2.5 Fonction de renforcement

Définissant l'objectif du RL, le principe de la fonction de renforcement r est de faire correspondre à chaque état-action une récompense sous forme scalaire $r : S \times A \rightarrow \mathbb{R}$ indiquant leur intérêt. A un instant donné, $r_t \in \mathbb{R}$ représente la performance momentanée de l'agent dans l'environnement. L'agent va ainsi chercher à maximiser le total des récompenses espérées (appelé encore retour) sur un horizon de temps H.

$$R_t = \sum_{k=0}^{H} r_{t+k+1} \tag{3.3}$$

Un horizon de temps fini est utilisé dans le cas où l'interaction agent-environnement se décompose naturellement en sous-séquences appelées épisodes. Dans le cas de tâches continues, l'horizon $H = \infty$, mais généralement une condition d'arrêt est prévue pour optimiser le comportement de l'agent. Afin de garantir la convergence de la fonction de renforcement, un facteur d'actualisation γ est généralement utilisé, où $0 < \gamma < 1$. Le retour actualisé s'écrit alors :

$$R_t = \sum_{k=0}^{H} \gamma^k \, r_{t+k+1} \tag{3.4}$$

Cette nouvelle forme du retour de l'environnement implique que les récompenses situées trop loin dans le futur sont évaluées à zéro étant donné que $\gamma < 1$. Plus γ est proche de zéro, plus l'effet de l'actualisation réduira le nombre de pas de temps disponibles à l'agent dans son appréciation de l'environnement à un instant donné.

3.2.6 Fonction de valeur

Dans la phase d'entraînement, l'agent construit des relations de cause à effet entre les états perçus et les actions les plus avantageuses. Il interagit avec l'environnement afin d'acquérir progressivement de l'expérience permettant de trouver des stratégies qui maximisent la récompense reçue dans une perspective à long terme. La fonction de valeur, également connue sous le nom de fonction d'utilité (Russell & Norvig, 2009) (notion à l'origine propre au jargon économique), vient répondre à ce besoin en effectuant une estimation de l'avantage espéré lorsque l'agent se trouve dans un état donné. Tandis que les fonctions de renforcement constituent un indicateur immédiat sur un état de l'environnement, les fonctions de valeurs quant à elles, offrent une vision à long terme en prenant en compte tout l'enchaînement d'états dont il est l'origine. Par exemple, un état peut générer une faible récompense immédiate tout en conservant une valeur élevée, car il est régulièrement suivi d'autres états qui génèrent des récompenses élevées. Formellement, soit $V^\pi : S \rightarrow \mathbb{R}$ la fonction de valeur associée à une politique π. Elle est définie comme l'espérance du retour d'un état s :

$$V^\pi(s) = \mathbb{E}_\pi \left[R_t | s_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{H} \gamma^k \, r_{t+k+1} | s_t = s \right] \tag{3.5}$$

Une extension de la fonction précédente est la fonction action-valeur $Q^\pi : S \times A \rightarrow \mathbb{R}$. Associée à une politique π, elle est définie comme l'espérance du retour d'un état s si

l'agent exécute l'action a :

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[R_t | s_t = s, a_t = a\right] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{H} \gamma^k \, r_{t+k+1} | s_t = s, a_t = a\right] \tag{3.6}$$

La relation entre les deux fonctions de valeur est :

$$V^{\pi}(s) = Q^{\pi}(s, \pi(s)) \tag{3.7}$$

Les fonctions de valeurs doivent être estimées à partir de séquences d'observations effectuées par un agent au cours de son cycle de vie, ce qui est beaucoup plus complexe que le calcul de récompenses attribuées directement par l'environnement. Par conséquent, l'une des problématiques les plus cruciales de l'application de RL est la détermination d'une méthode permettant d'estimer efficacement la fonction de valeur (Sutton & Barto, 2018).

3.2.7 Recherche d'optimalité

La recherche d'une politique optimale se base sur 3 méthodes fondamentales qui diffèrent par la manière dont la fonction de valeur est estimée :

- Programmation dynamique :
  Cette méthode suppose une connaissance complète de la dynamique de l'environnement $\mathcal{P}(s_{t+1}, r_{t+1} | s_t, a_t)$. Elle se base sur l'équation de Bellman (Bellman & Dreyfus, 1962) qui stipule que l'équation de la fonction de valeur (3.5) (raisonnement applicable également à l'équation (3.6)) satisfait la relation récursive suivante :

$$\begin{aligned}
V^{\pi}(s) &= \mathbb{E}_{\pi}\left[\sum_{k=0}^{H} \gamma^k \, r_{t+k+1} | s_t = s\right] \\
&= \mathbb{E}_{\pi}\left[r_{t+1} + \sum_{k=1}^{H} \gamma^k \, r_{t+k+1} | s_t = s\right] \\
&= \mathbb{E}_{\pi}\left[r_{t+1} + \sum_{k=0}^{H} \gamma^{k+1} \, r_{t+k+2} | s_t = s\right] \\
&= \mathbb{E}_{\pi}\left[r_{t+1} + \gamma \sum_{k=0}^{H} \gamma^k \, r_{t+k+2} | s_t = s\right] \\
&= \mathbb{E}_{\pi}\left[r_{t+1} + \gamma \, V^{\pi}(s_{t+1}) | s_t = s\right]
\end{aligned} \tag{3.8}$$

  En exprimant la fonction de valeur en fonction d'une récompense empirique et de son successeur, il est par la suite possible d'effectuer une itération de politique alternant entre une étape d'évaluation à travers l'équation de Bellman et une étape d'amélioration de politique gloutonne en dérivant $Q^{\pi}(s,a)$ des valeurs courantes de $V^{\pi}(s)$ et de $\pi(s)$:

$$\begin{aligned}
\pi(s) &= arg\ max_a Q^{\pi}(s,a) \\
&= arg\ max_a \sum_{s',r} \mathcal{P}(s', r | s, a)[r + \gamma V^{\pi}(s')]
\end{aligned} \tag{3.9}$$

Nous obtenons ainsi une séquence de politiques et de fonctions de valeur s'améliorant d'une manière continue et monotone :

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} V^{\pi_2} ... \xrightarrow{I} \pi^* \xrightarrow{E} V^{\pi^*} \tag{3.10}$$

où $\xrightarrow{E}$ correspond à une itération d'évaluation et $\xrightarrow{I}$ celle d'amélioration (Improvement). Dans le cas d'un MDP fini ce processus doit converger vers une politique optimale π^* et une fonction de valeur optimale V^{π^*}.

- Méthodes Monte Carlo :

  Pour pallier aux limites de la programmation dynamique, des méthodes de recherche d'optimalité applicable même si aucune connaissance préalable sur l'environnement n'est disponible ont été développées. Ainsi, si les probabilités de transition ne sont pas connues, nous pouvons remplacer le modèle en échantillonnant des transitions directement à partir de l'environnement afin d'obtenir des trajectoires complètes et d'acquérir des connaissances sur sa dynamique. Cette idée constitue la base des méthodes Monte Carlo, les premières approches model-free proposées dans la littérature. Les méthodes Monte Carlo ne sont pas directement utiles pour les approches présentées dans cette thèse, donc nous nous limitons à leur output.

  Dans les méthodes Monte Carlo nous pouvons faire la même distinction, comme pour la programmation dynamique, entre deux problèmes principaux : la prédiction (évaluation) et le contrôle (amélioration). Le contrôle reste le même que celui dans la programmation dynamique, c'est à dire à travers une politique gloutonne. La nouveauté réside au niveau de la prédiction où la fonction de valeur est estimée entièrement à partir du retour empirique R. Dans sa version moyenne incrémentale, la méthode Monte Carlo consiste à calculer les retours obtenus après la première visite d'un état jusqu'à la fin de la trajectoire. Ce processus de collecte de données est répété sur k épisodes. La fonction de valeur est estimée, au niveau de chaque épisode, comme suit :

$$V(s_t) \leftarrow V(s_t) + \frac{1}{N(s_t)}(R_t - V(s_t)) \tag{3.11}$$

  où $N(s)$ est un compteur représentant le nombre de trajectoires où l'état s est présent. Les méthodes Monte Carlo, qui sont des estimateurs non biaisés vu qu'elles utilisent les retours empiriques uniquement, impliquent que le calcul de la formule (3.11) n'est effectué qu'en fin de l'épisode.

- Méthodes de différence temporelle :

  L'apprentissage TD est une méthode prépondérante dans les approches RL en étant une combinaison des méthodes Monte Carlo et de programmation dynamique. En effet, il met à jour l'estimation des fonctions de valeur en joignant, grâce au bootstrapping, l'expérience empirique acquise et les estimations biaisées de la fonction de valeur. A la différence du cas Monte Carlo où l'évaluation est effectuée à la fin de chaque épisode, les mises à jour TD sont réalisées après chaque transition ou ensemble de transitions. Nous pouvons également souligner dans l'apprentissage TD les deux processus classiques de prédiction, dans laquelle intervient principalement

la fonction de valeur, et de contrôle mené essentiellement par l'erreur TD. Cette méthode de recherche de politique est expliquée en détail dans la section 3.5.1 vu qu'elle constitue l'un des éléments primordiaux de notre approche RL robuste pour la conduite autonome.

3.3 Apprentissage par renforcement profond

Au cours des dernières années, le RL est devenu de plus en plus populaire en raison de son aptitude à résoudre des problèmes décisionnels séquentiels compliqués grâce à l'intégration de l'apprentissage profond. La combinaison d'un cadre de modélisation pouvant englober tout l'AI (RL) (Russell & Norvig, 2009) et d'un approximateur universel (DNN) (Hornik et al., 1989) ont fait du RL profond (DRL) un outil puissant et unique capable de traiter de nombreux domaines complexes avec des performances élevées. Nous présentons dans cette section les avantages d'utilisation du DRL ainsi que les défis de son application à des problèmes du monde réel.

3.3.1 Contexte d'apparition

Le cadre théorique pour la prise de décision séquentielle basée sur l'expérience existe depuis plusieurs décennies. En effet, parmi les approches de l'AI, les méthodes RL sont les mieux qualifiées pour résoudre des problèmes de contrôle lorsque des modèles précis ne sont pas disponibles. Bien que le RL ait eu quelques succès par le passé, les approches précédentes manquaient d'extensibilité et se limitaient par nature à des problèmes relativement peu dimensionnels, ce qui réduisait le nombre de cas d'utilisation pouvant être résolus par RL. Ces limitations existent parce que les algorithmes RL rencontrent des problèmes de représentation et de dimensionnement (Stone et al., 2016) ainsi que de difficultés de conception d'extracteurs de caractéristiques (Munos & Moore, 2002). Lorsque Bellman (1957) a exploré le contrôle optimal dans des espaces discrets de haute dimension, il a noté une explosion exponentielle d'états et d'actions pour lesquels il a inventé le terme « malédiction de la dimensionnalité ».

Pour atténuer l'impact de ce problème, les chercheurs en RL utilisent traditionnellement une série de stratégies allant de la discrétisation adaptative à l'approximation de fonctions (Sutton & Barto, 2018). En élargissant ces approches, les développements récents en apprentissage profond ont ouvert la voie à de nouvelles solutions d'approximation de fonctions. La combinaison de RL avec des techniques d'apprentissage profond, appelée RL profond (DRL), est particulièrement utile dans les tâches avec des espaces d'état larges. Il a contribué à plusieurs réalisations dont principalement l'apprentissage avec succès à partir d'entrées de perception visuelle composées de milliers de pixels (Mnih et al., 2015). Le DRL consiste en général à entraîner des DNNs (figure 3.3) et plus spécifiquement les CNNs pour approximer une composante de RL : la politique optimale π^*, les fonctions de valeur V^* et Q^* et/ou la fonction de renforcement R^* (Tan et al., 2017).

Le DRL a pu ainsi profiter des avantages de l'apprentissage profond (étudiés dans le chapitre précédent) et s'adapter à des entrées sensorielles de grande dimension et à une logique de contrôle complexe, sans nécessiter l'intervention manuelle d'ingénierie ni se limiter à des modèles simples et insuffisamment expressifs (Bengio et al., 2013). L'utilisation de la descente de gradient stochastique pour la mise à jour des pondérations de DRL

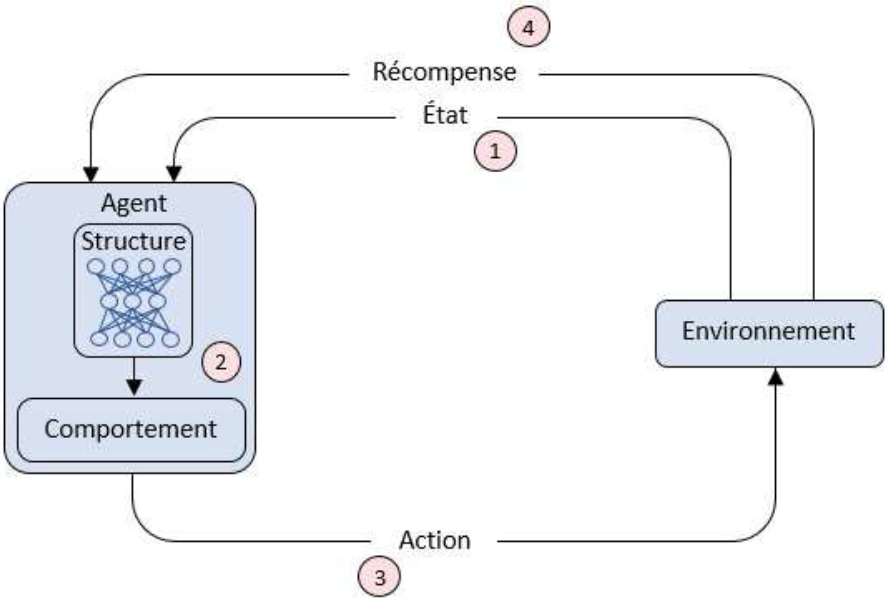

FIGURE 3.3 – L'apprentissage par renforcement profond consiste à utiliser une structure neuronale profonde pour approximer une des composantes impactant le comportement d'un agent : politique, fonctions de valeur ou de récompense.

ne modifie que légèrement les paramètres de la politique, contrairement aux méthodes tabulaires et non paramétriques traditionnelles dans lesquelles de grands sauts entre deux politiques estimées sont possibles. Cette propriété améliore la stabilité de l'entraînement et la convergence vers une politique optimale. D'un autre côté, une meilleure représentation des états de l'environnement est un facteur primordial pour le succès des approches RL. Dans ce cadre, les techniques d'apprentissage profond peuvent trouver automatiquement des représentations compactes pour des données de haute dimension permettant à l'agent RL de défier la « malédiction de dimensionnalité » et d'avoir une bonne perception et une exploration efficace de l'environnement (Böhmer et al., 2015).

Le début des succès du DRL remonte aux années **90**, avec les travaux de Tesauro (1995) utilisant une stratégie apprise par réseau de neurones qui a permis d'atteindre des résultats satisfaisants dans le jeu de Backgammon. Récemment, plusieurs travaux faisant appel au DRL se sont démarqués pour avoir surpassé le niveau d'experts humains dans les jeux tels que Atari (Mnih et al., 2015), Go (Silver et al., 2016) et le jeu de Poker (Brown et al., 2017). Les jeux vidéo peuvent constituer un défi intéressant, mais apprendre à les jouer n'est pas l'objectif final du DRL. Ce dernier offre également un potentiel pour des applications réelles telles que la robotique (Levine et al., 2016; Lillicrap et al., 2016), la finance (Deng et al., 2016) et les réseaux intelligents (François-Lavet et al., 2016). Il est devenu un cadre de modélisation qui permet de déplacer l'attention du ML de la reconnaissance de patterns vers la prise de décision séquentielle fondée sur l'expérience. Bien que largement confiné au monde de la recherche universitaire et industrielle au cours des dernières décennies, le RL connaît actuellement des succès pratiques dans le monde réel et promet de faire élargir l'échelle d'application de ML au-delà de la recherche (Stone et al., 2016).

Étant donnée la prépondérance de DRL dans les travaux de recherche actuels sur le RL, la distinction entre les **2** terminologies ne se pose généralement plus. En effet, le passage à

la haute dimensionnalité et la complexité des tâches du monde réel exige l'utilisation de l'apprentissage profond, à présent seul outil d'approximation de fonction permettant cette transition. Dans le reste de ce manuscrit, nous nous intéressons au DRL et utilisons les deux termes d'une manière équivalente, selon le contexte.

3.3.2 Défis d'application

Malgré les progrès significatifs récents du RL, les travaux réalisés pour les jeux vidéos et la robotique présentent certaines limites. En effet, les tâches traitées sont généralement caractérisées par l'utilisation d'environnements simulés, des règles déterministes, des espaces d'actions simples et des manipulations répétitives. Cependant, les tâches de contrôle du monde réel telles que la prédiction financière, les réseaux intelligents et la conduite autonome à laquelle nous nous intéressons pour l'évaluation de nos approches proposées dans le cadre de cette thèse, sont rarement déterministes en raison de la nature non-stationnaire de l'environnement et impliquent souvent des actions continues et évoluées.

D'un autre côté, deux aspects de l'intelligence humaine qui conditionnent l'apprentissage ne sont pas reflétés dans les méthodes actuelles de RL. Tout d'abord, un agent doté d'une politique sans orientation appropriée nécessite énormément d'exploration et de données d'entraînement avant de trouver une stratégie de contrôle stable, alors que l'être humain peut atteindre des performances raisonnables avec relativement peu d'expérience. Ce facteur reflète ce qu'on appelle l'inefficacité de données qui peut rendre intraitable des tâches de haute dimension. Deuxièmement, les systèmes RL se spécialisent généralement dans un domaine de tâches étroit, tandis que les apprenants humains peuvent s'adapter avec souplesse à la variabilité de différents problèmes.

Des critiques récentes ont souligné ces différences qui posent des défis directs aux recherches actuelles et l'extension du périmètre d'exploitation du RL en général (Lake et al., 2017). Même si l'apprentissage profond fournit un paradigme de bout en bout pour le développement de stratégies de contrôle optimales à partir des données, des avancées concrètes et notables doivent être accomplies pour parvenir à intégrer efficacement le RL à des applications du monde réel. Les méthodes développées doivent être capables d'explorer l'environnement efficacement et de généraliser un bon comportement dans un contexte variable et non-stationnaire. Nous exposons par la suite les principaux défis à surmonter pour atteindre cet objectif.

3.3.2.1 Observabilité partielle et non-stationnarité

Généralement, tous les systèmes réels pourraient être considérés partiellement observables. Par exemple, sur un système mécanique, il n'y a probablement pas d'observations sur l'usure des moteurs ou des articulations. Sur les systèmes de recommandation qui interagissent avec des utilisateurs, il n'y a aucune observation de l'état mental de ces derniers. Souvent, ces observabilités partielles se traduisent par un environnement non-stationnaire ou stochastique et peuvent être formulées sous la forme d'un processus de décision de Markov partiellement observable (POMDP). La principale différence par rapport à la formulation du MDP est que l'observation de l'agent est désormais séparée de l'état, avec une fonction d'observation $O(x|s)$ donnant la probabilité d'observer x étant donné l'état de l'environnement s.

Il existe dans la littérature des approches pour traiter l'observabilité partielle dépendant du domaine d'application. Une méthodologie consiste à générer un historique pour l'agent en empilant plusieurs entrées dans une même observation (Mnih et al., 2015) ou en utilisant des réseaux récurrents afin de récupérer l'état masqué (Hausknecht & Stone, 2015). Dans la perspective de concevoir des systèmes RL « robustes », d'autres approches ont été développées pour produire un agent qui résiste à la variabilité et la non-stationnarité des environnements réels (Mankowitz et al., 2018). Il s'agit en général d'entraîner un agent à diverses perturbations de l'environnement et de calculer la moyenne de ses erreurs d'apprentissage (Peng et al., 2018). Nagabandi et al. (2019) proposent une modélisation d'un environnement non-stationnaire avec une fonction de récompense variant dans le temps afin de trouver des politiques qui s'adaptent à cette non-stationnarité. Dans tous ces cas, les approches doivent délimiter quelles dimensions des perturbations sont pertinentes et examiner les effets des variations de ces dimensions sur les performances des politiques RL. Dans la section 3.5, nous nous proposons de construire une approche DRL robuste afin d'évaluer à quel degré elle serait capable de répondre aux exigences des applications du monde réel.

3.3.2.2 Espaces d'états et d'actions de grande dimension

Même si les algorithmes DRL peuvent traiter des entrées de grande dimension, il est compliqué d'entraîner des agents RL directement sur des entrées visuelles dans le monde réel en raison du grand nombre d'échantillons requis. Ainsi, de nombreux problèmes pratiques ayant des espaces d'état et d'action vastes posent des problèmes de dimensionnalité et d'insolubilité aux algorithmes RL. Nous pouvons mettre en exergue cette difficulté à travers les données sur l'apprentissage d'un jeu Atari dans (Mnih et al., 2015), Forstbite, qui est un exercice de dimensions beaucoup plus réduites que celles des tâches réelles, en termes d'espaces d'états et d'actions. Bien qu'il soit intéressant que le DRL proposé (DQN) apprenne à jouer à un niveau de performance humain en partant de très peu de connaissances préalables, il est fort probable qu'il soit en train d'apprendre d'une manière très différente. Un moyen d'examiner les différences est de considérer la quantité d'expérience requise pour son apprentissage. En effet, le DRL a été comparé à un joueur professionnel ayant reçu environ deux heures d'entraînement. L'algorithme a nécessité 200 millions d'images d'entraînement, ce qui équivaut à environ 924 heures de temps de jeu (environ 38 jours), soit près de 500 fois plus que l'expérience reçue par l'agent humain, pour atteindre une performance similaire.

Un certain nombre de travaux récents a été assigné à l'étude de ce problème. Zahavy et al. (2018) proposent un DQN pour l'élimination d'actions non pertinentes, He et al. (2016a) présentent un DRL de pertinence pour l'évaluation des espaces d'actions continus, Chandak et al. (2019) ont développé une méthode pour apprendre à intégrer des actions en fonction de leurs effets sur les transitions d'états. Une autre piste pour accélérer l'apprentissage de DRL est l'exploitation des connaissances déjà acquises à partir de tâches connexes, qui peut se présenter sous plusieurs formes : apprentissage par transfert, apprentissage multitâche (Caruana, 1997) et apprentissage par curriculum (Bengio et al., 2009). Il y a beaucoup d'avantages à transférer l'apprentissage d'une tâche à une autre, par exemple d'un entraînement sur des simulateurs vers des modèles dans le monde réel, accompagné généralement par un réglage fin. Cela peut être réalisé de manière naïve,

en utilisant directement le même réseau dans les phases simulée et réelle (Zhu et al., 2017), ou avec des procédures d'entraînement plus développées rajoutant des ressources supplémentaires lors du transfert du domaine (Rusu et al., 2017).

L'évaluation d'une politique face à de grands espaces d'actions et/ou d'états doit prendre en compte à la fois le périmètre à traiter et la qualité des métriques utilisées pour représenter ces espaces. Dans ce contexte, le défi de construire des modèles d'apprentissage s'inspirant de l'intelligence humaine revient à répondre aux deux questions suivantes : comment utiliser des connaissances antérieures riches pour apprendre de nouvelles tâches et résoudre de nouveaux problèmes aussi rapidement ? Quelle forme prennent ces connaissances antérieures et comment sont-elles construites ? Les éléments de réponse que nous proposons dans le chapitre 4 sur le méta-apprentissage offrent une voie pour relever ce défi.

3.3.2.3 Fonction de récompense multi-objectif

Le RL encadre l'apprentissage des politiques en optimisant une fonction de récompense globale. Mais de nombreux problèmes de la vie réelle impliquent des objectifs multiples qui peuvent être corrélés, contradictoires ou s'influencer mutuellement (Tesauro et al., 2008). Ainsi, une partie du travail de déploiement de RL sur des systèmes réels consiste à formuler la fonction de récompense en visant un équilibre entre plusieurs sous-objectifs. Par exemple, dans le cas de conduite autonome, il s'agirait de concilier entre accélération, minimisation du nombre de collisions et réduction du temps nécessaire pour atteindre un point d'arrivée. Il existe deux approches principales pour traiter des problèmes à objectifs multiples. Le moyen le plus courant consiste à utiliser une fonction linéaire qui transforme le problème à objectifs multiples en un problème à objectif unique standard. Une autre stratégie intéressante mais très coûteuse, qui pourrait constituer une perspective pour la consolidation de RL, est la séparation explicite des composants individuels de la fonction de récompense afin de mieux comprendre les compromis de la politique. Les méthodes qui découlent de cette stratégie se basent sur l'optimum de pareto (Roijers et al., 2013) et ne sont pas encore fréquemment appliquées pour des configurations RL.

3.3.3 Approches d'application

Il existe trois approches principales pour résoudre les problèmes RL, selon qu'elles reposent sur une fonction de valeur, une politique paramétrée ou une combinaison des deux pour prédire les actions (Konda & Tsitsiklis, 2003). Nous présentons dans ce qui suit ces approches et d'autres concepts utiles afin de mieux cerner le modèle de RL robuste proposé et expérimenté dans la dernière section de ce chapitre.

3.3.3.1 Méthodes value-based

Les méthodes value-based (critic-only) construisent une politique optimale en apprenant d'abord une fonction de valeur qui estime le rendement espéré. Ensuite, la politique optimale est déduite en choisissant une action qui maximise la fonction de valeur apprise. L'algorithme Q-learning (Watkins & Dayan, 1992) est l'un des algorithmes les plus simples et les plus populaires dans cette catégorie. Il utilise l'équation de Bellman (Bellman & Dreyfus, 1962) pour la fonction action-valeur Q^{π} (équation 3.6) afin de déduire la relation suivante :

$$Q^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) + \alpha[r_{t+1} + \gamma \, max_a \, Q^{\pi}(s_{t+1}, a) - Q^{\pi}(s_t, a_t)] \tag{3.12}$$

Où α est un taux d'apprentissage et γ un facteur d'actualisation. L'optimisation de Q^{π} se fait à travers la minimisation des erreurs TD (Sutton & Barto, 2018) des trajectoires rencontrées en suivant la politique (évaluation) :

$$\delta_t = r_{t+1} + \gamma \, Q^{\pi}(s_{t+1}, a) - Q^{\pi}(s_t, a_t) \tag{3.13}$$

La meilleure politique (amélioration) est choisie d'une manière gloutonne (greedy) en retenant l'action maximisant la fonction action-valeur à chaque état :

$$\pi(s_t) = arg \, max_a \, Q^{\pi}(s_t, a_t) \tag{3.14}$$

L'itération de politique évaluation-amélioration est répétée jusqu'à atteindre la convergence du système. Q-learning utilise une représentation tabulaire de la fonction $Q^{\pi}(s_t, a_t)$ de taille $|S| \times |A|$ rendant cette configuration simple inapplicable à des espaces action-état de grande dimension. Les propriétés d'approximation des fonctions par les réseaux de neurones vues dans la section précédente ont naturellement conduit à l'utilisation de l'apprentissage profond pour la régression des fonctions destinées aux agents RL. Dans ce contexte, Deep Q Network (DQN) a été proposé par Mnih et al. (2015). Il présente le premier modèle d'apprentissage profond permettant de maîtriser les politiques de contrôle directement à partir de données sensorielles de grande dimension en utilisant le RL (voir figure 3.4). Plus précisément, DQN prend comme entrée les images affichées sur l'émulateur Atari, en utilisant un CNN de paramètre θ pour traiter les données de l'image. Un algorithme Q-learning est utilisé pour prendre la décision où la représentation tabulaire de la fonction $Q^{\pi}(s, a)$ est remplacée par l'approximation du CNN :

$$Q^{\pi}(s_t, a_t) \approx Q^{\pi}(s_t, a_t, \theta) \tag{3.15}$$

Le problème RL consiste désormais à entraîner le CNN en minimisant sa fonction de perte $\mathcal{L}(\theta)$ par descente de gradient et obtenir le modèle d'apprentissage θ^* (équation 2.4).

D'autres approches ont succédé au DQN réalisant certaines avancées au niveau de l'efficacité de l'estimation de la fonction de valeur. Le Double DQN proposé par Hasselt et al. (2016) sépare la sélection et l'évaluation de l'action en apprenant deux fonctions de valeurs estimées par deux modèles d'apprentissage θ et θ'. Bellemare et al. (2017) appliquent la perspective distributionnelle de l'équation de Bellman au DQN pour accélérer l'apprentissage du modèle. En dernier exemple, Rainbow (Hessel et al., 2018) est un modèle qui combine 6 extensions intéressantes du DQN permettant de tester leur complémentarité et la contribution de chaque composante : DDQN, Prioritized replay, Dueling networks, Multi-step learning, Distributional RL et Noisy Nets.

Les méthodes value-based ont l'avantage de générer une variance réduite des retours estimés (Boyan, 2002), cependant elles présentent quelques limites. Premièrement, elles sont orientées vers la recherche de politiques déterministes, tandis que la politique optimale est souvent stochastique, sélectionnant différentes actions avec des probabilités spécifiques

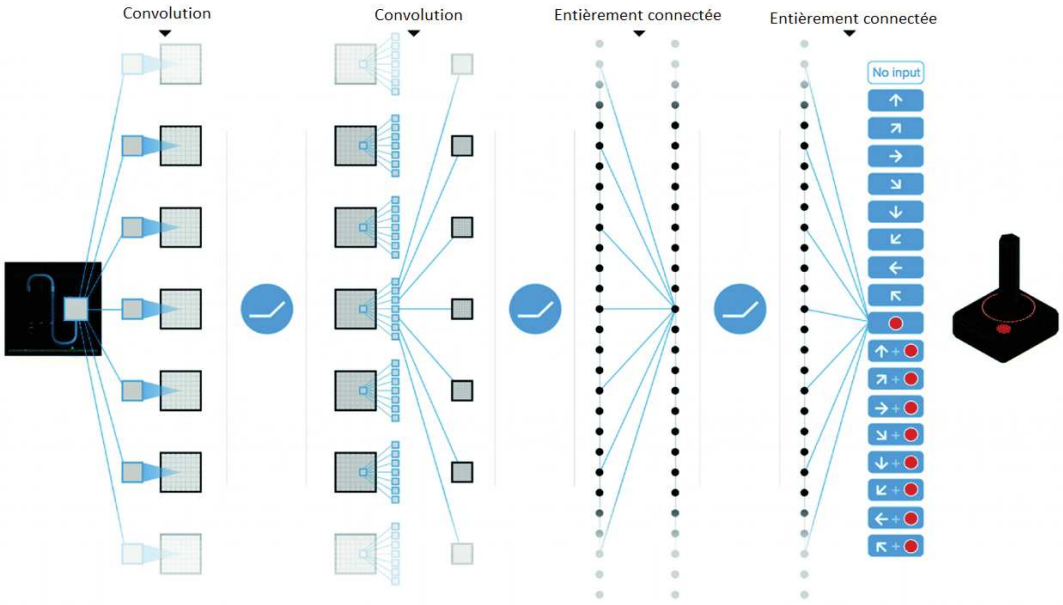

FIGURE 3.4 – Chaîne de traitement DRL de jeu Atari proposée dans (Mnih et al., 2015). L'approche permet de traduire des pixels d'images en actions de contrôle.

(Sutton et al., 2000). Deuxièmement, une modification arbitrairement faible de la valeur estimée d'une action peut entraîner sa sélection ou son exclusion. Ces modifications discontinues ont été identifiées comme un obstacle majeur à l'établissement de garanties de convergence pour ce type d'algorithmes (Grondman et al., 2012). Enfin, les méthodes value-based ont tendance à surestimer les valeurs Q^{π} car les erreurs positives sont propagées par l'opérateur max de l'équation 3.12, ce qui pourrait dégrader la précision du modèle d'apprentissage (Hasselt, 2010).

3.3.3.2 Méthodes policy-based

Les méthodes policy-based (actor-only), également appelées méthodes de recherche de politiques, apprennent une politique paramétrée maximisant une approximation du rendement espéré des échantillons de données sans apprendre la fonction de valeur. Lors de la construction directe de la politique, le système RL génère une distribution de probabilité résultant en une politique stochastique à partir de laquelle des actions sont prises directement. Pour ce type de méthodes, les gradients peuvent fournir un signal d'apprentissage fort quant à la manière d'améliorer une politique paramétrée. L'un des estimateurs de gradient les plus assignés aux tâches RL est la règle de REINFORCE (Williams, 1992), connue également sous le nom de fonction score (Fu, 2006) ou estimateur du rapport de vraisemblance (Glynn, 1990). Cette technique utilise le gradient pour mettre à jour les paramètres de la politique de sorte que la probabilité d'observer des retours d'échantillon élevés augmente. Similairement à l'optimisation de la fonction log-vraisemblance dans l'apprentissage supervisé, la montée progressive du gradient de la politique à l'aide de l'estimateur augmente la probabilité du log de l'action échantillonnée, pondérée par le retour.

Plus formellement, considérons la politique stochastique $\pi_\theta(a|s)$ paramétrée par θ et

la fonction objectif $J(\pi_\theta)$ qui permet de quantifier cette politique. Trouver une politique paramétrée optimale revient à trouver l'ensemble de paramètres θ^* satisfaisant la contrainte suivante :

$$\theta^* = arg\ max_\theta\ J(\pi_\theta) \tag{3.16}$$

L'idée de base derrière les algorithmes de gradient de politique est d'effectuer des ajustements de paramètres $\Delta\theta$ dans la direction de maximisation du gradient $\nabla_\theta J(\pi_\theta)$. Le théorème du gradient de politique (Sutton et al., 2000) stipule que pour une politique différentiable $\pi_\theta(a|s)$, le gradient de la politique est défini par :

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\pi_\theta}\Big[\nabla_\theta log\ \ \pi_\theta(a_t|s_t)\ \ Q^\pi(s_t, a_t)\Big] \tag{3.17}$$

Dérivée de ce théorème, la règle REINFORCE utilise un échantillon de retours pour estimer la fonction de valeur $Q^{\pi_\theta}(s_t, a_t)$ et déplace θ dans la direction :

$$\Delta\theta = \nabla_\theta\ log\, \pi_\theta(a_t|s_t) R_t \tag{3.18}$$

Plusieurs méthodes ont découlé de la règle REINFORCE proposant certains axes d'amélioration. Le gradient de politique naturel est une approche qui vise à améliorer la montée du gradient en restreignant la mise à jour de la politique grâce à la mesure de convergence de Kullback-Leibler (Kakade, 2002). En effet, empêcher les mises à jour de politique de trop s'écarter des stratégies précédentes conduit à une amélioration régulière des performances du RL. TRPO (Trust region policy optimisation) (Schulman et al., 2015) consiste à utiliser une région de confiance, où l'approximation de la fonction objectif est toujours efficace. L'algorithme récent d'optimisation de politique proximale (PPO) reprend la même idée du TRPO avec une implémentation plus simple, ne nécessitant que des informations de gradient de premier ordre (Schulman et al., 2017).

Comparées aux méthodes value-based, les méthodes de recherche de politique sont plus simples et offrent des garanties de convergence fortes vers une politique localement optimale dans des conditions de régularité standards (Sutton et al., 2000). Cependant, la variance des retours empiriques est généralement élevée impliquant une convergence très lente des méthodes policy-based (Berenji & Vengerov, 2003).

3.3.3.3 Actor-critic

Les méthodes actor-critic combinent les avantages des deux approches précédentes en consolidant l'étape d'itération de politique. Plus précisément, un algorithme actor-critic apprend à la fois une politique et une fonction de valeur afin de réduire la variance et d'accélérer l'apprentissage (Sutton & Barto, 2018). L'acteur (politique) apprend en utilisant le retour d'expérience du critique (fonction de valeur) sous forme d'erreur TD, comme montrée dans la figure 3.5. Cette interaction actor-critic implique un arbitrage entre la réduction de la variance des gradients et l'introduction d'un biais provenant des méthodes value-based (Konda & Tsitsiklis, 2003). Dans le contexte de DRL, l'acteur et le critique sont approximés par des réseaux de neurones (Mnih et al., 2016). En se référant aux équations précédentes (3.17) et (3.15), et en attribuant les paramètres θ et ω respectivement aux

approximations de la politique et de la fonction de valeur, le gradient de la politique d'un algorithme actor-critic est défini par :

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\pi_\theta}\Big[\nabla_\theta \, log\, \pi_\theta(a_t|s_t)\, Q^\pi(s_t, a_t, \omega)\Big] \tag{3.19}$$

Le travail de Barto et al. (1990) est considéré comme le point de départ définissant les bases des algorithmes actor-critic couramment utilisés dans les recherches de DRL récentes. Depuis lors, plusieurs algorithmes ont été développés avec des différentes directions d'amélioration. Wang et al. (2007) ont présenté un modèle fondé sur un réseau de neurones flou (fuzzy) permettant d'approximer à la fois, la fonction de valeur et la politique (FACRLN). Basés sur la même stratégie, Niedzwiedz et al. (2008) ont mis au point le modèle actor-critic consolidé (CACM). Jan et al. (2003) ont utilisé pour la première fois un gradient naturel (Amari & Douglas, 1998) pour les mises à jour de la politique dans leur algorithme actor-critic. Silver et al. (2014) ont présenté l'algorithme de gradient de politique déterministe (DPG) qui assigne une estimation de valeur apprise pour entraîner une politique déterministe. Récemment, Mnih et al. (2016) ont proposé l'algorithme A3C (Asynchronous Advantage Actor-Critic) dans lequel plusieurs agents fonctionnent en parallèle, ce qui permet la décorrélation des données et l'apprentissage de la diversité des expériences. La méthode actor-critic sera étudiée avec plus de détails dans la dernière section de ce chapitre où sera présentée notre première contribution de RL robuste.

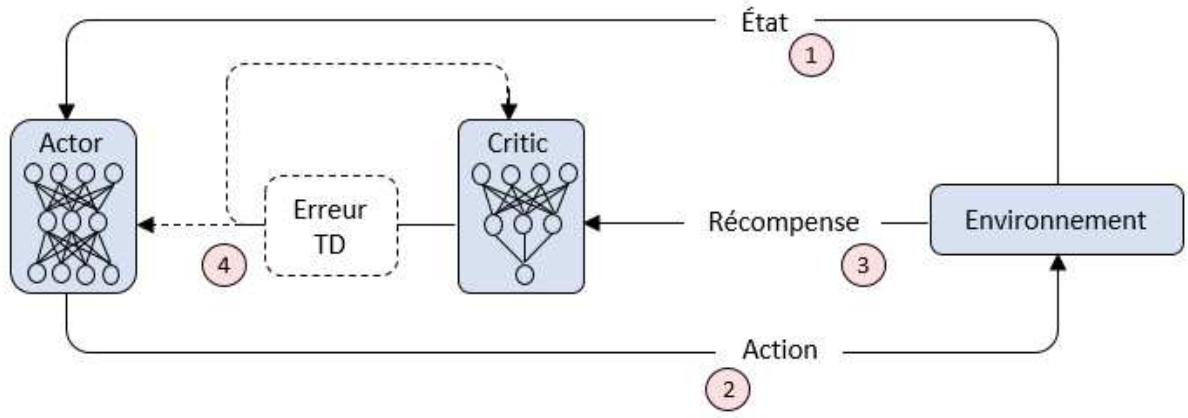

Figure 3.5 – Retour d'expérience par erreur TD dans un schéma RL de type actor-critic.

3.4 DRL pour la conduite autonome

Les véhicules autonomes sont des systèmes de prise de décision autonomes qui traitent des flux d'observations provenant de différentes sources embarquées, telles que caméras, radars, LiDAR, et GPS. Ces observations sont utilisées par l'ordinateur de bord pour prendre des décisions de conduite. Du fait de sa structure séquentielle, de l'incertitude et de la variabilité inhérentes à cette tâche, la conduite autonome constitue un domaine du monde réel naturellement compatible à l'application des approches RL.

D'un autre côté, l'utilisation de simulateurs offre un moyen précieux de reproduire une infinité de scénarios de conduite dans le but d'atteindre un haut degré d'optimisation et

de précision. Elle permet également d'éviter l'acquisition de prototypes coûteux pendant la phase de développement et d'accélérer la commercialisation des véhicules autonomes de nouvelle génération. Si le rôle des véhicules autonomes est de rendre les routes plus sûres et plaisantes, les simulateurs de conduite pourraient bien être la technologie habilitante qui les rendraient accessibles de manière abordable et dans les plus courts délais. Cette section se propose de détailler le cadre expérimental de nos approches, à savoir la conduite autonome comme domaine et le simulateur CARLA comme support de test.

3.4.1 Problématiques actuelles

Les véhicules autonomes se présentent comme la prochaine étape incontournable qui va révolutionner le développement du transport individuel et collectif à moyen terme. Les acteurs du secteur automobile investissent d'une manière massive dans les logiciels (basés sur l'intelligence artificielle), les équipements et les services qui vont encadrer l'entrée en exploitation de ce nouveau type de mobilité. Plus spécifiquement, les véhicules autonomes ont le potentiel de transformer le futur de la vie urbaine. En offrant la possibilité de disposer de moyens de transport sûrs, efficaces, accessibles et abordables, ils promettent non seulement un nouveau système de mobilité, mais également une nouvelle approche du mode de vie et du design urbains. Le défi DARPA 2007 (Buehler et al., 2009) a constitué un exploit historique en matière de robotique. En effet, 6 des 11 véhicules participant à l'étape finale ont navigué avec succès dans un environnement urbain pour atteindre la ligne d'arrivée. Le succès de cette compétition a conduit beaucoup d'intervenants à déclarer que la tâche de conduite totalement autonome était un « problème résolu », qui ne comptait que quelques détails à résoudre par les constructeurs automobiles avant de fournir un produit commercial final.

Aujourd'hui, plus de dix ans après DARPA 2007, la tâche de conduite reste encore une activité trop complexe, par rapport aux technologies disponibles actuellement, pour être formalisée en tant qu'un système robotique complètement autonome. La localisation, la cartographie, la perception de la scène, le contrôle du véhicule, l'optimisation de la trajectoire et bien d'autres décisions de planification de haut niveau associés au développement des véhicules autonomes sont encore, à différents degrés, des problématiques ouvertes et chargées de défis. Cette difficulté est accentuée par le fait que la conduite appartient à un groupe de tâches du monde réel pour lesquelles la marge d'erreur autorisée est extrêmement petite face à un nombre de cas particuliers très large. Tant que ces problèmes ne seront pas résolus, l'être humain restera une partie intégrante de la tâche de conduite, du moins pour surveiller les systèmes AI implémentés et intervenir en cas de défaillance (Favarò et al., 2017).

La norme SAE J3016 (SAE Committee, 2014) définit cinq niveaux d'automatisation de conduite. Comme indiqué dans le tableau 3.3, les niveaux débutent avec une assistance très limitée au conducteur et progressent vers une automatisation complète dans laquelle l'intervention humaine n'est plus nécessaire. De nos jours, la plupart des véhicules sur la route sont dotés d'une technologie d'autonomie de niveau 1 et de nombreux systèmes autonomes de niveau 2. Il est important de noter que la technologie ne progressera pas nécessairement de niveau en niveau. En fait, plusieurs entreprises expérimentent aujourd'hui l'automatisation de niveau 4 et proposent de sauter complètement l'automatisation de niveau 3. D'une manière générale, en raison du faible taux de renouvellement des véhicules,

ces derniers auront des autonomies hétérogènes sur la route posant encore des défis pendant une longue période de transition.

	Contrôle de direction, accélération et décélération	Surveillance de l'environnement de conduite	Fallback en cas d'échec d'automatisation	Capacités du système
Niveau 0 : Conducteur seul sans aucune automatisation	Conducteur humain	Conducteur humain	NA	NA
Niveau 1 : Conduite assistée	Conducteur humain	Conducteur humain	Conducteur humain	Quelques modes de conduite
Niveau 2 : Conduite partiellement automatisée	Système automatisé	Conducteur humain	Conducteur humain	Quelques modes de conduite
Niveau 3 : Conduite automatique sous conditions	Système automatisé	Système automatisé	Conducteur humain	Quelques modes de conduite
Niveau 4 : Niveau élevé d'automatisation	Système automatisé	Système automatisé	Système automatisé	Quelques modes de conduite
Niveau 5 : Autonomie complète	Système automatisé	Système automatisé	Système automatisé	Tous les modes de conduite

TABLE 3.3 – Les différents niveaux pour atteindre une automatisation complète de la tâche de conduite.

Le défi des véhicules autonomes est rendu particulièrement difficile en raison de l'immense variabilité inhérente à la tâche de conduite causée essentiellement par des caractéristiques associées à l'être humain, l'environnement et les technologies utilisées (Fridman et al., 2019). Tout d'abord, l'incertitude sous-jacente au comportement humain impacte largement l'interaction et la résolution de conflits entre les différents acteurs de la route tels que conducteurs, cyclistes et piétons. Ce comportement devient encore plus imprévisible en considérant la diversité des styles de conduite, l'appréhension du concept de l'automatisation et l'aptitude à intervenir et s'adapter aux nouvelles situations d'urgence. Deuxièmement, la complexité des environnements et le dynamisme des objets à reconnaître créent des problématiques spécifiques aux véhicules autonomes au niveau de l'identification et l'interprétation des scènes de conduite et des différents acteurs impliqués. D'autant plus qu'il faut prendre en compte des facteurs contextuels (météo, luminosité) instables et variés qui influencent considérablement la performance de la perception artificielle. Le

troisième facteur d'incertitude est d'ordre matériel et propre au véhicule lui même. En effet, ce dernier englobe une multitude de fonctionnalités (direction, freinage, accélération...) et de systèmes qui doivent interagir avec des logiciels de contrôle générant de nouveaux périmètres de vulnérabilité technologique qui nécessitent des mises à jour continues et une gestion d'erreurs informatiques.

3.4.2 Conduite par simulation

Des milliers de véhicules autonomes sont testés sur les routes par des entreprises telles que Waymo, Cruise, Uber et Tesla, dont certaines ont accumulé des données de tests sur des millions de kilomètres parcourus. En effet, il a été préconisé qu'un véhicule autonome doit être testé sur des centaines de millions de kilomètres dans des différentes conditions pour démontrer sa fiabilité statistique en termes de sécurité routière, ce qui pourrait nécessiter des dizaines d'années d'essais (Kalra & Paddock, 2016). D'un autre côté, de tels essais coûtent une fortune aux industriels. En moyenne, un véhicule autonome entièrement équipé peut coûter plus d'un demi-million de dollars, selon MSC Software [1]. Un petit parc de 20 véhicules représenterait donc un investissement de 10 à 12 millions de dollars, rien qu'en matériel. Un autre aspect à considérer pour les tests en mode réel est l'impossibilité de dérouler certains scénarios à haut risque. Par exemple, dans le cas d'autonomie de niveau 3 et 4, lorsque le système de contrôle est inopinément incapable de faire face à une situation imminente, combien de temps faut-il au conducteur pour comprendre la situation et reprendre le contrôle du véhicule, spécialement s'il était distrait par d'autres activités (lecture, conversation...).

Une solution possible à l'entraînement et à la validation des véhicules autonomes consiste à utiliser des systèmes de simulation, communs dans d'autres domaines tels que le secteur aéronautique, l'industrie de l'armement et la formation médicale. Les simulations de conduite peuvent servir à deux objectifs. Le premier consiste à tester et à valider la capacité de ces véhicules en termes de perception, navigation et contrôle de l'environnement. Le second objectif est de générer une grande quantité de données d'apprentissage pour entraîner des méthodes AI, par exemple les DNNs, largement adoptées dans la vision automatique (Gaidon et al., 2016). Le moyen de construire de tels simulateurs consiste à combiner des techniques d'infographie, de modélisation basée sur la physique et des techniques de planification de mouvement robotique afin de créer un environnement virtuel dans lequel les véhicules en mouvement peuvent être animés.

Traditionnellement, la simulation était utilisée pour restituer des données de capteurs collectées dans le monde réel qui servent à l'entraînement et la mise à jour des logiciels de conduite afin de gérer correctement certaines situations spécifiques. Récemment, des simulateurs plus avancés et de haute précision ont été développés dont par exemple CARLA d'Intel (Dosovitskiy et al., 2017), AirSim de Microsoft (Shah et al., 2018), Drive Constellation de NVIDIA [2] et Carcraft de Google-Waymo (Madrigal, 2017). Ils offrent des réseaux routiers virtuels très réalistes dans lesquels évolue un véhicule autonome en interaction avec plusieurs autres acteurs dynamiques. Cette caractéristique permet d'entraîner efficacement des agents AI avant de les lancer dans une phase de réglage fin dans le monde réel économisant considérablement les coûts et la durée d'entraînement

1. Source : https ://pages.mscsoftware.com/Engineering-Reality-Magazine-Winter-2018-Issue.html
2. https ://www.nvidia.com/en-us/self-driving-cars/drive-constellation/

d'un programme complet de conduite autonome.

3.4.3 Approche de bout en bout

La conduite autonome est un domaine d'application laborieux pour le ML, étant donné que le fait de « bien » conduire est sujet à des définitions subjectives, et qu'il n'est pas évident de pouvoir spécifier le comportement correct d'un système de conduite, ni de choisir les caractéristiques pertinentes pour aboutir à un apprentissage efficace (Talpaert et al., 2019). Devant la multitude de réponses possibles et acceptables à des situations similaires de conduite, les stratégies de contrôle basées sur le ML doivent surmonter le manque d'une métrique rigoureuse évaluant la qualité de conduite et l'absence d'un signal fort pour faciliter l'imitation d'un expert ou d'un benchmark. Alors que les méthodes d'apprentissage supervisé ne sont pas capables d'apprendre la dynamique d'un agent évoluant dans un environnement (Sutton & Barto, 2018), le RL est bien formulé pour gérer des processus de décision séquentiels, ce qui en fait une approche naturelle pour apprendre la tâche de conduite autonome.

Généralement, de tels systèmes sont élaborés sur la base de deux approches différenciées selon l'architecture de transmission d'information des capteurs vers les actionneurs (figure 3.6). Soit ils se basent sur une chaîne (pipeline) modulaire de tâches, soit ils suivent un mode d'apprentissage de bout en bout (end-to-end). Dans les deux conceptions, qui ont largement profité des avancées récentes en matière d'apprentissage profond, un moniteur est déployé pour assurer la sécurité de toute l'architecture. Les composantes de la chaîne modulaire (figure 3.6 (a)) peuvent être conçues en permutant entre méthodes AI et des approches classiques sans apprentissage (Paden et al., 2016). Structurées hiérarchiquement, elles correspondent aux 4 fonctions de base de la conduite autonome : (1) perception de l'environnement et localisation, (2) planification de trajectoire de haut niveau, (3) arbitrage de comportement pour la prise de décisions (actions) et (4) contrôle de mouvement réactif aux erreurs générées lors de l'exécution du mouvement planifié.

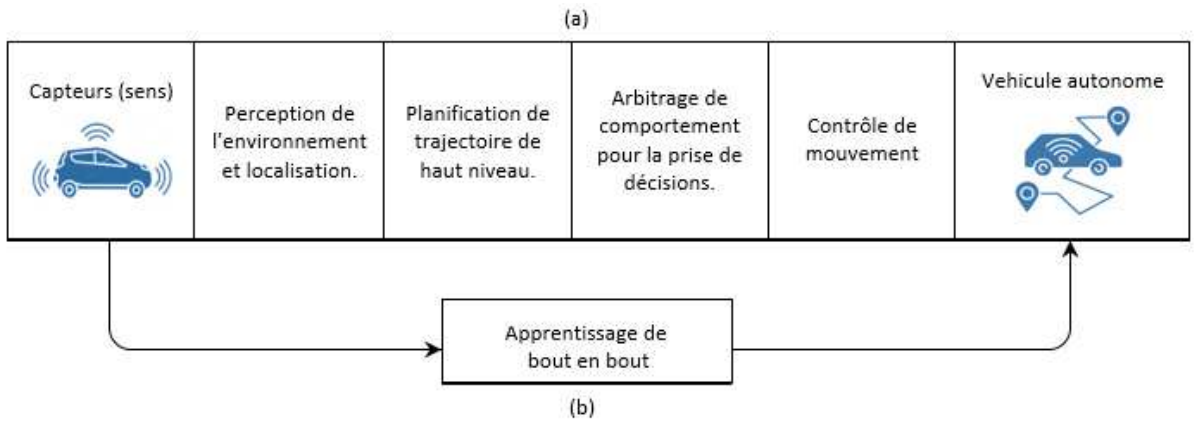

FIGURE 3.6 – Processus de la tâche de conduite autonome : (a) vision modulaire par fonction de base, (b) vision globaliste pour un apprentissage de bout en bout.

D'une manière plus globaliste, l'apprentissage de bout en bout (figure 3.6 (b)) consiste, de son côté, à coder le processus hiérarchique en une seule architecture d'apprentissage

profond qui traduit les observations sensorielles en commandes de contrôle (Xu et al., 2017). Dans cette thèse, nous nous situons dans ce deuxième cas de figure pour évaluer nos contributions de RL robuste et de méta-apprentissage. Le paradigme de l'apprentissage de bout en bout formulé en tant qu'un algorithme de rétropropagation a été introduit pour la première fois par Pomerleau (1989). Il a proposé ALVINN (the Autonomous Land Vehicle in a Neural Network), un système automatisé capable d'orienter un véhicule à partir de l'observation de la route. Le deuxième travail pionnier dans le cadre de ce même paradigme est l'approche DAVE (Darpa Autonomous VEhicle) développée par Muller et al. (2006) qui a permis à un véhicule autonome de parcourir avec succès une route présentant des obstacles, après une phase d'entraînement sur des heures de conduite réelle acquises à partir de scénarios similaires.

Au cours des dernières années, les progrès technologiques réalisés au niveau des ressources de calcul informatique ont facilité l'apprentissage de bout en bout. L'algorithme de rétropropagation pour l'estimation du gradient dans les DNNs est désormais efficacement mis en œuvre sur les unités de traitement graphiques (GPU) parallèles. Un premier groupe des travaux sur le contrôle de bout en bout utilise des DNNs pour effectuer un apprentissage offline sur les données du monde réel ou générées par un simulateur et produire des modèles d'apprentissage reliant directement la scène (des pixels) aux commandes de conduite (Bojarski et al., 2017; Rausch et al., 2017). Le principe commun à ces méthodes consiste à composer une base d'images et de commandes de direction recueillies dans des scénarios de conduite avec des conditions de route, éclairage et météorologiques variées. Avant l'entraînement par un CNN ou un réseau de neurones récurrents (RNN), il est possible d'enrichir les données par augmentation en intégrant des changements et des rotations aux images initiales.

Le deuxième groupe de méthodes conçoit des systèmes de conduite de bout en bout en adoptant une démarche DRL (Sallab et al., 2017; Jaritz et al., 2018). L'entraînement dans ce cas est principalement réalisé online à l'aide de simulateurs où un agent explore des environnements virtuels et construit ses stratégies de conduite. Tandis que la perception passive par DNN alimentant un système de contrôle risque de ne pas s'adapter aux environnements de grande variabilité, l'utilisation d'un cadre RL pour l'apprentissage de la conduite permettrait une perception active de la tâche de contrôle et une plus grande capacité d'appréhender la complexité de la conduite dans le monde réel (Talpaert et al., 2019). Le tableau 3.4 présente un ensemble de travaux adoptant le paradigme de bout en bout et qui ont été implémentés sur des simulateurs bien connus de la communauté de recherche.

Méthode	Outils	Description
TORCS DRL (Sallab et al., 2017)	Environnement : Simulateur de jeu de course open-source (TORCS [3]) Technique ML : DRL par Deep Q Network (DQN)	Une méthode de bout en bout qui intègre des entrées de capteur à partir du simulateur TORCS et utilise le DQN pour prédire les actions de conduite. Le modèle est capable de gérer des scénarios partiellement observables. De plus, il propose d'intégrer les avancées récentes dans les modèles d'attention afin d'extraire uniquement les informations pertinentes, ce qui le rend adapté aux systèmes embarqués en temps réel.

WRC6 DRL (Jaritz et al., 2018)	Environnement : Simulateur de jeu de course World Rally Championship 6 (WRC6) Technique ML : DRL par A3C	Une méthode de bout en bout qui intègre des entrées de capteur collectées par un agent navigant sur le simulateur WRC6 et utilisant un algorithme A3C pour générer les actions de conduite. La fonction de récompense n'est pas le score du jeu, mais une agrégation pondérée de trois variables : la différence d'angle entre le cap du véhicule et la ligne de la route, la vitesse et la distance par rapport au milieu de la route.
Agile (Pan et al., 2019)	Environnement : Simulateur Gazebo (Koenig & Howard, 2004) Technique ML : DRL par imitation	Ce travail propose un système d'apprentissage par imitation de bout en bout pour une conduite agile et autonome, utilisant uniquement des capteurs embarqués à faible coût. En imitant un contrôleur optimal de commande prédictive, un RL est entraîné dans le simulateur Gazebo pour transcoder les observations en commandes de direction et d'accélération. Le test s'est déroulé sur le terrain en mode réel.
J-Net (Kocić et al., 2019)	Environnement : Udacity, Inc. [4] , simulateur de voiture autonome Technique ML : CNN	Une solution simplifiée d'un apprentissage prenant comme input les images d'une caméra (pixel brut) afin de prédire l'angle de direction. La collecte de données est effectuée dans le simulateur Udacity, Inc. Un conducteur humain conduit le véhicule dans le simulateur et enregistre simultanément les images et les mesures de direction qui servent à entraîner l'architecture de CNN.
Politique DNN (Rausch et al., 2017)	Environnement : Simulateur CARSIM [5] (commercial) Technique ML : CNN	L'approche consiste à développer un contrôleur basé sur un CNN. La collecte des images labellisées par les angles de direction nécessaire à l'entraînement du CNN est assurée par un conducteur humain qui dirige un véhicule par une manette de jeu dans le simulateur CARSIM.

TABLE 3.4 – Etat de l'art des approches de bout en bout pour la conduite autonome.

3.4.4 Simulateur CARLA

Bien que l'utilisation de la simulation dans la recherche sur la conduite autonome soit répandue, le nombre de plates-formes de simulation adéquates à cette tâche en zones urbaines est réduit. Par exemple, les simulateurs de course open-source comme TORCS ne

3. http ://torcs.sourceforge.net/
4. https ://github.com/udacity/self-driving-car-sim
5. https ://en.wikipedia.org/wiki/CarSim

reflètent pas la complexité de la conduite urbaine en l'absence de piétons, intersections et d'autres caractéristiques qui la distinguent de la course sur piste. D'un autre côté, les jeux commerciaux qui simulent des environnements urbains de haute fidélité tels que GTA 5 (Richter et al., 2017), présentent des limitations dues largement à leur nature commerciale à source fermée. Ils ne prennent pas en charge l'implémentation de scénarios spécifiques, ont une marge limitée de personnalisation et de contrôle sur l'environnement et ne fournissent pas de retour d'information détaillée sur le déroulement de la tâche. Le simulateur CARLA (Dosovitskiy et al., 2017; Palanisamy, 2018) choisi pour l'implémentation de nos méthodes constitue un outil précieux pour la recherche sur la conduite autonome. En effet, il est relativement plus accessible grâce à son module client API et supporte le développement, l'entraînement et l'analyse détaillée des performances des systèmes RL. Dans la suite, nous présentons avec plus de détails les aspects techniques du simulateur CARLA.

3.4.4.1 Structure client-serveur

Nous menons l'expérimentation de nos approches RL avec le simulateur open-source CARLA (Car Learning to Act) dans sa version la plus stable (0.8.2)[6] . Il offre une interface évoluée permettant à l'agent de contrôler un véhicule et d'interagir avec un environnement dynamique. Comparativement aux simulateurs existants, CARLA propose des conditions de conduite urbaine paramétrables et relativement réalistes, avec un ensemble de fonctionnalités avancées permettant de contrôler le véhicule et de recueillir, en retour, des informations sur l'environnement. CARLA est conçu comme un système client-serveur où le serveur implémenté dans Unreal Engine 4 (UE4)[7] exécute les commandes de simulation et renvoie les lectures de la scène. Le client API implémenté en Python envoie les actions prédites par l'agent transcodées en tant que commandes de conduite (détaillées dans le tableau 3.5) et reçoit les mesures de simulation résultantes qui seront interprétées comme des récompenses pour le système RL.

Commande	Description
Direction	L'angle du volant est représenté par un nombre réel compris entre -1 et 1, où -1 et 1 correspondent respectivement aux positions complètement à gauche et à droite.
Accélération	La pression sur la pédale d'accélérateur, représentée par un nombre réel compris entre 0 et 1.
Freinage	La pression sur la pédale de frein, représentée par un nombre réel compris entre 0 et 1.
Freinage à main	Valeur booléenne indiquant si le frein à main est activé ou non.
Marche arrière	Valeur booléenne indiquant si la marche arrière est activée ou non.

Table 3.5 – Description de l'espace d'actions envoyées par le client API.

6. https ://github.com/carla-simulator/carla/releases/tag/0.8.2
7. https ://www.unrealengine.com

3.4.4.2 Environnement

L'environnement $3D$ de CARLA comprend des objets statiques tels que bâtiments, routes, végétation, et panneaux de signalisation ainsi que des acteurs dynamiques, principalement des piétons et des véhicules. La version que nous utilisons comprend deux villes virtuelles dont les cartes sont indiquées dans la figure 3.7 : ville 1 et ville 2 avec respectivement $2{,}9$ et $1{,}4$ km de routes. Les acteurs dynamiques ont été conçus d'une manière à simuler des comportements réalistes et à pouvoir se détecter et s'éviter. Les véhicules sont basés sur le modèle de véhicule UE4 standard (PhysXVehicles) muni d'un contrôleur pour le bon suivi de la voie, respect des feux de circulation, des limitations de vitesse et la prise de décision aux intersections. Les piétons parcourent les rues selon une carte de navigation spécifique à la ville avec un coût encourageant la marche le long des trottoirs et des intersections marquées. Si une voiture entre en collision avec un piéton, ce dernier est supprimé de la simulation et un nouveau piéton apparaît à un autre endroit après un bref intervalle de temps.

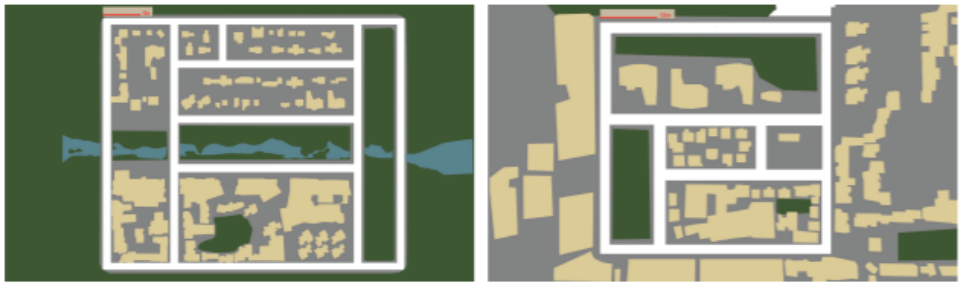

FIGURE 3.7 – Cartes de deux villes utilisées dans CARLA.

Pour augmenter la diversité visuelle, chaque piéton est habillé aléatoirement d'une tenue différente et est équipé d'un ou plusieurs des éléments suivants : smartphone, sacs à provisions, valise, parapluie, etc. En outre, chaque véhicule est peint selon un ensemble de matériaux spécifique au modèle. Le contexte visuel est également enrichi par d'autres paramétrages du serveur tels que la densité de la circulation (nombre d'objets dynamiques), les conditions météorologiques et les régimes d'éclairage. Ces derniers peuvent varier par le choix de la position et la couleur du soleil, l'intensité du rayonnement, la couverture nuageuse, le degré des précipitations et la présence de flaques d'eau dans les rues. Ainsi, pour la version $0.8.2$ il est possible de configurer 15 combinaisons éclairage-météo, dont quelques exemples sont illustrés dans la figure 3.8. Les trois paramètres de configuration de scènes de conduite sont expliqués dans le tableau 3.6.

3.4.4.3 Capteurs

CARLA dispose d'une caméra RGB et de deux pseudo-capteurs qui fournissent la profondeur de vérité-terrain et une segmentation sémantique tels qu'illustrés dans la figure 3.9. 3 paramètres permettent de configurer les capteurs au niveau du client : le nombre, le type et la position $3D$. Le pseudo-capteur de segmentation fournit 12 classes sémantiques : route, marquage de voie, panneau de signalisation, trottoir, clôture, poteau, mur, bâtiment, végétation, véhicule, piéton et autre. Outre les lectures des capteurs, CARLA fournit une

FIGURE 3.8 – Exemples de variabilité des environnements paramétrables dans CARLA.

Paramètres visuels	Description
Nombre de véhicules	Nombre entier de véhicules non contrôlés par le client à générer dans la ville.
Nombre de piétons	Nombre entier de piétons à générer dans la ville.
Condition météo	Un index des préréglages météo / éclairage supportés par la version 0.8.2 : Clear Midday, Clear Sunset, Cloudy Midday, Cloudy Sunset, Soft Rain Midday, Soft Rain Sunset, Medium Rain Midday, Cloudy After Rain Midday, Cloudy After Rain, Sunset, Medium Rain, Hard Rain Midday, Hard Rain Sunset, After Rain Noon, After Rain Sunset.

TABLE 3.6 – Les paramètres de configuration de scènes.

gamme de mesures associées à l'état de l'agent et au respect des règles de circulation (vitesse, nombre de collisions, etc.) qui jouent un rôle important dans l'entraînement et l'évaluation des politiques de conduite. L'ensemble de retour d'information du simulateur est décrit dans le tableau 3.7.

Retours de l'environnement	Description
Position	Position 3D du véhicule par rapport au système de coordonnées de référence (semblable à GPS).
Vitesse	Vitesse du véhicule en kilomètres/heure.
Collisions	Nombre de collisions avec autres voitures, piétons ou objets statiques.
Intersection voie opposée	Fraction de chevauchement de la voie opposée.
Intersection trottoir	Fraction de chevauchements du trottoir.
Heure	Heure actuelle de la conduite.
Accélération	Vecteur $3D$ de l'accélération du véhicule par rapport au système de coordonnées de référence.
Orientation	Un vecteur de longueur unitaire correspondant à l'orientation de la voiture de l'agent.
Lectures des capteurs	Les lectures actuelles de l'ensemble des capteurs et de la caméra.
Information acteurs non-joueurs	Informations sur les acteurs non contrôlés par le client : positions, orientations et cadres englobant pour tous les piétons et voitures présents dans l'environnement.
Feux de circulation	Position et état des feux de circulation.
Panneaux de limitation de vitesse	Position et lecture des panneaux de limitation de vitesse.

TABLE 3.7 – Retour d'information sur l'environnement.

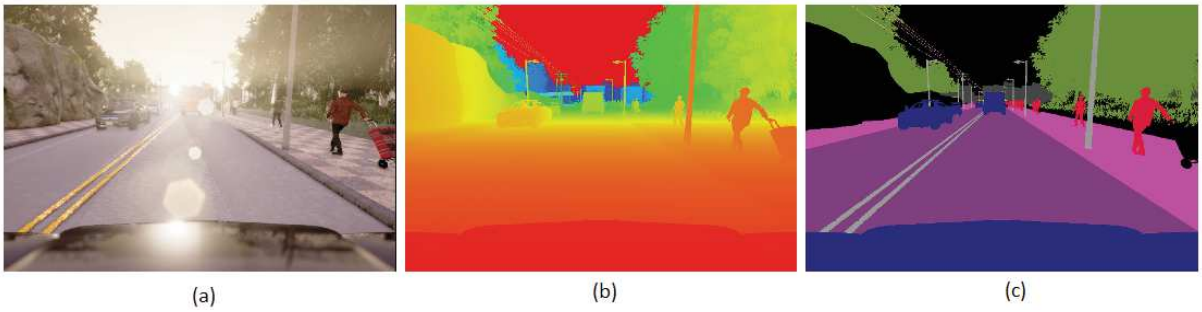

FIGURE 3.9 – Les trois modalités de détection fournies par CARLA. (a) Caméra de vision normale RGB, (b) image en profondeur et (c) segmentation sémantique.

3.5 RL robuste pour la tâche de conduite autonome

Nous avons vu tout au long des sections précédentes que les avancées réalisées dans l'apprentissage profond ont contribué considérablement à l'application, avec succès, du

RL à une multitude de tâches relativement complexes, principalement les jeux vidéos et certaines manipulations robotiques. Cependant, il a été montré dans la littérature que l'inefficacité des données et la tendance à se spécialiser à un domaine bien précis empêchent l'extension des approches DRL aux tâches du monde réel, caractérisées par leurs non-stationnarité et grande dimensionnalité (voir section 3.3.2). Plus spécifiquement, la tâche de conduite autonome est particulièrement difficile à gérer par le système RL en raison de ses dynamiques complexes et contradictoires. En effet, l'agent de conduite doit interagir, dans des conditions météorologiques et d'éclairage changeantes, avec plusieurs acteurs pouvant se comporter de manière inattendue, identifier les règles de circulation et estimer la vitesse et les distances appropriées.

L'inadaptabilité entre les spécificités de la solution DRL et la variabilité des tâches du monde réel aboutit à des gradients de politique de haute variance et un apprentissage instable et non convergent (Haarnoja et al., 2018). Notre première contribution pour traiter cette problématique consiste à développer une approche RL robuste pour la tâche de conduite autonome, dans le sens où l'agent parvient à généraliser sa politique à toutes les perturbations et les changements de paramètres de l'environnement (Pattanaik et al., 2018). Afin d'atteindre cet objectif, nous introduisons un critique avec retours multi-step (MSRC pour multi-step returns critic) réduisant la variance et stabilisant l'apprentissage de l'agent de conduite. L'expérimentation permettra d'évaluer la performance d'une approche RL robuste face à la non-stationnarité et la haute dimensionnalité de l'environnement et à motiver le recours au méta-apprentissage pour des politiques plus généralisables dans le chapitre 4.

3.5.1 Modèle

Nous proposons dans notre approche une instanciation des méthodes actor-critic décrites dans le paragraphe (3.3.3.3) en intégrant une composante MSRC dans le fonctionnement d'un agent RL de type policy-based (3.3.3.2). Le point de départ du modèle proposé est un cadre markovien (3.2.3) où nous introduisons la notion de trajectoire pour permettre un traitement par apprentissage TD multi-step et une évaluation épisodique de la tâche de conduite autonome (3.4).

3.5.1.1 Recherche de politique

La tâche RL T_i considérée est un processus MDP défini en fonction du $n-$uplet $(S, A, r, Tr, \gamma, \rho_0, H)$ où, à titre de rappel, S est l'ensemble des états, A est l'ensemble des actions, Tr est la fonction de transition calculant la probabilité $\mathcal{P}(s_{t+1}, r_{t+1}|s_t, a_t)$ de transition vers un état s_{t+1}, r est une fonction de récompense, γ est le facteur d'actualisation et H un horizon fini. Nous rajoutons également la variable ρ_0 pour désigner la distribution initiale d'état. La configuration RL proposée consiste à apprendre une politique stochastique π de paramètres θ qui associe chaque état à une action optimale permettant de maximiser le retour R_t d'une trajectoire $\tau_{(t,t+H-1)} = (s_t, a_t, \ldots, s_{t+H-1}, a_{t+H-1}, s_{t+H})$ de T_i.

$$R_t = \sum_{k=t}^{t+H-1} \gamma^{k-t} r_{k+1} \tag{3.20}$$

Dans les approches DRL de type policy-based, une solution à ce problème revient à utiliser le théorème du gradient de politique (équation 3.17), ensuite appliquer la règle

REINFORCE ou l'une de ses variantes approximant directement la fonction de valeur $Q^{\pi_\theta}(s_t, a_t)$ par le retour empirique R_t de la trajectoire afin de mettre à jour θ dans la direction de l'équation (équation 3.18). Afin de profiter des garanties de convergence de telles démarches tout en évitant la forte variance des retours empiriques déstabilisant l'apprentissage de l'agent, nous procédons différemment à l'approximation de la fonction de valeur $Q^{\pi_\theta}(s_t, a_t)$ en adoptant un critique pour l'évaluation de la politique.

3.5.1.2 Réduction de la variance

Dans leur travail de référence sur les gradients de politiques RL, Sutton et al. (2000) établissent comme critère de convergence d'une approximation paramétrable $Q^{\pi_\theta}(s_t, a_t, \omega)$ de $Q^{\pi_\theta}(s_t, a_t)$ sa capacité à avoir une moyenne zéro pour chaque état :

$$\sum_a \pi_\theta(s_t, a_t)\, Q^{\pi_\theta}(s_t, a_t, \omega) = 0,\ \forall s \in S \tag{3.21}$$

Dans ce sens, il est plus pertinent de considérer une approximation de la fonction avantage $A^{\pi_\theta}(s_t, a_t) = Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)$ (Baird, 1993) plutôt que $Q^{\pi_\theta}(s_t, a_t)$ dans la formulation du gradient (équation 3.17) qui devient :

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\pi_\theta}\Big[\nabla_\theta log\, \pi_\theta(a_t|s_t)\, A^{\pi_\theta}(s_t, a_t)\Big] \tag{3.22}$$

Cette expression intégrant $V^{\pi_\theta}(s_t)$ comme baseline vise à obtenir la valeur relative des actions dans chaque état, et non pas la valeur absolue. Elle fournit une mesure de comparaison de chaque action au rendement espéré de l'état. Ceci aboutit à des magnitudes inférieures à celle de $Q^{\pi_\theta}(s_t, a_t)$ et une réduction de la variance de l'estimateur du gradient sans modification des espérances. L'utilisation d'une fonction baseline est inspirée par des travaux antérieurs traitant des problèmes de variance dans les systèmes RL (Williams, 1992; Dayan, 1991).

L'introduction d'un critique pour l'évaluation de la fonction avantage est possible grâce à l'application de l'apprentissage TD pour l'estimation de $A^{\pi_\theta}(s_t, a_t)$. Rappelons que cette technique est une combinaison des méthodes Monte Carlo en apprenant directement de l'expérience sans un modèle de l'environnement et de l'équation de Bellman (Bellman & Dreyfus, 1962) en mettant à jour une estimation sur la base d'une estimation future, sans attendre la fin d'une trajectoire (bootstrapping) (Sutton & Barto, 2018). Il en résulte la méthode 1-step TD ou TD(0) qui définit le retour de l'environnement par bootstrapping (TD target) :

$$R_t = r_{t+1} + \gamma V^{\pi_\theta}(s_{t+1}) \tag{3.23}$$

En approximant $Q^{\pi_\theta}(s_t, a_t)$ par R_t dans la formulation de la fonction avantage, nous obtenons une estimation de cette dernière par l'erreur TD (notée δ_t) qui quantifie la différence entre la valeur estimée $V^{\pi_\theta}(s_t)$ et une meilleure estimation $r_{t+1} + \gamma\, V^{\pi_\theta}(s_{t+1})$:

$$A^{\pi_\theta}(s_t, a_t) \approx \delta_t = r_{t+1} + \gamma\, V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t) \tag{3.24}$$

Ce qui nous conduit à la version finale du gradient de la politique π_θ après mise à jour de l'équation 3.22 :

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\pi_\theta}\Big[\nabla_\theta log\, \pi_\theta(a_t|s_t)\, \delta_t\Big] \tag{3.25}$$

3.5.1.3 Généralisation de l'apprentissage TD

Compte tenu de la nature complexe de la tâche de conduite autonome en milieu urbain, nous utilisons une version généralisée de l'apprentissage TD, appelés n-step TD ou TD(n), en prolongeant le bootstrapping sur plusieurs transitions. Il a été démontré que les méthodes TD(n) améliorent les performances d'apprentissage, en particulier dans les systèmes DRL (Mnih et al., 2016; Munos et al., 2016; Espeholt et al., 2018). En effet, cela permet au critique d'approfondir sa connaissance de l'environnement, en collectant plus d'informations, avant de mettre à jour la politique de l'acteur. D'un autre côté, le bootstrapping fonctionne mieux avec des changements de statut significatifs et reconnaissables, ce qui est plus probable à travers n transitions qu'une seule (Sutton & Barto, 2018). Le TD(n) se situe entre les méthodes de Monte Carlo qui présentent une grande variance à cause de l'utilisation de toute la trajectoire pour estimer la fonction $V^{\pi_\theta}(s_{t+1})$ et le TD(0) qui souffre d'un biais élevé en se basant uniquement sur une transition. Il propose ainsi un compromis biais-variance pouvant le rendre plus efficace que ces deux approches extrêmes (Asis & Sutton, 2018).

Plus formellement, reprenons l'expression du retour de l'environnement par Monte Carlo (équation 3.4) :

$$R_t = \sum_{k=t}^{t+H-1} \gamma^{k-t}\, r_{k+1} \tag{3.26}$$

L'apprentissage TD permet de son côté de formuler ce retour par bootstrapping :

- 1-step TD : $r_{t+1} + \gamma\ V^{\pi_\theta}(s_{t+1})$, où $\gamma V^{\pi_\theta}(s_{t+1})$ corrige l'absence des termes $\sum_{k=t+1}^{t+H-1} \gamma^{k-t} r_{k+1}$
- 2-step TD : $r_{t+1} + \gamma r_{t+2} + \gamma^2\ V^{\pi_\theta}(s_{t+2})$, où $\gamma^2\ V^{\pi_\theta}(s_{t+2})$ corrige l'absence des termes $\sum_{k=t+2}^{t+H-1} \gamma^{k-t} r_{k+1}$

  ...
- n-step TD : $r_{t+1} + \gamma r_{t+2} + ... + \gamma^{n-1} r_{t+n} + \gamma^n\ V^{\pi_\theta}(s_{t+n})$, où $\gamma^n\ V^{\pi_\theta}(s_{t+n})$ corrige l'absence des termes $\sum_{k=t+n}^{t+H-1} \gamma^{k-t} r_{k+1}$

Ainsi, nous définissons, d'une manière générale, un retour paramétrable sur n transitions (n-step TD target) par :

$$R_t = \Big[\sum_{k=t}^{t+n-1} \gamma^{k-t} r_{k+1} \Big] + \gamma^n V^{\pi_\theta}(s_{t+n}) \tag{3.27}$$

La partie n transitions $\sum_{k=t}^{t+n-1} \gamma^{k-t} r_{k+1}$ est peu biaisée avec une variance élevée, tandis que la partie estimation $\gamma^n V^{\pi_\theta}(s_{t+n})$ est source de biais, mais de variance réduite. A titre comparatif, la figure ci-dessous illustre les différents types de retours.

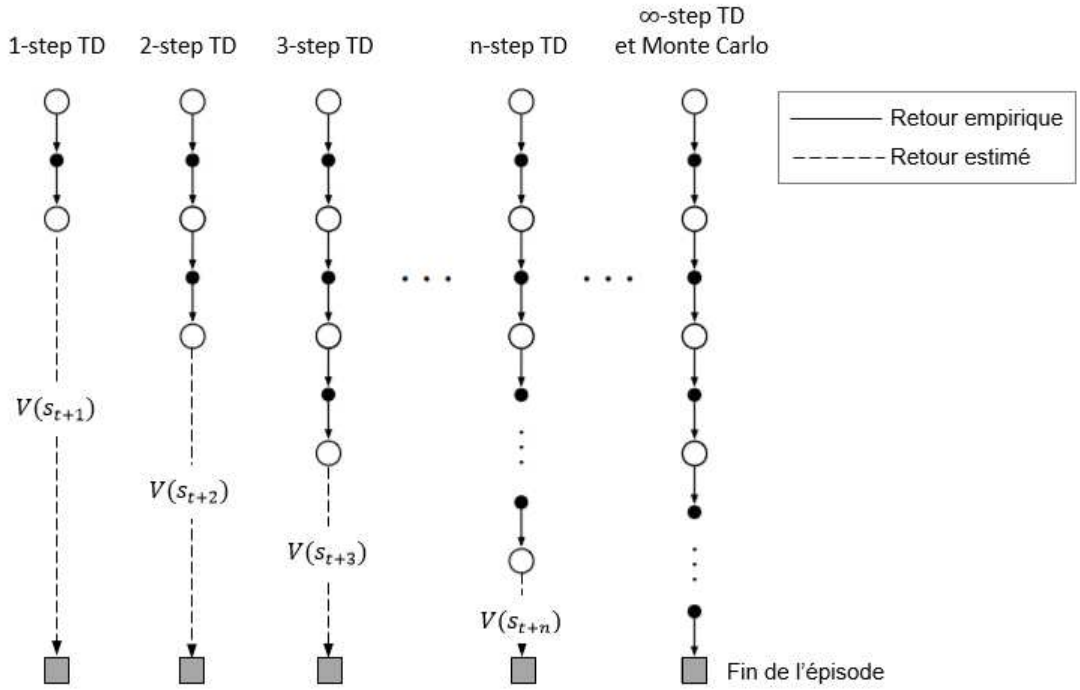

FIGURE 3.10 – Illustration des méthodes n-step return : du TD(0) aux méthodes de Monte Carlo.

Dans la littérature, la meilleure façon de choisir le paramètre n pour arbitrer entre biais et variance n'est pas établie et semble varier d'un domaine à un autre (De Asis et al., 2018). En intégrant la nouvelle formulation n-step du retour R_t (3.27) dans l'équation 3.24, l'erreur TD(n) est définie par :

$$\delta_t = \Big[\sum_{k=t}^{t+n-1} \gamma^{k-t} r_{k+1} \Big] + \gamma^n V^{\pi_\theta}(s_{t+n}) - V^{\pi_\theta}(s_t) \tag{3.28}$$

3.5.1.4 Fonctionnement du MSRC

Nous récapitulons, à travers les outputs des deux derniers paragraphes, le fonctionnement de notre approche que nous appelons MSRC dans la suite de cette section pour simplifier la notation. Tout d'abord, rappelons que par analogie à l'équation (3.15), la fonction de valeur du critique est approximée par une fonction paramétrable de paramètres ω :

$$V^{\pi_\theta}(s_t) \approx V^{\pi_\theta}(s_t, \omega) \tag{3.29}$$

La mise à jour des paramètres de $V^{\pi_\theta}(s_t, \omega)$ se fait par montée de gradient exploitant l'erreur TD(n) avec un taux d'apprentissage β (Grondman et al., 2012) :

$$\omega \leftarrow \omega + \beta \nabla_\omega V^{\pi_\theta}(s_t, \omega)\, \delta_t \tag{3.30}$$

De même, la politique stochastique $\pi_\theta(a_t|s_t)$ de l'acteur est optimisée avec un taux d'apprentissage α selon (l'équation 3.25) :

$$\theta \leftarrow \theta + \alpha \nabla_\theta \, log \, \pi_\theta(a_t|s_t) \, \delta_t \tag{3.31}$$

Pour calculer une mise à jour de son modèle d'apprentissage à un instant t, l'algorithme MSRC sélectionne d'abord une suite de k actions sur une trajectoire τ à l'aide de sa politique initiale π_θ pour des étapes allant jusqu'à n ($k = n$) ou jusqu'à ce qu'un état terminal s_H soit atteint en fin d'épisode ($k < n$). Au bout de ce processus, l'agent reçoit k récompenses de l'environnement depuis sa dernière mise à jour. L'algorithme calcule par la suite les retours et les TD targets intermédiaires pour chacune des paires état-action rencontrées tout au long de la trajectoire, afin d'en déduire les n-step TD error et le gradient final. Chaque TD target intermédiaire utilise le plus long retour possible, ce qui donne le retour d'une étape pour le dernier état, un retour sur deux étapes pour l'avant-dernier, et ainsi de suite, pour un total de k retours. Le gradient calculé est appliqué en une seule montée de gradient pour générer les nouveaux modèles d'apprentissage de l'acteur ($\pi_{\theta'}$) et du critique ($V^{\pi_\theta}(s_t, \omega')$). Une description détaillée de l'approche est donnée dans l'algorithme 2.

3.5.2 Configuration de l'expérimentation

Nous examinons les performances de l'algorithme MSRC sur la tâche complexe de conduite autonome urbaine selon une configuration d'apprentissage de bout en bout telle que décrite dans le paragraphe 3.4.3. L'évaluation expérimentale a pour objectif de démontrer que l'incorporation d'une composante critique avec n-step TD dans un cadre DRL renforce la robustesse de l'agent en contrôlant et en guidant sa stratégie d'apprentissage. Empiriquement, nous nous attendons à (1) une réduction de la variance du gradient des acteurs, (2) une tendance ascendante des rendements épisodiques et plus généralement (3) de meilleures performances comparativement au cas où la composante MSRC est désactivée dans l'algorithme DRL.

3.5.2.1 Espaces d'observations et d'actions

L'agent interagit avec l'environnement en générant des actions et en recevant des observations sur des intervalles de temps réguliers. L'espace d'action sélectionné pour nos expériences est construit sur la base de trois commandes de conduite parmi la liste décrite dans le tableau 3.5 : direction, freinage et accélération. Nous avons opté pour une discrétisation de ces commandes afin de simplifier les calculs et les modèles déployés et de se focaliser sur les traitements assez lourds du DRL et méta-apprentissage par la suite. Le traitement d'espaces d'actions continus constitue une perspective intéressante pour notre travail et une considération plus réaliste de la tâche de conduite autonome. La discrétisation consiste à définir un vecteur de commande à deux dimensions limitées à l'intervalle $[-1, 1]$. Le premier attribut est assigné à l'accélération $[0, 1]$ et le freinage $[-1, 0]$. Le deuxième attribut est relatif à la direction déjà décrite dans le tableau 3.5. Nous fixons 9 instances de ce vecteur de commande de manière à réduire au minimum le nombre d'actions possibles tout en permettant le bon contrôle du véhicule dans l'environnement de simulation contre une dégradation acceptable de la précision. La transposition des actions retenues pour l'expérimentation est définie dans la figure 3.11.

Algorithme 2 : n-step TD actor-critic (MSRC)

Entrée : Initialisation aléatoire des paramètres θ de la politique π
Initialisation aléatoire des paramètres ω de la fonction valeur V_π
γ= Facteur d'actualisation
n = Nombre de step

Sortie : θ^* et ω^*

Variable : $fin\_episode$ = Variable booléenne
R = Somme de récompense par trajectoire
$start$ = Indice du 1^{er} retour de la trajectoire

pour *chaque épisode* **faire**
  $fin\_episode \leftarrow Faux$
  Recevoir l'état initial s_t de l'environnement

  répéter
    Initialiser $R \leftarrow 0$
    $start \leftarrow t+1$
    répéter
      $a_t \leftarrow \pi(s_t, \theta)$
      $v_t \leftarrow V(s_t, \omega)$
      Recevoir la valeur de récompense r_{t+1} et la mise à jour de l'environnement s_{t+1}
      Mémoriser r_{t+1} , s_{t+1}
      si s_{t+1} *est un état terminal* **alors**
        $fin\_episode \leftarrow Vrai$
      fin
      $t \leftarrow t+1$

    jusqu'à $t = start + n - 1$ *ou* $fin\_episode = Vrai$;
    `// Calcul de n-step TD target`

$$\mathrm{R} = \begin{cases} 0 & si \quad fin\_episode = Vrai \\ V(s_t, \omega) & sinon. \end{cases}$$

    pour $i \in \{t, ..., start\}$ **faire**
      $R \leftarrow \gamma R + r_i$
    fin

    Calculer δ_t en utilisant l'équation 3.28
    Mettre à jour θ et ω en utilisant respectivement les équations 3.31 et 3.30

  jusqu'à $fin\_episode = Vrai$;
fin

Les observations qui alimentent l'acteur et le critique sont constituées de captures de scènes réalisées par une caméra RGB installée en avant du véhicule. Le reste des observations physiques décrites dans le tableau 3.7 sont utilisées pour générer les récompenses empiriques qui rentrent dans la formule de calcul des retours d'actions prises par l'acteur et de l'erreur TD fournie par le critique.

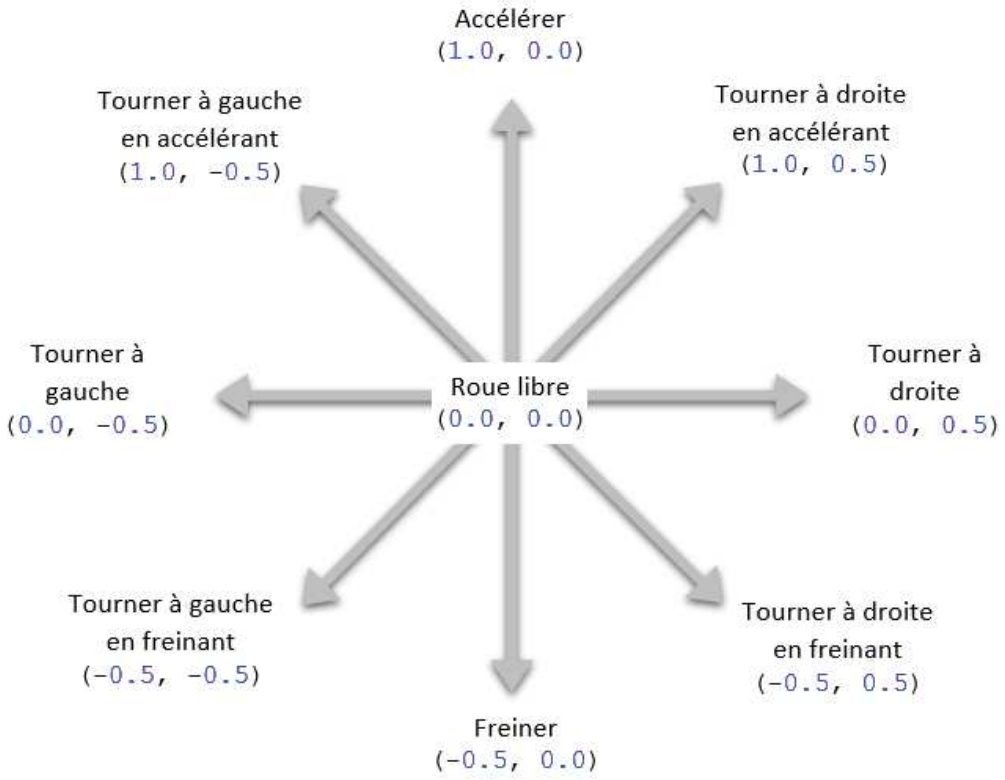

FIGURE 3.11 – Discrétisation de l'espace d'actions pour le contrôle du véhicule autonome dans CARLA.

3.5.2.2 Fonction de récompense

La fonction de récompense qui constitue la partie empirique du retour de l'environnement R_t est conçue linéairement sous la forme d'une somme pondérée de certaines observations de l'environnement afin de refléter les objectifs de performance et de sécurité de la conduite. L'idée est de calculer un écart à l'instant t entre la mesure actuelle (étape t) et la mesure précédente (étape $t-1$) de l'observation sélectionnée reflétant la marge d'évolution réalisée par l'agent entre deux instants successifs. Chaque écart est répercuté par la suite avec un coefficient positif ou négatif sur la récompense globale. Les variables pondérées positivement sont la distance restante pour atteindre la cible en km (DT pour distance to tatget) et la vitesse en km/h (SP pour speed). Les variables pondérées négativement sont le nombre de collisions avec véhicules, piétons et autres (CD pour collision damage), l'intersection avec le trottoir (SI pour sidewalk intersection) et la voie opposée (OI pour opposite lane intersection). Par exemple, l'agent recevra une récompense si la distance au but diminue et une pénalité chaque fois qu'une collision ou une intersection avec la voie opposée est enregistrée. Les coefficients utilisés dans la fonction de récompense sont ceux proposés par Dosovitskiy et al. (2017) :

$$\begin{aligned} r_t = 1000\,(DT_{t-1} - DT_t) \; + \; 0.05\,(SP_t - SP_{t-1}) \; - \; 0{,}00002\,(CD_t - CD_{t-1}) \\ - \; 2\,(SI_t - SI_{t-1}) \; - 2\,(OI_t - OI_{t-1}) \end{aligned} \tag{3.32}$$

3.5.2.3 Contrôle continu

L'entraînement de l'agent RL suit une navigation orientée objectif sur une route droite avec un seul virage. Ainsi il est initialisé quelque part en ville et doit atteindre un point de destination. Dans notre expérience, l'agent est autorisé à ignorer les limitations de vitesse et les feux de signalisation. Un épisode est terminé lorsque la destination cible est atteinte, après une collision avec un acteur dynamique ou après l'écoulement d'une durée de temps prédéfinie. Nous utilisons 2 CNNs de même architecture pour approximer la fonction de valeur du critique et la politique de l'acteur dont les paramètres correspondront aux poids des deux réseaux profonds. L'architecture retenue se compose de 4 couches à convolution, 3 couches de max-pooling et une couche de sortie entièrement connectée. En guise de perspective, il est possible d'expérimenter des techniques issues de la branche de recherche récente NAS (neural architecture search, voir section 4.4) pour déterminer des architectures appropriées à la nature de la tâche. Il faudrait toutefois apporter des solutions efficaces au coût computationnel important engendré par les méthodes existantes.

Au niveau des paramètres du modèle MSRC, le taux d'actualisation γ est fixé à $0,9$ en compatibilité avec la complexité de l'environnement et favorisant une plus grande considération des récompenses futures pour son appréciation courante. Similairement à Mnih et al. (2016) et Dosovitskiy et al. (2017), les taux d'apprentissage α et β sont établis aléatoirement puis recuits linéairement à zéro au cours de l'entraînement afin de permettre une meilleure convergence vers l'optimalité. Le nombre d'étapes relatives au bootstrapping de l'apprentissage TD est déterminé empiriquement ($n = 10$) de manière à équilibrer entre la variance des récompenses collectées et le biais induit par l'estimation de la fonction de valeur. Nous effectuons ainsi une montée de gradient stochastique pour mettre à jour les paramètres de la politique à chaque 10 pas d'entraînement. Le modèle θ résultant de la descente n'est retenu que si sa performance (cumul de récompenses) dépasse celle du dernier stocké. A la fin de l'entraînement, le modèle de la politique le plus performant θ^* est déployé en phase de test.

3.5.3 Résultats et interprétation

L'approche proposée est développée en Python et implémentée à l'aide de la bibliotèque TensorFlow et l'IDE Pycharm. L'expérimentation est exécutée avec un serveur NVIDIA P5000 (1GPU, 16 GB RAM). La phase d'entraînement s'étale sur 10 millions pas de temps (environ 10000 épisodes) pour 72 heures de conduite continue simulée. En l'absence de travaux benchmark similaires au notre sur le récent simulateur CARLA, nous comparons deux versions de notre algorithme : la version complète MSRC, et la version actor-only (sans la composante critique), que nous notons Deep RL. D'un autre côté, les quelques travaux réalisés dans l'environnement CARLA (Dosovitskiy et al., 2017; Liang et al., 2018) indiquent le pourcentage d'épisodes terminés avec succès. Ce type d'évaluation quantitative ne répond pas aux objectifs de l'expérience mentionnés au début de cette section, qui consistent à évaluer et interpréter la contribution du MSRC dans des tâches complexes telles que la conduite autonome. Guidés par plusieurs travaux sur les stratégies RL dans différents domaines (Mnih et al., 2016; Parisi et al., 2019), nous choisissons les mesures de récompenses épisodiques et de récompenses unitaires (par pas de temps) pour évaluer notre approche.

Deux environnements sont configurés pour le déroulement des scénarios d'entraînement

FIGURE 3.12 – Environnement 1 : Ville 2 et « Clear noon weather » .

FIGURE 3.13 – Environnement 2 : Ville 1 et « Hard rainy conditions » .

et de test. « Town 2 et Clear noon weather » (Env1) offre un temps clair à midi et « Town 1 et hard rainy conditions » (Env2) produit des scènes moins éclairées dans un contexte pluvieux. En outre, la densité de la circulation a été paramétrée différemment pour Env1 et Env2. Des captures des deux environnements sont présentées dans les figures 3.12 et 3.13. Les résultats sont collectés et visualisés grâce au module TensorBoard disponible dans la bibliothèque de TensorFlow. C'est un outil qui permet de tracer les métriques quantitatives, déboguer les implémentations et comprendre le fonctionnement des algorithmes DRL.

Les figures 3.14 et 3.15 montrent l'évolution des récompenses générées au cours de la phase d'entraînement menée dans Env1. Nous utilisons la récompense épisodique pour décrire les performances globales des méthodes évaluées et la récompense par pas de temps pour mettre en exergue la variance des retours. Nous pouvons faire quelques observations à cet égard. En termes de performance, notre approche MSRC a mis environ 4000 épisodes avant de surpasser nettement l'algorithme RL standard sur le reste du parcours de conduite, confirmant l'efficacité de la stratégie de RL contrôlée par le n-step TD (figure 3.14). La lenteur relative d'apprentissage constatée sur le premier tiers du parcours d'entraînement pourrait être interprétée par l'existence de 2 modèles de CNNs dans notre approche (relatifs à l'acteur et au critique) dont l'interactivité mettrait plus de temps à montrer son efficacité par rapport au seul modèle côté Deep RL standard.

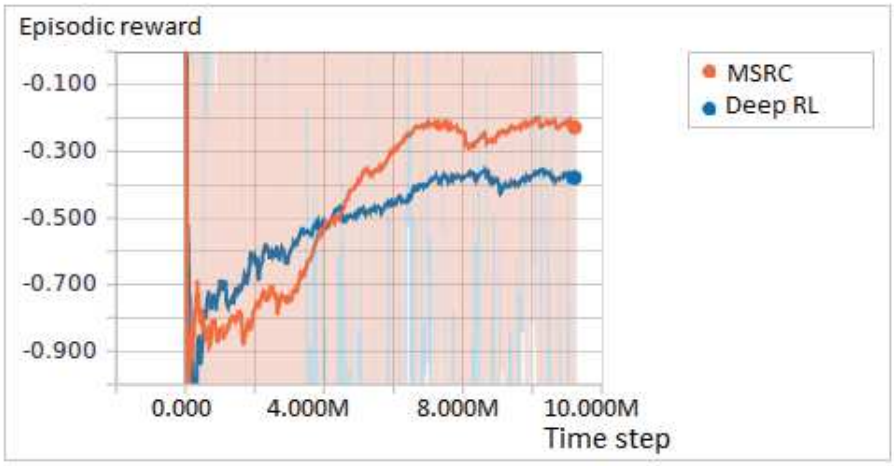

FIGURE 3.14 – Phase d'entraînement dans Env1. Comparaison de la performance de notre approche MSRC et du Deep RL selon les récompenses épisodiques.

En outre, nous constatons qu'au niveau des meilleurs modèles stockés, le MSRC n'a retenu que quelques modèles (5) sur les 2000 premiers épisodes, puis ce nombre est passé à 100 modèles sur les 8000 épisodes restants. Cela signifie que notre méthode a achevé relativement tôt la phase d'exploration et est passée, à partir de 2000 épisodes d'entraînement, à l'étape d'exploitation. En revanche, le Deep RL ne totalise que 10 modèles stockés au cours de la phase d'entraînement, ce qui reflète la faible efficacité d'une stratégie RL aléatoire pour résoudre un problème complexe tel que la conduite autonome. Une dernière interprétation visuelle que nous pouvons déduire du graphe relatif aux récompenses par pas de temps (figure 3.15), est que la variance des prédictions du MSRC est significativement réduite par rapport au Deep RL, confirmant le rôle du n-step TD dans la stabilisation de l'apprentissage.

Au niveau de la phase de test, afin d'évaluer la capacité de généralisation de notre approche, deux différents scénarios ont été adoptés :

- Scénario 1 : entraînement et test sur le même environnement (Env1, figure 3.12).
- Scénario 2 : test effectué sur un environnement différent de celui de l'entraînement (Env2, figure 3.13).

La figure 3.16 récapitule l'évaluation de la performance. Nous pouvons observer que notre approche surpasse largement le Deep RL pour le premier scénario, ce qui signifie que l'entraînement avec un critique conduit à des modèles RL plus performants. Dans le deuxième scénario, le MSRC est toujours plus compétitif affichant des capacités de généralisation, à un nouveau contexte inconnu auparavant, supérieures à celle du Deep RL. Néanmoins, le rendement du MSRC diminue nettement entre les deux scénarios de test, reflétant une certaine fragilité et un manque d'adaptation du modèle face à la variabilité et la non-stationnarité de l'environnement.

Nous pouvons interpréter cette régression de performance à travers les figure 3.12 et 3.13 qui reflètent l'impact de l'intensité d'éclairage sur des scènes similaires capturées dans différentes conditions météorologiques. Une stratégie d'apprentissage naïve conclurait que la couleur du sol est suffisante pour délimiter le trottoir de la route, ce qui pourrait ne pas être

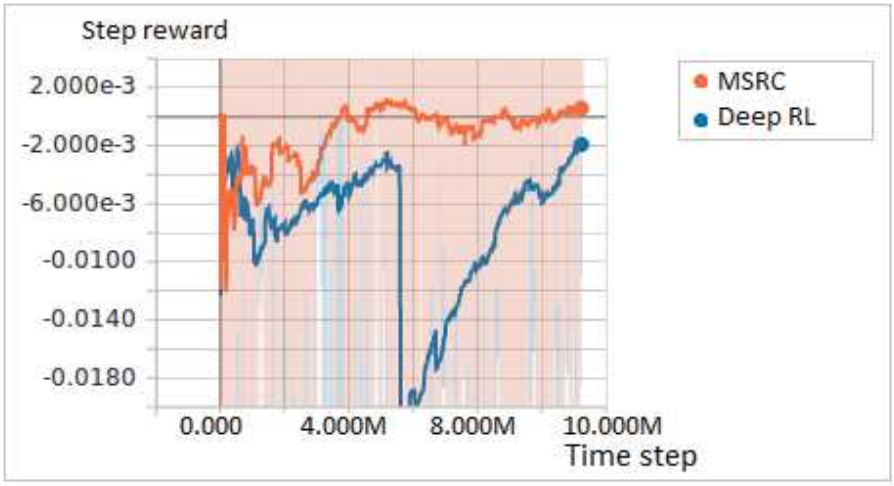

FIGURE 3.15 – Phase d'entraînement dans Env1. Comparaison de la performance de notre approche MSRC et du Deep RL selon les récompenses unitaires par pas de temps.

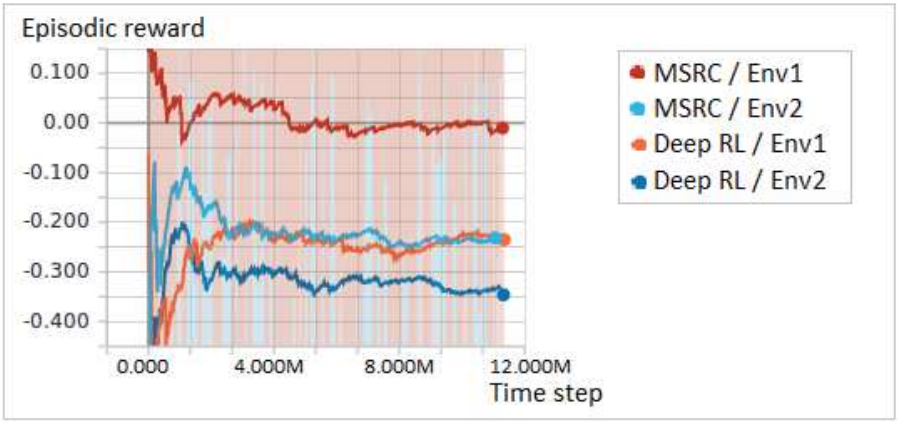

FIGURE 3.16 – Phase de test. Comparaison de la performance de notre approche et du Deep RL standard entraînés et testés dans deux environnements différents.

une règle généralisable à toutes les situations. Cependant, une stratégie d'apprentissage plus sensible au contexte devrait pouvoir prendre en compte une tâche différente correspondant à chaque environnement (scène) où le marquage de voie constituerait un critère commun et plus pertinent pour prédire la meilleure action de conduite. C'est dans ce cadre que nous explorons dans le prochain chapitre la capacité du méta-apprentissage à apporter plus de stabilité et de robustesse à la politique RL à travers une perception multitâche des environnements non-stationnaires.

3.6 Conclusion

Partageant certains principes de la programmation dynamique classique jugée inefficace pour des problèmes à grande échelle ou complexes, le RL se distingue par le fait qu'il ne

nécessite pas le calcul ou le stockage des probabilités de transition définissant le modèle de l'environnement. Cela lui confère la capacité de résoudre, de manière quasi-optimale, un plus grand périmètre de problèmes MDP en intégrant des approximations de fonctions adaptatives basées essentiellement sur des méthodes d'apprentissage profond. Nous avons développé dans ce chapitre une approche DRL robuste adoptant une stratégie hiérarchique d'apprentissage de bout en bout assurée par des architectures de CNNs capables de traduire les observations sensorielles de grande dimension en commandes de contrôle pour la conduite autonome.

Le DRL robuste proposé intègre également un processus d'itération de politique adaptatif renforcé par l'apport de l'apprentissage TD générant des bonus supplémentaires pour guider davantage l'agent vers des meilleures politiques de contrôle. Nous avons réussi dans une première étape à stabiliser l'apprentissage du système RL et à produire un modèle plus performant en termes de récompenses générées tout au long des trajectoires de conduite. L'évaluation des capacités de généralisation de l'approche a toutefois soulevé une dégradation de la performance au passage à de nouveaux scénarios de test non rencontrés lors de l'entraînement. Cela nous permet, dans une deuxième étape, de souligner l'intérêt de rechercher des stratégies plus intelligentes dans l'exploration des environnements complexes, dans le sens où elle seraient plus attentives au contexte d'apprentissage « context-aware » pour une meilleur adaptation du biais des fonctions d'approximation. C'est dans cette optique de gestion multitâche de la conduite autonome en milieu urbain que nous envisageons, dans le prochain chapitre, de recourir au méta-apprentissage pour partir d'une méthode RL robuste vers encore plus de stabilité et de maîtrise des systèmes de contrôle incertains et non-stationnaires.

Chapitre 4

Contrôle adaptatif par méta-apprentissage

4.1 Introduction

Le cheminement méthodologique vers le méta-apprentissage a été guidé par les nouveaux besoins inhérents à la dimensionnalité des données et la variabilité des tâches, et supporté par des avancées technologiques substantielles au niveau des ressources informatiques de stockage et de calcul. La représentation limitée nécessitant des prétraitements manuels rend les techniques superficielles de ML incapables d'explorer et de comprendre la complexité accrue des modèles de données, plus spécifiquement dans le domaine du Big Data. Avec l'avènement de l'apprentissage profond endossé par des percées concrètes dans le développement d'algorithmes d'entraînement, l'AI a franchi un pas de plus vers l'automatisation et le traitement des tâches complexes et de grande dimensionnalité. Des anciens paradigmes dont principalement le RL, ont été revisités et révolutionnés grâce aux nouveaux concepts de représentation hiérarchique et automatique de bout en bout offerts par l'apprentissage profond aboutissant à des algorithmes sophistiqués dépassant la performance humaine dans diverses tâches. La transition naturelle du ML de la sphère de recherche vers l'arène du monde réel grâce à ces évolutions importantes a engendré des nouveaux défis de généralisation et d'optimisation face à la haute diversité et non-stationnarité de ce nouveau périmètre de tâches. Des solutions innovantes s'imposent ainsi pour remédier à l'inefficience de données et la spécialisation de l'apprentissage profond au domaine traité. Le méta-apprentissage présente dans ce cadre un espace de conception de systèmes ML intéressant pour répondre à ces défis en adoptant des stratégies d'apprentissage plus intelligente imitant la capacité humaine à capitaliser l'expérience à travers les tâches et à s'adapter rapidement aux nouvelles situations.

Le terme méta-apprentissage trouve ses origines dans la psychologie de l'éducation faisant référence à l'acquisition de connaissance relative à une expérience antérieure afin de prendre le contrôle d'un processus d'apprentissage et de le guider en fonction du contexte d'une application spécifique (Biggs, 1985). Ce terme a commencé à apparaître dans le domaine de ML vers les années 1990, bien que le concept lui-même remonte au milieu des années 1970 avec les travaux de Rice (1976) sur le problème de sélection d'algorithme. Souvent désigné comme l'action d'apprendre à apprendre, il consiste à observer systématiquement l'évolution de différents modèles d'apprentissage sur un large éventail de tâches, puis à tirer des conclusions de cette expérience ou des métadonnées qui lui sont associées pour apprendre de nouvelles tâches beaucoup plus efficacement. Au niveau d'architecture, l'apprentissage est opéré à deux échelles. D'un côté, un système de base est affecté à l'apprentissage individuel de chaque tâche. D'un autre côté, un méta-système utilise les résultats du niveau précédent pour apprendre graduellement à travers les tâches (Wang et al., 2017).

La caractéristique principale du méta-apprentissage est qu'il se distingue de l'apprentissage standard par sa capacité d'adaptation. En effet, ce dernier possède un biais fixe et accumule de l'expérience par rapport à une tâche spécifique, alors que le méta-apprentissage ajuste dynamiquement son biais selon le contexte d'application et accumule de l'expérience à travers plusieurs tâches (Vilalta & Drissi, 2002). L'acquisition de biais antérieurs permet au méta-apprentissage d'examiner les imperfections des modèles observés soutenant une adaptation rapide aux nouveaux contextes à partir de quelques échantillons d'entraînement (Clavera et al., 2018). En conséquence, le méta-apprentissage peut réduire le temps consacré à l'implémentation, au réglage et à la maintenance des méthodes ML, améliorer considérablement leur fonctionnement dans des conditions variables et promouvoir la conception automatique de nouvelles approches apprises à partir des données.

Dans ce chapitre, nous donnons un aperçu de l'état de l'art sur ce domaine innovant et en constante évolution. Plus spécifiquement, nous étudions comment le regain d'intérêt récent aux DNNs a permis le passage du méta-apprentissage tabulaire classique issu des théories de sélection d'algorithme et « No Free Lunch » vers le méta-apprentissage profond dédié à des tâches plus complexes et de grande dimensionnalité. Nous consacrons également une section à la conception automatique d'architecture de CNN afin de mieux cerner les contraintes liées à l'application du méta-apprentissage aux tâches complexes du monde réel. Dans la dernière section, nous proposons et évaluons une approche MRL basée sur le gradient qui a fait l'objet d'une publication (Jaafra et al., 2019b), afin de répondre aux limites de notre premier algorithme DRL dans la résolution de la tâche non-stationnaire de conduite autonome, discutées dans le chapitre 3.

4.2 Apprentissage par méta-caractéristiques

L'une des premières contributions au méta-apprentissage est le problème classique de sélection d'algorithme (ASP) proposé par Rice (1976) considérant la relation entre les caractéristiques du problème et l'algorithme approprié pour le résoudre. Ensuite, basé sur le concept de l'ASP, le théorème No Free Lunch (NFL) (Wolpert & Macready, 1997) a démontré que la performance de généralisation de tout apprenant est égale à **0**. L'apprenant universel est donc un mythe et chaque algorithme est meilleur sur un ensemble de tâches délimitant son domaine de compétence.

L'ASP et le théorème NFL ont déclenché de nombreuses recherches sur la recommandation de paramètres et d'algorithmes. Leur objectif est de prédire le meilleur modèle pour résoudre une tâche spécifique en appréhendant la relation entre les spécificités des données, appelées méta-caractéristiques, et les performances des algorithmes de base (Jankowski & Grabczewski, 2011; Brazdil et al., 2008; Soares et al., 2004; Pfahringer et al., 2000). Divers méta-apprenants ont été utilisés et consistent généralement en des algorithmes « superficiels », tels que les arbres de décision, les plus proches voisins et les séparateurs à vaste marge (Vanschoren, 2018). Cette section présente la schématisation classique du méta-apprentissage qui se distingue, principalement, par une expression explicite et sous forme tabulaire de ses méta-caractéristiques. Ses fondements permettront de mieux comprendre les approches récentes de méta-apprentissage utilisées dans ce chapitre pour le développement de notre système de conduite par MRL.

4.2.1 Contexte

Le domaine d'apprentissage automatique a produit une multitude d'algorithmes de différentes natures pour traiter les problèmes complexes de fouille de données (Mitchell, 1999). Il s'agit principalement de mettre au point des fonctions de prédiction à partir de base de données d'observations. Déterminer le modèle d'apprentissage le plus approprié à la nature du problème à résoudre n'est pas une tâche évidente. En effet, des travaux tels que (Michie et al., 1994) et (Mooney et al., 1989), révèlent l'inexistence d'un algorithme globalement meilleur sur l'ensemble des tâches à traiter. Ces constatations empiriques ont été confirmées ultérieurement par le théorème NFL (Wolpert & Macready, 1997), stipulant que l'algorithme d'apprentissage le plus performant sur quelques fonctions ne l'est certainement pas sur d'autres (Brazdil et al., 2003) (voir figure 4.1).

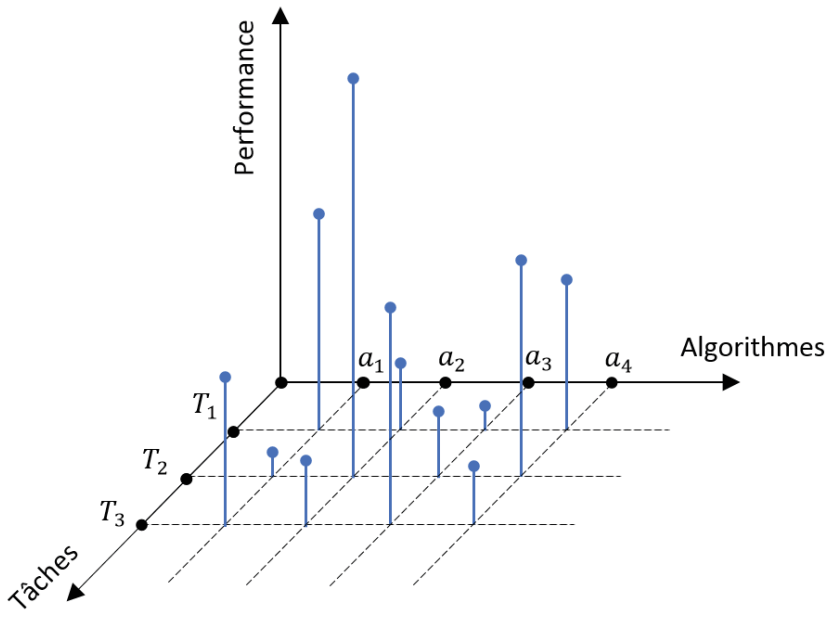

FIGURE 4.1 – Illustration du théorème NFL : chaque algorithme possède une « supériorité sélective » par rapport à l'ensemble des tâches.

En d'autres termes, chaque algorithme possède une « supériorité sélective » (Brodley, 1995) par rapport aux autres algorithmes concurrents, dépendant des caractéristiques du problème à résoudre. Cette dépendance revient au fait qu'un algorithme d'apprentissage se distingue par un « biais inductif » spécifique construit sur la base de ses hypothèses de généralisation de règles induites à partir des données d'entraînement et applicables sur les nouveaux cas reçus pour traitement. Mitchell (1997) définit ce biais comme étant « l'ensemble de toutes les hypothèses qui, conjointement avec la base d'entraînement, justifient, par déduction, la classification de prochaines instances par l'algorithme d'apprentissage ».

Par conséquent, l'identification du modèle d'apprentissage le plus approprié au problème posé requiert de la part de l'analyste une procédure d'évaluation essai-erreur assez coûteuse qui pourrait se compliquer davantage avec l'augmentation du volume de données à examiner. Afin de remédier à cet obstacle, des approches d'extraction de méta-connaissance ont vu le jour, permettant d'établir le lien entre la nature de la tâche et les outils d'apprentissage requis (Brazdil et al., 2003). Ce concept de méta-apprentissage par méta-caractéristiques

accroît la capacité d'un système d'apprentissage d'apprendre à apprendre à travers l'expérience en lui offrant la possibilité d'adaptation au domaine ou à la tâche étudiée. Cette adaptation est concrétisée par la génération de connaissance reliant la performance des algorithmes aux caractéristiques des instances du problème.

4.2.2 Architecture générale

La stratégie de méta-apprentissage par méta-caractéristiques permet au niveau pratique d'aider l'utilisateur à choisir l'algorithme d'apprentissage adéquat pour une tâche d'exploration de données particulière en analysant les règles de corrélation entre les caractéristiques des jeux de données et la performance de cet algorithme. Vilalta et al. (2004) présentent une conception générale de méta-apprentissage composée de deux phases qui seront détaillées dans les paragraphes suivants.

- Acquisition de connaissances : extraction des méta-caractéristiques de l'échantillon de problèmes à traiter et évaluation de la performance des algorithmes de base (base-learner) appliqués à ces données. Une base de méta-connaissance est construite à partir des paires méta-caractéristiques / performances.
- Phase consultative : un algorithme de méta-apprentissage (meta-learner) est appliqué à la base de méta-connaissance pour apprendre la fonction de correspondance entre méta-caractéristiques et algorithme de base approprié. Cette fonction est appliquée aux nouvelles instances à traiter.

4.2.2.1 Mode d'acquisition de connaissances

L'objectif de l'étape d'acquisition de connaissances est d'apprendre le processus d'apprentissage lui-même. La Figure 4.2 illustre ce mode opératoire. Le point d'entrée du système est supposé être des jeux de données constitués par des paires de vecteurs d'attributs et classes (A). A l'arrivée de chaque jeu de données, un composant est invoqué pour l'extraction des méta-caractéristiques de l'ensemble d'exemples reçu (B). Il existe plusieurs techniques de caractérisation d'un jeu de données qui peuvent être groupées en :

- Paramètres statistiques : ces mesures incluent le nombre de classes, d'attributs, le degré de corrélation entre attributs et concept cible, entropie moyenne et conditionnelle des classes, asymétrie, aplatissement, etc. Ces mesures ont fourni des résultats positifs et tangibles (Hilario & Kalousis, 2001).
- Caractérisation basée modèle : la base d'exemples peut être représentée sous forme d'un modèle dont les propriétés sont exploitées dans la caractérisation de cette base. Un exemple courant est celui de la construction d'un arbre de décision à partir d'un jeu de données. La caractérisation est déduite des propriétés de l'arbre lui-même, telles que nœuds par attribut, profondeur de l'arbre, forme, déséquilibre, etc. (Peng et al., 2002).
- Utilisation de descripteurs (Landmarking) : l'information nécessaire à la caractérisation est obtenue à travers la performance d'apprenants simples (landmarks) dont la précision (taux d'erreur) permet d'identifier les zones d'expertise de chaque descripteur. Ainsi, cette information est utilisée par la suite pour sélectionner l'algorithme d'apprentissage approprié au jeu de données analysé (Soares et al., 2001).

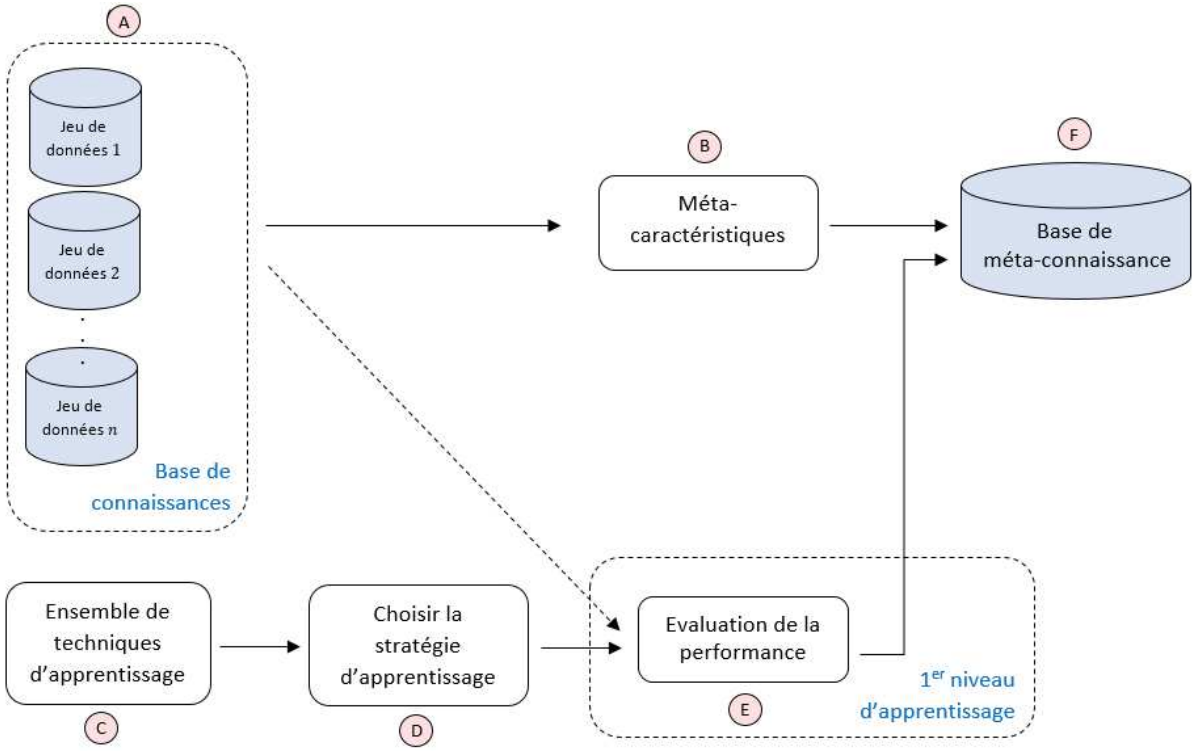

FIGURE 4.2 – Mode d'acquisition de connaissances.

A partir d'un ensemble de techniques d'apprentissage disponibles pour l'analyse (C), une série de stratégies d'apprentissage (telles que classifieurs ou combinaison de classifieurs (D)) est appliquée aux différents jeux de données d'entraînement. Par la suite, une procédure d'évaluation permet de déterminer les performances des stratégies d'apprentissages pour chaque instance (E). Finalement, l'information issue de la génération de méta-caractéristiques et de l'évaluation de performance est intégrée sous forme de paires au sein d'une base de méta-connaissance (F). Cette dernière est le résultat le plus important de la phase d'acquisition de connaissances qui reflète l'expérience accumulée à travers toutes les tâches considérées.

4.2.2.2 Mode consultatif

La méta-connaissance acquise dans la phase précédente permet l'exploitation des caractéristiques des nouvelles instances en mode consultatif (voir figure 4.3). En effet, les méta-caractéristiques (B) du nouveau jeu de données (A) sont appariées avec la base de méta-connaissances (F) afin de produire une recommandation quant à la meilleure stratégie d'apprentissage disponible. A ce niveau, l'apport du méta-apprentissage consiste à passer d'un apprentissage de base conventionnel vers un modèle éclairé de sélection ou de combinaison de techniques d'apprentissage (C). La performance de ce modèle évolue proportionnellement au nombre de jeux de données d'entraînement.

L'appariement décrit ci-dessus est réalisé par une fonction de correspondance des méta-caractéristiques vers les stratégies d'apprentissage. Cette fonction est le résultat de l'entraînement de l'algorithme de méta-apprentissage (meta-learner) sur la base de connaissance. Le classifieur ou combinaison de classifieurs (D) retenus pour résoudre le

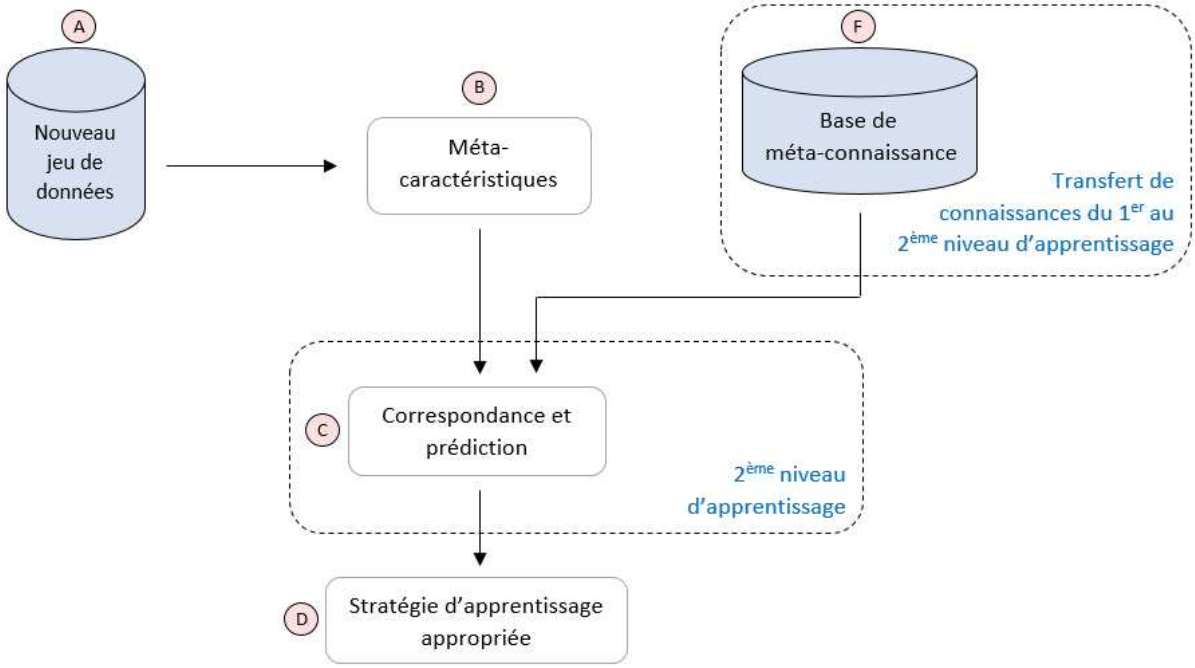

FIGURE 4.3 – Mode consultatif.

nouveau problème à traiter sont élus pour effectuer cette tâche sur la base des connaissances passées obtenues dans la première phase d'acquisition. Le méta-apprentissage permet ainsi l'instauration d'un pouvoir de sélection dynamique de la stratégie d'apprentissage se différenciant de la sorte d'une approche classique où le choix de stratégie est soit aléatoire ou issue d'une phase d'essai-erreur coûteuse et subjective.

4.2.3 Formes d'application

Plusieurs formes de méta-apprentissage ont été développées afin d'automatiser la sélection, le paramétrage ou la combinaison d'algorithmes d'apprentissage (Vilalta et al., 2004; Anderson & Oates, 2007). Ces tâches se différencient essentiellement par la manière de constitution de la méta-connaissance :

- Sélection d'algorithme.

  Cette approche fait partie de la problématique générale de l'ASP qui a été décrite pour la première fois par Rice (1976). Elle est définie comme étant l'apprentissage de règles de correspondance entre l'espace de caractéristiques des problèmes et l'espace de performance d'algorithmes. Cela consiste à prédire, à partir d'un sous-ensemble de l'espace d'algorithmes, celui qui est susceptible de produire la meilleure performance en fonction des caractéristiques mesurables d'une collection d'instances de l'espace de problèmes. Formellement, considérons les espaces de problèmes P, d'algorithmes A, de performances Y et de caractéristiques F et la fonction de caractérisation de P vers F associant à chaque $x \in P$ l'ensemble de ses caractéristiques $f(x)$ dans F. Résoudre l'ASP revient à déterminer la fonction de sélection $s(f(x))$ de F vers A, de telle sorte que l'algorithme sélectionné $a \in A$ maximise la mesure de performance $y(a, x) \in Y$. La figure 4.4 présente une modélisation de l'ASP.

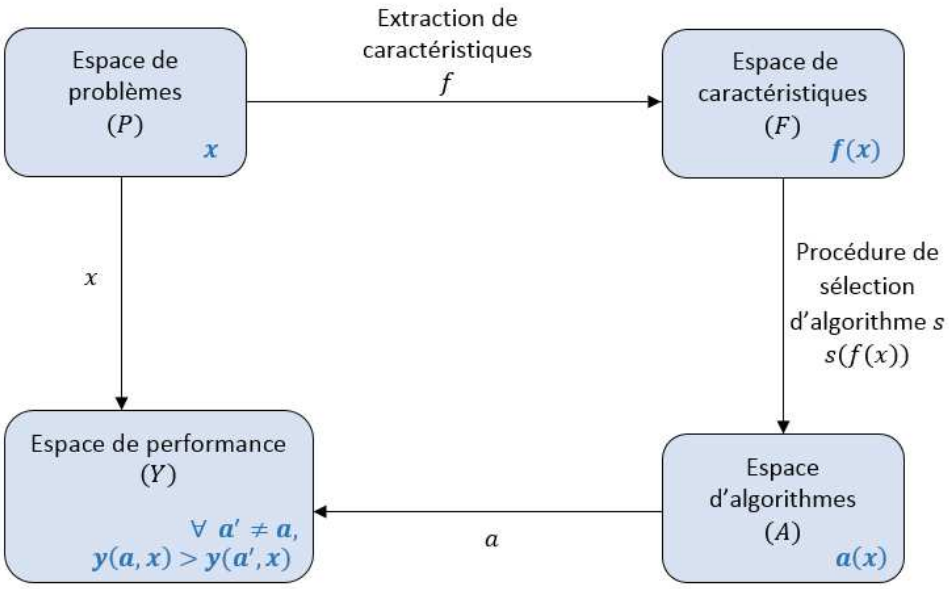

FIGURE 4.4 – Modélisation du problème de sélection d'algorithme (Rice, 1976).

- Paramétrage d'algorithme.

  Généralement, les algorithmes d'apprentissage nécessitent l'ajustement d'une multitude de paramètres qui influencent leur performance et qui sont strictement corrélés à la nature des données à traiter (Molina et al., 2012). Étant donné la complexité de cette tâche même pour des experts, plusieurs solutions ont été proposées telles que la réduction du nombre de paramètres, la présence de valeurs par défaut et le développement d'algorithmes zéro-paramètres. Toutefois, l'automatisation de la tâche de réglage est la solution qui a suscité le plus d'intérêt (Gomes et al., 2012). Dans ce cadre, le méta-apprentissage, qui a été conçu initialement pour répondre à la problématique de sélection d'algorithmes, a été adapté à la sélection des paramètres d'algorithmes d'apprentissage en constituant des bases de méta-connaissance composées par les vecteurs de méta-caractéristiques de chaque jeu de données d'entraînement appariés avec la configuration de paramétrage optimale (Wolpert, 1992).

- Combinaison d'algorithmes.

  Dans cette approche, le méta-apprentissage consiste à apprendre à partir de la combinaison des prédictions des apprenants de base. L'une des méthodologies de combinaison les plus reportées dans la littérature est celle de stacked generalization (généralisation empilée) initiée par les travaux de Wolpert (1992). Il s'agit d'exploiter, au niveau du méta-apprentissage, la variabilité des résultats produits par l'application d'un ensemble d'algorithmes de base à un même jeu de données. La stratégie de méta-apprentissage conventionnelle de sélection du meilleur candidat est substituée par l'apprentissage des corrélations entre les biais des apprenants de base permettant de surpasser leurs performances individuelles (Fan et al., 1996). A la suite de l'application des apprenants sur un jeu de données (à travers une procédure de validation croisée par exemple), leurs prédictions sont intégrées dans la représentation des caractéristiques initiale afin de construire la nouvelle base d'entraînement du méta-apprenant (voir

figure 4.5).

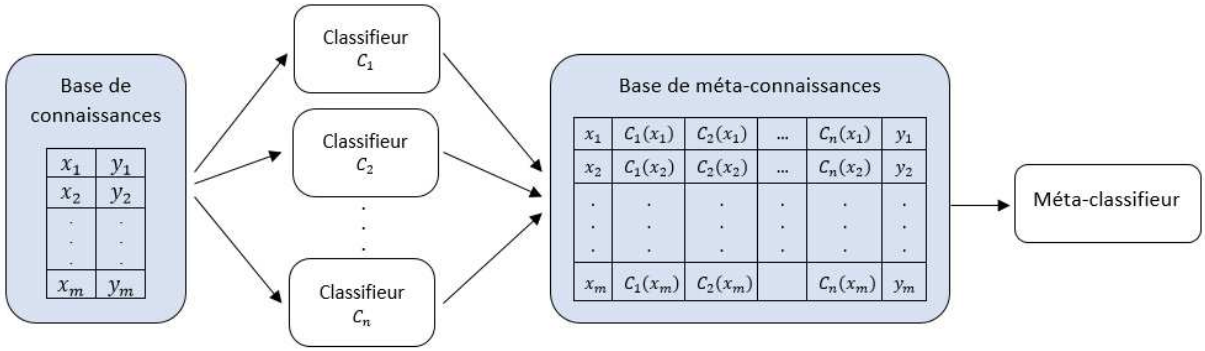

FIGURE 4.5 – Approche de stacked generalization illustrée à travers une tâche de classification.

4.2.4 Contraintes d'application

Les utilisateurs de systèmes prédictifs sont confrontés à un choix difficile parmi un nombre croissant de solutions techniques. Le méta-apprentissage peut aider à contrôler le coût d'expérimentation en fournissant des conseils dynamiques pour réduire l'effort qu'il faut consacrer à la sélection et au réglage d'algorithmes et promouvoir ainsi l'apprentissage automatique en dehors du milieu de la recherche. Toutefois, le méta-apprentissage classique a rencontré certains obstacles qui ont freiné son plein essor. Tout d'abord on retrouve le coût de réalisation de la méta-connaissance qui décrit les performances de l'ensemble d'algorithmes d'apprentissage explorés sur plusieurs jeux de données (Gama & Brazdil, 1995). Afin de construire cette méta-connaissance, il faut donc passer par un processus coûteux d'entraînement, test et évaluation unitaires. Ceci pose un véritable problème pour les tâches d'apprentissage à grande échelle où le temps d'exécution est aussi important que la performance.

D'un autre côté, plusieurs variables de l'architecture d'un système de méta-apprentissage sont considérés dans la littérature comme un problème de « black art » dans le sens où il n'y a pas de recommandations spécifiques sur le sujet. Ce problème concerne principalement la sélection d'algorithmes pour les deux niveaux d'apprentissage (Caffé et al., 2012). En effet, le problème avec la recommandation d'un candidat particulier est que le portefeuille éligible est potentiellement infini. Le traitement est encore plus complexe si on prend en compte les paramètres qui, généralement, impactent sensiblement les performances du même algorithme sur différents jeux de données. Certains auteurs proposent même de considérer des versions d'un algorithme avec plusieurs réglages de paramètres comme différents candidats à la méta-sélection (Gomes et al., 2012).

Le défi majeur entravant l'application de cette approche de méta-apprentissage est la difficulté du choix des méta-caractéristiques. Identifiées dans la littérature comme un problème récurrent du domaine (Giraud-Carrier et al., 2004), ces dernières permettent l'estimation de la similarité des tâches et constituent ainsi un facteur dont dépend fortement la performance du méta-niveau (Kalousis & Hilario, 2001). Un critère utilisé doit être suffisamment discriminant et informatif sur les différents aspects de la tâche d'apprentissage afin d'induire un pouvoir prédictif efficace au système de méta-apprentissage. Dans plusieurs

travaux, on ne trouve pas une justification claire sur les méta-caractéristiques adoptées ce qui rend difficile la définition de caractéristiques réellement pertinentes et utiles pour des nouvelles applications. Cela revient probablement à l'absence d'études systématiques exclusivement dédiées à la discussion de ce thème.

4.3 Méta-apprentissage par réseau de neurones

Comparativement aux premières approches tabulaires orientées caractéristiques, où la classe d'algorithmes pouvant être méta-appris est souvent restreinte à l'apprentissage supervisé, la formulation moderne du méta-apprentissage a largement tiré parti des avantages des DNNs, qui sont expressifs, évolutifs et susceptibles d'optimisation. Ainsi, après l'émergence du DRL, c'est au tour du méta-apprentissage d'être métamorphosé par les apports de l'approximation universelle que représente l'apprentissage profond. Dans cette section nous explorons comment la recherche est passée du méta-apprentissage classique issu des théories de sélection d'algorithmes et No Free Lunch, vers le méta-apprentissage par réseaux de neurones orienté davantage aux tâches de grande dimension et le « few shot learning » (FSL) où les métadonnées sont issues de la structure et des paramètres des modèles d'apprentissage eux-mêmes. Nous présentons 3 approches du périmètre de méta-apprentissage profond qui sont les approches métriques, récurrentes et par optimiseur. Nous nous intéressons à cette dernière et plus spécifiquement à sa variante basée sur le méta-gradient qui a été introduite avec le Model-Agnostic Meta-Learning (MAML) (Finn et al., 2017). En effet, elle offre plusieurs avantages qui la rendent compatible avec les configurations RL et donc appropriée à notre approche MRL pour le contrôle.

4.3.1 Contexte

Le regain d'intérêt récent aux réseaux de neurones artificiels et plus particulièrement l'apprentissage profond supporté par l'avènement d'importantes bases de données d'entraînement et de ressources informatiques puissantes a permis la résurgence du méta-apprentissage par réseaux de neurones (ou profond) (Li et al., 2018). La vocation initiale de cette catégorie de méta-apprentissage est le traitement de problèmes d'apprentissage supervisé dans le cadre d'une perspective FSL. En effet, le succès de l'apprentissage en général est indissociable de la disponibilité d'une grande quantité de données labellisées. Bien que les DNNs réalisent, en tant qu'apprenants de base, de grandes performances dans ce régime Big Data, il y a un intérêt croissant pour réduire la quantité de données requises.

Plusieurs domaines d'application peuvent exprimer ce besoin comme par exemple les grands détaillants de commerce électronique (Amazon) et les systèmes de recommandation personnalisés (Netflix). Dans ces cas bien précis, il n'est pas pratique de collecter des millions d'exemples labellisés pour identifier des nouvelles catégories ou profils qui s'ajoutent par milliers en des laps de temps courts. Il est essentiel pour la survie et la compétitivité de ces secteurs de pouvoir s'adapter aux caractéristiques d'un nouveau profil dès le début et en utilisant très peu d'échantillons. Dans ce type de problématiques, il est nécessaire d'éviter les classifieurs à biais statiques et recourir plutôt à un algorithme qui puisse s'entraîner dynamiquement et se comporter plus efficacement que les systèmes d'apprentissage standards. Un tel algorithme doit incorporer transversalement des informations spécifiques

à différents domaines, de sorte qu'il n'ait pas besoin de s'appuyer sur de nouvelles données pour exploiter ces connaissances.

Le méta-apprentissage, qui apprend un algorithme d'apprentissage à partir de données, est une approche prometteuse pour atteindre cet objectif. En s'entraînant sur un grand nombre de tâches connexes, où chaque tâche ne peut avoir qu'une petite quantité de données labellisées, le méta-apprentissage produit automatiquement un algorithme adaptatif au contexte qui capture les connaissances communes parmi les tâches d'entraînement et peut être utilisé pour résoudre rapidement des nouvelles tâches. Les réseaux de neurones sont particulièrement adaptés à ce rôle de méta-apprenant par transfert de connaissance du fait de leurs capacités internes d'abstraction de caractéristiques de données et d'inductions de règles reflétées dans leurs poids et biais de connexion. Inspiré par les progrès récents du méta-apprentissage pour l'apprentissage supervisé, il est naturel d'envisager des techniques similaires pour le RL, afin d'établir des politiques de contrôle encore plus robustes et généralisables que celles générées par le DRL standard.

4.3.2 Approche d'optimiseur

Une des approches de méta-apprentissage les plus convoitées dans la littérature actuellement est la recherche de représentations internes qui peuvent être ajustées facilement pour de nouvelles tâches. Concrètement, il s'agit d'optimiser les paramètres initiaux d'un modèle de manière à le rendre généralisable à différents contextes à la suite d'une adaptation par réglage fin de ces paramètres. La possibilité d'une instanciation simple et efficace de telles méthodes à des configurations RL sont parmi les raisons pour lesquelles nous suivons cette démarche dans le développement de notre approche MRL pour le contrôle d'un système de conduite dans un milieu urbain.

4.3.2.1 Fondement

Dans le contexte d'apprentissage profond, une représentation interne est un modèle d'apprentissage qui se manifeste sous la forme de paramètres d'un réseau de neurones. Ce modèle est considéré efficace dans un régime de FSL s'il est généralisable à de nouvelles situations suite à une reconfiguration de paramètres basée sur un petit échantillon d'entraînement. Techniquement, on qualifie ces approches de méta-apprentissage par optimiseur vu qu'elles incluent généralement un méta-apprenant (optimiseur) pour rechercher les paramètres du réseau de neurones constituant le modèle généralisable. Une grande partie de ces approches se focalisent sur l'apprentissage et l'amélioration des méthodes de descente de gradient. S'apparentant au concept « apprendre ce que nous apprenons », le méta-apprentissage par optimiseur collecte et fusionne la connaissance acquise à travers un batch de tâches avant de l'intégrer dans le modèle final. Ce dernier se distingue de l'apprentissage conventionnel par retro-propagation qui requiert un grand volume de données d'entraînement, par sa capacité de converger rapidement à partir de quelques exemples seulement. Formellement, nous considérons les ensembles suivants :

- Les tâches antérieures $T_i \in T$.
- Les modèles d'apprentissage $l_i \in L$ appris sur les tâches T_i et définis par leur configuration (paramètres, architecture, etc.) $\theta_i \in \Theta$.

- Les performances $P_i = P(\theta_i, T_i) \in P$, des modèles l_i sur les tâches T_i, reflétant l'évaluation de la configuration θ_i selon une mesure prédéfinie (taux de précision, récompense, perte, etc.).

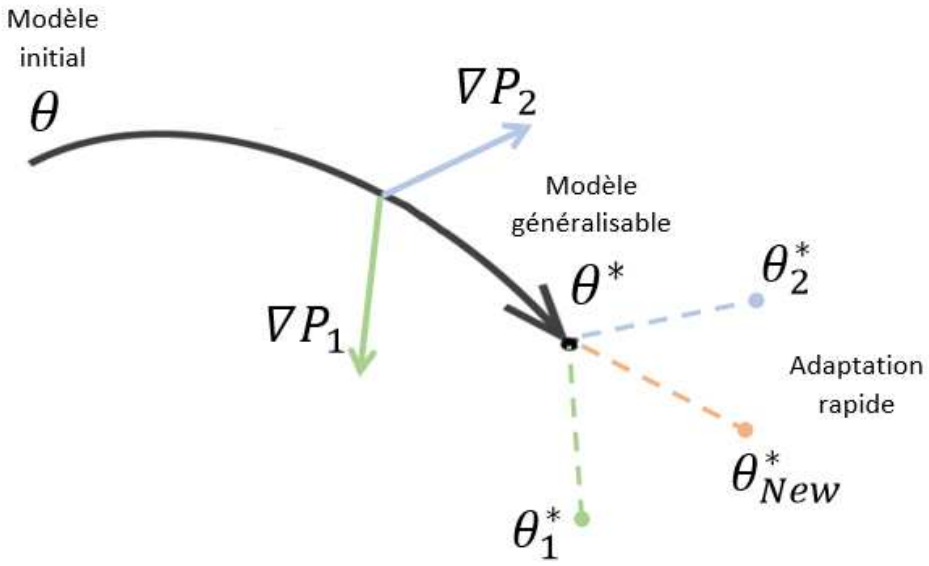

FIGURE 4.6 – Apprentissage d'un modèle d'initialisation θ^* capable de s'adapter rapidement aux nouvelles tâches.

L'objectif du méta-apprentissage par optimiseur est d'entraîner un méta-apprenant à travers les métadonnées (l_i, P_i) afin de générer un modèle d'initialisation l^* de paramétrage θ^* constituant un point de départ efficace pour traiter une nouvelle tâche T_{new}. Cela implique que le modèle obtenu est sensible aux modifications induites par T_{new} et que de petits changements au niveau des paramètres θ^* entraîneront de grandes améliorations sur la fonction de performance. La stratégie de méta-apprentissage par optimiseur est schématisée dans la figure 4.6. D'un point de vue architecture, nous définissons l'opérateur de méta-apprentissage $g(f(l_i))$ composite de deux opérateurs d'apprentissage g et f. Le méta-apprentissage consiste alors à établir une outer loop $g : L \rightarrow L$ pour apprendre la règle d'apprentissage f et une inner loop $f : L \rightarrow L$ pour apprendre le modèle l_i sur une tâche donnée T_i. Au cours du méta-entraînement, la inner loop génère l'ensemble des métadonnées (l_i, P_i) qui sera transféré à la outer loop pour en déduire le modèle d'initialisation l^*.

4.3.2.2 Model-Agnostic Meta-Learning (MAML)

Dans les approches de méta-apprentissage par optimiseur, nous nous intéressons aux méthodes dont l'application pourrait s'étendre au RL et utilisant les gradients pour effectuer le méta-apprentissage. La stratégie basée sur le gradient a été introduite à l'origine par Finn et al. (2017) avec leur algorithme Model-Agnostic Meta-Learning (MAML). Le MAML vise principalement à générer une initialisation de modèles sensible aux changements et à obtenir des résultats optimaux sur un nouveau scénario après seulement quelques mises à jour de gradient.

L'approche proposée est simple et présente de nombreux avantages. Finn & Levine (2018) démontrent que MAML est une procédure d'apprentissage universelle dans le sens où elle est capable d'approximer n'importe quel algorithme. Elle ne fait aucune hypothèse sur la forme du modèle, ainsi elle est indépendante de la tâche traitée, adaptable à tout modèle entraîné par descente de gradient et applicable à une variété de problèmes d'apprentissage différents, y compris la classification, la régression et le RL. D'un autre côté, le méta-niveau du MAML n'engendre aucun paramètre supplémentaire par rapport à l'apprenant de base. Enfin, l'optimiseur introduit, qui est la SGD, est bien connu et étudié dans la littérature permettant d'apporter plus facilement des évolutions et des améliorations à l'approche de référence.

4.3.2.3 Modèles dérivés du MAML

Les avantages cités ci-dessus ont donné lieu à plusieurs méthodes issues du MAML. Meta-SGD (Li et al., 2017) utilise une descente de gradient stochastique pour méta-apprendre, en plus d'une initialisation de modèle, le taux d'apprentissage de la inner loop et le sens de mise à jour des paramètres de la politique. Dans Reptile (Nichol et al., 2018), les auteurs conçoivent une version de premier ordre du MAML qui, sur le plan de calcul, s'avère moins coûteuse que la méthode d'origine incluant une dérivée du gradient de deuxième ordre. Dans la même perspective d'optimisation de la performance, Zintgraf et al. (2019) développent la méthode CAVIA (pour Fast context adaptation via meta-learning) dans laquelle les paramètres θ sont partitionnés en spécifiques (de contexte) et indépendants (partagés) de la tâche. Au niveau de la inner loop, seulement les paramètres de contexte sont adaptés aux tâches individuelles, alors qu'au niveau de la outer loop ce sont les « partagés » qui sont méta-entraînés à travers les tâches. Shedivat et al. (2018) proposent une vue probabiliste du MAML pour une adaptation continue des paramètres du RL. Au niveau de l'évaluation de l'approche, un environnement multi-agents compétitif (RoboSumo) a été conçu pour exécuter des jeux d'adaptation itérative.

Nous présentons dans le tableau 4.1 une revue des principales approches de méta-apprentissage par optimiseurs de type gradient. Afin de standardiser l'écriture, le modèle l_i est directement exprimé par sa configuration de paramètres θ_i. La mesure de performance P_i est reflétée par la fonction de perte $\mathcal{L}_i(\theta)$.

Méthode	Inner loop	Outer loop	Tâche
MAML (Finn et al., 2017)	Apprentissage par descente de gradient des tâches individuelles et mise à jour des paramètres du modèle : $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_i(\theta)$. Évaluation des modèles individuels pour obtenir l'échantillon de pertes $\mathcal{L}_i(\theta_i')$.	Méta-apprentissage par descente de gradient des performances de l'ensemble des tâches pour générer le modèle d'initialisation : $\theta^* = \theta - \beta \nabla_\theta \sum \mathcal{L}_i(\theta_i')$.	Régression d'un signal sinusoïdal. Classification sur Omniglot. RL pour navigation 2D.

Meta-SGD (Li et al., 2017)	Le taux d'apprentissage α n'est pas un scalaire fixe, mais un vecteur de paramètres à apprendre en plus des paramètres θ du modèle. La descente de gradient s'écrit alors : $\theta_i^{'} = \theta - \alpha \circ \nabla_\theta \mathcal{L}_i(\theta)$. où $\alpha \circ \nabla_\theta \mathcal{L}_i(\theta)$ est appelé terme d'adaptation.	Méta-apprentissage par descente de gradient des performances de l'ensemble des tâches pour générer une initialisation des paramètres du modèle et du taux d'apprentissage : $\theta^* = \theta - \beta \nabla_\theta \sum \mathcal{L}_i(\theta_i^{'})$ $\alpha^* = \alpha - \beta \nabla_\alpha \sum \mathcal{L}_i(\theta_i^{'})$	Régression d'un signal sinusoïdal. Classification sur Omniglot. RL pour navigation 2D.
Reptile (Nichol et al., 2018)	Extension de la version de 1^{er} ordre de MAML. Apprentissage par descente de gradient des tâches individuelles et mise à jour des paramètres du modèle : $\theta_i^{'} = \theta - \alpha \nabla_\theta \mathcal{L}_i(\theta)$.	Remplacer la méta-descente de gradient (dérivée de $2^{ème}$ ordre) par une minimisation de la distance euclidienne entre les paramètres d'initialisation recherchés et ceux issus de la inner loop. $\theta^* = \theta + \varepsilon \sum(\theta_i^{'} - \theta)$	Classification sur Omniglot et MiniImageNet.
CAVIA (Zintgraf et al., 2019)	Partitionner les paramètres du modèle en 2 groupes : ϕ et θ présentent respectivement les paramètres de contexte et les paramètres partagés. Dans la inner loop, l'apprentissage des tâches individuelles porte seulement sur la mise à jour des paramètres de contexte : $\phi_i^{'} = \phi - \alpha \nabla_\phi \mathcal{L}_i(\phi, \theta)$. Calculer l'échantillon de pertes $\mathcal{L}_i(\phi_i^{'}, \theta)$.	Méta-apprentissage des paramètres partagés par descente de gradient : $\theta^* = \theta - \beta \nabla_\theta \sum \mathcal{L}_i(\phi_i^{'}, \theta)$. Avantage de la méthode CAVIA : éviter le sur-apprentissage du modèle d'initialisation sur les tâches individuelles.	Régression d'un signal sinusoïdal. Classification sur MiniImageNet. RL pour locomotion dans MuJoCo.
Robosumo (Shedivat et al., 2018)	Une vue probabiliste du MAML destinée aux configurations RL. Les tâches sont définies selon les dynamiques Markoviennes $\mathcal{P}(x)$ et $\mathcal{P}(x_{t+1}\|x_t, a_t)$ en fonction des actions (a) et des états (x) à un instant donné (t). Le principe d'optimisation au niveau de la inner et la outer loop suit la même logique que MAML .		RL pour des compétitions multi-agents.

TABLE 4.1 – Revue des approches de méta-apprentissage par optimiseur gradient.

4.3.3 Approche métrique

Dans un contexte d'apprentissage supervisé, il serait naturel de comparer un objet à traiter aux exemples labellisés disponibles, d'en tirer celui qui lui ressemble le plus et de lui affecter sa classe comme étiquette. C'est l'idée derrière les approches métriques ou basées sur la similarité. Elles ont pour objectif de déterminer un espace métrique dans lequel l'apprentissage est particulièrement efficace en établissant une fonction de distance entre les données traitées. Le schéma de comparaison entre les instances de test non labellisées (query set) et les exemples d'entraînement labellisés (support set) est assuré par des fonctions de type similarité cosinus, distance l_1, l_2, etc.

Le méta-apprentissage métrique est effectué à l'aide de réseaux de neurones optimisés par descente de gradient. Cette approche s'est avérée très performante pour la résolution de tâches de classification en régime FSL, cependant elle n'a pas été appliquée à d'autres domaines tels que la régression et le RL.

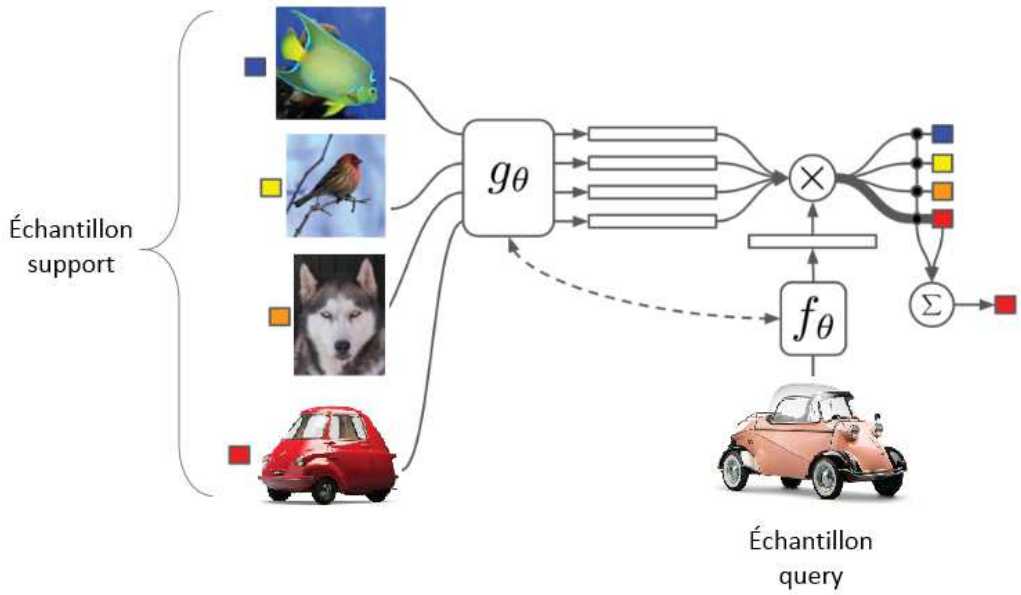

FIGURE 4.7 – Cas de classification par approche métrique adapté de (Vinyals et al., 2016).

D'une manière générale, on considère deux fonctions f et g encodant respectivement les échantillons query (x^{test}) et support (x^i) vers un espace Z et une mesure S utilisée pour calculer la similarité entre $f(x^{test})$ et $g(x^i)$ pour chaque x^i. L'instance x^{test} est assignée à la classe de l'instance x^i la plus similaire. Dans la littérature, les fonctions f et g peuvent être identiques ou différentes selon l'architecture adoptée. La figure 4.7 montre une illustration de la stratégie décrite ci-dessus. Nous présentons également dans le tableau 4.2 les détails des approches métriques les plus citées dans la littérature.

Méthode	f	g	S	Tâche
Réseaux de neurones siamois (Koch et al., 2015) : Une architecture siamoise constituée de deux CNN jumeaux partageant les mêmes paramètres et reliés par une fonction de distance. Elle permet d'apprendre une mesure de similarité à partir de deux entrées indépendantes.	CNN	CNN	Distance l_1 pondérée	Classification sur la base de données Omniglot.
Réseaux d'appariement (Vinyals et al., 2016) : Cette méthode se distingue par la prise en compte de tout l'échantillon support lors de l'encodage de chaque x^i vers l'espace Z, d'où l'utilisation d'un LSTM. Ainsi la fonction g se présente sous la forme $g(x^i, X)$ où X présente l'échantillon support.	CNN	LSTM bi-directionnel	Similarité cosinus	Classification sur Omniglot, ImageNet et Penn Treebank.
Réseaux prototypiques (Snell et al., 2017) : L'apport de cette méthode est la représentation de chaque classe par son cluster (prototype) c_k. Ainsi la distance d'une instance query est calculée par rapport aux prototypes de chaque classe et non pas aux instances de l'échantillon support. L'objectif est d'éviter le sur-apprentissage (overfitting).	CNN	CNN	Carré distance euclidienne (l_2)	Classification sur Omniglot et miniImageNet.
Réseaux de relations (Sung et al., 2017) : Dans cette méthode, la métrique de distance n'est plus fixe mais apprise par un module de relation. Ainsi, les encodages par CNN (cartes de caractéristiques) sont concaténés sous la forme $C(f(x^{test}), g(x^i))$ puis transmis à un module de relation (CNN) afin d'apprendre la similarité entre les instances query et support.	CNN	CNN	Similarité apprise par un CNN.	Classification réalisée sur Omniglot et miniImageNet.

Table 4.2 – Comparaison d'approches métriques.

4.3.4 Modèle d'apprentissage récurrent

Le méta-apprentissage peut être formalisé comme un problème de séquence-à-séquence en adoptant un méta-contrôleur de type RNN (Elman, 1990) capable d'internaliser et de se référer à l'expérience passée afin de générer de nouveaux candidats pour le processus de recherche. L'optimiseur RNN apprend à utiliser une mémoire pour stocker des informations sur les requêtes et évaluations de fonctions précédentes, et apprend également à accéder à cette mémoire pour prendre des décisions sur les parties du domaine à explorer par la suite. Entraîné d'une manière itérative par descente de gradient, le RNN utilise la séquence de requêtes passées et les réponses correspondantes pour proposer la requête suivante maximisant l'amélioration observée de ces réponses. Dans sa forme la plus basique illustrée dans la figure 4.8, un RNN est constitué d'une couche cachée qui à l'instant t dépend de l'entrée x_t, mais aussi de la sortie de la même couche cachée à l'instant $t-1$.

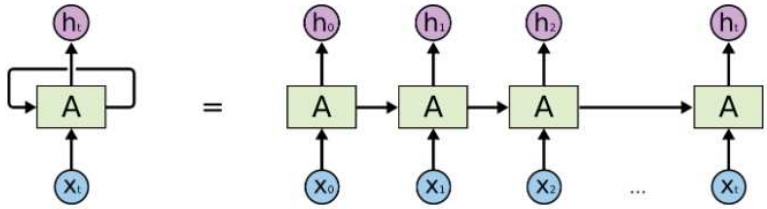

FIGURE 4.8 – A gauche un exemple d'une structure RNN A traitant une entrée x_t et générant une valeur h_t. A droite, sa représentation dépliée. [2].

Au cours des dernières années, le RNN a été utilisé avec succès pour diverses applications dont essentiellement le traitement automatique du langage naturel (NLP) (Graves et al., 2013). Ce succès est principalement dû aux performances des cellules de mémoire implémentées dans plusieurs variantes du RNN. En général, la mémoire peut être intégrée dans un modèle de réseau de neurones de deux manières : elle peut être interne ou externe. Dans la première configuration, la mémoire consiste en une couche de neurones enrichie de dynamiques spécifiques qui permettent le maintien des représentations internes. Par exemple, c'est le cas du LSTM (Long Short-Term Memory) introduit par Hochreiter & Schmidhuber (1997) qui adopte un mécanisme de déclenchement de mémoire : l'unité neuronale est en fait équipée de variables de déclenchement utilisées pour décider quand stocker le nouveau stimulus ou effacer le contenu de la mémoire actuelle.

Dans le deuxième cas, les réseaux de neurones augmentés de mémoire (MANN pour memory augmented neural networks) utilisent une matrice de mémoire externe, où les informations pertinentes peuvent être écrites et extraites à l'aide de têtes d'écriture et de lecture qui adressent des emplacements de mémoire spécifiques. La mémoire est dite «externe» car elle n'est pas strictement intégrée dans la voie de traitement du signal n'ayant pas de connexions directes avec l'entrée ou la sortie du réseau. Plus formellement, lors de l'entraînement, le méta-contrôleur g reçoit des données d'entrée à partir d'une fonction à apprendre f et produit un ensemble de vecteurs d'interaction, appelés clés de lecture, qui sont transmis à la fonction mémoire M. Celle-ci effectue une comparaison de cet output avec les exemples encodés à sa disposition et émet une valeur de lecture en

2. https ://colah.github.io/posts/2015-08-Understanding-LSTMs/

réponse. Le contrôleur procède à l'évaluation de la réponse reçue et en déduit une mise à jour de ses paramètres et de la mémoire à travers les clés d'écriture. Enfin, en fonction de son état interne (paramètres) et de la lecture de la mémoire, le contrôleur produit un vecteur de sortie émis vers la fonction f. Ce processus de méta-apprentissage séquentiel est itéré sur toutes les instances de la base d'entraînement.

Les modèles de méta-apprentissage récurrent reprennent ainsi une spécificité des approches métriques en utilisant parfois les mêmes techniques d'appariement pour retrouver l'information pertinente au niveau de leur mémoire. D'un autre côté, cette approche emprunte le mode de fonctionnement global de l'apprentissage par optimiseur dans le sens où un méta-apprenant entraîne un RNN pour optimiser les paramètres d'un modèle de base. Ce qui distingue le modèle récurrent, c'est la manière de traitement séquentiel des données et la procédure d'écriture et de lecture à partir de sa mémoire. En effet, l'utilisation de RNN pour le méta-apprentissage nécessite le recours à des mémoires capables d'encoder et capturer rapidement les informations sur les nouvelles tâches et, en même temps, offrir un accès facile et stable aux représentations stockées. Il en découle que les principales méthodes de méta-apprentissage récurrent se différencient par les spécificités de la mémoire introduite dans le processus d'apprentissage. Les traitements de mémoires effectués par les MANNs ne rentrent pas dans le cadre de notre recherche, toutefois il est possible d'avoir plus de détails sur ce thème dans les travaux résumés au niveau du tableau 4.3.

Méthode	Contrôleur	Mémoire	Tâche
MANN (Santoro et al., 2016)	LSTM : les entités d'entraînement x_t sont présentées comme input en décalage temporel avec les valeurs cibles $f(x_{t-1})$ afin que le contrôleur apprenne à conserver l'input x_t en mémoire en attendant que la valeur correcte $f(x_t)$ soit présentée ultérieurement.	Lecture mémoire : comparaison de la clé de lecture produite par le contrôleur aux exemples encodés dans la mémoire en utilisant la similarité cosinus. Écriture mémoire : appliquer la procédure d'arbitrage LRUA (Least Recent Used Access) afin de décider de sauvegarder la clé d'écriture soit dans l'emplacement de mémoire le moins utilisé soit dans le dernier emplacement de mémoire utilisé.	Régression Classification
SNAIL (Mishra et al., 2017)	Convolution temporelle pour un traitement séquentiel de données doté d'une plus grande capacité d'accès à l'information passée par rapport à un RNN classique.	Mécanisme de self-attention (Vaswani et al., 2017) pour une lecture-écriture basée sur le contenu plus efficace. Il permet de contourner la nature séquentielle des modèles récurrents en offrant un accès parallèle aux données d'une même séquence, donc donner plus de performance aux traitements des positions distantes.	Classification RL

MetaNet (Munkhdalai & Yu, 2017)	LSTM : la spécificité de cette approche est l'intégration, lors de l'entraînement, de paramètres fast (appris par un autre réseau de neurone) au niveau de la fonction de base f en plus de ses paramètres slow calculés par descente de gradient. L'objectif est d'accélérer la convergence.	La mémoire est utilisée pour sauvegarder les paramètres fast calculés au niveau méta à partir des gradients de perte générés par la fonction de base f. Lecture mémoire : comparaison de la clé de lecture aux exemples encodés dans la mémoire en utilisant la similarité cosinus.	Classification

TABLE 4.3 – Revue d'approches de méta-apprentissage récurrent.

4.4 Recherche d'architecture neuronale (NAS)

Nous entamons dans cette section une dernière approche de méta-apprentissage fondée sur les réseaux de neurones artificiels qui tire ses spécificités du domaine d'application plutôt que de la méthodologie de construction du système de méta-apprentissage lui-même. Il s'agit de la conception automatique d'architecture de CNN qui est devenue récemment une branche de recherche active connue sous le nom de NAS. L'intérêt pour nous en abordant ce thème, sur lequel nous avons publié une revue de la littérature (Jaafra et al., 2019c), est de tirer profit des outputs des différents travaux sur le sujet afin de mieux cerner les exigences et les contraintes inhérentes à l'application du méta-apprentissage aux tâches complexes du monde réel. D'un autre côté, cela permettra d'ouvrir des horizons pour certaines problématiques reliées à notre thème d'étude dont notamment les techniques d'accélération d'apprentissage qui faciliteront un accès équitable à la technologie d'apprentissage profond sans les conditions préalables actuelles de disponibilité de moyens de calculs hyperpuissants.

4.4.1 Contexte

La performance d'un réseau de neurones et notamment d'un CNN dépend principalement de la configuration de la structure du modèle, du processus d'entraînement et de la représentation des données. Toutes ces variables sont contrôlées par un certain nombre d'hyperparamètres et ont un impact important sur l'évolution de l'apprentissage. Afin d'optimiser les performances des CNNs, ces hyperparamètres incluant la profondeur du réseau, le taux d'apprentissage, le type de couche, le nombre d'unités par couche, le taux de dropout, etc., doivent être soigneusement sélectionnés. Toutefois, l'avènement d'architectures modernes plus profondes et plus complexes a augmenté considérablement le nombre et le type d'hyperparamètres à contrôler. L'étape de réglage de ces hyperparamètres et plus généralement, la conception d'architecture CNN est, par conséquent, devenue très coûteuse et lourde pour une procédure d'essai-erreur manuelle conduite par un expert.

D'un autre côté, la conception d'un CNN est considérée comme un problème d'optimisation de type boîte noire (Bengio et al., 2013) en raison de la nature inconnue de la correspondance reliant l'architecture, la performance et la tâche d'apprentissage. Dans ce contexte de complexité et d'absence de cadre théorique justifiant le choix de paramétrage, les solutions de conception automatique de CNN sont largement requises par les utilisateurs finaux et suscitent un grand volume de recherche. En réponse à ce besoin, plusieurs méthodes de méta-modélisation ont été développées et consistent essentiellement à appliquer des techniques d'apprentissage automatique à la conception d'architectures CNN. La méta-modélisation est décrite plus en détail dans les paragraphes suivants en fonction du concept d'apprentissage utilisé pour l'optimisation.

4.4.2 Architectures de CNN modernes

Nous présentons dans ce qui suit une revue des architectures de CNN conçues manuellement (hand-crafted) qui ont eu un impact important sur les travaux récents de conception d'architecture automatique. La plupart de ces travaux ont remporté au moins un des défis de « ImageNet Large Scale Visual Recognition Competition » (ILSVRC) (Russakovsky et al., 2015). Cela permet d'établir le lien entre les apports de ces modèles manuels modernes et les orientations d'évolution de la NAS.

4.4.2.1 LeNet

LeNet (Lecun et al., 1998) constitue le travail initiateur des réseaux à convolution. Il a été expérimenté avec succès pour classifier des chiffres manuscrits sans aucun traitement préalable de l'image d'entrée. L'architecture de LeNet est illustrée dans la figure 4.9. Elle comprend une couche d'entrée et une couche de sortie de tailles respectives $32 * 32$ et 10, ainsi que 6 couches cachées. L'idée de base de cette conception est de faire fonctionner plusieurs convolutions (3) avec du pooling intermédiaire (2), puis de transmettre le signal final via une couche entièrement connectée vers la couche de sortie. Malheureusement, faute de données d'entraînement et de puissance de calcul adéquates, il n'a pas été possible d'étendre l'application de cette architecture vers des tâches plus complexes.

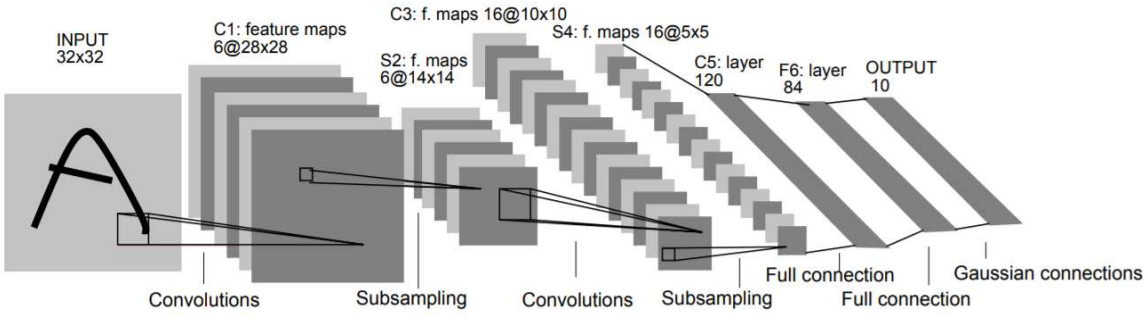

FIGURE 4.9 – Architecture LeNet (Lecun et al., 1998).

4.4.2.2 AlexNet

AlexNet est l'un des CNN les plus influents qui a remporté les compétitions ILSVRC en 2012 (Krizhevsky et al., 2012). Il n'est pas très différent de LeNet, toutefois son architecture est relativement plus profonde avec 8 couches au total, 5 convolutions et 3 entièrement connectées. La contribution effective d'AlexNet réside dans plusieurs particularités de conception et d'entraînement. Premièrement, il a introduit la fonction d'activation de l'unité de rectification linéaire (ReLU pour Rectified Linear Unit) qui a permis de surmonter le problème de la disparition du gradient et de dynamiser l'entraînement. De plus, AlexNet implémente une étape de dropout (Srivastava et al., 2014) qui consiste à mettre à zéro un pourcentage prédéfini de poids par couche. Cette technique diminue les paramètres appris et contrôle la corrélation entre les neurones afin de limiter l'impact du sur-apprentissage. Troisièmement, la convergence des processus d'entraînement est accélérée avec les techniques de l'élan (momentum) et de la réduction conditionnelle du taux d'apprentissage (par exemple, lorsque l'apprentissage stagne). Enfin, le volume de données d'apprentissage est augmenté artificiellement en générant des variations des images d'origine qui sont pivotées de manière aléatoire.

4.4.2.3 VGGNet

Soumis à ILSVRC 2014, VGGNet (Simonyan & Zisserman, 2015) a remporté la deuxième place et démontré que des architectures plus profondes donnent de meilleurs résultats. En effet, avec ses 19 couches cachées, il était beaucoup plus profond que les CNN précédents. Afin de permettre une augmentation de profondeur sans croissance exponentielle du nombre de paramètres, des filtres de convolution plus petits $(3 * 3)$ ont été utilisés dans toutes les couches (taille inférieure aux filtres de $11 * 11$ adoptés dans AlexNet). L'utilisation de filtres de petite taille présente également l'avantage de réduire le chevauchement des pixels balayés, ce qui produit des cartes de caractéristiques plus riches en détails (Zeiler & Fergus, 2014).

4.4.2.4 GoogLeNet

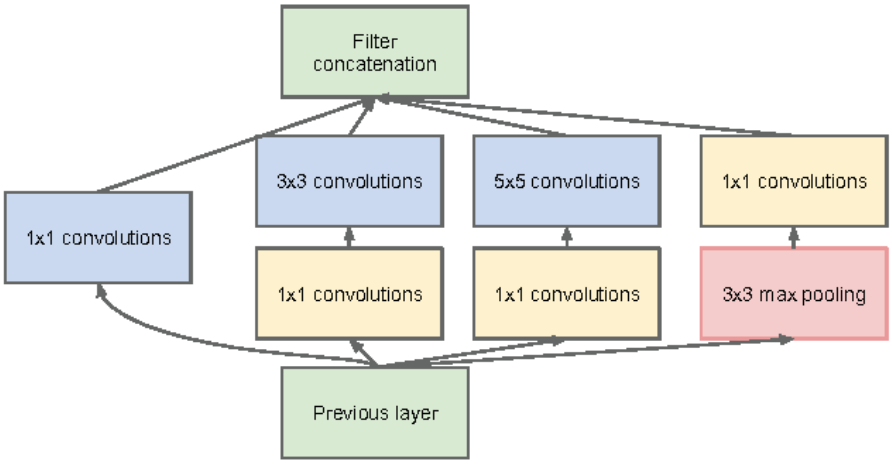

FIGURE 4.10 – L'architecture d'un module « inception » (Szegedy et al., 2015).

Depuis qu'il a été démontré que la taille d'une architecture de CNN est positivement corrélée à ses performances, les efforts se sont focalisés sur la manière d'augmenter la profondeur d'un CNN tout en conservant un nombre acceptable de paramètres. Gagnant de l'ILSVRC 2014, GoogLeNet (Szegedy et al., 2015) a innové en matière de conception de réseau en remplaçant la stratégie classique d'alternance de couches de convolution et de pooling avec des modules « inception » superposés illustrés dans la figure 4.10.

En dépit d'être plus profond que VGGNet avec 22 couches cachées, GoogLeNet nécessite remarquablement moins de paramètres en raison de cette technique de connexion parcimonieuse. Dans un module « inception », plusieurs convolutions à différentes échelles et pooling sont effectuées en parallèle puis concaténées en une seule couche. Cela permet au CNN de détecter des motifs de différentes tailles au sein de la même couche et d'éviter des redondances lourdes et inutiles de paramètres.

4.4.2.5 ResNet

Deep Residual Network (He et al., 2016b) a été le premier réseau de neurones à surpasser la performance humaine sur ImageNet Challenge (ILSVRC 2015). Grâce aux connexions résiduelles, ce type d'architecture est devenu plus profond et a été implémenté avec plusieurs versions de 34, 50, 101 et 152 couches. En effet, l'une des difficultés rencontrées lors de l'entraînement de réseaux très profonds réside dans la disparition du gradient lors de la retro-propagation des erreurs, ce qui pénalise la mise à jour appropriée des poids des couches précédentes.

La contribution principale de ResNet consiste à diviser les couches de convolution en blocs résiduels. Chaque bloc est contourné par une connexion résiduelle (skip) qui transmet l'entrée de bloc à l'aide d'une fonction identitaire. La sortie finale est la somme des sorties du bloc et de la connexion, comme illustré dans la figure 4.11. En ajoutant des connexions résiduelles, la rétro-propagation peut être exploitée sans aucune interférence avec les couches précédentes, ce qui permet d'éviter la disparition du gradient et de former des architectures encore plus profondes.

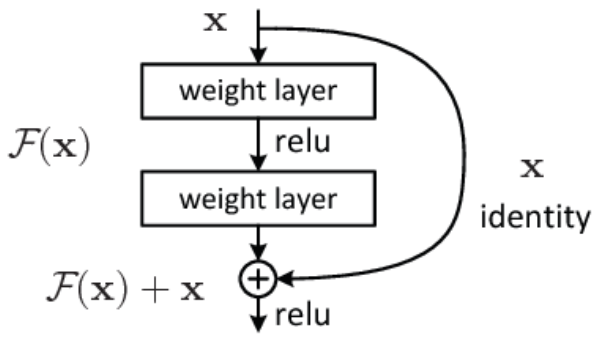

Figure 4.11 – Apprentissage résiduel : un exemple de bloc de construction (He et al., 2016b).

Après le succès de ResNet, les CNN « hand-crafted » modernes continuent à être conçus sur la base des modèles précédents recherchant plus d'efficacité et moins de coût d'entraînement. Inception-v4 (Szegedy et al., 2017) est une nouvelle version de GoogLeNet

qui implique beaucoup plus de couches que la version initiale. Inception-ResNet (Szegedy et al., 2017) est construit sous forme d'une combinaison de blocs « inception » et de connexions résiduelles. DenseNet (Huang et al., 2017), propose une extension des connexions résiduelles à toutes les couches ultérieures permettant ainsi l'apprentissage de nouvelles caractéristiques.

Les aboutissements des architectures CNN détaillées ci-dessus fournissent les éléments de base utilisés dans les approches de NAS qui seront décrites dans les paragraphes suivants. Ils offrent des alternatives de conception et des solutions efficaces aux problèmes les plus courants rencontrés par les algorithmes de CNN profonds.

4.4.3 Méta-modélisation

Pour traiter la tâche de NAS, les approches de méta-modélisation se proposent d'effectuer une sélection itérative à partir de l'espace des hyperparamètres et construisent des architectures de CNN associées qui sont ensuite entraînées et évaluées. Les résultats d'évaluation (taux de précision de classification par exemple) sont transmis aux méta-contrôleurs pour guider la conception des architectures suivantes. Les premiers travaux sur la NAS étaient dominés par les méthodes basées sur l'optimisation génétique et Bayésienne (Bergstra et al., 2011; Domhan et al., 2015; Floreano et al., 2008; Stanley et al., 2009). Toutefois, ces deux méthodes ont rencontré des difficultés relatives à la performance et au volume de données traitées, cédant la place récemment aux méta-contrôleurs RL largement adaptés aux DNNs et aux techniques de calcul de gradient.

- Optimisation génétique.
  La stratégie des algorithmes évolutifs pour l'optimisation d'hyperparamètres consiste à modifier un ensemble de solutions candidates (population) sur la base d'un certain nombre de règles (opérateurs). Après une procédure itérative de mutation, croisement et sélection (Eiben & Smith, 2015), un algorithme évolutif initialise, dans une première étape, un ensemble de N architectures de CNNs aléatoires afin de créer une population primaire. La deuxième étape consiste à introduire une fonction fitness pour évaluer chaque réseau grâce à sa précision de classification et conserver les réseaux les mieux classés constituant la génération suivante. Le processus évolutif se poursuit jusqu'à ce que les critères d'arrêt soient remplis, ce qui correspond généralement à un nombre maximal de générations autorisées.

- Optimisation Bayésienne.
  Les méthodes Bayésiennes sont un moyen efficace pour optimiser les fonctions objectif de type boîte noire $f : X \rightarrow \mathbb{R}$ lentes à évaluer (Brochu et al., 2010). Elles visent à trouver une entrée $x = arg\ min_{x \in X}\ f(x)$ qui minimise globalement f où, dans le contexte d'un algorithme d'apprentissage automatique, x fait référence à l'ensemble des hyperparamètres à optimiser. Le problème avec ce type d'optimisation est que l'évaluation de la fonction objectif est très coûteuse en raison du grand nombre d'hyperparamètres et de la nature complexe des modèles tels que les réseaux neuronaux profonds. Par conséquent, les approches Bayésiennes proposent une reconstruction probabiliste de substitution (surrogate) de la fonction objectif $\mathcal{P}(f|D)$ où D est un ensemble d'observations passées. L'évaluation de la fonction empirique est beaucoup moins coûteuse que la fonction objectif réelle (Klein et al., 2017). Les

modèles de substitution probabilistes les plus utilisés sont les processus gaussiens (Rasmussen & Williams, 2005), les forêts aléatoires (Breiman, 2001) et l'estimateur de Parzen (Bergstra et al., 2011).

- Méta-contrôleur RL.

  Malgré quelques succès obtenus par les 2 stratégies d'optimisation précédentes, elles ont montré des faiblesses structurelles vis à vis de la tâche de NAS. En effet, les algorithmes génétiques utilisent des ressources de calcul excessives et sont incapables d'exploiter les informations portées par le gradient dans les réseaux de neurones profonds (Xie et al., 2019). De leur côté, les méthodes Bayésiennes ne sont pas adaptées aux modèles à taille variable (Perez-Rua et al., 2018).

  Récemment, les méta-contrôleurs RL ont été proposés comme un substitut compétitif aux méthodes de recherche précédentes. Le cadre d'optimisation qu'offre le RL à travers un agent ajustant ses transitions vers de nouveaux états selon une politique maximisant la récompense de l'environnement s'est avéré compatible avec la NAS. Ainsi, le travail pionnier présenté dans Zoph & Le (2017) qui a reproduit les résultats de l'état de l'art sur des benchmarks de classification bien connus (CIFAR-10 et Penn Treebank) a renforcé l'attrait des méta-modèles RL pour la NAS et déclenché une série d'études intéressantes sur ce thème de recherche.

4.4.4 Méthodes de recherche d'architecture

Plusieurs stratégies ont été développées pour la conception automatique d'architectures CNN dont la majorité a opté pour le RL en tant que méta-contrôleur. Cette section est destinée à examiner en détail les dernières approches de NAS prometteuses, groupées en fonction des spécificités des espaces de recherche et du niveau de complexité.

4.4.4.1 Conception d'architecture basique

Certaines approches de NAS se focalisent sur la conception d'architecture de CNNs basiques, composé exclusivement de couches classiques, principalement de convolution, pooling et entièrement connectées. L'espace de recherche qui en résulte est relativement simple et la contribution de ces approches réside globalement au niveau de la stratégie de conception.

- MetaQNN

  Le modèle MetaQNN (Baker et al., 2017) repose sur le Q-learning pour sélectionner séquentiellement des couches de réseau et leurs paramètres parmi un espace fini. Cette méthode implique d'abord de définir l'état de l'agent d'apprentissage en tant qu'une couche avec tous les paramètres pertinents associés. À titre d'exemple, le tableau 4.4 représente 5 couches : convolution (C), pooling (P), entièrement connectée (FC), pooling global moyen (GAP) et softmax (SM).

Type de couche	Paramètres de couche	Valeurs de paramètre
Convolution	$i \sim$ Profondeur de couche $f \sim$ Taille du champ récepteur $l \sim$ Stride $d \sim$ Nombre de champs récepteurs $n \sim$ Taille de la représentation	< 12 Square $\in \{1,3,5\}$ Toujours égal à 1 $\in \{64,128,256,512\}$ $\in \{(\infty,8],(8,4],(4,1]\}$
Pooling	$i \sim$ Profondeur de couche $(f,l) \sim$ (Taille du champ récepteur, Stride) $n \sim$ Taille de la représentation	< 12 Square $\in \{(5,3),(3,2),(2,2)\}$ $\in \{(\infty,8],(8,4],(4,1]\}$
Entièrement connectée	$i \sim$ Profondeur de couche $n \sim$ Nombre de couches entièrement connectées consécutives $d \sim$ Nombre de neurones	< 12 < 3 $\in \{128,256,512\}$
État de terminaison	$s \sim$ État précédent $t \sim$ Type	Pooling global moyen / softmax.

TABLE 4.4 – Les paramètres de l'espace d'états (Baker et al., 2017).

Deuxièmement, l'espace d'action de l'agent est assimilé aux couches possibles vers lesquelles l'agent peut se déplacer en fonction d'un certain nombre de contraintes définies intentionnellement, afin de permettre une convergence plus rapide.
La figure 4.12 illustre un ensemble d'états et d'actions et un chemin éventuel de l'agent RL permettant de concevoir une architecture CNN. MetaQNN a produit des résultats compétitifs avec des CNNs hand-crafted ainsi que des méthodes de NAS antérieures.

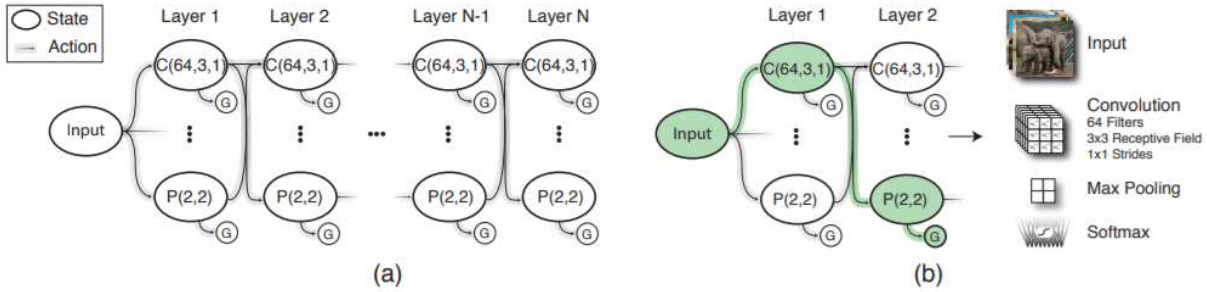

FIGURE 4.12 – MetaQNN : (a) Ensemble d'états.(b) Un chemin choisi par l'agent (Baker et al., 2017).

- NAS

  En utilisant une approche RL, Zoph & Le (2017) entraînent un réseau de neurones récurrent pour construire des CNNs. La figure 4.13 montre un contrôleur RNN générant séquentiellement des paramètres associés à des couches de convolution. Chaque sortie de séquence est prédite par une fonction softmax, puis utilisée comme

entrée pour la séquence suivante. Le jeu de paramètres comprend la hauteur et la largeur du filtre, la taille de la foulée et le nombre de filtres par couche. La conception d'une architecture prend fin lorsque le nombre de couches atteint une valeur prédéfinie qui augmente tout au long de l'entraînement. Le taux de précision de l'architecture élaborée est utilisé comme récompense pour entraîner le contrôleur RNN dans le cadre d'une configuration de RL dont l'objectif est de maximiser la précision espérée des prochaines architectures. L'expérimentation de l'approche globale a permis d'obtenir des résultats satisfaisants sur les benchmarks CIFAR-10 et Penn Treebank.

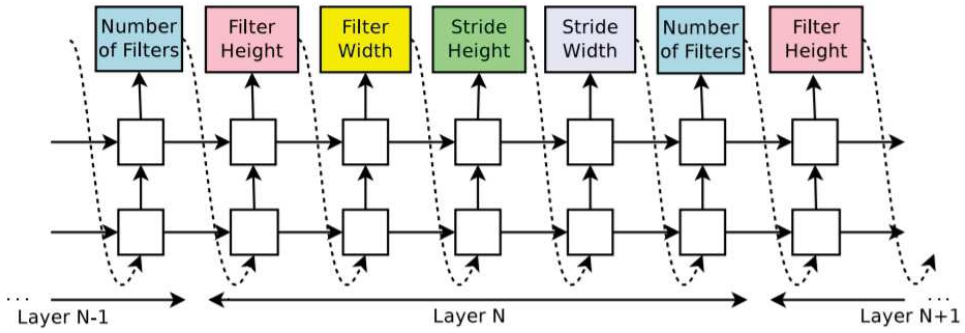

FIGURE 4.13 – Illustration du mode de sélection des hyperparamètres par un RNN (Zoph & Le, 2017).

- EAS

  La méthode récente « Efficient Architecture Search » (Cai et al., 2018a) met en œuvre des techniques de transformation de réseau de neurones permettant de réutiliser des modèles préexistants et d'explorer efficacement l'espace de recherche pour la conception d'architecture automatique. EAS diffère des méthodes précédentes au niveau de la définition des états et des actions du RL. L'état correspond à l'architecture CNN actuelle, tandis que l'action implique des opérations de transformation telles que l'ajout, l'élargissement et la suppression de couches. Les architectures de départ utilisées dans les expériences sont de type CNN simple, composées uniquement de couches de convolution, pooling et entièrement connectées.

  L'approche EAS s'inspire de la technique Net2Net introduite dans le travail de Chen et al. (2016) et basée sur l'idée de construire des réseaux de neurones « élèves » plus profonds pour reproduire le même traitement effectué par un réseau « enseignant » associé. Comme le montre la figure 4.14, un réseau d'encodeurs implémenté avec un RNN bidirectionnel (Schuster & Paliwal, 1997) alimente les réseaux d'acteurs avec des architectures données. Les réseaux d'acteurs sélectionnés effectuent 2 types de transformation : élargir les couches en nombre d'unités et de filtres et insérer de nouvelles couches. En plus de sa performance démontrée, EAS présente l'avantage attrayant d'utiliser des ressources de calcul beaucoup moins importante.

4.4.4.2 Conception d'architecture modulaire

La plupart des travaux récents portant sur la NAS sont basés sur des structures modulaires (multi-branches) plus complexes inspirées des architectures modernes présentées

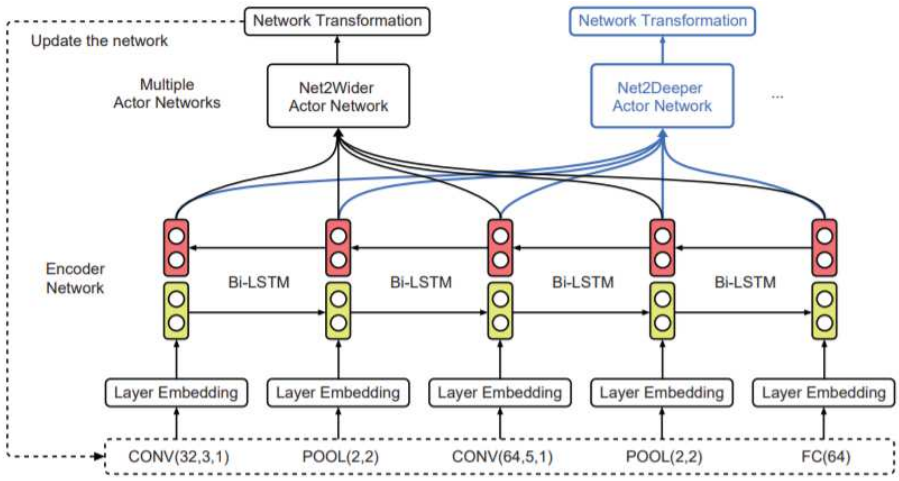

FIGURE 4.14 – Méta-contrôleur pour la transformation de réseaux de neurones (Cai et al., 2018a).

dans le paragraphe 4.4.2. Plutôt que de mener la recherche fastidieuse sur des réseaux de neurones entiers, cette deuxième série d'approches se concentre sur la recherche de blocs de construction empruntés, par exemple, aux modèles GoogLeNet et ResNet. Ces éléments multi-branches sont ensuite empilés répétitivement en faisant appel à des connexions résiduelles afin de créer l'architecture profonde finale. Les architectures « par blocs » réduisent considérablement l'espace de recherche, accélèrent le processus de conception, améliorent les performances des CNN générés et leur confèrent une capacité de généralisation à travers des adaptations mineures.

- BlockQNN

  L'une des premières méthodes mettant en œuvre la recherche d'architecture par blocs est BlockQNN (Zhong et al., 2018). Elle consiste à créer automatiquement des CNN en utilisant la technique de renforcement Q-Learning (Watkins, 1989) avec epsilon-greedy comme stratégie d'exploration (Mnih et al., 2015). La structure d'un bloc, inspirée par les réseaux modernes ResNet et Inception (GoogLeNet), inclut des connexions résiduelles et des combinaisons de couches multi-branches.

Nom	Index	Type	Taille du kernel	Pred1	Pred2
Convolution	T	1	1,3,5	K	0
Max Pooling	T	2	1,3	K	0
Average Pooling	T	3	1,3	K	0
Identité	T	4	0	K	0
Elemental Add	T	5	0	K	K
Concat	T	6	0	K	K
Terminal	T	7	0	0	0

TABLE 4.5 – La codification de l'espace de recherche de blocs (Zhong et al., 2018).

  L'espace de recherche de blocs est détaillé dans le tableau 4.5 et comprend 5 paramètres : index de la couche (sa position dans le bloc), type d'opération (sélectionné

parmi 7 types couramment utilisés), taille du kernel de la couche et positions des couches précédentes (Pred1 et Pred2). La figure 4.15 illustre 2 échantillons de blocs différents, l'un avec une structure à plusieurs branches et le second illustrant une connexion résiduelle. L'ensemble du réseau est construit en empilant le bloc définis séquentiellement N fois.

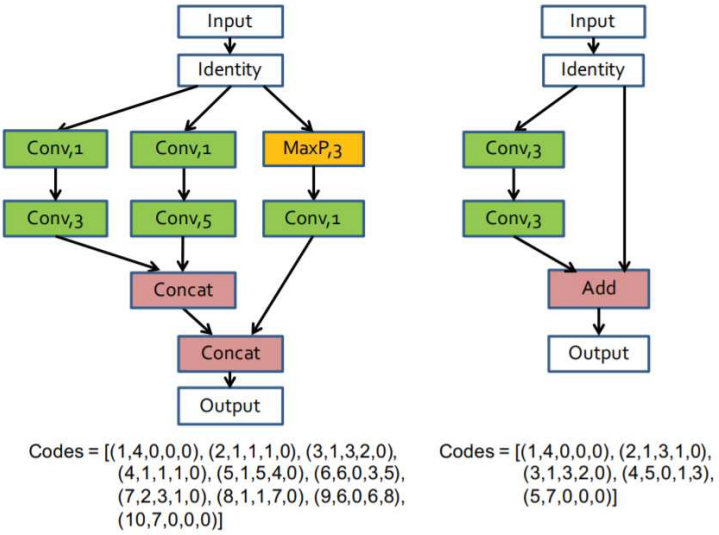

FIGURE 4.15 – Deux exemples de blocs avec leurs codes d'architecture (Zhong et al., 2018).

- PNAS

  La recherche d'architecture neuronale progressive (Liu et al., 2018) propose d'explorer l'espace des structures modulaires, en partant de modèles simples pour ensuite évoluer vers des modèles plus complexes. Les structures peu performantes sont écartées au fur et à mesure de l'apprentissage. Dans cette approche, la structure modulaire s'appelle une cellule et consiste en un nombre fixe de blocs. Chaque bloc est une combinaison de 2 opérateurs parmi 8 sélectionnés, tels que la fonction d'identité, pooling et convolution. Une structure de cellule est apprise en premier lieu, puis empilée N fois afin de construire le CNN final. La première étape consiste à construire, entraîner et évaluer toutes les cellules possibles à un bloc. La cellule est ensuite élargie à 2 blocs, ce qui explose le nombre total de combinaisons.

  L'innovation apportée par PNAS consiste à prédire les performances des cellules du second niveau en entraînant un RNN sur les performances des cellules de niveau précédent. Seules les K meilleures cellules (c'est-à-dire les plus prometteuses) sont transférées à l'étape suivante de l'expansion de la cellule. Ce processus est répété jusqu'à ce que le nombre maximal de blocs autorisé soit atteint. Avec une précision comparable à l'approche NAS (Zoph & Le, 2017), PNAS est jusqu'à 5 fois plus rapide avec une taille maximale de cellule de 5 blocs et K égale à 256. Ce résultat est dû au fait que la prédiction de performance prend beaucoup moins de temps que l'entraînement complet des cellules conçues. La meilleure architecture de cellule est illustrée dans la figure 4.16.

- ENAS

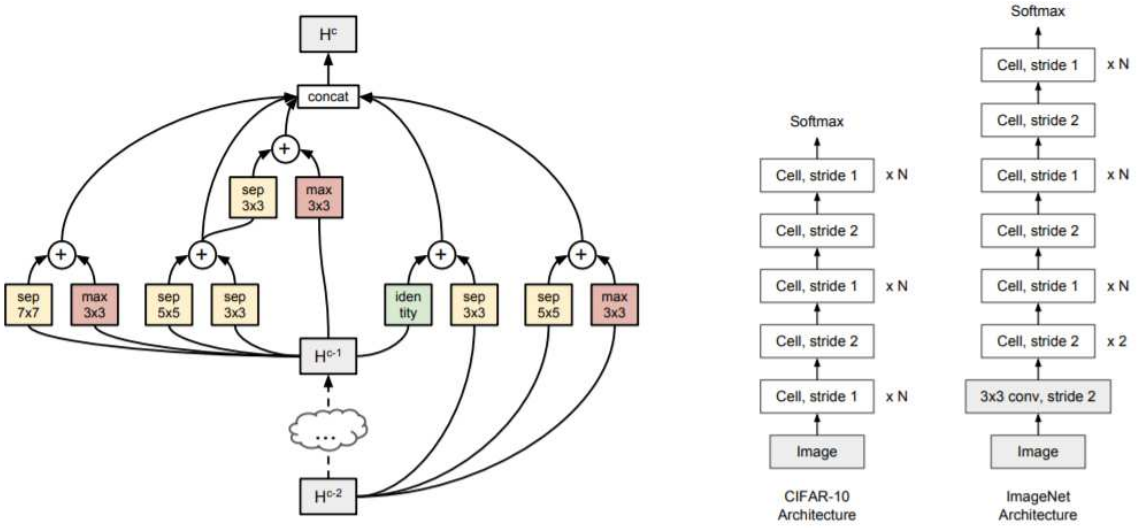

FIGURE 4.16 – La meilleure architecture de cellules sélectionnée par PNAS (Liu et al., 2018).

La recherche d'architecture neuronale efficace (Pham et al., 2018) s'inscrit dans la continuité des travaux antérieurs NAS (Zoph & Le, 2017) et PNAS (Liu et al., 2018). Cette méthode explore un espace de recherche basé sur des cellules via un contrôleur RNN entraîné sous une configuration RL. La structure de la cellule est similaire au modèle PNAS où le concept de bloc est remplacé par un nœud composé de 2 opérations et de deux connexions résiduelles. Le contrôleur RNN gère ainsi 2 types de décisions sur chaque nœud. Premièrement, il identifie 2 nœuds prédécesseurs avec lesquels il peut se connecter, incluant la possibilité des connexions résiduelles. Deuxièmement, le contrôleur sélectionne 2 opérations à mettre en œuvre parmi un ensemble composé d'une fonction d'identité, 2 convolutions depthwise de tailles de filtre différentes (Chollet, 2017a), max-pooling et average-pooling. Au sein de chaque nœud, les résultats des opérations sont ajoutés afin de constituer une entrée pour le nœud suivant.

La figure 4.17 illustre la conception d'une cellule à 4 nœuds. A la fin, l'ensemble du CNN est construit en empilant N fois les cellules obtenues. Au niveau de l'optimisation des coûts de calcul, ENAS propose d'effectuer une présélection des CNN construits suite à une évaluation sur des mini-lots de données. Seuls les meilleurs modèles sont ensuite évalués sur la totalité des données disponibles. D'un autre côté, un partage de poids est effectué entre les nœuds afin de réduire le temps d'entraînement. ENAS fournit des résultats compétitifs sur les jeux de données CIFAR-10 et Penn Treebank.

- EAS avec transformation path-level

  Une version mise à jour de EAS (Cai et al., 2018a) qui adopte la transformation de réseau de neurone pour une NAS plus efficace est présentée dans le travail de Cai et al. (2018b). La nouvelle méthode s'attaque à la contrainte de EAS de n'effectuer que des modifications d'architecture simples (niveau de couche), tel que l'ajout (suppression) d'unités, de filtres et de couches. Le modèle proposé est similaire à EAS où le méta-contrôleur RL échantillonne des actions de transformation pour créer de

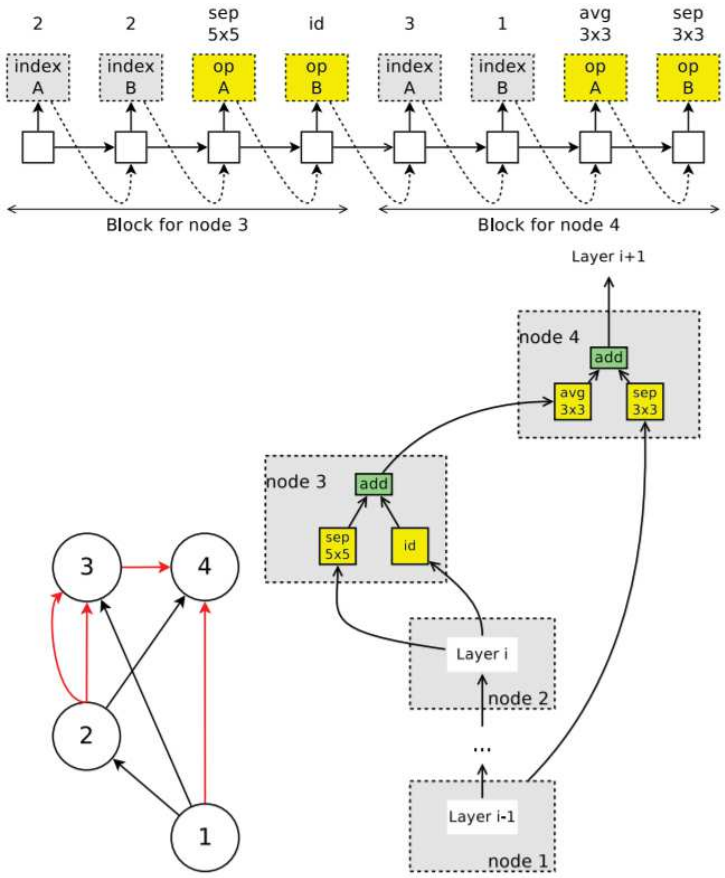

FIGURE 4.17 – Illustration d'une cellule à 4 nœuds (Pham et al., 2018).

nouvelles architectures. L'évaluation de ces dernières est retournée sous forme de récompense pour mettre à jour le méta-contrôleur. L'apport consiste à introduire un nouveau type de transformation appelé path-level afin de s'adapter à la conception d'architecture modulaire. Ainsi un nouveau groupe d'action approprié au concept de bloc a été rajouté comprenant essentiellement la réplication, le saut (skip) et la concaténation.

La figure 4.18 présente un exemple de transformation effectuée par le méta-contrôleur. En sélectionnant ResNet et DenseNet comme architecture de départ, l'approche de transformation path-level permet de reproduire les résultats de l'état de l'art avec des ressources de calcul relativement faibles.

4.4.5 Accélérateurs d'apprentissage

Les méthodes RL offrent des pistes prometteuses pour la résolution de la tâche de NAS. Bien que les structures multi-branches et les connexions résiduelles semblent améliorer l'efficacité de la recherche automatique d'architectures, cette dernière reste toujours coûteuse en temps de calcul (des centaines d'heures GPU, voir annexe A pour un benchmarking de performance par rapport à une tâche de classification) et nécessite davantage d'études pour accélérer le processus de méta-apprentissage. Ainsi, outre les méthodes assignées à l'optimisation de NAS et à la construction de nouveaux composants complexes, certaines techniques sont développées pour accélérer l'apprentissage des méta-contrôleurs et sont

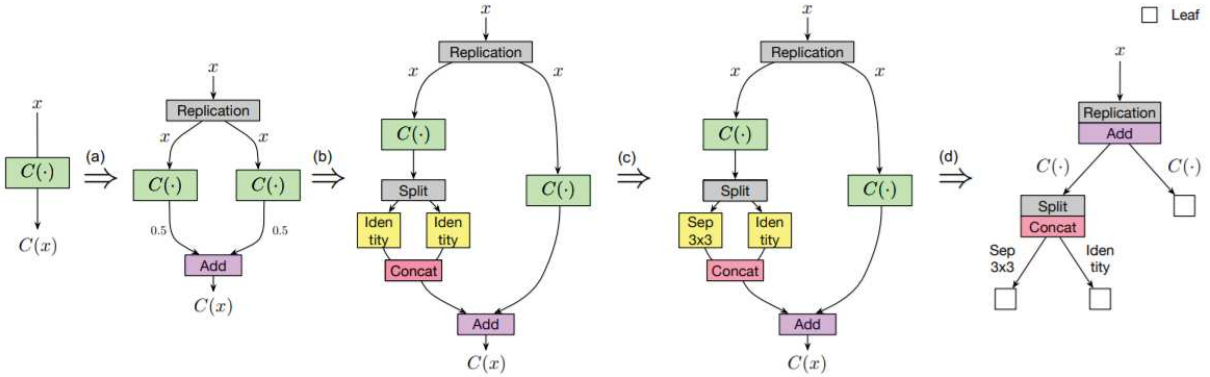

FIGURE 4.18 – Un exemple de transformation path-level partant d'une couche vers une structure modulaire (Cai et al., 2018b).

décrites dans ce paragraphe.

La stratégie d'arrêt précoce d'apprentissage proposée dans (Zhong et al., 2018) permet une convergence rapide de l'agent d'apprentissage tout en maintenant un niveau d'efficacité acceptable. Ceci est possible en prenant en compte les récompenses intermédiaires ignorées dans les travaux précédents (fixées à zéro et retardant la convergence du RL (Sutton & Barto, 2018)). Dans ce cas, l'agent interrompt la recherche à un stade précoce de l'entraînement car les récompenses retournées par les architectures évaluées atteignent des niveaux élevés en quelques itérations. La fonction de récompense est redéfinie afin d'inclure la complexité et la densité des blocs conçus et d'éviter une mauvaise performance pouvant résulter d'un entraînement précoce.

Une deuxième technique est présentée dans Zhong et al. (2018) qui consiste en un cadre de calcul asynchrone distribué assemblant 3 nœuds avec des fonctions différentes. Le nœud central (maître) est l'endroit où les structures de blocs sont échantillonnées par l'agent. Ensuite, dans le nœud du contrôleur, l'ensemble du réseau est construit à partir de blocs générés et transmis à plusieurs nœuds de calcul pour entraînement. L'approche proposée est une sorte de serveur de paramètres simplifié (Dean et al., 2012) qui permet d'entraîner en parallèle les CNNs conçus dans chaque nœud de calcul. Par conséquent, l'ensemble du processus de conception et d'apprentissage est exploité sur plusieurs machines et GPUs. Zoph & Le (2017) utilisent le même schéma de serveur de paramètres avec réplication de contrôleurs afin d'entraîner différentes architectures en parallèle.

Les stratégies RL utilisent les performances des architectures explorées comme récompense pour les mises à jour de leur politique d'exploration. L'entraînement et l'évaluation de chaque architecture construite (parmi des centaines) sur la totalité des données de test sont responsables de la majeure partie de la charge de calcul. L'extraction des performances de l'architecture a donc été soumise à plusieurs tentatives d'estimation. Un certain nombre d'approches se focalise sur la prévision de la performance à partir d'observations passées. La plupart de ces techniques reposent sur une extrapolation de la courbe d'apprentissage (Domhan et al., 2015) et des modèles de substitution (surrogate) utilisant des RNN (Liu et al., 2018) visant à prédire et à éliminer les architectures médiocres avant le passage à l'évaluation. Une autre idée pour estimer les performances et de classer les architectures conçues consiste à utiliser des métriques simplifiées (proxy) pour l'entraînement, telles que des sous-ensembles de données (Pham et al., 2018) et des données sous-échantillonnées

(généralement des images avec une résolution dégradée) (Hinz et al., 2018).

Dans une tentative originale de résoudre ce problème, Ying et al. (2019) présentent NAS-Bench-101, la première base de données d'architectures pour NAS. Les auteurs ont entraîné et évalué des milliers d'architectures de CNN sur CIFAR-10 nécessitant plusieurs années TPU (Tensor Processing Unit). Le résultat consiste en une large table reliant près de 423 milles architectures à leurs taux de précision et à d'autres métriques à l'aide d'une connectivité graphique spécifique. Cela permet de réaliser des expériences NAS en un temps record en interrogeant cette table au lieu de traiter intégralement la procédure d'évaluation coûteuse des architectures explorées. Bien que NAS-Bench-101 soit un nouvel outil et nécessite encore une évaluation approfondie sur des benchmarks de l'état de l'art, il représente une alternative intéressante pour optimiser le travail de NAS.

La transformation de réseaux de neurones est l'une des techniques les plus récentes visant à accélérer la recherche automatique d'architecture neuronale (Cai et al., 2018b; Thomas Elsken, 2018). Il s'agit d'entraîner des architectures explorées réutilisant des réseaux déjà entraînés ou existants. Cette caractéristique de modélisation permet de traiter une limitation des approches de méta-modélisation par RL où l'entraînement est effectué avec une initialisation aléatoire des poids. Ainsi, l'extension du morphisme de réseaux de neurones (Wei et al., 2017) pourrait servir à mieux initier la recherche d'architecture à travers le transfert de connaissance. Celle-ci est reflétée par des poids de réseaux de neurones réutilisés permettant au méta-contrôleur d'analyser efficacement l'espace de recherche.

4.5 Méta-apprentissage pour la conduite adaptative

Tel que constaté à travers la revue de littérature effectuée dans les sections précédentes, l'apprentissage profond a favorisé le passage d'un méta-apprentissage tabulaire, basé sur des méta-caractéristiques statistiques et dédié à la sélection d'algorithmes, vers un méta-apprentissage orienté aux régimes FSL avec une plus grande capacité d'abstraction des modèles à apprendre et une adaptabilité à une large variétés de domaines. Il s'attaque à la complexité et à la non-stationnarité en tant qu'un problème d'apprentissage multitâche permettant un traitement adéquat de l'inefficacité des données et l'amélioration des capacités de généralisation des agents artificiels. Imitant l'intelligence humaine dans sa capacité d'apprendre avec peu d'expérience et de s'adapter à des perturbations inattendues, le méta-apprentissage se présente comme une réponse adéquate aux limites des algorithmes DRL dans la résolution de tâches complexes et de grandes dimensions qui ont été abordées dans le chapitre 3.

En effet, l'approche proposée (MSRC) parvient à améliorer nettement la performance de l'agent RL sur la tâche épineuse de conduite autonome urbaine en profitant des avantages des méthodes actor-critic et de la version généralisée de l'apprentissage TD. Toutefois, ses capacités de généralisation diminuent considérablement lorsque l'entraînement et le test se déroulent dans des environnements et des conditions météorologiques différents. L'expérimentation souligne un « manque à gagner», en termes de récompenses épisodiques, de la stratégie RL qui aurait pu être évité si la méthode était plus robuste à la variabilité de l'environnement. Dans la continuité de notre première contribution de RL robuste, nous développons dans cette section une approche MRL combinant le méta-apprentissage par

optimiseur, plus spécifiquement le MAML, et l'algorithme MSRC. Ce choix est justifié par les avantages inhérents à ce type de méta-apprentissage (voir paragraphe 4.3.2.2) dont principalement son universalité et son adaptabilité à tout modèle entrainé par descente de gradient tel que le DRL.

4.5.1 Modèle

Nous construisons une approche de méta-apprentissage compatible avec une configuration DRL. Notre contribution consiste à combiner (1) un méta-apprenant basé sur le gradient, inspiré du MAML (Finn et al., 2017), pour apprendre une initialisation de modèles généralisable et (2) un contrôleur DNN (MSRC) pour assurer une adaptation robuste et continue. Cette approche MRL est constituée d'une inner loop dédiée à l'apprentissage spécifique à chaque tâche et d'une outer loop apprenant graduellement à travers les tâches (voir paragraphe 4.3.2.1). Nous commençons par présenter les préliminaires relatives au méta-apprentissage par optimiseur de type gradient avant d'en déduire une instanciation pour le RL et plus spécifiquement pour l'approche MSRC discutée dans le chapitre précèdent.

4.5.1.1 Méta-optimiseur par gradient

Dans le cadre de ce scénario de méta-apprentissage, nous considérons une distribution sur les tâches $T_i \sim p(T)$ et le méta-opérateur $g(f(\theta))$ défini dans (section 4.3.2). La fonction $f(\theta)$ est entraînée sur un échantillon tiré de la tâche courante T_i afin de déduire le modèle $\theta_i^{'}$ à partir de la perte $\mathcal{L}_{T_i}(\theta)$. $f(\theta_i^{'})$ est par la suite évaluée sur un nouveau échantillon de T_i pour générer la perte $\mathcal{L}_{T_i}(\theta_i^{'})$. L'ensemble des pertes spécifiques issues de l'évaluation de $f(\theta_i^{'})$ sur le batch de T_i sont transférées au méta-niveau pour un entraînement transverse du méta-apprenant g produisant le modèle d'initialisation généralisable θ^*. Plus formellement, les modèles individuels de base sont calculés en utilisant une mise à jour par descente de gradient suite à chaque entraînement sur T_i :

$$\theta_i^{'} = \theta - \alpha \nabla_\theta \mathcal{L}_{T_i}(\theta) \tag{4.1}$$

Le méta-modèle est issu de l'optimisation des performances de $f(\theta_i^{'})$ sur les tâches échantillonnées à partir de $p(T)$. En effet, le test de chaque modèle individuel $\theta_i^{'}$ permet de constituer des appréhensions différentes de ce qui pourrait être la vrai dynamique du domaine dans sa globalité. Ces appréciations concrétisées par la génération des pertes $\mathcal{L}_{T_i}(\theta_i^{'})$ permettent de construire le méta-objectif suivant :

$$\min_\theta \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}(\theta_i^{'}) \quad o\grave{u} \quad \theta_i^{'} = \theta - \alpha \nabla_\theta \mathcal{L}_{T_i}(\theta) \tag{4.2}$$

Ainsi, nous pouvons définir g en tant qu'une optimisation à travers les tâches, effectuée par descente du méta-gradient $\nabla_\theta \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}(\theta_i^{'})$, dont le rôle est de mettre à jour les paramètres initiaux θ par rapport aux pertes générées au niveau de l'apprentissage de base :

$$\theta \leftarrow \theta - \eta \nabla_\theta \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}(\theta_i^{'}) \tag{4.3}$$

4.5.1.2 MRL avec contrôle multistep

Nous avons vu dans le paragraphe précédent les grandes lignes d'une approche générique et simple de méta-apprentissage par optimiseur de type gradient. Une instanciation de ce concept pour le DRL, et plus spécifiquement le MSRC, garde la même démarche de base mais engendre des adaptations de la fonction de perte et de la manière dont les données sont générées par tâche et présentées au modèle. Nous rappelons les outputs de notre méthode MSRC (section 3.5) et la combinons au méta-optimiseur par gradient. Pour cela, nous distinguons les trois composants principaux de l'approche MRL : l'acteur, le critique et le méta-gradient. A travers ces composants, nous décrivons l'intégration des notions de multitâche et de transfert de connaissance vers un méta-niveau.

Le point de départ du MSRC est un agent actor-only dont le gradient (de la politique) dépend du retour empirique (équation 3.18). Le problème de ce type de RL est la haute variance des retours empiriques spécialement dans un domaine d'application complexe, incertain et de grande dimension tel que la conduite autonome. La solution proposée consiste à modifier le schéma RL en intégrant une étape d'évaluation et d'amélioration de la politique qui génère des bonus supplémentaires pour guider l'agent vers de nouveaux états. Ce rôle est assuré par un critique grâce à l'apprentissage n-step TD et le calcul de l'erreur TD (équation 3.28). Cette dernière intègre l'expression du gradient de la politique (équation 3.25) et permet la mise à jour itérative des modèles de la fonction de valeur (équation 3.30) et de la politique (équation 3.31).

Notre objectif est d'implémenter une couche de méta-apprentissage dans la méthode RL décrite ci-dessus pour obtenir une approche MRL avec contrôle multistep robuste et généralisable aux environnements non-stationnaires. Nous commençons par apparier les variables du système RL définis dans le paragraphe 3.5.1.1 avec les termes de l'approche de méta-apprentissage proposée dans le paragraphe précédent en contextualisant par rapport à T_i. Ainsi les tâches $T_i \sim p(T)$ sont des processus MDP, la fonction de base $f(\theta)$ est la politique stochastique π_θ de paramètres θ qui maximise le retour R_{T_i} d'une trajectoire $\tau_{(t,t+H-1)} = (s_t, a_t, \ldots, s_{t+H-1}, a_{t+H-1}, s_{t+H})$ de T_i, la fonction de perte $\mathcal{L}_{T_i}(\theta)$ correspond à la fonction objectif de la politique $J_{T_i}(\pi_\theta)$ exprimée en fonction de l'erreur empirique n-step TD ($\delta_{T_i}(\theta)$). L'expression individuelle du gradient de la politique devient spécifique à la tâche courante T_i :

$$\nabla_\theta J_{T_i}(\pi_\theta) = \mathbb{E}_{\pi_\theta}\Big[\nabla_\theta \, log\, \pi_\theta(a_t|s_t)\, \delta_{T_i}(\theta)\Big] \tag{4.4}$$

Par conséquent, la mise à jour des modèles de base au niveau de la inner loop est effectuée en fonction de l'erreur n-step TD selon la formule suivante :

$$\theta_i^{'} = \theta + \alpha\, \nabla_\theta \, log\, \pi_\theta(a_t|s_t)\, \delta_{T_i}(\theta) \tag{4.5}$$

Remarquons qu'il ne s'agit plus d'une descente mais d'une montée de gradient puisque il y a eu passage d'une minimisation de perte à une maximisation de l'estimation des retours de l'environnement. Le test des $\theta_i^{'}$ sur des nouveaux échantillons des T_i permet de générer les erreurs TD d'évaluation $\delta_{T_i}(\theta_i^{'})$ à travers lesquelles la connaissance relative aux tâches individuelles est transférée au méta-apprenant. Nous en déduisons le méta-objectif de la outer loop :

$$\max_{\theta} \sum_{T_i \sim p(T)} \delta_{T_i}(\theta_i^{'}) \tag{4.6}$$

En dernier lieu, une montée du méta-gradient par rapport aux erreurs TD permet de générer le modèle d'initialisation final :

$$\theta \leftarrow \theta + \eta \nabla_{\theta} \sum_{T_i \sim p(T)} \delta_{T_i}(\theta_i^{'}) \tag{4.7}$$

La dynamique inner-outer loop de l'approche MRL avec contrôle multistep est illustrée dans la figure ci-dessous.

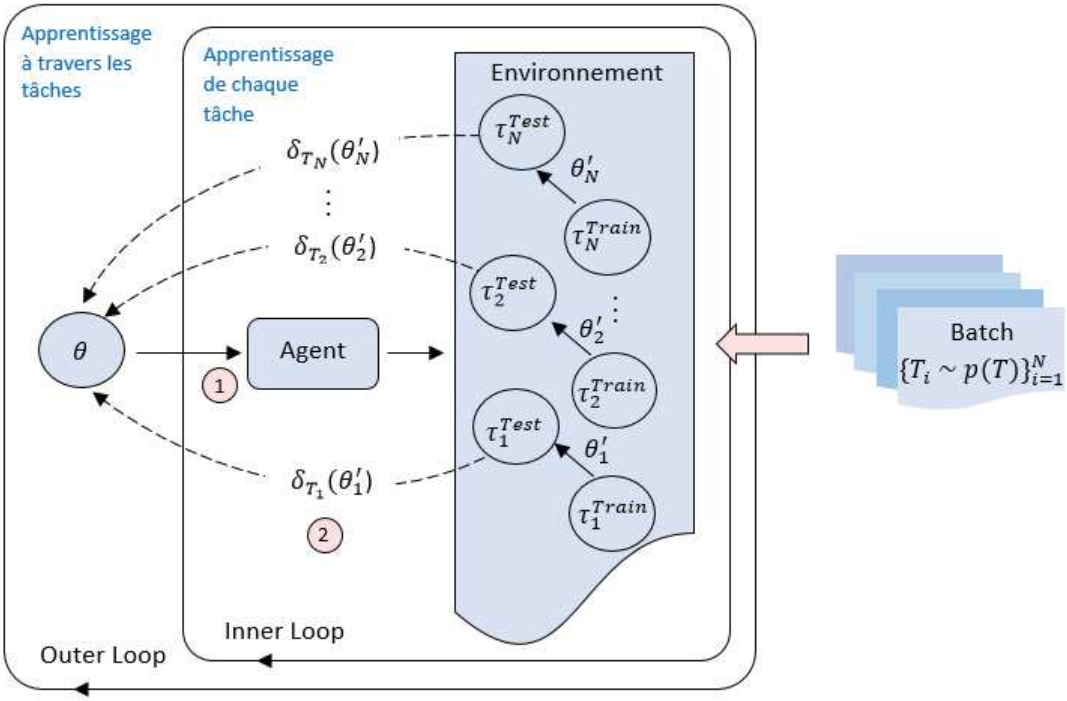

FIGURE 4.19 – Approche de méta-apprentissage par renforcement : interaction entre inner et outer loop.

4.5.1.3 Implémentation algorithmique

Avant de passer à l'expérimentation, nous présentons l'algorithme général de notre approche MRL (voir algorithme 3). Tout d'abord, les modèles de la politique θ et de la fonction de valeur ω sont initialisés aléatoirement. Lors de chaque itération de la inner loop, pour un batch courant de MDPs $\{T_i\}_{i=1}^{N}$, nous procédons comme suit. Une trajectoire τ_i^{train} est collectée en exécutant la politique π_{θ}. Les paramètres $\theta_i^{'}$ et $\omega_i^{'}$ spécifiques à la tâche T_i sont calculés selon la méthode MSRC (algorithme 2). Ensuite, une autre trajectoire τ_i^{test} est collectée en fonction de la politique mise à jour $\pi_{\theta_i^{'}}$ produisant une évaluation par l'erreur n-step TD (δ_{T_i}), toujours selon la même méthode. Une fois cette opération effectuée pour toutes les N tâches T_i du batch en cours de traitement, la outer loop est activée. Elle consiste à mettre à jour les paramètres de départ θ de la politique selon

l'équation 4.7 sous contrainte de la maximisation du retour moyen à travers les tâches. Les étapes détaillées précédemment sont parcourus en boucle jusqu'à ce qu'une performance souhaitée soit atteinte. Le modèle θ^* qui en résulte permettra l'initialisation de politiques généralisables et rapidement adaptables à la variabilité de l'environnement de la conduite autonome en zone urbaine.

Algorithme 3 : Méta-apprentissage par renforcement avec un contrôle multi-step

Entrée : Initialisation aléatoire des paramètres θ de la politique π
Initialisation aléatoire des paramètres ω de la fonction valeur V_π
α= Pas d'apprentissage de l'apprenant de base
η = Pas d'apprentissage du méta-apprenant
Sortie : θ^* `// Modèle d'initialisation généralisable`
répéter
  `// Outer Loop`
  Sélectionner un batch de N tâches $T_i \sim p(T)$

  pour $i \in \{1,..,N\}$ **faire**
    `// Inner Loop : apprentissage de l'apprenant de base (entraînement et test)`
    Exécuter une trajectoire $\tau_i^{train} \sim T_i$ d'horizon H en utilisant $\pi(a_t|s_t,\theta)$
    $\theta_i^{'}$, $\omega_i^{'} \leftarrow MSRC\,(\tau_i^{train},\theta,\omega)$ `/* Mettre à jour la politique et la fonction valeur en utilisant la fonction MSRC (algorithme 2) */`
    Exécuter une trajectoire $\tau_i^{test} \sim T_i$ d'horizon M en utilisant $\pi(a_t|s_t,\theta_i^{'})$
    $\delta_{T_i} \leftarrow MSRC\,(\tau_i^{test},\theta^{'},\omega^{'})$ `/* Calcul de n-step TD error en utilisant la fonction MSRC (algorithme 2) */`
    Mémoriser δ_{T_i}

  fin
  Mettre à jour $\theta \leftarrow \theta + \eta\,\nabla_\theta \sum_{T_i \sim p(T)} \delta_{T_i}(\theta_i^{'})$ `/* Entraînement du méta-niveau à travers les tâches */`
jusqu'à *Jusqu'à la convergence*;

4.5.2 Configuration de l'expérimentation

Nous évaluons les performances de l'approche MRL avec adaptation continue sur la tâche de conduite autonome en milieu urbain exécutée dans le simulateur CARLA. Notre expérience a pour objectif de démontrer l'efficacité de l'apprentissage au niveau méta associé à un contrôleur DNN afin d'optimiser la stratégie RL et d'obtenir un apprentissage plus robuste des environnements complexes et de grande dimension. Nous présentons les résultats de notre étude évaluant deux hypothèses de base : l'agent MRL (1) s'adapte plus rapidement au moment de d'entraînement et (2) affiche de meilleures capacités de généralisation à de nouveaux environnements non inclus dans la phase d'entraînement.

Nous utilisons la même configuration expérimentale décrite au niveau de l'évaluation de l'approche MSRC dans (la section 3.5.2) qui inclut, à titre de rappel :

- Un espace d'actions constitué de 9 commandes de conduite discrètes : Roue libre, Tourner à gauche, Tourner à droite, Accélérer, Freiner, Tourner à gauche en accélérant, Tourner à gauche en freinant, Tourner à droite en accélérant, Tourner à droite en freinant.
- Un espace d'observation construit à partir de captures de scènes réalisées par une caméra RGB installée en avant du véhicule.
- La fonction de récompense basée sur les observations physiques retournées par l'environnement et formulée dans l'équation 3.32.
- Les paramètres inhérents à l'algorithme MSRC définis dans (le paragraphe 3.5.2.3) dont principalement les taux d'apprentissage et d'actualisation, la profondeur du bootstrapping de l'apprentissage TD, les conditions de terminaison d'un épisode d'entraînement et l'architecture des CNNs utilisés pour approximer la fonction de valeur et de la politique.

A travers l'expérimentation, nous essayons d'examiner la tâche de conduite autonome dans des conditions variables sous l'angle d'un problème multitâche, où chaque tâche correspond à un épisode déroulé dans un contexte spécifique. Par conséquent, nous induisons des environnements non stationnaires pour l'agent RL au cours des épisodes d'entraînement en faisant varier plusieurs paramètres du simulateur CARLA :

- La complexité de la tâche : sélectionner l'une des 2 villes disponibles, ainsi que différentes positions de départ et d'arrivée (conduite en ligne droite ou avec virage).
- La densité du trafic : changer le nombre d'objets dynamiques tels que les piétons et les véhicules.
- Conditions météorologiques : choisissez une combinaison de conditions météorologiques et d'éclairage afin de diversifier les effets visuels.

La combinaison de cette multitude de paramètres nous permet d'attribuer un sous-ensemble d'environnements à l'entraînement que nous qualifions de « seen » dans la suite de cette section. D'un autre côté, nous dédions un deuxième sous-ensemble d'environnements, notés « unseen » (non vus au cours de l'entraînement), exclusivement à la phase d'adaptation du modèle optimal. Un échantillon d'environnements utilisés dans cette expérimentation est montré dans la figure 3.8.

4.5.3 Résultats et interprétation

Le déploiement de notre algorithme de méta-apprentissage est effectué avec les mêmes ressources utilisées dans le RL robuste (paragraphe 3.5.3) : Python et TensorFlow pour le développement et Nvidia P5000 (1 GPU, 16 GB RAM) pour l'exécution. La récompense épisodique est également retenue pour mesurer la performance de conduite de l'agent. Nous adoptons une méthodologie d'évaluation similaire à Finn et al. (2017) consistant à comparer 3 modes d'initialisation des paramètres θ de la politique de l'agent actor-critic :

- Le modèle de la politique est initialisé par les paramètres θ^{rand} échantillonnés aléatoirement et sans aucun entraînement. Ce mode est utilisé comme un « cas témoin » pour relativiser et mieux apprécier l'apport des modes suivants.

- Le modèle de la politique est initialisé par θ^{msrc} obtenus suite à l'application de l'algorithme 2 avec les mêmes modalités d'entraînement décrite dans (paragraphe 3.5.2), principalement 72 heures de conduite dans le simulateur CARLA et même méthode de déduction du meilleur modèle final.

- Le modèle de la politique est initialisé par θ^{mrl} obtenus suite au déroulement de (l'algorithme 3) avec les mêmes modalités d'entraînement du modèle précédent θ^{msrc} pour garantir le maximum d'objectivité de comparaison entre les deux versions. Notons toutefois que le nombre d'épisodes dans le cas du θ^{mrl} est inférieur à celui réalisé avec θ^{msrc} vu le coût de calcul supplémentaire engendré par l'exécution du méta-gradient et l'utilisation de 5 tâches par batch ($N = 5$). En outre, il est important de souligner que les modèles θ^{mrl} et θ^{msrc} sont appris dans les mêmes conditions de conduite alternant un grand nombre d'environnements différents.

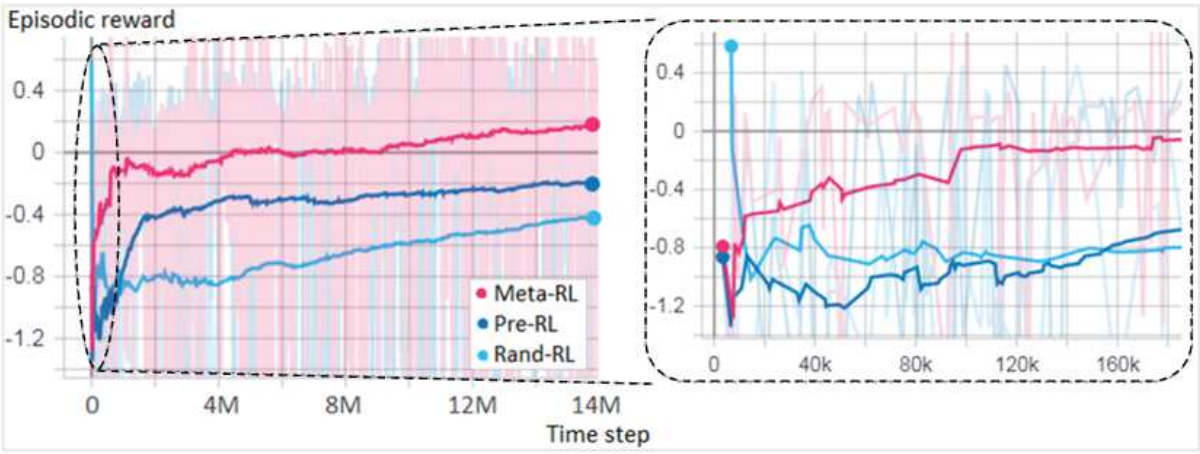

FIGURE 4.20 – Gauche : comparaison de notre approche (Meta-RL) avec l'algorithme pré-entraîné (Pre-RL) et celui initialisé aléatoirement (Rand-RL) dans la phase d'entraînement adaptatif sur des environnements « unseen ». Droite : un zoom sur les étapes initiales de conduite.

Nous transférons les modèles d'initialisation θ^{rand}, θ^{msrc} et θ^{mrl} à l'algorithme RL robuste (MSRC) pour en générer trois versions, notées respectivement Rand-RL, Pre-RL et Meta-RL. Nous entamons par la suite la phase d'entraînement adaptatif sur des environnements « unseen ». La figure 4.20 illustre les performances d'adaptation des 3 approches Rand-RL, Pre-RL et Meta-RL. Les résultats confirment que notre approche Meta-RL génère des modèles s'adaptant plus rapidement à des nouveaux environnements non rencontrés lors de la phase d'entraînement, comparativement aux stratégies Rand-RL et Pre-RL.

En effectuant un zoom sur les étapes de conduite initiales (figure 4.20 à droite), nous remarquons que notre méthode a nettement dépassé le Pre-RL après environ 10000 pas de temps d'adaptation. Il serait intéressant de mener, dans des travaux futurs, des tests supplémentaires pour identifier un seuil benchmark de FSL (en termes de nombre minimum de montées de gradients ou de pas de temps) spécifique à l'apprentissage des tâches RL de grande dimension telles que la conduite autonome.

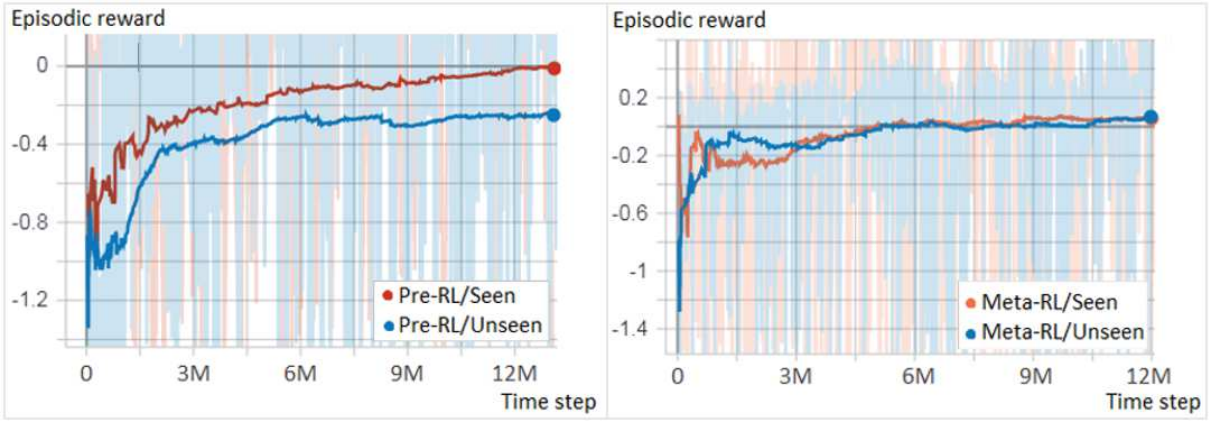

FIGURE 4.21 – Capacités de généralisation de notre approche (à droite) et du RL pré-entraîné (à gauche) : comparaison des résultats d'adaptation dans des environnements «seen» et «unseen».

Afin de valider l'hypothèse de généralisation, nous reprenons la phase d'adaptation en suivant deux scénarios de conduite distincts : un avec des environnements « seen », l'autre avec des « unseen ». Le Rand-RL est exclu de cette expérimentation étant donné qu'il a été initialisé aléatoirement, donc n'appartenant pas au périmètre de la notion « seen ». La figure 4.21 ne révèle pas un écart de performance significatif pour notre approche Meta-RL entre les 2 scénarios, prouvant sa robustesse dans des conditions de conduite non stationnaires. Au contraire, les performances du Pre-RL diminuent considérablement dans les environnements « unseen » par rapport au scénario « seen » montrant des limites au niveau de ses capacités de généralisation.

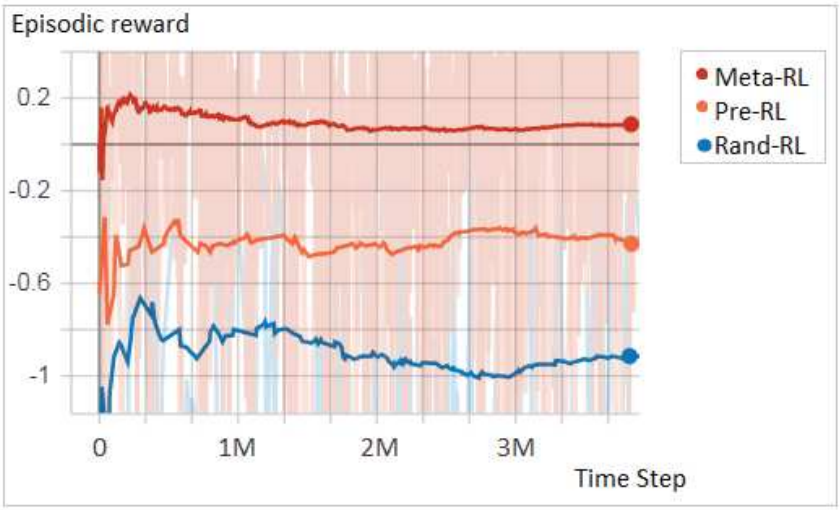

FIGURE 4.22 – Phase de test sur des environnements « unseen »

A la suite de l'étape d'adaptation, nous obtenons trois nouveaux modèles de politique qui sont évalués dans la troisième phase de test. Dans la figure 4.22, le Rand-RL est confirmée comme étant inefficace pour des tâches complexes réalisant la plus faible récompense épisodique au cours des 3 millions pas de temps de test. D'un autre côté, notre approche

Meta-RL est toujours dominante par rapport au Pre-RL, démontrant la contribution du méta-apprentissage à l'élaboration de stratégies RL robustes et efficaces dans des contextes incertains et de grande dimension.

4.6 Conclusion

Imitant l'intelligence humaine dans sa capacité d'apprendre avec peu d'expérience et de s'adapter efficacement à la variabilité des tâches, le méta-apprentissage se présente en tant qu'une architecture de transfert de connaissance dans le cadre de laquelle les méthodes ML sont plus aptes à développer des capacités de généralisation dans des contextes de non-stationnarité. Dans le cadre de notre quête de stratégies RL robuste pour les tâches du monde réel, et en continuité avec notre première contribution décrite dans le chapitre précédent, nous faisons appel au méta-apprentissage avec optimiseur de type gradient dont les principes sont spécifiquement compatibles avec notre objectif. Ce type d'architecture qui vise à générer une initialisation de modèles sensible aux changements, présente l'avantage de constituer une procédure d'apprentissage universelle dans le sens où elle ne fait aucune hypothèse préalable sur (1) la nature de la tâche traitée et (2) le modèle de base à apprendre sauf le fait qu'il soit entraîné par descente de gradient. D'un autre côté, l'implémentation d'un méta-gradient n'engendre aucun paramètre supplémentaire dans l'algorithme global minimisant les coûts de traitement additionnels.

Les résultats de l'évaluation du MRL proposé pour la conduite autonome en milieu urbain valident les hypothèses d'entrée du modèle stipulant que l'agent MRL s'adapte plus rapidement au moment de l'entraînement et affiche de meilleures capacités de généralisation à de nouveaux contextes de conduite. Plus concrètement, l'expérimentation montre que le méta-apprentissage permet à la politique de contrôle par MSRC de rattraper la dégradation de performance constatée dans le chapitre 3 au passage de l'agent à de nouveaux scénarios de test du périmètre « unseen » (environnements non rencontrés lors de l'entraînement). D'un autre côté, nous avons pu mettre en exergue tout au long de ce chapitre un certain nombre de perspectives intéressantes. A titre d'exemple, il serait intéressant d'examiner la notion de FSL, à l'origine spécifique à l'apprentissage supervisé, dans une configuration RL et essayer d'identifier un seuil benchmark en termes de nombre minimum de montées de gradients nécessaires à l'apprentissage de tâches de grande dimension. Un deuxième élément à considérer est l'architecture des CNNs utilisés dans le processus de vision de la conduite autonome. Ce type de réseaux de neurones nécessite une configuration d'hyperparamètres lourde et compliquée influençant profondément ses performances, ce qui justifierait l'utilisation des techniques de NAS pour concevoir des CNNs appropriés aux scènes de conduite urbaines.

Chapitre 5

Conclusion

Dans cette thèse, nous avons étudié une instanciation du méta-apprentissage pour le contrôle adaptatif, un procédé indispensable à la majorité des activités humaines, des plus rudimentaires (assistance, ménage, loisirs, sécurité, etc.) aux plus sophistiquées (transport, industrie, médecine, exploration des océans et de l'espace, etc.). L'approche proposée est construite sur la base d'une formulation classique du RL évoluant graduellement et d'une manière justifiée par l'intégration de différentes techniques émanant de la programmation dynamique et de l'apprentissage profond. Concrètement, ces apports répondent aux contraintes et défis imposés par les applications du monde réel qui résultent, d'une part, de la grande dimensionnalité et la complexité des espaces d'observation, d'autre part, de la variabilité et l'incertitude du contexte de progression des systèmes de contrôle. Après avoir rappelé les éléments de la modélisation Markovienne adoptée comme point de départ pour notre approche, nous présentons le bilan de nos contributions avec leurs perspectives correspondantes selon 3 axes d'amélioration : la dimensionnalité des espaces d'états, la variance des rendements et la capacité de généralisation du modèle.

Modèle préliminaire. Nous avons considéré un contrôleur (agent, décideur) interagissant avec un processus (environnement), au moyen de trois signaux : un signal décrivant l'état du processus, un signal d'action qui permet au contrôleur d'influencer le processus et un signal de récompense scalaire qui fournit au contrôleur des informations sur ses performances immédiates. À chaque pas de temps discret, le contrôleur reçoit la mesure d'état et applique une action, qui fait transiter le processus dans un nouvel état. Une récompense est générée évaluant la qualité de cette transition. Le contrôleur reçoit la nouvelle mesure d'état et le cycle entier se répète. Le comportement du contrôleur est dicté par sa politique, une fonction des états en actions. Le comportement du processus est décrit par sa dynamique, qui détermine comment l'état change à la suite des actions du contrôleur. Les transitions examinées dans notre étude sont stochastiques, dans le sens où l'état suivant est une variable aléatoire. La règle selon laquelle les récompenses sont générées est décrite par la fonction de récompense. La dynamique du processus et la fonction de récompense, ainsi que l'ensemble des espaces d'états et d'actions sont formalisés dans le cadre d'un processus de décision de Markov dont l'objectif est de trouver une politique optimale qui maximise le rendement espéré calculé à travers sa fonction de valeur.

Dimensionnalité. Un défi central des algorithmes RL dans leur forme originale est qu'ils ne peuvent pas être implémentés que lorsque les espaces d'état et d'action sont constitués d'un nombre réduit d'éléments discrets parce qu'ils nécessitent la représentation exacte des fonctions de valeur ou des politiques. En effet, une représentation de fonction de valeur exacte ne peut être obtenue qu'en stockant des estimations du retour pour chaque paire état-action (lorsque les fonctions Q sont utilisées) ou pour chaque état (dans le cas des fonctions V). De même, pour représenter exactement les politiques, des actions distinctes doivent être stockées pour chaque état. Or, la plupart des problèmes

de contrôle automatique du monde réel dont la conduite autonome ont des espaces d'état et d'action vastes ou continus augmentant exponentiellement le coût de la représentation exacte. Ce problème, appelé la « malédiction de la dimensionnalité » rend les algorithmes RL classiques inapplicables dans des contextes de grande dimensionnalité. Ainsi, lorsque certaines variables ont un nombre très grand ou infini de valeurs possibles, les fonctions de valeur et les politiques doivent être représentées approximativement.

Nous avons utilisé l'apprentissage profond et plus spécifiquement les CNNs pour effectuer une approximation adaptative paramétrique (où le nombre de paramètres est connu a priori) de la politique et de la fonction de valeur et développer par conséquent une approche DRL model-free entraînée online. Nous avons pu ainsi obtenir une méthode robuste adoptant une stratégie hiérarchique d'apprentissage de bout en bout assurée par des architectures CNNs capables de traduire des états (capture de scène) composées de milliers de pixels en commandes de contrôle pour la conduite autonome. Nous avons montré que le DRL profite des avantages de l'apprentissage profond en s'adaptant à des entrées sensorielles de grande dimension et à une logique de contrôle complexe, sans requérir l'ingénierie manuelle de caractéristiques ni des modèles préalables d'un environnement de simulation quasi-réaliste, éventuellement contraignants et imprécis. D'un autre côté, l'utilisation de la descente de gradient stochastique pour la mise à jour des pondérations des CNNs d'approximation ne modifie que légèrement les paramètres de la politique évitant de grands écarts d'estimation, ce qui a permis de renforcer davantage la stabilité de l'entraînement et la convergence vers une politique efficace.

L'architecture des CNNs utilisés dans l'approximation de l'environnement nécessite une configuration empirique influençant largement ses performances. En effet, la structure du modèle, le processus d'entraînement et la représentation des données dépendent étroitement d'un certain nombre d'hyperparamètres incluant la profondeur du réseau, le taux d'apprentissage, le type de couches, etc. En guise de perspective, nous planifions de recourir à la NAS, étudiée dans le chapitre 4, en tant qu'une méthodologie de méta-apprentissage pour concevoir des CNNs appropriés aux scènes de conduite urbaines. Il est donc possible d'automatiser cette sélection d'hyperparamètres qualifiée de boite noire dans la littérature (Bengio et al., 2013) en raison de la nature inconnue de la relation entre l'architecture du réseau et la performance de l'apprentissage. Que ce soit sous sa forme basique (Baker et al., 2017) ou « multi-branche » modulaire (Liu et al., 2018), la NAS propose une méta-modélisation capable d'effectuer une sélection itérative à partir de l'espace des hyperparamètres et construire des architectures de CNN adaptées au contexte d'application. Les modèles produits par la NAS sont prometteurs et compétitifs avec les CNNs « hand-crafted » modernes tels que ResNet (He et al., 2016b), GoogLeNet (Szegedy et al., 2015) et Inception (Szegedy et al., 2017) avec une focalisation actuelle sur l'accélération de l'apprentissage des méta-contrôleurs (Zhong et al., 2018) pour une meilleure accessibilité à cette technologie innovante.

Stabilité. Les méthodes de recherche de politiques (policy-based), également appelées actor-only, se concentrent sur la maximisation de la récompense à long terme par essai-erreur afin de contrôler des tâches complexes sans un modèle préalable du système. Elles peuvent ainsi apprendre des politiques stochastiques maximisant une approximation du rendement espéré des échantillons de données sans apprendre la fonction de valeur. Cela se fait au prix d'un apprentissage lent et de haute variance nécessitant des millions d'itérations

à apprendre. En effet, la variation entre les cycles d'apprentissage peut être très élevée, ce qui signifie que certaines actions d'un algorithme RL réussissent tandis que d'autres échouent en fonction du caractère aléatoire de l'initialisation et de l'échantillonnage de la trajectoire. Malgré que dans le cas des méthodes policy-based, les gradients peuvent fournir un signal d'apprentissage fort quant à la manière d'améliorer une politique paramétrée, leur grande variabilité constitue un obstacle important à l'application du RL sur des tâches du monde réel étant donné que l'apprentissage devient peu fiable.

Afin de stabiliser l'apprentissage, nous avons remplacé, dans la formulation du gradient de la politique, le retour empirique défini dans les méthodes Monte Carlo par la fonction avantage. L'utilisation d'une baseline qui permet d'exprimer la valeur relative de chaque action par rapport au rendement espéré de l'état a généré des magnitudes inférieures à celles de l'expression classique des retours empiriques, diminuant significativement la variabilité des prédictions issues du gradient de la politique. Nous avons introduit également un critique pour l'évaluation continue de la fonction avantage estimée à travers une version généralisée de l'erreur TD. Combinant les avantages des méthodes Monte Carlo et de la programmation dynamique, cette implémentation actor-critic a été capable d'apprendre à la fois la politique et sa fonction de valeur et de mener en conséquence une itération de politique efficace tout au long des trajectoires de conduite. La réduction des magnitudes des rendements par la fonction avantage, l'arbitrage par bootstrapping entre variabilité empirique et biais d'estimation et l'alternance continue des deux processus de programmation dynamique (prédiction et contrôle) ont permis de réduire la variance des gradients et d'accélérer sensiblement l'apprentissage de l'agent RL dans la tâche de conduite autonome en milieu urbain.

L'apprentissage par TD est une méthode prépondérante dans les configurations RL en constituant une solution intermédiaire entre l'approximation empirique des retours et l'estimation biaisée de la fonction de valeur. Une perspective d'évolution intéressante relative à ce thème est l'utilisation des traces d'éligibilité $\lambda \in [0, 1]$, généralement compatibles avec la majorité des méthodes TD. Lorsque l'apprentissage TD est complété par des traces d'éligibilité (noté TD(λ)), il produit une famille d'algorithmes délimités par les méthodes Monte Carlo ($\lambda = 1$) et les méthodes TD ($\lambda = 0$). Entre ces deux bornes, se trouvent des méthodes intermédiaires qui sont souvent plus performantes (Sutton & Barto, 2018). L'idée derrière les traces d'éligibilité est simple. Chaque fois qu'un état est visité, il initie un processus de mémoire à court terme, une trace, qui se désagrège ensuite progressivement au cours des pas de temps. Cette trace marque l'état comme éligible à l'apprentissage. Si une action est exécutée alors que la trace n'est pas nulle, l'état reçoit une récompense qui lui est proportionnelle (Singh & Sutton, 1996). Par rapport au n-step TD, les traces d'éligibilité offrent plusieurs avantages de calcul vu qu'elles ne requièrent qu'un seul vecteur de trace plutôt qu'une mémoire des n derniers vecteurs de retours et que l'apprentissage peut se produire et affecter la politique de l'agent immédiatement après la rencontre d'un état au lieu d'être retardé de n pas de temps.

Généralisation. Les axes d'amélioration détaillés antérieurement ont permis de stabiliser l'apprentissage de notre approche DRL sur la tâche complexe de conduite autonome en milieu urbain. Toutefois, l'amplification de la variabilité et de l'incertitude de l'environnement et l'introduction de nouvelles situations « unseen » par rapport aux scénarios d'entraînement ont mis en exergue certaines limites de généralisation du DRL robuste

proposé face à des simulations plus fidèles au contexte de conduite dans le monde réel. Ces résultats sont prévisibles vu que le succès des configurations RL profondes est fortement fondé sur l'utilisation d'un large volume d'épisodes d'entraînement monotones au sein d'un environnement plutôt stationnaire, aboutissant à des algorithmes corrélés à des périmètres étroits de scènes de conduite. En effet, les DNNs représentant la politique et la fonction de valeur disposent souvent de centaines, voir de milliers de paramètres initialisés aléatoirement. Ces derniers doivent être ajustés en fonction d'une grande quantité de données afin de s'adapter correctement à la tâche tout en empêchant le sur-apprentissage, d'où l'affichage d'une mauvaise généralisation des modèles appris aux données collectées en dehors du périmètre d'entraînement.

L'inefficacité de données et la spécialisation au domaine sont deux caractéristiques inhérentes à l'apprentissage profond et par conséquent aux approches DRL, qui ont été largement confirmées dans la littérature (Mnih et al., 2015). Ces caractéristiques sont en contraste évident avec la cognition humaine qui est plus à l'aise à apprendre avec peu d'expérience et à s'adapter à des perturbations imprévisibles. Nous avons développé une approche MRL qui a permis de combler cet écart de performance en instaurant des politiques RL plus intelligentes et plus attentives au contexte d'apprentissage dans des environnements de conduite complexes et variables. L'implémentation d'un méta-optimiseur de type gradient apprenant les évaluations du modèle actor-critic de base à travers la variabilité des environnements a créé une dynamique inner-outer loop aboutissant à une meilleure adaptation du biais des fonctions d'approximation des politiques. Plus concrètement, à l'issue de la montée itérative du méta-gradient sur l'ensemble des trajectoires échantillonnées pour l'entraînement à partir d'un environnement non-stationnaire, nous avons produit un modèle d'initialisation de paramètres de politique généralisable. Ce modèle a montré une plus grande sensibilité et une adaptation rapide aux nouveaux contextes de conduite traduites par des améliorations de rendement notables suite à seulement un petit nombre d'ajustements de ses paramètres.

Tel que mentionné précédemment, notre approche MRL est model-free où la politique prédit directement des actions à partir des états observés, sans modélisation explicite et préalable des dynamiques de l'environnement. Cependant, la plupart des applications du monde réel, dont notamment la conduite autonome, nécessitent que l'agent agisse en respectant certaines normes (de sécurité par exemple) dans l'environnement. Dans ce cas, une exploration totalement non supervisée qui conduirait à des échecs catastrophiques est à revoir. Le RL model-based constitue un moyen d'intégration de tels schémas coercitifs dans l'apprentissage, étant donné qu'il agit différemment en essayant de construire un modèle de l'environnement sur la base duquel il résout un problème de planification et en déduit les actions optimales de l'agent. Ce type de RL fait souvent un meilleur usage d'une quantité limitée d'expérience et aboutit à une politique optimale en opérant moins d'interactions avec l'environnement (Chebotar et al., 2017). D'un autre côté, le RL model-free est plus simple, nécessite moins de ressources de calcul et n'est pas affecté par les biais de conception d'un modèle éventuellement imprécis de l'environnement. Étant donné les avantages complémentaires des deux types de RL, nous envisageons dans un travail futur de développer une approche hybride qui incorpore les connaissances sur les dynamiques de l'environnement dans le processus de contrôle afin de réduire l'incertitude dans les itérations d'approximations stochastiques.

Un moyen efficace pour profiter des connaissances préalables sur l'environnement est

l'utilisation d'algorithmes RL model-based en conjonction avec la commande prédictive (MPC pour model predictive control) (Hafner et al., 2019). La MPC est une technique pionnière de contrôle qui peut prendre en compte, au stade de la conception, les prévisions futures et les contraintes d'exploitation du système (Morari & Lee, 1999). Cela en fait un choix approprié pour l'entraînement des véhicules autonomes où le système est confronté à un environnement en évolution dynamique et doit satisfaire des contraintes de sécurité cruciales (telles que l'évitement d'obstacles et le maintien sur la voie). Dans la MPC, le modèle de l'environnement est utilisé pour prédire l'évolution future du système avant d'entamer un processus de compensation continu des erreurs qui peuvent en découler. Le fonctionnement de la MPC consiste ainsi à planifier H pas de temps vers l'avant à l'aide du modèle et de sa fonction de coût, exécuter la première action basée sur ce plan (appelée action de contrôle), et puis replanifier une nouvelle trajectoire après avoir reçu l'observation résultante de l'environnement. Dans l'annexe B, nous présentons une illustration de haut niveau d'une approche hybride intégrant la MPC et le méta-apprentissage, qui nous permettrait de réaliser des progrès concrets dans la recherche davantage d'optimalité et de généralisation dans les tâches de contrôle du monde réel.

Tout au long de cette thèse, nous avons pu explorer, au niveau de la revue de la littérature, du développement des contributions et des perspectives, plusieurs techniques et paradigmes du ML qui ont révolutionné ce domaine. L'universalité de l'apprentissage profond a permis de redécouvrir le potentiel du RL à englober une large partie de l'AI et à résoudre des problèmes de grande échelle. Toutefois, nous sommes convaincus que l'adoption d'architectures de méta-apprentissage reprenant des aspects fondamentaux de la cognition humaine poussera encore plus loin le ML dans le défi de la complexité du monde réel. Si un jour nous parvenions à atteindre le niveau 5 dans l'autonomie des véhicules autonomes, ça serait fort probablement grâce à un déploiement plus large et plus profond du méta-apprentissage.

Annexe A

NAS : benchmarking de performance

Dans le cadre de notre recherche sur le méta-apprentissage, nous avons mené quelques travaux sur la NAS, et plus spécifiquement la conception d'architectures de CNN pour la reconnaissance d'objets dans des nuages de points 3D projetés en images 2D (voir figure A.1). Nous présentons les résultats empiriques d'une approche basée sur le modèle PNAS décrit dans le paragraphe 4.4.4.2. Elle a été développée dans le cadre d'un projet traitant l'automatisation de l'inventaire et de la surveillance des installations dans les raffineries et les plateformes pétrochimiques. L'objectif de ce travail, qui a fait l'objet d'une communication présentée à la GPU Technology Conference, Europe 2018 (Munich) [1], est de se constituer un benchmark par rapport au coût d'application de la NAS sur ce type de données et avec les capacités de calculs dont nous disposons.

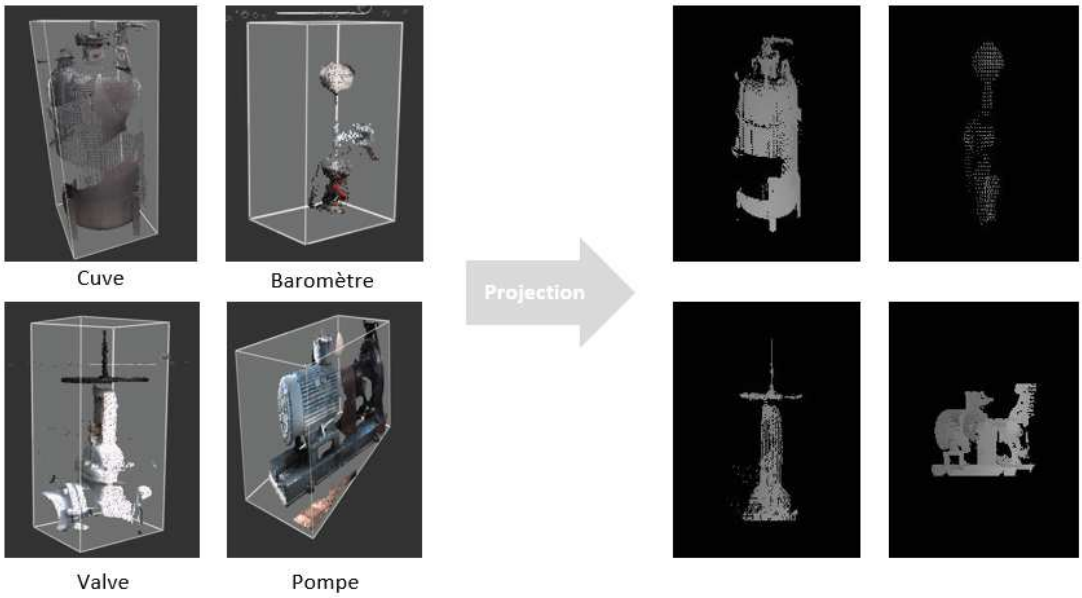

Figure A.1 – Projection en 2D des images d'expérimentation.

La base de données utilisées comporte 12000 images à 3 bandes (RGB) de taille 200×200 pixels réparties sur 7 classes : baromètre, cuve, pompe, valve, réservoir, extincteur et tuyau. Les paramètres utilisés sont inspirés par (Liu et al., 2018), principalement un nombre maximal de blocs par cellule B égal à 5 et un nombre d'architectures sélectionnées pour le passage au niveau suivant K égal à 256. Le nombre d'époques (passages sur l'ensemble des images d'entraînement) par architecture dépend de la capacité de calcul disponible vu la nature coûteuse, en termes de temps d'exécution, du processus d'entraînement de centaines de CNNs sur des milliers d'exemples. Chaque architecture constituée automatiquement est

1. https ://www.nvidia.com/en-eu/gtc/poster-gallery/deep-learning/

complétée par une couche d'average pooling suivie d'une couche softmax afin de pouvoir entraîner le modèle empilé sur la tâche de classification.

Les expériences ont été menées en utilisant le serveur NVIDIA P5000 (1GPU, 16 GB RAM) et la station NVIDIA DGX (4 GPU Tesla V100, 128 Go de RAM). Les figures A.2, A.3 et A.4 montrent l'évolution de la performance de classification en fonction du nombre de CNNs entraînés dans différents cas de figure. Nous remarquons que pour les 3 contextes d'expérimentation, le taux de précision de classification évolue avec l'augmentation du nombre de modèles entraînés, ce qui confirme que le système d'apprentissage améliore sa stratégie de sélection d'architecture avec l'expérience. D'un autre côté, les taux de précision des meilleures architectures croissent nettement de 60% à 89,1% en augmentant la consistance de l'entraînement (évolution du volume de données et du nombre d'époques entre les figures A.2 et A.3) au prix, toutefois, d'une charge de calcul supplémentaire importante de 20 jours d'entraînement. Nous constatons enfin que la faisabilité et la performance des implémentations de la NAS dépendent étroitement de la puissance du hardware utilisé. Ainsi, nous avons réalisé un taux de précision de 81% pour seulement 3 jours d'entraînement grâce à la puissance de calcul de la station NVIDIA DGX (voir figure A.4).

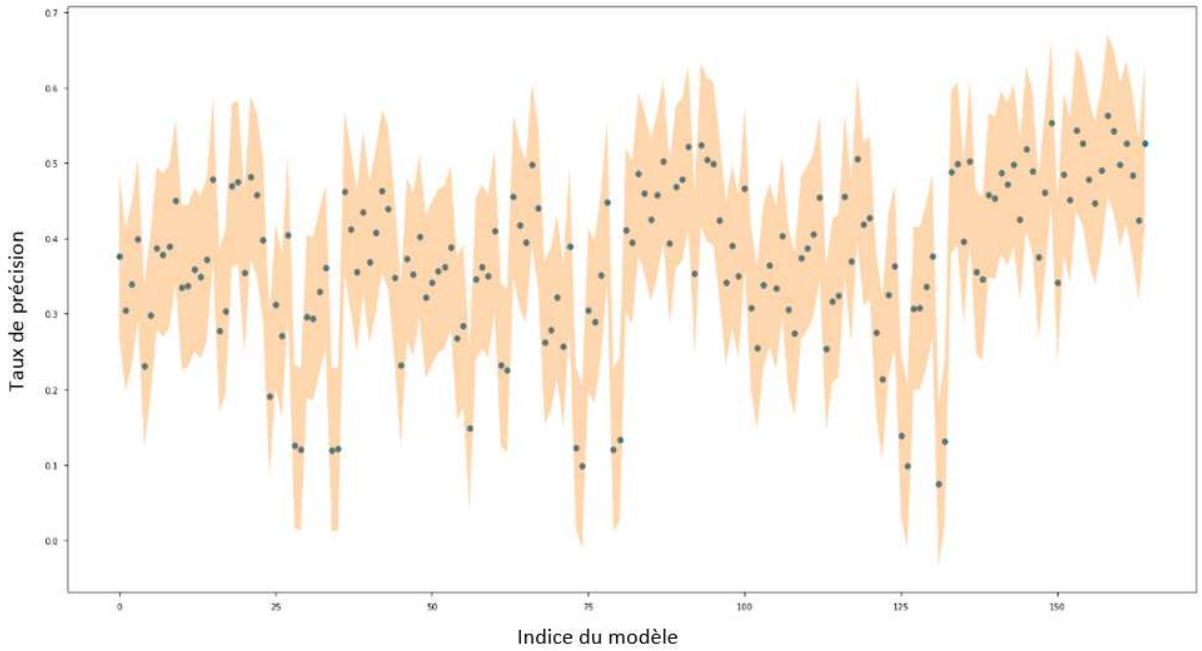

FIGURE A.2 – Expérimentation 1 (P5000 - 16 GB RAM) : avec 2 époques d'entraînement, 200 architectures explorées et 10 jours d'exécution, le taux de précision de la meilleure architecture a atteint 60%.

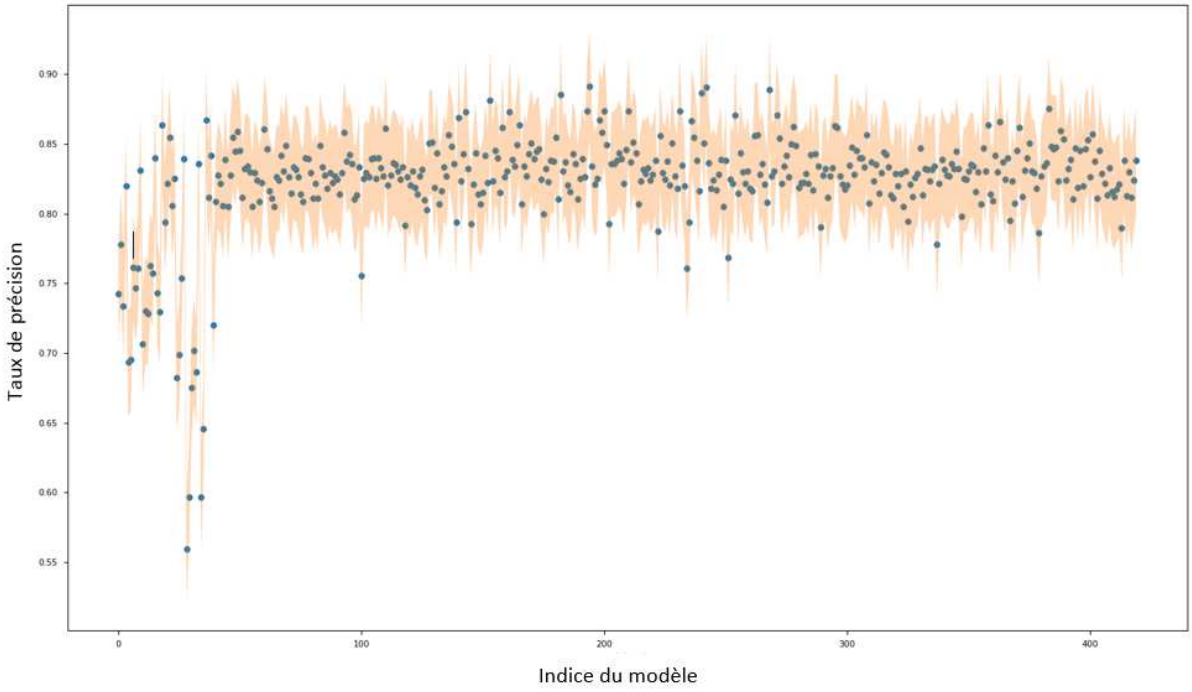

FIGURE A.3 – Expérimentation 2 (P5000 - 16 GB RAM) : avec 100 époques d'entraînement, 450 architectures explorées et 30 jours d'exécution, le taux de précision de la meilleure architecture a atteint 89,15%

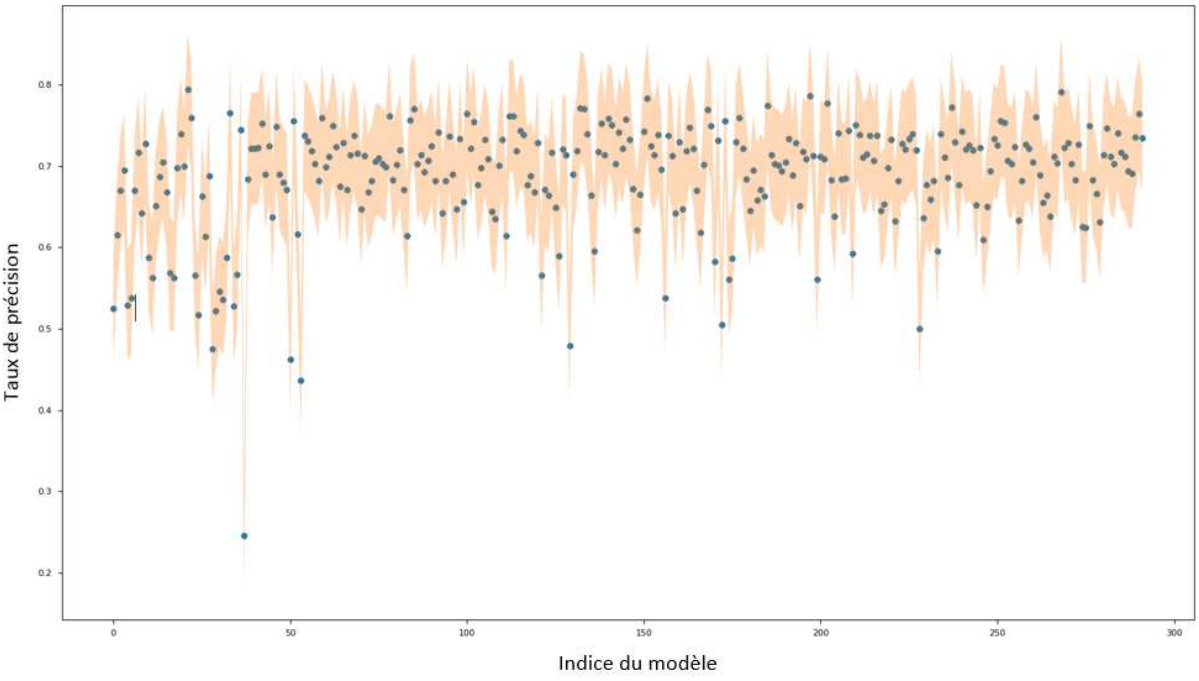

FIGURE A.4 – Expérimentation 3 (DGX – 128 GB RAM, 4 GPUs Tesla V100) : avec 10 époques d'entraînement, 300 architectures explorées et 3 jours d'exécution, le taux de précision de la meilleure architecture a atteint 80%.

Annexe B

Perspective : approche hybride

La combinaison de l'efficacité des échantillons du RL model-based et de la précision du RL model-free permet d'incorporer une source de données supplémentaires accélérant l'apprentissage des tâches de contrôle continu. En règle générale, les systèmes model-based tentent d'apprendre un modèle des dynamiques de l'environnement qui peut être utilisé par la suite, de différentes manières, pour améliorer l'apprentissage d'une politique. Plus spécifiquement, le schéma hybride que nous proposons comme perspective consiste à utiliser un RL model-based pour apprendre les dynamiques de l'environnement de conduite et une MPC pour initialiser la politique de notre approche MRL. L'initialisation du gradient de la politique en utilisant des trajectoires optimisées, observées au cours de l'entraînement de l'agent model-based, améliore l'efficacité de l'échantillonnage de la politique apprise. Concrètement, l'architecture hybride illustrée dans la figure B.1 comprend un module model-based constitué d'un modèle de dynamiques et d'un contrôleur MPC, et un module model-free comportant l'acteur et le critique détaillés dans notre approche MRL (4.5.1).

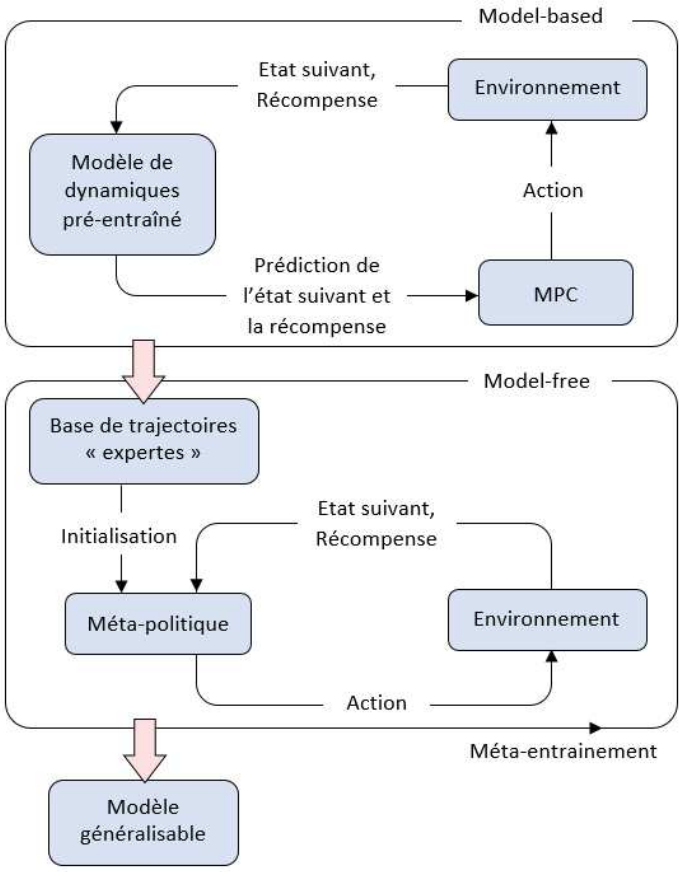

Figure B.1 – Approche hybride proposée comme perspective.

Le modèle des dynamiques de l'environnement consiste en un DNN qui prend l'état et l'action actuels en entrée et prédit l'état suivant et sa récompense immédiate (Sutton & Barto, 2018). Il joue ainsi le rôle d'un guide de référence et d'un composant médiateur entre la perception des scènes de conduite et la MPC. A partir de la prédiction du modèle des dynamiques, le contrôleur MPC optimise un indice de performance (fonction de coût) à chaque pas de temps t par rapport à une entrée de séquence de mouvements futurs. Le premier de ces mouvements optimaux (action de contrôle) est appliqué à l'environnement à l'instant t. A $t+1$, une nouvelle optimisation est résolue sur un horizon de prédiction décalé de 1 pas de temps. La MPC permet en général à un véhicule autonome de prévoir des actions anticipées (par exemple freinage préventif avant d'atteindre un virage) et d'éviter des obstacles qui surgissent soudainement à proximité du véhicule. L'étape suivante consiste à entraîner une politique par descente de gradient à apprendre les trajectoires « expertes » générées par la MPC (Nagabandi et al., 2018). Le module MRL incorpore la politique résultant de l'initialisation par MPC au niveau de l'acteur et continue son apprentissage model-free.

Bibliographie

Alpaydin, E. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010. ISBN 026201243X, 9780262012430. (Cité en page 9.)

Amari, S. and Douglas, S. C. Why natural gradient ? In *ICASSP*, pp. 1213–1216. IEEE, 1998. (Cité en page 43.)

Anderson, M. L. and Oates, T. A review of recent research in metareasoning and metalearning. *AI Magazine*, 28(1) :12, Mar. 2007. doi : 10.1609/aimag.v28i1.2025. URL https://www.aaai.org/ojs/index.php/aimagazine/article/view/2025. (Cité en page 71.)

Asis, K. D. and Sutton, R. S. Per-decision multi-step temporal difference learning with control variates. In *UAI*, 2018. (Cité en page 56.)

Baird, L. C. Advantage updating. Technical report, WRIGHT LAB WRIGHT-PATTERSON AFB OH, 1993. (Cité en page 55.)

Baker, B., Gupta, O., Naik, N., and Raskar, R. Designing neural network architectures using reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017. (Cité en pages vi, vii, 88, 89 et 106.)

Barron, A. R. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3) :930–945, 1993. (Cité en page 13.)

Barto, A. G., Sutton, R. S., and Anderson, C. W. Artificial neural networks. chapter Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems, pp. 81–93. IEEE Press, Piscataway, NJ, USA, 1990. (Cité en page 43.)

Basu, S., Ganguly, S., Mukhopadhyay, S., DiBiano, R., Karki, M., and Nemani, R. Deepsat : a learning framework for satellite imagery. In *Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems*, pp. 37. ACM, 2015. (Cité en page 15.)

Bellemare, M. G., Dabney, W., and Munos, R. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 449–458. JMLR. org, 2017. (Cité en page 40.)

Bellman, R. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957. (Cité en pages 3, 25, 26 et 35.)

Bellman, R. E. and Dreyfus, S. E. *Applied Dynamic Programming*. Princetown University Press, 1962. (Cité en pages 33, 39 et 55.)

Bengio, Y. Deep learning of representations : Looking forward. In *International Conference on Statistical Language and Speech Processing*, pp. 1–37. Springer, Springer Berlin Heidelberg, 2013. (Cité en page 2.)

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48. ACM, 2009. (Cité en page 38.)

Bengio, Y., Courville, A., and Vincent, P. Representation learning : A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8) :1798–1828, August 2013. ISSN 0162-8828. (Cité en pages 35, 84 et 106.)

Berenji, H. R. and Vengerov, D. A convergent actor-critic-based frl algorithm with application to power management of wireless transmitters. *IEEE Transactions on Fuzzy Systems*, 4(11) :478–485, 2003. (Cité en page 42.)

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24*, pp. 2546–2554. Curran Associates, Inc., 2011. (Cité en pages 87 et 88.)

Biggs, J. The role of meta-learning in study process. In *The British Psychological Society*, volume 55, pp. 185–212, 1985. (Cité en pages 2 et 66.)

Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738. (Cité en page 9.)

Böhmer, W., Springenberg, J. T., Boedecker, J., Riedmiller, M., and Obermayer, K. Autonomous learning of state representations for control : An emerging field aims to autonomously learn state representations for reinforcement learning agents from their real-world sensor observations. *KI-Künstliche Intelligenz*, 29(4) :353–362, 2015. (Cité en page 36.)

Bojarski, M., Yeres, P., Choromanska, A., Choromanski, K., Firner, B., Jackel, L. D., and Muller, U. Explaining how a deep neural network trained with end-to-end learning steers a car. *CoRR*, abs/1704.07911, 2017. (Cité en page 48.)

Boser, B. E., Guyon, I. M., and Vapnik, V. N. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pp. 144–152, New York, NY, USA, 1992. ACM. (Cité en page 9.)

Boyan, J. A. Technical update : Least-squares temporal difference learning. In *Machine Learning*, pp. 233–246, 2002. (Cité en page 40.)

Brazdil, P., Carrier, C., Soares, C., and Vilalta, R. *Metalearning : Applications to Data Mining*. Cognitive Technologies. Springer Berlin Heidelberg, 2008. ISBN 9783540732624. (Cité en page 67.)

Brazdil, P. B., Soares, C., and Pinto da Costa, J. Ranking learning algorithms : Using IBL and meta-learning on accuracy and time results. *Machine Learning*, 50(3) :251–277, 2003. (Cité en page 68.)

Breiman, L. Random forests. *Mach. Learn.*, 45(1) :5–32, 2001. (Cité en page 88.)

Brochu, E., Brochu, T., and de Freitas, N. A bayesian interactive optimization approach to procedural animation design. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 103–112, Goslar Germany, Germany, 2010. Eurographics Association. (Cité en page 87.)

Brodley, C. E. Recursive automatic bias selection for classifier construction. *Mach. Learn.*, 20(1-2) :63–94, July 1995. ISSN 0885-6125. (Cité en page 68.)

Brown, N., Sandholm, T., and Machine, S. Libratus : The superhuman ai for no-limit poker. In *IJCAI*, pp. 5226–5228, 2017. (Cité en page 36.)

Buehler, M., Iagnemma, K., and Singh, S. *The DARPA Urban Challenge : Autonomous Vehicles in City Traffic*. Springer Publishing Company, Incorporated, 1st edition, 2009. (Cité en page 44.)

Caffé, M. I. R., Perez, P. S., and Baranauskas, J. A. Evaluation of stacking on biomedical data. *Journal of Health Informatics*, 4(3), 2012. (Cité en page 73.)

Cai, H., Chen, T., Zhang, W., Yu, Y., and Wang, J. Efficient architecture search by network transformation. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7*, pp. 2787–2794, 2018a. (Cité en pages vi, 90, 91 et 93.)

Cai, H., Yang, J., Zhang, W., Han, S., and Yu, Y. Path-level network transformation for efficient architecture search. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 678–687, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018b. PMLR. (Cité en pages vi, 93, 95 et 96.)

Caruana, R. Multitask learning. *Mach. Learn.*, 28(1) :41–75, July 1997. ISSN 0885-6125. URL https://doi.org/10.1023/A:1007379606734. (Cité en pages 1 et 38.)

Chandak, Y., Theocharous, G., Kostas, J., Jordan, S., and Thomas, P. Learning action representations for reinforcement learning. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 941–950, Long Beach, California, USA, 09–15 Jun 2019. PMLR. (Cité en page 38.)

Chebotar, Y., Hausman, K., Zhang, M., Sukhatme, G., Schaal, S., and Levine, S. Combining model-based and model-free updates for trajectory-centric reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 703–711. JMLR.org, 2017. URL http://dl.acm.org/citation.cfm?id=3305381.3305454. (Cité en page 108.)

Chen, T., Goodfellow, I., and Shlens, J. Net2Net : Accelerating learning via knowledge transfer. In *International Conference on Learning Representations (ICLR)*, 2016. (Cité en page 90.)

Chollet, F. Xception : Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1800–1807, July 2017a. (Cité en page 93.)

Chollet, F. *Deep Learning with Python*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2017b. ISBN 1617294438, 9781617294433. (Cité en pages 17 et 20.)

Clavera, I., Rothfuss, J., Schulman, J., Fujita, Y., Asfour, T., and Abbeel, P. Model-based reinforcement learning via meta-policy optimization. In *CoRL*, volume 87, pp. 617–629. Proceedings of Machine Learning Research, 2018. (Cité en pages 2 et 67.)

Dayan, P. Reinforcement comparison. In *Connectionist Models*, pp. 45–51. Elsevier, 1991. (Cité en page 55.)

De Asis, K., Hernandez-Garcia, J. F., Holland, G. Z., and Sutton, R. S. Multi-step reinforcement learning : A unifying algorithm. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. (Cité en page 57.)

Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., and Ng, A. Y. Large scale distributed deep networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, pp. 1223–1231, USA, 2012. Curran Associates Inc. (Cité en page 95.)

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet : A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, 2009. (Cité en page 15.)

Deng, Y., Bao, F., Kong, Y., Ren, Z., and Dai, Q. Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*, 28(3) :653–664, 2016. (Cité en page 36.)

Domhan, T., Springenberg, J. T., and Hutter, F. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pp. 3460–3468. AAAI Press, 2015. (Cité en pages 87 et 95.)

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. CARLA : An open urban driving simulator. In Levine, S., Vanhoucke, V., and Goldberg, K. (eds.), *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pp. 1–16. PMLR, 2017. (Cité en pages 4, 46, 50, 60 et 61.)

Eiben, A. E. and Smith, J. E. *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 2nd edition, 2015. (Cité en page 87.)

Elman, J. L. Finding structure in time. *Cognitive science*, 14(2) :179–211, 1990. (Cité en page 81.)

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. IMPALA : Scalable distributed deep-RL with importance weighted actor-learner architectures. In *Proceedings*

of the 35th International Conference on Machine Learning, volume 80, pp. 1407–1416, Stockholmsmässan, Stockholm Sweden, 2018. (Cité en page 56.)

Fan, D. W., Chan, P. K., and Stolfo, S. J. A comparative evaluation of combiner and stacked generalization. In *Proceedings of AAAI-96 workshop on Integrating Multiple Learned Models*, pp. 40–46. Citeseer, 1996. (Cité en page 72.)

Favarò, F. M., Nader, N., Eurich, S. O., Tripp, M., and Varadaraju, N. Examining accident reports involving autonomous vehicles in california. *PLoS one*, 12(9), 2017. (Cité en page 44.)

Finn, C. and Levine, S. Meta-learning and universality : Deep representations and gradient descent can approximate any learning algorithm. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018. URL https://openreview.net/forum?id=HyjC5yWCW. (Cité en page 77.)

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1126–1135, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. (Cité en pages 2, 5, 74, 76, 77, 97 et 101.)

Floreano, D., Dürr, P., and Mattiussi, C. Neuroevolution : from architectures to learning, 2008. (Cité en page 87.)

François-Lavet, V., Taralla, D., Ernst, D., and Fonteneau, R. Deep reinforcement learning solutions for energy microgrids management. In *European Workshop on Reinforcement Learning (EWRL 2016)*, 2016. (Cité en page 36.)

Fridman, L., Brown, D. E., Glazer, M., Angell, W., Dodd, S., Jenik, B., Terwilliger, J., Patsekin, A., Kindelsberger, J., Ding, L., et al. Mit advanced vehicle technology study : Large-scale naturalistic driving study of driver behavior and interaction with automation. *IEEE Access*, 7 :102021–102038, 2019. (Cité en pages 5 et 45.)

Fu, M. C. Gradient estimation. *Handbooks in operations research and management science*, 13 :575–616, 2006. (Cité en page 41.)

Gaidon, A., Wang, Q., Cabon, Y., and Vig, E. Virtual worlds as proxy for multi-object tracking analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4340–4349, 2016. (Cité en page 46.)

Gama, J. and Brazdil, P. Characterization of classification algorithms. In *Proceedings of the 7th Portuguese Conference on Artificial Intelligence : Progress in Artificial Intelligence*, EPIA '95, pp. 189–200, London, UK, UK, 1995. Springer-Verlag. ISBN 3-540-60428-6. (Cité en page 73.)

Giraud-Carrier, C., Vilalta, R., and Brazdil, P. Introduction to the special issue on meta-learning. *Machine Learning*, 54(3) :187–193, Mar 2004. (Cité en page 73.)

Glynn, P. W. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10) :75–84, 1990. (Cité en page 41.)

Gomes, T. A., Prudêncio, R. B., Soares, C., Rossi, A. L., and Carvalho, A. Combining meta-learning and search techniques to select parameters for support vector machines. *Neurocomputing*, 75(1) :3–13, 2012. (Cité en pages 72 et 73.)

Graves, A., Mohamed, A.-r., and Hinton, G. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 6645–6649. IEEE, 2013. (Cité en page 81.)

Griffiths, T. L., Callaway, F., Chang, M. B., Grant, E., Krueger, P. M., and Lieder, F. Doing more with less : meta-reasoning and meta-learning in humans and machines. *Current Opinion in Behavioral Sciences*, 29 :24–30, 2019. (Cité en page 1.)

Grondman, I., Busoniu, L., Lopes, G. A. D., and Babuska, R. A survey of actor-critic reinforcement learning : Standard and natural policy gradients. *Trans. Sys. Man Cyber Part C*, 42(6) :1291–1307, 2012. (Cité en pages 41 et 57.)

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic : Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1861–1870, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/haarnoja18b.html. (Cité en pages 5 et 54.)

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pp. 2555–2565, Long Beach, California, USA, 09–15 Jun 2019. (Cité en page 109.)

Hasselt, H. V. Double q-learning. In *Advances in Neural Information Processing Systems*, pp. 2613–2621, 2010. (Cité en page 41.)

Hasselt, H. V. and Wiering, M. A. Reinforcement learning in continuous action spaces. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pp. 272–279, April 2007. (Cité en page 30.)

Hasselt, H. V., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pp. 2094–2100, 2016. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389. (Cité en page 40.)

Hausknecht, M. and Stone, P. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*, 2015. (Cité en page 38.)

He, J., Chen, J., He, X., Gao, J., Li, L., Deng, L., and Ostendorf, M. Deep reinforcement learning with a natural language action space. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*, pp. 1621–1630, Berlin, Germany, August 2016a. Association for Computational Linguistics.

doi : 10.18653/v1/P16-1153. URL https://www.aclweb.org/anthology/P16-1153. (Cité en page 38.)

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, June 2016b. (Cité en pages vi, 86 et 106.)

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow : Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. (Cité en page 40.)

Hilario, M. and Kalousis, A. Fusion of meta-knowledge and meta-data for case-based model selection. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pp. 180–191. Springer, 2001. (Cité en page 69.)

Hinton, G. E., Osindero, S., and Teh, Y.-W. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7) :1527–1554, 2006. (Cité en pages 3 et 15.)

Hinz, T., Navarro-Guerrero, N., Magg, S., and Wermter, S. Speeding up the Hyperparameter Optimization of Deep Convolutional Neural Networks. *International Journal of Computational Intelligence and Applications*, 17(2), June 2018. (Cité en page 96.)

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8) : 1735–1780, 1997. (Cité en page 81.)

Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5) :359–366, July 1989. URL http://dx.doi.org/10.1016/0893-6080(89)90020-8. (Cité en pages 13 et 35.)

Huang, G., Liu, Z., v. d. Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, July 2017. (Cité en page 87.)

Hubel, D. H. and Wiesel, T. N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1) :106–154, 1962. (Cité en page 16.)

Hughes, G. On the mean accuracy of statistical pattern recognizers. *IEEE transactions on information theory*, 14(1) :55–63, 1968. (Cité en page 21.)

Jaafra, Y., Laurent, J. L., Deruyver, A., and Naceur, M. S. Robust reinforcement learning for autonomous driving. In *Deep Reinforcement Learning Meets Structured Prediction, ICLR 2019 Workshop, New Orleans, Louisiana, United States, May 6, 2019*, 2019a. URL https://sites.google.com/view/iclr2019-drlstructpred/. (Cité en page 26.)

Jaafra, Y., Laurent, J. L., Deruyver, A., and Naceur, M. S. Meta-reinforcement learning for adaptive autonomous driving. In *Proceedings of the 1 st Adaptive Multitask Learning Workshop, ICML, Long Beach, California*, 2019b. URL https://www.amtl-workshop.org. (Cité en page 67.)

Jaafra, Y., Laurent, J. L., Deruyver, A., and Naceur, M. S. Reinforcement learning for neural architecture search : A review. *Image and Vision Computing, Elsevier*, 89 :57 – 66, 2019c. ISSN 0262-8856. doi : https://doi.org/10.1016/j.imavis.2019.06.005. URL http://www.sciencedirect.com/science/article/pii/S0262885619300885. (Cité en page 83.)

Jan, P., Sethu, V., and Stefan, S. Reinforcement learning for humanoid robotics. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids2003)*, Karlsruhe, Germany, Sept.29-30, 2003. (Cité en page 43.)

Jankowski, N. and Grabczewski, K. Universal meta-learning architecture and algorithms. In *Meta-Learning in Computational Intelligence*, 2011. (Cité en page 67.)

Jaritz, M., de Charette, R., Toromanoff, M., Perot, E., and Nashashibi, F. End-to-end race driving with deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2070–2075. IEEE, 2018. (Cité en pages 48 et 49.)

Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. What is the best multi-stage architecture for object recognition ? In *2009 IEEE 12th international conference on computer vision*, pp. 2146–2153. IEEE, Sept 2009. (Cité en page 14.)

Kakade, S. M. A natural policy gradient. In *Advances in neural information processing systems*, pp. 1531–1538, 2002. (Cité en page 42.)

Kalousis, A. and Hilario, M. Feature selection for meta-learning. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 222–233. Springer, 2001. (Cité en page 73.)

Kalra, N. and Paddock, S. M. Driving to safety : How many miles of driving would it take to demonstrate autonomous vehicle reliability ? *Transportation Research Part A : Policy and Practice*, 94 :182–193, 2016. (Cité en pages 5 et 46.)

Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS 2017)*, volume 54 of *Proceedings of Machine Learning Research*, pp. 528–536. PMLR, apr 2017. (Cité en page 87.)

Koch, G., Zemel, R., and Salakhutdinov, R. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning workshop*, 2015. (Cité en page 80.)

Kocić, J., Jovičić, N., and Drndarević, V. An end-to-end deep neural network for autonomous driving designed for embedded automotive platforms. *Sensors*, 19(9) :2064, 2019. (Cité en page 49.)

Koenig, N. and Howard, A. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *In IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2149–2154, 2004. (Cité en page 49.)

Konda, V. R. and Tsitsiklis, J. N. On actor-critic algorithms. *SIAM J. Control Optim.*, 42(4) :1143–1166, April 2003. URL https://doi.org/10.1137/S0363012901385691. (Cité en pages 39 et 42.)

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pp. 1097–1105, USA, 2012. Curran Associates Inc. (Cité en pages 15 et 85.)

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. Building machines that learn and think like people. *The Behavioral and brain sciences*, 40, 2017. (Cité en pages 1, 2 et 37.)

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pp. 396–404, 1990. (Cité en page 16.)

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11) :2278–2324, 1998. (Cité en pages vi, 9 et 84.)

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1) :1334–1373, January 2016. (Cité en pages 3, 25 et 36.)

Li, D., Yang, Y., Song, Y.-Z., and Hospedales, T. M. Learning to generalize : Meta-learning for domain generalization. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018. (Cité en page 74.)

Li, Z., Zhou, F., Chen, F., and Li, H. Meta-sgd : Learning to learn quickly for few shot learning. *In Proc. Conf. Neural Information Processing Systems*, 2017. (Cité en pages 77 et 78.)

Liang, X., Wang, T., Yang, L., and Xing, E. CIRL : controllable imitative reinforcement learning for vision-based self-driving. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y. (eds.), *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII*, volume 11211, pp. 604–620. Springer, 2018. (Cité en page 61.)

Liaw, A. and Wiener, M. Classification and Regression by randomForest. *R News*, 2(3) : 18–22, 2002. URL `http://CRAN.R-project.org/doc/Rnews/`. (Cité en page 9.)

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *ICLR*, 2016. (Cité en pages 3, 25, 30 et 36.)

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y. (eds.), *Computer Vision – ECCV 2018*, pp. 19–35, Cham, 2018. Springer International Publishing. (Cité en pages vi, 92, 93, 95, 106 et 110.)

Madrigal, A. C. Inside waymo's secret world for training self-driving cars. *The Atlantic*, 23, 2017. (Cité en page 46.)

Mankowitz, D. J., Mann, T. A., Bacon, P.-L., Precup, D., and Mannor, S. Learning robust options. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. (Cité en page 38.)

Michie, D., Spiegelhalter, D. J., Taylor, C. C., and Campbell, J. (eds.). *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, Upper Saddle River, NJ, USA, 1994. ISBN 0-13-106360-X. (Cité en page 68.)

Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. Meta-learning with temporal convolutions. *CoRR*, abs/1707.03141, 2017. URL http://arxiv.org/abs/1707.03141. (Cité en page 82.)

Mitchell, T. M. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. (Cité en pages 9 et 68.)

Mitchell, T. M. Machine learning and data mining. *Commun. ACM*, 42(11) :30–36, 1999. URL https://doi.org/10.1145/319382.319388. (Cité en page 68.)

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518(7540) :529–533, February 2015. (Cité en pages v, 3, 25, 35, 36, 38, 40, 41, 91 et 108.)

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pp. 1928–1937, New York, New York, USA, 2016. PMLR. (Cité en pages 42, 43, 56 et 61.)

Molina, M., Luna, J., C.Romero, and S.Ventura. Meta-learning approach for automatic parameter tuning : A case study with educational datasets. *International Educational Data Mining Society*, 2012. (Cité en page 72.)

Mooney, R., Shavlik, J., Towell, G., and Gove, A. An experimental comparison of symbolic and connectionist learning algorithms. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'89, pp. 775–780, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. (Cité en page 68.)

Morari, M. and Lee, J. H. Model predictive control : past, present and future. *Computers & Chemical Engineering*, 23(4-5) :667–682, 1999. (Cité en page 109.)

Muller, U., Ben, J., Cosatto, E., Flepp, B., and Cun, Y. L. Off-road obstacle avoidance through end-to-end learning. In *Advances in neural information processing systems*, pp. 739–746, 2006. (Cité en page 48.)

Munkhdalai, T. and Yu, H. Meta networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp. 2554–2563. Proceedings of Machine Learning Research, 2017. (Cité en page 83.)

Munos, R. and Moore, A. Variable resolution discretization in optimal control. *Machine learning*, 49(2-3) :291–323, 2002. (Cité en page 35.)

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1054–1062, 2016. (Cité en page 56.)

Murphy, K. P. *Machine learning : a probabilistic perspective*. MIT press, 2012. (Cité en page 10.)

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7559–7566. IEEE, 2018. (Cité en page 114.)

Nagabandi, A., Clavera, I., Liu, S., and Fearing, R. S. Learning to adapt in dynamic, real-world environments via meta-reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2019. (Cité en page 38.)

Naik, D. K. and Mammone, R. J. Meta-neural networks that learn by learning. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 1, pp. 437–442 vol.1, June 1992. (Cité en page 2.)

Nichol, A., Achiam, J., and Schulman, J. On first-order meta-learning algorithms. *CoRR*, abs/1803.02999, 2018. (Cité en pages 77 et 78.)

Niedzwiedz, C., Elhanany, I., Liu, Z., and Livingston, S. A consolidated actor-critic model with function approximation for high-dimensional pomdps. *AAAI 2008 Workshop for Advancement in POMDP*, pp. 37–42, 2008. (Cité en page 43.)

Nielsen, M. A. Neural networks and deep learning, 2018. URL `http://neuralnetworksanddeeplearning.com/`. (Cité en pages 17 et 24.)

Paden, B., Čáp, M., Yong, S. Z., Yershov, D., and Frazzoli, E. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, 1(1) :33–55, 2016. (Cité en page 47.)

Palanisamy, P. *Hands-On Intelligent Agents with OpenAI Gym : Your Guide to Developing AI Agents Using Deep Reinforcement Learning*. Packt Publishing, 2018. (Cité en page 50.)

Pan, Y., Cheng, C.-A., Saigol, K., Lee, K., Yan, X., Theodorou, E. A., and Boots, B. Imitation learning for agile autonomous driving. In *International Journal of Robotics Research (IJRR)*, 2019. (Cité en page 49.)

Parisi, S., Tangkaratt, V., Peters, J., and Khan, M. E. Td-regularized actor-critic methods. *Machine Learning*, 2019. (Cité en page 61.)

Pattanaik, A., Tang, Z., Liu, S., Bommannan, G., and Chowdhary, G. Robust deep reinforcement learning with adversarial attacks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 2040–2042, 2018. (Cité en page 54.)

Pavlov, I. P. and Anrep, G. V. *Conditioned Reflexes : An Investigation of the Physiological Activity of the Cerebral Cortex*. Oxford University Press : Humphrey Milford, 1927. (Cité en page 26.)

Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8. IEEE, 2018. (Cité en page 38.)

Peng, Y., Flach, P. A., Brazdil, P., and Soares, C. Decision tree-based data characterization for meta-learning. *IDDM-2002*, pp. 111–122, 2002. (Cité en page 69.)

Perez-Rua, J., Baccouche, M., and Pateux, S. Efficient progressive neural architecture search. In *BMVC*, pp. 150. BMVA Press, 2018. (Cité en page 88.)

Pfahringer, B., Bensusan, H., and Giraud-Carrier, C. G. Meta-learning by landmarking various learning algorithms. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 743–750, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. (Cité en page 67.)

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. Efficient neural architecture search via parameters sharing. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 4095–4104. PMLR, 10–15 Jul 2018. (Cité en pages vi, 93, 94 et 95.)

Pomerleau, D. A. Alvinn : An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pp. 305–313, 1989. (Cité en page 48.)

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, Workshop Track Proceedings*, 2018. (Cité en page 13.)

Rasmussen, C. E. and Williams, C. K. I. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X. (Cité en page 88.)

Rausch, V., Hansen, A., Solowjow, E., Liu, C., Kreuzer, E., and Hedrick, J. K. Learning a deep neural net policy for end-to-end control of autonomous vehicles. In *2017 American Control Conference (ACC)*, pp. 4914–4919. IEEE, 2017. (Cité en pages 48 et 49.)

Ravi, S. and Larochelle, H. Optimization as a model for few-shot learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017. (Cité en page 2.)

Rescorla, R. A. and Wagner, A. R. A theory of Pavlovian conditioning : Variations on the effectiveness of reinforcement and non-reinforcement. In Black, A. H. and Prokasy, W. F. (eds.), *Classical conditioning II : Current research and theory*, pp. 64–99. Appleton-Century-Crofts, New York, 1972. (Cité en page 26.)

Rice, J. R. The algorithm selection problem. *Advances in Computers*, 15 :65–118, 1976. (Cité en pages v, 66, 67, 71 et 72.)

Richter, S. R., Hayder, Z., and Koltun, V. Playing for benchmarks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2213–2222, 2017. (Cité en page 50.)

Roijers, D. M., Vamplew, P., Whiteson, S., and Dazeley, R. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48 :67–113, 2013. (Cité en page 39.)

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Parallel distributed processing : Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pp. 318–362. MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-68053-X. URL http://dl.acm.org/citation.cfm?id=104279.104293. (Cité en page 14.)

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(3) :211–252, dec 2015. (Cité en page 84.)

Russell, S. and Norvig, P. *Artificial Intelligence : A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009. ISBN 0136042597, 9780136042594. (Cité en pages 1, 3, 8, 27, 28, 29, 32 et 35.)

Rusu, A. A., Večerík, M., Rothörl, T., Heess, N., Pascanu, R., and Hadsell, R. Sim-to-real robot learning from pixels with progressive nets. In *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pp. 262–270. PMLR, 13–15 Nov 2017. URL http://proceedings.mlr.press/v78/rusu17a.html. (Cité en page 39.)

SAE Committee. Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems. *SAE Standard J*, 3016 :1–16, 2014. (Cité en page 44.)

Sallab, A. E., Abdou, M., Perot, E., and Yogamani, S. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19) :70–76, 2017. (Cité en page 48.)

Santamaría, J. C., Sutton, R. S., and Ram, A. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behaviour*, 6(2) :163–217, 1997. (Cité en page 30.)

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. Meta-learning with memory-augmented neural networks. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1842–1850, New York, New York, USA, 2016. PMLR. (Cité en pages 2 et 82.)

Schmidhuber, J. and Huber, R. Learning to generate artificial fovea trajectories for target detection. *Int. J. Neural Syst.*, 2(1-2) :125–134, 1991. (Cité en page 2.)

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897, 2015. (Cité en page 42.)

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017. (Cité en page 42.)

Schuster, M. and Paliwal, K. Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11) :2673–2681, 1997. (Cité en page 90.)

Shah, S., Dey, D., Lovett, C., and Kapoor, A. Airsim : High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pp. 621–635. Springer, 2018. (Cité en page 46.)

Shedivat, M., Bansal, T., Burda, Y., Sutskever, I., Mordatch, I., and Abbeel, P. Continuous adaptation via meta-learning in nonstationary and competitive environments. In *ICLR*. OpenReview.net, 2018. (Cité en pages 77 et 78.)

Shrestha, A. and Mahmood, A. Review of deep learning algorithms and architectures. *IEEE Access*, 7 :53040–53065, 2019. (Cité en page 3.)

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pp. 387–395. JMLR.org, 2014. (Cité en page 43.)

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529 :484–489, January 2016. (Cité en pages 3, 25 et 36.)

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015. (Cité en page 85.)

Singh, S. P. and Sutton, R. S. Reinforcement learning with replacing eligibility traces. *Machine learning*, 22(1-3) :123–158, January 1996. ISSN 0885-6125. URL http://dx.doi.org/10.1007/BF00114726. (Cité en page 107.)

Snell, J., Swersky, K., and Zemel, R. Prototypical networks for few-shot learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 4077–4087. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6996-prototypical-networks-for-few-shot-learning.pdf. (Cité en page 80.)

Soares, C., Petrak, J., and Brazdil, P. Sampling-based relative landmarks : Systematically test-driving algorithms before choosing. In *Portuguese conference on artificial intelligence*, pp. 88–95. Springer, 2001. (Cité en page 69.)

Soares, C., Brazdil, P. B., and Kuba, P. A meta-learning method to select the kernel width in support vector regression. *Machine Learning*, 54(3) :195–209, Mar 2004. (Cité en page 67.)

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout : A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1) : 1929–1958, 2014. (Cité en pages 15 et 85.)

Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. A hypercube-based encoding for evolving large-scale neural networks. *Artif. Life*, 15(2) :185–212, apr 2009. ISSN 1064-5462. (Cité en page 87.)

Stone, P., Brooks, R., Brynjolfsson, E., Calo, R., Etzioni, O., Hager, G., Hirschberg, J., Kalyanakrishnan, S., Kamar, E., Kraus, S., et al. Artificial intelligence and life in 2030. one hundred year study on artificial intelligence : Report of the 2015-2016 study panel. *Stanford University, Stanford, CA*, 6, 2016. (Cité en pages 35 et 36.)

Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P. H. S., and Hospedales, T. M. Learning to compare : Relation network for few-shot learning. *CoRR*, abs/1711.06025, 2017. URL http://arxiv.org/abs/1711.06025. (Cité en page 80.)

Sutton, R. S. and Barto, A. G. *Reinforcement Learning : An Introduction*. The MIT Press, second edition, 2018. (Cité en pages 3, 26, 27, 33, 35, 40, 42, 47, 55, 56, 95, 107 et 114.)

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In Solla, S. A., Leen, T. K., and Müller, K. (eds.), *Advances in Neural Information Processing Systems 12*, pp. 1057–1063. MIT Press, 2000. (Cité en pages 41, 42 et 55.)

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, June 2015. (Cité en pages vi, 85, 86 et 106.)

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. Inception-v4, inception-resnet and the impact of residual connections on learning. In Singh, S. P. and Markovitch, S. (eds.), *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pp. 4278–4284. AAAI Press, 2017. (Cité en pages 86, 87 et 106.)

Talpaert, V., Sobh, I., Kiran, B., Mannion, P., Yogamani, S., El-Sallab, A., and Pérez, P. Exploring Applications of Deep Reinforcement Learning for Real-world Autonomous Driving Systems. In *14th International Conference on Computer Vision Theory and Applications*, pp. 564–572. SCITEPRESS - Science and Technology Publications, 2019. URL https://hal.archives-ouvertes.fr/hal-02140352. (Cité en pages 47 et 48.)

Tan, F., Yan, P., and Guan, X. Deep reinforcement learning : from q-learning to deep q-learning. In *International Conference on Neural Information Processing*, pp. 475–483. Springer, 2017. (Cité en page 35.)

Tesauro, G. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3) :58–68, 1995. (Cité en page 36.)

Tesauro, G., Das, R., Chan, H., Kephart, J., Levine, D., Rawson, F., and Lefurgy, C. Managing power consumption and performance of computing systems using reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1497–1504, 2008. (Cité en page 39.)

Thomas Elsken, Jan Hendrik Metzen, F. H. Simple and efficient architecture search for convolutional neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018. (Cité en page 96.)

Thrun, S. and Pratt, L. Learning to learn. chapter Learning to Learn : Introduction and Overview, pp. 3–17. Kluwer Academic Publishers, Norwell, MA, USA, 1998. URL http://dl.acm.org/citation.cfm?id=296635.296639. (Cité en page 2.)

Vanschoren, J. Meta-learning : A survey. *CoRR*, abs/1810.03548, 2018. URL http://arxiv.org/abs/1810.03548. (Cité en page 67.)

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6000–6010, USA, 2017. Curran Associates Inc. ISBN 978-1-5108-6096-4. URL http://dl.acm.org/citation.cfm?id=3295222.3295349. (Cité en page 82.)

Vilalta, R. and Drissi, Y. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2) :77–95, 2002. (Cité en pages 2 et 67.)

Vilalta, R., Giraud-Carrier, C. G., Brazdil, P., and Soares, C. Using meta-learning to support data mining. *IJCSA*, 1(1) :31–45, 2004. (Cité en pages 69 et 71.)

Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D. Matching networks for one shot learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 3637–3645, USA, 2016. Curran Associates Inc. (Cité en pages vi, 79 et 80.)

Wahlström, N., Schön, T. B., and Deisenroth, M. P. From pixels to torques : Policy learning with deep dynamical models. *arXiv preprint arXiv :1502.02251*, 2015. (Cité en page 4.)

Wang, J. X., Kurth-Nelson, Z., Soyer, H., Leibo, J. Z., Tirumala, D., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. V. Learning to reinforcement learn. In *Proceedings of the 39th Annual Meeting of the Cognitive Science Society, CogSci 2017, London, UK, 16-29 July 2017*, 2017. URL https://mindmodeling.org/cogsci2017/papers/0252/index.html. (Cité en page 66.)

Wang, X.-S., Cheng, Y.-H., and Yi, J.-Q. A fuzzy actor-critic reinforcement learning network. *Information Sciences*, 177(18) :3764–3781, 2007. (Cité en page 43.)

Watkins, C. J. C. H. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, 1989. (Cité en page 91.)

Watkins, C. J. C. H. and Dayan, P. Q-learning. *Machine Learning*, 8(3) :279–292, May 1992. URL https://doi.org/10.1007/BF00992698. (Cité en page 39.)

Wei, T., Wang, C., and Chen, C. W. Modularized morphing of neural networks. In *International Conference on Learning Representations, Workshop*, 2017. (Cité en page 96.)

Widmer, G. Tracking context changes through meta-learning. *Mach. Learn.*, 27(3) :259–286, June 1997. ISSN 0885-6125. URL https://doi.org/10.1023/A:1007365809034. (Cité en page 1.)

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3) :229–256, May 1992. ISSN 1573-0565. doi : 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696. (Cité en pages 41 et 55.)

Wolpert, D. H. Stacked generalization. *Neural networks*, 5(2) :241–259, 1992. (Cité en page 72.)

Wolpert, D. H. and Macready, W. G. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1) :67–82, 1997. (Cité en pages 67 et 68.)

Wu, H. and Gu, X. Max-pooling dropout for regularization of convolutional neural networks. In *International Conference on Neural Information Processing*, pp. 46–54. Springer, 2015. (Cité en page 16.)

Xie, S., Zheng, H., Liu, C., and Lin, L. SNAS : stochastic neural architecture search. In *International Conference on Learning Representations*, 2019. (Cité en page 88.)

Xu, H., Gao, Y., Yu, F., and Darrell, T. End-to-end learning of driving models from large-scale video datasets. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2174–2182, 2017. (Cité en pages 5 et 48.)

Ying, C., Klein, A., Real, E., Christiansen, E., Murphy, K., and Hutter, F. Towards reproducible neural architecture search. *ICML 2018 - Reproducibility in Machine Learning Workshop*, abs/1902.09635, 2019. (Cité en page 96.)

Zahavy, T., Haroush, M., Merlis, N., Mankowitz, D. J., and Mannor, S. Learn what not to learn : Action elimination with deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 3562–3573, 2018. (Cité en page 38.)

Zeiler, M. and Fergus, R. Stochastic pooling for regularization of deep convolutional neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2013. (Cité en page 17.)

Zeiler, M. D. and Fergus, R. Visualizing and understanding convolutional networks. In Fleet, D., Pajdla, T., Schiele, B., and Tuytelaars, T. (eds.), *Computer Vision – ECCV 2014*, pp. 818–833, Cham, 2014. Springer International Publishing. (Cité en page 85.)

Zhong, Z., Yan, J., Wu, W., Shao, J., and Liu, C.-L. Practical block-wise neural network architecture generation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. (Cité en pages vi, vii, 91, 92, 95 et 106.)

Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., and Farhadi, A. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3357–3364. IEEE, 2017. (Cité en page 39.)

Zintgraf, L. M., Shiarlis, K., Kurin, V., Hofmann, K., and Whiteson, S. Fast context adaptation via meta-learning. *Thirty-sixth International Conference on Machine Learning (ICML)*, June 2019. (Cité en pages 77 et 78.)

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017. (Cité en pages vi, 88, 89, 90, 92, 93 et 95.)

Yesmina JAAFRA

Méta-Apprentissage par Renforcement pour le Contrôle Adaptatif

Résumé : Avec l'avènement de l'apprentissage profond, l'intelligence artificielle a franchi un pas décisif vers l'automatisation des tâches de grande dimensionnalité. L'apprentissage par renforcement a été révolutionné grâce aux nouveaux concepts de représentation profonde. Toutefois, l'extension de l'application de ce paradigme vers la sphère du monde réel a engendré des nouveaux défis de généralisation et d'optimisation face à la non-stationnarité des tâches. Dans cette thèse, nous nous intéressons à l'évolution méthodologique récente de l'apprentissage automatique vers le méta-apprentissage afin de remédier aux limites de l'apprentissage profond. L'approche proposée est construite sur la base d'une formulation Markovienne évoluant graduellement selon 2 axes d'amélioration. Au niveau de la robustesse de l'apprentissage, nous intégrons dans l'expression du gradient de la politique la fonction avantage estimée par une version généralisée de l'apprentissage par différence temporelle. Concernant la capacité de généralisation, nous implémentons un méta-optimiseur de type gradient apprenant les évaluations de l'algorithme de base à travers les tâches. Le modèle généralisable obtenu a montré une adaptation rapide aux nouveaux contextes de conduite autonome en milieu urbain.

Meta-Reinforcement Learning for Adaptive Control

Abstract : With the advent of deep learning supported by substantial technological advances, the Artificial Intelligence has taken a decisive step towards the automation of large dimension tasks. Reinforcement learning has been revolutionized thanks to new representation concepts introduced by deep learning. However, the extension of this paradigm application to the real world has triggered new challenges of generalization and optimization associated with higher level of tasks non-stationarity. In this thesis, we are interested in the recent methodological evolution of machine learning towards meta-learning in order to remedy the deep learning limits. The proposed approach is built on the basis of a Markovian formulation gradually evolving along 2 axes of improvement. In terms of learning robustness, we integrate in the policy gradient expression, the advantage function estimated by a generalized version of temporal difference learning. Regarding the generalization capacity, we implement a gradient meta-optimizer, learning the evaluations of the base-level algorithm across tasks. The resulting generalizable model showed a rapid adaptation to new contexts of urban autonomous driving.