الجمهـــــــــــــــــــــورية الجـــــــــــــــــــــزائـــــرية الديمـــقراطية الشعـــــــــــــــــــبية
RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
وزارة التعـــــــــــــــــــــــــــليم العالـــــــــــــــي والبـــــــحث العلمـــــــــــــــــــــــــي
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

جـــــــــــــــــــــــــامعة أبي بكر بالقايد - تلمســـــــــــان -
Université Abou Bekr BelKaïd - Tlemcen -

Faculté de Technologie

MEMOIRE

Présenté pour l'obtention du **diplôme** de **Master**

En : Télécommunications

Spécialité : Réseaux et Télécommunications

Par : Mlle. Mokhtari Wissam & Mlle. Laidouni Ahlem

Sujet :

Détection intelligente des intrusions dans les systèmes via l'apprentissage par renforcement

Soutenu publiquement, le 29 juin 2025, devant le jury composé de :

M. MERZOUGUI Rachid	Professeur	Université de Tlemcen	Président
Mme. TALEB Sarra	MCB	Université de Tlemcen	Examinateur
M. HADJILA Mourad	Professeur	Université de Tlemcen	Encadrant

Année Universitaire : 2024-2025

Remerciement

Nous remercions, tout d'abord "Allah", notre créateur, de nous avoir donné le courage, la volonté, la force et de la patience durant toutes ces années écoulées d'études, d'entamer et d'achever ce mémoire dans les bonnes conditions.

Nous remercions vivement notre encadrant, monsieur HADJILA Mourad, Maitre de conférences classe A à l'université de Tlemcen, d'avoir encadré ce travail avec ses énormes compétences et ses grandes qualités humaines, merci pour l'acuité de ses critiques, ses recommandations de perfectionniste et ses conseils éclairés ainsi pour sa disponibilité, sa gentillesse et toute l'assistance qu'il nous avons rapportées au cours de cette recherche.

Nous remercions également tous les membres de nos familles notamment nos chers parents pour les sacrifices qu'ils ont faits pour que nous achevions nos études avec excellence.

Nous tenons à adresser nos amples remerciements et le grand respect aux membres de jury : Monsieur MERZOUGUI Rachid, Professeur à l'université de Tlemcen pour avoir accepté de présider le jury de ce Mémoire, et Madame Taleb Sara, en tant qu'examinatrice, d'avoir accepté l'évaluation de ce travail, pour toutes leurs remarques et critiques signalées.

Nos remerciements s'adressent également à tous les professeurs, les enseignants, intervenants et toutes les personnes à tous les niveaux qui par leurs paroles, leurs écrits, leurs conseils et leurs critiques ont guidé nos réflexions et ont accepté de nous rencontrer et de répondre à nos questions et ont contribué à bien pour terminer nos recherches et pour l'ensemble du savoir que nous avons acquis grâces à eux durant nos études.

Enfin, nous adressons toutes nous sympathies à tous nos collègues et nos amis, qui ont toujours été là pour nous, pour leur soutien inconditionnel et leurs encouragements qui ont été d'une grande ampleur et pour tous les moments agréables. Un grand merci pour tous ceux qui ont contribué de près ou de loin pour la réalisation de ce mémoire. À toutes ces personnes, nous présentons notre remerciement, nos respects et notre gratitude

À nos familles, qui nous ont soutenus tout au long de ce parcours.

Table des matière

Liste des Figures

Liste des Tableaux

Liste des Abréviations

2FA : Authentification à deux facteurs
AES: Advanced Encryption Standard
ARP: Address Resolution Protocol
A2C: Advantage Actor-Critic
A3C: Asynchrone Advantage Actor-Critic
Adam : Adaptive Moment Estimation
AWID : Aegean Wi-Fi Intrusion Dataset
CNN : Convolutional Neural Network
CIC-IDS-2017 : Canadian Institute for Cybersecurity - Intrusion Detection System - 2017
DoS: Denial of Service
DDoS: Distributed Denial of Service
DES: Data Encryption Standard
DR: Detection Rate
DRL: Deep Reinforcement Learning
DQN : Deep Q-Networks
DDQN: Double Deep Q-Network
DDPG: Deep Deterministic Policy Gradient
FTP: File Transfer Protocol
HIDS: Host-based IDS
IA: Intelligence Artificielle
IDS: Intrusion Detection System
IoT: Internet of Things
MSE: Mean Squared Error
MITM: Man-in-the-Middle
MAC: Media Access Control
MFA : Authentification multifacteur
MIB: Management Information Base
ML : Machine Learning
MDP : Processus de Décision Markovien
MCTS: Monte Carlo Tree Search
MPC: Model Predictive Control
MLPs: Multilayer Perceptron
NIST: National Institute of Standards and Technology
NGFW: Next-generation firewall
NIDS: Network-based IDS
POMDP: Partially Observable Markov Decision Process
PER : Rejeu Expérience Prioritaire
PPO : Proximal Policy Optimization
R2L: Remote to Local

RSA: Rivest-Shamir-Adleman
RL: Reinforcement Learning
RNN: Recurrent Neural Network
ReLU: Rectified Linear Unit
SAC: Soft Actor-Critic
SGD : Stochastic Gradient Descent
SQL: Structured Query Language
SSH: Secure Shell
TD : Temporal Difference
TRPO: Trust Region Policy Optimization
TRPO: Trust Region Policy Optimization
U2R: User to Root
VPN: Virtual Private Network
XSS: Cross-Site Scripting

Résumé

L'émergence de cybermenaces de plus en plus sophistiquées et furtives rend indispensable le développement de systèmes de détection d'intrusions (IDS) intelligents, capables de prendre des décisions de manière autonome sans intervention humaine constante. Dans ce contexte, l'apprentissage par renforcement profond (Deep Reinforcement Learning, DRL) constitue une solution prometteuse, permettant aux agents d'apprendre des stratégies de détection optimales à travers leur interaction avec l'environnement. Ce mémoire explore l'application des réseaux de neurones à Q-learning profond (Deep Q-Networks, DQN) à la détection d'intrusions, en évaluant le modèle sur trois ensembles de données de référence : NSL-KDD, CIC-IDS2017 et AWID. Une attention particulière est portée à l'optimisation des hyperparamètres afin d'améliorer les performances d'apprentissage. Les résultats expérimentaux obtenus sur ces trois ensembles de données montrent qu'un facteur d'actualisation faible ($\gamma = 0{,}001$) permet d'atteindre une précision de détection significativement améliorée pour différentes classes d'intrusions. Le système IDS basé sur DQN proposé surpasse plusieurs approches classiques d'apprentissage automatique. Ces résultats confirment l'efficacité du DRL pour améliorer les performances des IDS et leur capacité d'adaptation face à l'évolution constante des cyberattaques dans les infrastructures réseaux modernes.

Mots-clés — Sécurité des réseaux, Deep Q-Networks, Apprentissage profond, Apprentissage par renforcement, Epsilon-Greedy, Systèmes de détection d'intrusions, NSL-KDD, CIC-IDS2017, AWID, Intelligence artificielle.

Abstract

The emergence of increasingly sophisticated and evasive cyber threats necessitates the development of intelligent intrusion detection systems (IDS) capable of autonomous decision-making without constant human intervention. In this context, deep reinforcement learning (DRL) offers promising solutions by enabling agents to learn optimal detection strategies through interaction with their environment. This thesis explores the application of Deep Q-Networks (DQN) to intrusion detection, evaluating the model on three benchmark datasets : NSL-KDD, CIC-IDS2017, and AWID. Special attention is given to the tuning of hyperparameters to optimize learning performance. Experimental results on the three datasets demonstrate that a low discount factor ($\gamma = 0.001$) leads to significantly improved detection accuracy across various intrusion classes. Our proposed DQN-based IDS outperforms several traditional machine learning approaches. These findings underline the effectiveness of DRL in enhancing IDS performance and adapting to the evolving landscape of cyber-attacks in modern network infrastructures.

Index Terms — Network Security, Deep Q-Networks, Deep Learning, Reinforcement Learning, Epsilon-Greedy, Intrusion Detection Systems, NSL-KDD, CIC-IDS2017, AWID, Artificial Intelligence

الملخص

أدى تصاعد التهديدات الإلكترونية، التي أصبحت أكثر تعقيدًا وتخفيًا، إلى ضرورة تطوير أنظمة ذكية لكشف التسللات(IDS) ، قادرة على اتخاذ قرارات بشكل مستقل دون الحاجة إلى تدخل بشري مستمر. في هذا السياق، يُعد التعلم العميق بالتعزيز (Deep Reinforcement Learning - DRL) من الحلول الواعدة، حيث يسمح للوكلاء بتعلم استراتيجيات فعّالة للكشف من خلال التفاعل المباشر مع البيئة. يتناول هذا البحث دراسة تطبيق الشبكات العصبية العميقة المعتمدة على خوارزمية Q-Learning (Deep Q-Networks - DQN) في مجال كشف التسللات، مع تقييم النموذج على ثلاث مجموعة بيانات مرجعية : NSL-KDD، CIC-IDS2017 و AWID.كما يركز البحث على تحسين ضبط المعاملات الفائقة (Hyperparameters) بهدف تعزيز كفاءة التعلم. أظهرت النتائج التجريبية على هذه المجموعات الثلاث أن استخدام عامل خصم منخفض ($\gamma = 0.001$) يساهم في تحسين ملحوظ لدقة الكشف عبر مختلف أنواع الهجمات. وقد تفوق نظام الكشف المقترح، المعتمد علىDQN ، على العديد من نماذج التعلم الآلي التقليدية. تؤكد هذه النتائج فعالية DRL في تعزيز أداء أنظمة الكشف، وقدرتها على التكيف مع التطور المستمر للهجمات السيبرانية في بيئات الشبكات الحديثة.

الكلمات المفتاحية :أمن الشبكات، Deep Q-Networks، التعلم العميق، التعلم بالتعزيز، استراتيجية epsilon-greedy، أنظمة كشف التسللات، NSL-KDD، CIC-IDS2017، AWID، الذكاء الاصطناعي.

Introduction Générale

Face à l'augmentation constante des cybermenaces en termes de volume, de complexité et de furtivité, les systèmes de détection d'intrusions (IDS – *Intrusion Detection Systems*) traditionnels montrent aujourd'hui leurs limites. Généralement conçus sur des bases statiques, ces systèmes classiques requièrent une supervision humaine continue, manquent de capacités d'adaptation face aux attaques émergentes, et sont souvent inefficaces contre les menaces inconnues, notamment les attaques de type *zero-day*. Dans ce contexte, le besoin de solutions plus intelligentes, autonomes et évolutives devient de plus en plus crucial.

L'apprentissage par renforcement (RL – *Reinforcement Learning*) s'impose comme une alternative prometteuse pour répondre à ces défis. En permettant à un agent d'apprendre par interaction avec son environnement, le RL rend possible l'élaboration de stratégies de défense dynamiques et adaptées. Grâce à un mécanisme d'essais-erreurs, l'agent peut affiner ses décisions en fonction des récompenses ou pénalités reçues, et ainsi améliorer ses performances au fil du temps sans nécessiter d'expertise humaine préalable. Cette approche a donné naissance à de nombreuses solutions d'IDS autonomes dans des domaines variés, tels que l'Internet des Objets (IoT), les réseaux sans fil ou encore le cloud computing.

Dans ce travail, nous explorons l'application d'une méthode issue de l'apprentissage par renforcement profond : le Deep Q-Network (DQN). En combinant l'algorithme de Q-Learning avec la puissance des réseaux de neurones profonds, le DQN permet d'estimer les valeurs d'action Q(s,a) associées à chaque état de l'environnement, et de sélectionner l'action optimale à entreprendre à chaque instant. Cette capacité de généralisation du DQN est particulièrement adaptée à des environnements complexes comme ceux liés à la cybersécurité. Afin de simuler ces environnements, nous utilisons trois ensembles de données de référence dans le domaine de la détection d'intrusions : NSL-KDD, CIC-IDS2017 et AWID. Ces ensembles de données sont prétraités rigoureusement pour garantir une qualité optimale d'apprentissage, notamment par la suppression des valeurs manquantes ou redondantes, l'encodage des variables catégorielles, la normalisation des données numériques, et l'équilibrage des classes.

Notre approche repose sur la conception d'un agent DQN autonome, entraîné dans ces environnements simulés, afin d'apprendre à détecter et classifier différents types d'attaques. Le modèle est évalué selon quatre métriques standards de classification permettant une mesure complète de ses performances. Les principales contributions de ce mémoire sont organisées comme suit :

- **Chapitre 1** introduit les systèmes de détection d'intrusions (IDS) à travers leurs fondements théoriques, leur architecture, leurs fonctions principales, ainsi que les différentes approches et typologies existantes. Ce chapitre met également en lumière les limites des IDS classiques face à la complexité croissante des cyberattaques modernes, ce qui justifie le recours à des solutions plus intelligentes et autonomes.

- **Chapitre 2** présente les notions clés de l'apprentissage automatique, avec un focus particulier sur l'apprentissage par renforcement (RL) et l'apprentissage par renforcement profond (DRL). Il propose une exploration des principaux algorithmes utilisés dans ce domaine, notamment DQN et DDQN, ainsi qu'une explication des concepts fondamentaux tels que les processus de décision markoviens, les fonctions de valeur et les stratégies d'exploration. Le rôle des réseaux de neurones dans l'approximation des politiques et l'influence des hyperparamètres sur les performances y sont également abordés.

- **Chapitre 3** est consacré à l'implémentation de notre approche. Il décrit les ensembles de données utilisés, le modèle DQN développé, les métriques d'évaluation retenues, ainsi que les résultats expérimentaux obtenus dans différents contextes d'intrusion.

Enfin, nous concluons notre mémoire par une synthèse des apports de ce travail et des perspectives futures pour améliorer encore les performances des IDS intelligents.

Chapitre 1

Système de détection d'intrusions

1.1 Introduction

1.1.1 Contexte

La sécurité informatique est devenue un enjeu majeur dans les systèmes modernes. Avec l'essor des réseaux et leur interconnexion sur des infrastructures publiques, le nombre ainsi que la gravité des menaces ont rapidement augmenté. En 2025**,** des données récentes indiquent que près de 4 000 nouvelles cyberattaques sont enregistrées quotidiennement, soit environ une attaque toutes les 22 secondes. Cette statistique met en évidence l'intensification des menaces cybernétiques à l'échelle mondiale [1].

Bien que diverses techniques de prévention des intrusions aient été mises en place, telles que l'autorisation/l'authentification des utilisateurs (via mots de passe ou biométrie), la correction des erreurs de conception et de programmation, la protection de l'information par le chiffrement des données ou encore les pare-feu, ces mesures restent insuffisantes. En effet, à mesure que les systèmes deviennent plus complexes, il existe toujours des failles exploitables en raison d'erreurs de conception et de programmation, de mauvaises configurations et opérations du système, ou encore de diverses techniques de pénétration basées sur l'ingénierie sociale.

Les politiques qui équilibrent la commodité et le contrôle strict d'un système ainsi que l'accès à l'information rendent également impossible la sécurité totale d'un système opérationnel.

Pour renforcer la protection, de nombreux systèmes intègrent des systèmes de détection des intrusions (IDS - Intrusion Detection System), qui ajoutent une couche de sécurité supplémentaire.

Le terme « intrusion **»** fait référence aux tentatives de compromission de la confidentialité, de l'intégrité ou de la disponibilité d'une ressource, ou au contournement des mécanismes de sécurité d'un système informatique ou réseau [2].

Un IDS surveille les événements au sein d'un réseau ou d'un système afin de détecter des activités suspectes. Il analyse ces événements en quête d'anomalies et génère des alertes pour avertir les opérateurs en cas d'intrusion potentielle.

1.1.2 Les problèmes existants

À ce jour, deux problèmes majeurs persistent dans les systèmes de détection des intrusions (IDS) :
La quantité et la qualité des alertes générées.

Le premier problème est lié au volume des alertes émises. Un IDS a tendance à produire un grand nombre d'alarmes, ce qui peut rapidement submerger l'opérateur chargé de les analyser. Face à cette surcharge, ces alertes risquent d'être ignorées, rendant ainsi le système de détection inefficace.

Le second problème concerne la dégradation des performances. Le comportement normal d'un système évolue en permanence, tandis que de nouvelles attaques apparaissent continuellement. Cela entraîne une baisse de la qualité des alertes de deux manières : d'une part, l'IDS peut ne pas détecter certaines attaques réelles (faux négatifs) ; d'autre part, il peut générer trop d'alertes inutiles sur des comportements normaux (faux positifs). Une alerte manquée sur une attaque réelle met en péril la sécurité du système surveillé, tandis qu'un excès de fausses alertes gaspille le temps de l'opérateur et réduit sa confiance dans l'IDS. Si les alarmes ne sont pas fiables, elles risquent d'être systématiquement ignorées, rendant l'IDS inutile.

1.1.2.1 Gestion des alertes

Les IDS sont conçus pour surveiller un réseau ou un système informatique et signaler toute activité suspecte. Les notifications peuvent être envoyées sous différentes formes : e-mail, signal sonore, etc.

Les alertes générées par un IDS sont censées apporter une valeur ajoutée à l'opérateur système. Toutefois, pour être réellement efficaces, elles nécessitent une intervention humaine : l'opérateur ou l'analyste doit évaluer si une alerte correspond réellement à une intrusion ou s'il s'agit d'un faux positif.

Cependant, un IDS peut produire un volume extrêmement élevé d'alertes en peu de temps. Par exemple, un administrateur peut recevoir jusqu'à 1 000 e-mails par jour contenant chacun environ 8 alertes. Si ce dernier utilise les paramètres de détection par défaut, il doit impérativement ajuster le profil de signature des menaces. Sans cela, les alertes risquent d'être complètement ignorées au bout de deux semaines, faute de fiabilité [2].

Lorsqu'un IDS génère 8 000 alertes par jour, le temps alloué à l'opérateur pour analyser chaque alerte est très limité. Une estimation rapide montre qu'un seul opérateur travaillant 24h/24 ne disposerait que d'environ 10,8 secondes pour examiner chaque alerte [2]. Dans la réalité, cette durée est souvent insuffisante pour prendre une décision éclairée sur la nature de l'alerte. Pire encore, si le nombre d'opérateurs qualifiés est insuffisant, la situation devient critique.

1.1.2.2 Maintien des performances

L'efficacité d'un système de détection d'intrusion repose sur la capacité de l'utilisateur à réagir correctement aux véritables menaces. Toutefois, une grande partie des alertes générées sont des fausses alarmes. Une étude menée par [i] Neustar montre que 43% des entreprises sont victimes de faux positifs dans plus de 20% des cas. 15% des sondés affirment même ce taux peut atteindre 50% des alertes. La moyenne se situe à 26%, ce qui représente plus d'un quart des alertes reçues [3].

Les cyberattaques exploitant de nouvelles vulnérabilités apparaissent de manière rapide et fréquente, exposant en permanence le système à des risques. Les systèmes de détection basée sur la reconnaissance de signatures présentent des limites dans l'identification de ces nouvelles menaces (zero day attack).

Les systèmes de détection d'anomalies, qui s'appuient principalement sur l'exploration de données, nécessitent des données d'entraînement de haute qualité. Or, collecter toutes les données pertinentes sur de nouvelles attaques avant qu'elles ne surviennent est une tâche pratiquement impossible. De ce fait, les systèmes de détection d'anomalies ont souvent un taux de fausses alertes élevé.

De plus, avec l'évolution constante des matériels et logiciels, les comportements des utilisateurs et du système changent, entraînant une dégradation des performances des modèles de détection. Ainsi, un modèle statique n'est pas adapté à un système de détection d'intrusion, nécessitant un ajustement continu après son déploiement.

1.2 Les attaques et contre-mesures en sécurité réseau

1.2.1 Les piliers fondamentaux de sécurité

La sécurité informatique est un domaine vaste qui englobe un large éventail de techniques, de cibles à protéger et d'acteurs impliqués. Son objectif principal est de limiter la vulnérabilité des systèmes face aux menaces, qu'elles soient accidentelles ou intentionnelles. Bien que certains principes fassent l'objet de débats, il existe néanmoins des objectifs fondamentaux que l'on cherche à satisfaire dans la plupart des cas. Ces objectifs, également appelés attributs de la sécurité ou services de sécurité, définissent les garanties essentielles attendues par les utilisateurs en matière de protection des données et de fiabilité des systèmes.

1.2.1.1 Assurance

Le National Institute of Standards and Technology (NIST) définit l'assurance de la sécurité comme une mesure de confiance indiquant que les fonctionnalités de sécurité, les pratiques, les procédures et l'architecture d'un système d'information appliquent et font respecter la politique de sécurité de manière précise et efficace [4] pour prévenir, détecter et répondre aux cybermenaces.

1.2.1.2 Authentification

L'authentification est faite sous la forme d'une clé. Il s'agit du processus de vérification de l'identité d'un utilisateur, d'un processus ou d'un appareil avant d'accorder l'accès aux ressources du système. Il garantit qu'une entité est bien celle qu'elle prétend être, empêchant ainsi tout accès non autorisé et vol d'identité.

L'authentification repose sur des mécanismes de vérification tels que des mots de passe (sont généralement le facteur d'authentification le plus courant, et le plus ancien), des données biométriques, l'authentification à deux facteurs (2FA) ou multifacteur (MFA) (ces processus nécessitent la vérification réussie d'une ou plusieurs modalités avant d'accorder l'accès à un système) [5].

1.2.1.3 Autorisation

L'autorisation est faite sous la forme de permissions, est le processus consistant à accorder aux utilisateurs l'autorisation d'accéder à un emplacement physique ou à une ressource informationnelle telle qu'un document, une base de données, une application ou un site Web. L'authentification précède toujours l'autorisation. Les utilisateurs doivent prouver leur identité avant que le système puisse leur accorder l'accès.

1.2.1.4 Confidentialité

La confidentialité consiste à empêcher tout accès, lecture ou divulgation non autorisés, que ce soit lors du stockage, du traitement ou de la transmission. Elle assure que seules les personnes ou systèmes disposant des droits nécessaires peuvent consulter les informations sensibles. Pour assurer cette confidentialité, des mécanismes tels que le chiffrement, les contrôles d'accès et l'authentification sont mis en place afin de prévenir toute fuite ou exposition des données.

1.2.1.5 Disponibilité

La disponibilité en sécurité informatique désigne la capacité d'un système, d'un service ou d'une donnée à être accessible aux utilisateurs autorisés dans un délai approprié. Elle garantit que les ressources informatiques restent opérationnelles malgré les attaques (comme les attaques par déni de service DoS ou DDoS) ou les défaillances techniques, assurant ainsi un accès fiable et continu aux informations et aux services.

1.2.1.6 Intégrité

Larousse définit le terme « intégrité » comme un "état de quelque chose qui a conservé sans altération ses qualités, son état originel". En sécurité informatique, L'intégrité garantit que les données et les systèmes restent inchangés et protégés contre toute modification non autorisée. Elle se divise en deux aspects : l'intégrité des données, qui veille à ce que les informations restent exactes, cohérentes et non altérées pendant leur stockage, traitement ou transmission ; et l'intégrité des systèmes, qui assure qu'un système fonctionne sans manipulation et selon son objectif initial.

1.2.1.7 Non-répudiation

La non-répudiation est une propriété qui garantit qu'aucune des parties dans une communication ne puisse nier avoir effectué une action ou reçu un message. Elle est essentielle pour assurer la fiabilité des transactions et des échanges d'informations. En pratique, cela implique qu'un émetteur ne peut pas contester l'envoi de données, et qu'un récepteur ne peut pas nier la réception d'un message.

1.2.1.8 Responsabilité

La responsabilité est un concept essentiel qui vise à assurer que les actions effectuées au sein d'un système d'information puissent être attribuées de manière unique aux entités correspondantes. Cet objectif

garantit que les utilisateurs et les administrateurs sont tenus responsables des comportements ayant un impact sur la sécurité des informations [2].

L'illustration suivante (voir Figure 1.1) montre la relation entre les cinq principaux objectifs de sécurité : Confidentialité, Intégrité, Disponibilité, Responsabilité (Accountability) et Assurance. Elle met en évidence le fait que ces concepts sont interconnectés et qu'une politique de sécurité efficace doit garantir l'ensemble de ces objectifs pour assurer une protection optimale des données et des systèmes.

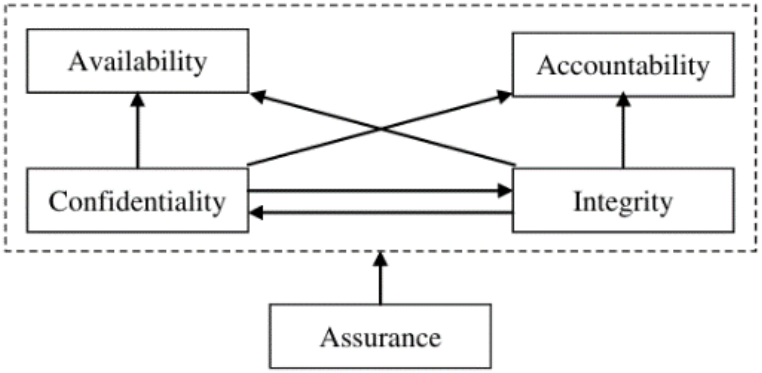

Figure 1.1 - Relation entre les cinq principaux objectifs de sécurité [2]

1.2.2 Types d'attaques

Chaque ordinateur connecté à un réseau informatique est potentiellement exposé à des attaques. Une "attaque" désigne l'exploitation d'une vulnérabilité présente dans un système informatique (qu'il s'agisse du système d'exploitation, d'un logiciel ou même de l'utilisateur) dans un but malveillant, souvent inconnu de l'opérateur du système.

De telles attaques surviennent en continu sur Internet, avec un nombre moyen impressionnant d'attaques par minute ciblant chaque appareil connecté. La majorité de ces offensives sont lancées de manière automatisée à partir d'ordinateurs compromis (infectés par des virus, chevaux de Troie, vers, etc.), généralement à l'insu de leurs propriétaires. Elles peuvent également être le fruit d'actions ciblées menées par des hackers. Face à cette menace omniprésente, il devient indispensable de connaître les principales formes d'attaques existantes afin de mettre en place des mesures préventives et des stratégies de défense efficaces.

1.2.2.1 Attaque DoS & DDoS

Une attaque par déni de service (DoS, pour *Denial of Service*) est une tentative malveillante visant à perturber ou à interrompre le bon fonctionnement d'un serveur, d'un service ou d'un réseau ciblé, en le submergeant de requêtes illégitimes qui provoquent un crash ou un ralentissement [6]. Cette surcharge empêche le système ciblé de traiter les demandes légitimes des utilisateurs, rendant ainsi le service inaccessible, lent ou complètement indisponible.

Les attaques par déni de service peuvent prendre plusieurs formes, telles que :

- **Les attaques par saturation**, consistant à submerger une machine de requêtes, afin qu'elle ne soit plus capable de répondre aux requêtes réelles.
- **Les attaques par exploitation de vulnérabilités**, où l'attaquant exploite une faille dans le système ou le protocole de communication pour rendre le service ou l'application inutilisable.

Ces attaques sont généralement utilisées pour nuire à la réputation d'une organisation en ligne ou pour perturber son fonctionnement, sans pour autant voler ou altérer les données. Les attaques DoS peuvent être réalisées à l'aide d'un seul ordinateur ou, dans le cas d'attaques distribuées (DDoS), par un réseau de machines compromises (botnet), ce qui permet d'amplifier l'attaque et d'en rendre l'origine plus difficile à identifier.

Les attaques DoS n'ont pas pour but de voler des informations ou des actifs, mais elles peuvent causer d'importants dommages financiers et réputationnels en rendant des services essentiels inaccessibles pour une durée indéterminée.

Une attaque par déni de service distribué (en anglais Distributed Denial of Service, abrégé en DDoS) est une variante de l'attaque DoS, mais où l'attaque provient de plusieurs sources, souvent en utilisant un réseau de machines compromises (botnet). L'attaque DDoS amplifie l'impact d'une attaque DoS en envoyant des requêtes depuis de multiples dispositifs, rendant plus difficile l'identification et l'arrêt de l'attaque. Le but est toujours de rendre un service indisponible sur pour les utilisateurs légitimes, mais la méthode est beaucoup plus complexe et plus difficile à contrer [7].

1.2.2.2 Attaque par Injection

Une **attaque par injection** est une tentative malveillante d'influencer ou de détourner l'exécution normale d'un programme ou d'une requête, en y injectant des données non sécurisées. Ces données, interprétées par le système comme des instructions, permettent à l'attaquant de modifier le comportement prévu du programme. Ce type d'attaque fait partie des plus anciens et des plus dangereux visant les applications web [8].

Les deux formes les plus courantes d'attaques par injection sont :

- **L'injection SQL (SQLi),** cible les bases de données.
- **Le Cross-Site Scripting (XSS),** cible les navigateurs des utilisateurs pour injecter du code JavaScript malveillant.

a) L'injection SQL (SQLi)

L'injection SQL (Structured Query Language) est une attaque consistant à insérer des instructions SQL malveillantes dans un champ d'entrée d'une application web, dans le but de manipuler ou d'accéder illégalement à la base de données [9]. Cela peut lui permettre de : Consulter, modifier ou supprimer des données sensibles, Usurper l'identité d'un administrateur, Perturber les transactions ou accéder à l'ensemble du système.

Les attaques SQLi sont particulièrement dangereuses dans les applications qui ne valident pas correctement les entrées utilisateurs. Selon les statistiques, l'injection SQL représente près de 65,1 **%** des attaques ciblant les applications web [10].

b) Le Cross-Site Scripting (XSS)

Le Cross Site Scripting (XSS) est une attaque par injection de code dans laquelle un adversaire insère un code malveillant au sein d'un site web légitime [11]. Ce code s'exécute ensuite comme un script infecté dans le navigateur web de l'utilisateur, permettant à l'attaquant de voler des informations sensibles ou d'usurper l'identité de l'utilisateur. Les forums web, les tableaux de messages, les blogs et les autres sites permettant aux utilisateurs de publier leur propre contenu sont les plus vulnérables aux attaques XSS.

1.2.2.3 Attaques par Brute Force

Une attaque par force brute est une méthode de cyberattaque basée sur un processus automatisé d'essais et d'erreurs, visant à deviner des identifiants d'authentification (nom d'utilisateur, mot de passe) ou des clés de chiffrement. Cette technique consiste à tester systématiquement toutes les combinaisons possibles jusqu'à obtenir l'accès. Grâce à la puissance de calcul croissante et à l'utilisation de dictionnaires, les attaquants peuvent tester des millions de combinaisons en peu de temps [12], [13].

Les attaques par force brute peuvent cibler différents services en ligne ou protocoles d'accès à distance. Selon le vecteur exploité, elles se déclinent en plusieurs formes spécifiques, notamment :

a) Attaque par force brute sur SSH (Brute Force SSH)

Le protocole SSH (Secure Shell) est un protocole réseau cryptographique permettant une communication sécurisée sur un réseau non sécurisé, généralement utilisé pour l'administration distante de serveurs. Cependant, il peut devenir une cible pour les attaques par force brute, où des tentatives répétées d'identifiants sont effectuées jusqu'à trouver une combinaison valide [14].

Ce type d'attaque est facilité lorsque le service SSH est exposé sur Internet sans mesures de protection telles que le changement du port par défaut, l'utilisation de clés SSH, ou la limitation des tentatives de connexion.

b) Attaque par force brute sur FTP (Brute Force FTP)

Le protocole FTP (File Transfer Protocol) est un protocole de communication utilisé pour transférer des fichiers entre un client et un serveur. Lorsqu'il est mal configuré, notamment avec une authentification en clair ou des identifiants faibles, il devient vulnérable aux attaques par force brute. De nombreuses requêtes de connexion avec des identifiants différents sont alors envoyées pour accéder au serveur de fichiers.

c) Attaque par force brute sur les applications web (Brute Force Web)

Les applications web, en particulier sur les pages d'authentification, sont également exposées à ce type d'attaque. Une série de requêtes HTTP (souvent de type POST ou GET) est envoyée avec diverses combinaisons d'identifiants et de mots de passe jusqu'à trouver une correspondance valide. Ce processus peut inclure l'utilisation d'attaques par dictionnaire ou de combinaisons alphanumériques classiques.

1.2.2.4 Attaques Man-in-the-Middle (MITM)

Une attaque Man-in-the-Middle (MITM), ou attaque de l'homme du milieu, est une forme avancée de cyberattaque dans laquelle un acteur malveillant s'interpose furtivement dans une communication entre deux parties légitimes. En se plaçant dans ce canal d'échange, l'attaquant est capable de surveiller, intercepter, voire modifier les données échangées, sans que les deux parties ne s'en rendent compte [15]. Ce type d'attaque peut viser différents types de communications : échanges entre utilisateurs, connexions à des applications web, transactions financières, ou encore messageries instantanées.

Son objectif principal est de collecter illicitement des informations sensibles telles que des identifiants de connexion, des mots de passe ou des données personnelles en exploitant des failles dans les protocoles de communication, notamment en cas de chiffrement faible ou inexistant [16].

Au sein des réseaux locaux, l'une des méthodes les plus courantes pour réaliser une attaque MITM est l'ARP Spoofing, qui tire parti des failles du protocole ARP (Address Resolution Protocol). Ce protocole joue un rôle fondamental dans les réseaux locaux (LAN), en permettant l'association entre une adresse IP et une adresse MAC (Media Access Control), indispensables pour permettre la communication entre

les appareils sur un même réseau. Chaque machine maintient une table ARP locale contenant ces correspondances. En l'absence d'une correspondance, l'hôte émet une requête ARP diffusée sur le réseau pour obtenir l'adresse MAC liée à une IP donnée.

a) L'attaque ARP Spoofing

L'ARP Spoofing, ou usurpation ARP, est une forme spécifique d'attaque MITM consiste à injecter de fausses réponses ARP dans le réseau afin d'associer l'adresse IP d'un autre périphérique généralement une passerelle ou un serveur à l'adresse MAC de l'attaquant [17]. Cela redirige les paquets vers l'attaquant qui peut alors les intercepter, les analyser, voire les modifier avant de les retransmettre à leur destination finale.

1.2.2.5 Attaques d'Infiltration

L'infiltration, en cybersécurité, fait référence à un type d'attaque informatique où l'attaquant obtient un accès non autorisé à un réseau sécurisé, un système informatique ou une application dans le but de collecter des informations sensibles ou de préparer des attaques futures. Ces attaques impliquent souvent des techniques avancées telles que des campagnes de phishing, l'exploitation de vulnérabilités ou l'utilisation de données d'identification compromises pour obtenir un premier accès.

Une fois à l'intérieur, l'attaquant peut établir une porte dérobée (backdoor) pour poursuivre d'autres actions malveillantes, telles que l'exfiltration de données ou la perturbation des opérations du système.

a) Attaques par Backdoor

Une attaque par Backdoor est un type de violation de sécurité où l'attaquant installe une méthode cachée permettant de contourner l'authentification normale et d'accéder de manière non autorisée à un système ou à un réseau. Les backdoors sont souvent utilisées par les attaquants pour maintenir un accès constant à un système compromis, même après que la vulnérabilité initiale ait été corrigée ou découverte.

Ces accès cachés sont particulièrement redoutables, car ils peuvent rester actifs pendant des mois voire des années sans être détectés, contrairement aux attaques bruyantes ou destructrices, offrant ainsi aux cybercriminels une présence continue dans le système, tout en évitant les signes évidents d'une compromission.

1.2.2.6 Attaques de Scan (Probing Attacks)

Les attaques de scan et de reconnaissance, également appelées Probing Attacks, sont des techniques utilisées par des attaquants pour collecter des informations détaillées sur une cible (système, réseau ou infrastructure), dans le but d'identifier ses vulnérabilités potentielles avant de lancer une attaque plus intrusive [18]. Ce type d'attaque consiste à sonder les adresses IP, les ports ouverts, les services actifs, ainsi que la configuration du système et du réseau (topologie, systèmes d'exploitation, mécanismes de sécurité, etc.). Ces informations permettent ensuite à l'attaquant d'élaborer une stratégie d'intrusion adaptée.

L'un des types les plus courants est le port scanning (balayage de ports), qui consiste à analyser systématiquement les ports d'un hôte distant pour détecter lesquels sont ouverts et, par conséquent, potentiellement vulnérables. Ces ports peuvent ensuite être utilisés comme points d'entrée pour une intrusion, une compromission ou une attaque plus poussée. Les exemples de ce type d'attaque sont : Ipsweep, Nmap, etc.

Bien que le scan soit une étape légitime dans les tests d'intrusion éthiques (ethical hacking), il est également largement utilisé dans les campagnes d'attaques automatisées visant à cartographier des infrastructures cibles pour ensuite y déployer des exploits.

1.2.2.7 Botnets

Le botnet, contraction des mots "robot" et "network" (réseau), est un réseau de dispositifs informatiques infectés par des logiciels malveillants et contrôlés à distance par un attaquant [19], souvent appelé "bot-master". Ces dispositifs compromis, appelés "bots" ou "zombis", sont utilisés pour mener diverses cyberattaques de manière automatisée et massive.

Les botnets se forment généralement lorsque des utilisateurs non informés cliquent sur des liens malveillants, ouvrent des pièces jointes infectées ou visitent des sites web compromis. Une fois l'attaque réussie, un logiciel malveillant s'installe sur la machine de l'utilisateur, la transformant en un bot qui sera contrôlé à distance par l'attaquant via des canaux de communication sécurisés. Ils sont principalement utilisés pour des attaques comme les attaques par déni de service (DoS/DDoS). Les bots d'un botnet sont des appareils connectés à Internet, tels que des ordinateurs, serveurs ou objets connectés (IoT).

1.2.2.8 Attaques User to Root (U2R)

L'objectif de cette classe d'attaques est d'obtenir la main de l'administrateur système (Root) à partir d'un simple compte utilisateur par l'exploitation des vulnérabilités, Les exploits les plus connus sont les débordements réguliers des Buffers (buffer overflows) dus aux erreurs de programmation.

a) Buffer Overflow (Dépassement de Tampon)

Le dépassement de tampon est une technique d'attaque dans laquelle un programme informatique reçoit plus de données que celles qu'il attend. Lorsque cela se produit, le programme tente de placer ces données dans une zone mémoire au-delà de la taille allouée pour un tampon (espace de stockage réservé). Cette écriture excédentaire peut corrompre les données, provoquer un plantage du programme, ou être exploitée pour exécuter du code malveillant, permettant ainsi à un attaquant de prendre un contrôle non autorisé du système ciblé [20].

1.2.2.9 Attaque Remote to Local (R2L)

Une attaque Remote to Local désigne une technique où un attaquant, ayant un accès limité à un réseau ou à un système distant, cherche à obtenir des privilèges locaux sur une machine cible afin d'exploiter ce système. L'attaquant commence généralement par obtenir un accès non autorisé à une machine ou un service distant par l'exploitation des failles de sécurité telles que les bugs des applications installées dans la machine cible, faible authentification, ou les erreurs de configuration. L'objectif principal de cette attaque est d'obtenir un contrôle total du système local.

1.2.3 Contre-mesures des attaques

Afin de faire face aux différentes attaques informatiques, de nombreuses méthodes ont été conçues et proposées. Bien qu'elles ne permettent pas d'éliminer toutes les menaces, elles contribuent à renforcer le niveau de sécurité d'un système logiciel. Une contre-mesure n'est pas nécessairement destinée à contrer une seule attaque spécifique. Au contraire, certaines d'entre elles peuvent offrir une protection contre plusieurs types d'attaques.

Ces contre-mesures peuvent être mises en œuvre à différents niveaux d'un système informatique, notamment au niveau physique ou réseau, au niveau du système d'exploitation (incluant la gestion du système de fichiers), au niveau de la gestion des bases de données, ou encore au niveau des applications.

Dans ce qui suit, nous présenterons quelques grandes catégories de contre-mesures.

1.2.3.1 Le contrôle d'accès

Le contrôle d'accès regroupe l'ensemble des mécanismes permettant aux administrateurs d'un système d'exercer une influence de direction ou de restriction sur les comportements, l'utilisation et le contenu du système. Il permet de définir ce que les utilisateurs sont autorisés à faire, quelles ressources ils peuvent consulter et quelles opérations ils peuvent effectuer. Cette technique est également connue sous le nom d'autorisation.

Le contrôle d'accès joue un rôle fondamental dans la sécurité des systèmes logiciels, car il constitue la base d'objectifs de sécurité plus avancés tels que la confidentialité et l'intégrité. Lorsqu'il est bien mis en œuvre, il peut empêcher certaines attaques, notamment celles liées à l'accès non autorisé [2].

Il est important de distinguer les politiques de contrôle d'accès et les mécanismes de contrôle d'accès. Les politiques représentent des directives de haut niveau qui déterminent comment les accès doivent être contrôlés et selon quels critères les décisions d'autorisation sont prises. Les mécanismes, quant à eux, sont des fonctions logicielles ou matérielles de bas niveau, configurables pour appliquer concrètement une politique de contrôle d'accès donnée [21].

1.2.3.2 Les Pare-Feu (Firewall)

Un pare-feu (firewall en anglais) est une solution de sécurité essentielle pour les réseaux informatiques. Il sert de première ligne de défense en protégeant les informations privées des réseaux internes contre les utilisateurs des réseaux externes [22]. Un pare-feu peut être un ensemble de programmes (software firewall) ou un dispositif matériel (hardware firewall) situé à un point d'accès du réseau, qui filtre le trafic entrant et sortant en fonction d'une politique de sécurité définie.

Il contrôle et surveille le flux de données afin de prévenir l'accès non autorisé, les activités malveillantes et les menaces potentielles, en examinant les paquets de données entrant ou sortant d'un réseau et en les comparant à un ensemble de règles prédéfinies. Selon le type de pare-feu, ces règles peuvent être basées sur différents critères, tels que les adresses IP, les numéros de port et le contenu des paquets. Si un paquet correspond à une règle, il est soit autorisé à passer à travers le pare-feu, soit bloqué. Dans une entreprise, un pare-feu empêche les utilisateurs extérieurs d'accéder aux ressources internes et régule l'accès des utilisateurs internes aux ressources externes.

Il existe de nombreux types de pare-feu, souvent classés en fonction du système protégé, de la forme du dispositif, de l'emplacement dans le réseau et de la méthode de filtrage des données [23], incluant : Hardware firewall, Software firewall, Next-generation firewall (NGFW), Packet filtering firewall, Proxy firewall, etc.

a) Pare-feu matériel (Hardware Firewall)

Un pare-feu matériel est un dispositif autonome physique situé entre un réseau et ses dispositifs connectés. Il surveille et contrôle le trafic réseau entrant et sortant en fonction de politiques de sécurité prédéfinies.

Le déploiement d'un pare-feu matériel nécessite **des compétences spécialisées** pour garantir une configuration correcte et une gestion continue. Ce type de pare-feu est particulièrement adapté pour des réseaux de grande taille, car il protège tous les dispositifs du réseau contre les menaces externes et contrôle l'accès à partir d'internet ou d'autres réseaux externes.

b) Pare-feu logiciel (Software Firewall)

Un pare-feu logiciel fonctionne au sein d'un serveur ou d'une machine virtuelle. Il s'exécute sur un système d'exploitation orienté sécurité, généralement superposé à des ressources matérielles génériques. Il peut être déployé rapidement, souvent à l'aide d'outils d'automatisation basés sur le cloud.

Ce type de pare-feu est installé directement sur des ordinateurs ou des dispositifs individuels, protégeant chaque appareil contre les accès non autorisés. Les pare-feu logiciels sont particulièrement utiles pour les dispositifs individuels et offrent une flexibilité accrue, tout en étant moins coûteux à déployer que les pare-feu matériels.

c) Pare-feu à filtrage de paquets (Packet Filtering Firewalls)

Un pare-feu à filtrage de paquets examine les paquets de données qui passent à travers le pare-feu et prend des décisions quant à leur autorisation ou leur blocage en fonction de règles prédéfinies. Ces règles sont généralement basées sur des critères tels que les adresses IP source et de destination, les numéros de port et les protocoles. Bien que ces pare-feu soient relativement simples et efficaces, ils ne disposent pas de fonctionnalités de sécurité avancées et ne sont pas en mesure d'analyser le contenu des paquets, ce qui les rend moins efficaces contre les attaques sophistiquées.

d) Pare-feu proxy (Proxy Firewalls)

Les pare-feu proxy agissent comme des intermédiaires entre les réseaux internes et externes, interceptant et scrutant le trafic avant de le transmettre à sa destination prévue. En agissant en tant que proxy, ces pare-feu offrent une sécurité supplémentaire en masquant les adresses IP internes du réseau et en filtrant le contenu malveillant. Contrairement aux pare-feu traditionnels, les pare-feu proxy analysent en profondeur le trafic des protocoles d'application, offrant ainsi une protection renforcée contre les cyberattaques et les malwares. Ils empêchent également les connexions directes aux réseaux internes, contribuant ainsi à une protection plus stricte.

e) Pare-feu de nouvelle génération (Next-Generation Firewalls, NGFWs)

Les pare-feu de nouvelle génération (NGFW) combinent les fonctionnalités traditionnelles des pare-feu avec des fonctions de sécurité supplémentaires telles que la détection et la prévention des intrusions, la gestion des applications et l'inspection approfondie des paquets.

Ces pare-feu sont conçus pour offrir une protection avancée contre les menaces cybernétiques modernes. Les NGFW vont au-delà de l'inspection traditionnelle en fournissant une visibilité et un contrôle sur le trafic des applications, en intégrant des mécanismes de prévention des intrusions et en utilisant des informations de menaces provenant du cloud. Grâce à leur capacité à examiner plus en profondeur les paquets de données et à utiliser des sources d'intelligence sur les menaces, les NGFW sont capables de détecter des attaques sophistiquées et de mieux protéger les réseaux contre des menaces émergentes.

1.2.3.3 Réseau Privé Virtuel (VPN)

Un VPN (Virtual Private Network), ou Réseau Privé Virtuel, est un service de sécurité Internet qui permet de créer une connexion chiffrée et sécurisée entre un appareil et un réseau, en passant par Internet [24]. Le VPN est également très utilisé pour contourner la censure, accéder à des contenus géo-restreints, ou se connecter à un réseau d'entreprise en télétravail.

Lorsqu'un utilisateur se connecte à un VPN, son appareil établit un tunnel chiffré avec un serveur VPN où toutes les données échangées sont cryptées, ce qui empêche leur lecture par des tiers non autorisée. Le serveur VPN joue alors le rôle d'intermédiaire entre l'utilisateur et Internet, en masquant son adresse IP et sa localisation [25]. Cette technologie s'appuie sur des protocoles VPN tels qu'OpenVPN, WireGuard, ou IKEv2/IPSe.

On distingue principalement deux types de VPN :

a) VPN d'accès à distance (Remote Access VPN)

Dans un contexte professionnel, le VPN d'accès à distance, permet à un employé (un utilisateur individuel, appelé endpoint) de se connecter à distance au réseau de son entreprise de façon sécurisée, comme s'il était physiquement dans les locaux. Cette technologie est également disponible sur les appareils mobiles.

b) VPN site-à-site (Site-to-Site VPN)

Le VPN site-à-site relie directement deux réseaux distants entre eux, permettant aux appareils de chaque réseau de communiquer comme s'ils étaient localement connectés. Il utilise des équipements dédiés pour maintenir une connexion chiffrée permanente entre les deux sites.

1.2.3.4 La cryptographie

La cryptographie est une discipline de la sécurité informatique qui consiste à transformer des informations lisibles (texte en clair) en un format illisible (texte chiffré), à l'aide d'algorithmes et de clés spécifiques. Ce procédé permet de sécuriser la transmission des données sur des canaux potentiellement non sécurisés, en garantissant que seules les personnes autorisées, disposant de la clé appropriée, peuvent les déchiffrer et les comprendre. Elle est un pilier fondamental de la cybersécurité, largement utilisée dans les systèmes d'information, les communications électroniques et les transactions numériques. La figure 1.2 montre le principe de la cryptographie.

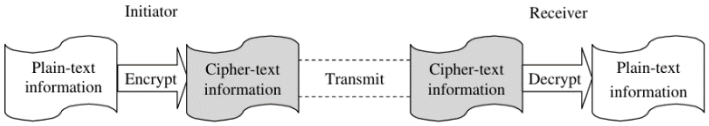

Figure 1.2 - La cryptographie [2]

Il existe trois principaux types de cryptographie, chacun ayant ses particularités et ses usages

a) La cryptographie à clé symétrique

La cryptographie à clé symétrique aussi appelée cryptographie à clé partagée, elle utilise une seule et même clé pour le chiffrement et le déchiffrement des données. L'expéditeur et le destinataire doivent donc disposer de cette clé commune. Bien que ce système soit rapide et relativement simple, il présente un défi majeur : le partage sécurisé de la clé. Les algorithmes AES (Advanced Encryption Standard) et DES (Data Encryption Standard) sont parmi les plus utilisés dans ce cadre [26].

b) La cryptographie à clé asymétrique

La cryptographie à clé asymétrique également connue sous le nom de cryptographie à clé publique, elle repose sur une paire de clés liées mathématiquement : une clé publique, partagée avec tous, et une clé privée, conservée secrète par son propriétaire. Lorsqu'un message est chiffré avec la clé publique du destinataire, seul ce dernier peut le déchiffrer à l'aide de sa clé privée. Ce système renforce la sécurité, en particulier pour l'échange de données sensibles. L'algorithme RSA (Rivest-Shamir-Adleman) est le plus emblématique de cette catégorie.

c) Les fonctions de hachage

Les fonctions de hachage contrairement aux deux systèmes précédents, les fonctions de hachage ne nécessitent aucune clé. Elles génèrent une empreinte numérique (ou valeur de hachage) de longueur fixe à partir d'un texte en clair. Cette empreinte est unique et irréversible, ce qui rend pratiquement impossible la reconstitution des données d'origine. Les fonctions de hachage sont largement utilisées pour stocker des mots de passe ou vérifier l'intégrité des données dans divers systèmes informatiques.

Par ailleurs, des systèmes hybrides combinent souvent les avantag**es des approches symétriques et** asymétriques, notamment pour sécuriser les échanges de clés dans des environnements à grande échelle.

1.3 Système de détection d'intrusions

À l'ère du numérique, les systèmes d'information et les réseaux informatiques occupent une place centrale dans notre société. Plus les volumes de données collectées, traitées et échangées augmentent, plus la sécurité de ces systèmes devient un enjeu crucial.

Une intrusion peut être définie comme une suite d'actions visant à compromettre l'intégrité, la confidentialité ou la disponibilité d'une ressource. L'identification des intrusions consiste à surveiller en continu les événements se produisant dans un système ou un réseau, puis à analyser ces événements pour détecter d'éventuels signes d'activités malveillantes. L'objectif est double : détecter les utilisateurs non autorisés, mais aussi les utilisateurs légitimes qui abusent de leurs privilèges.

Les systèmes de détection d'intrusions (IDS, pour *Intrusion Detection Systems*) peuvent être implémentés sous forme de logiciels ou de dispositifs matériels. Ils permettent d'automatiser le processus de surveillance et d'analyse des incidents de sécurité.

Pour être efficace, un IDS doit fonctionner de manière continue, s'adapter aux comportements évolutifs des utilisateurs, traiter de grands volumes de données, être configurable, avoir une faible empreinte sur les ressources système, et pouvoir redémarrer automatiquement après une panne sans nécessiter une réinitialisation manuelle.

Il existe principalement deux approches de détection : la détection d'anomalies et la détection d'abus. La première repose sur l'établissement de profils de comportement normal des utilisateurs du système (à partir d'indicateurs comme les actions sur le réseau), afin de détecter les écarts inhabituels pouvant signaler une intrusion. La seconde consiste à rechercher des signatures connues d'attaques ou de comportements malveillants.

1.3.1 Définition

Un système de détection d'intrusion (*IDS – Intrusion Detection System*) constitue un composant fondamental de la sécurité des réseaux informatiques. Il est conçu pour surveiller en continu les événements qui se produisent dans un système ou un réseau, dans le but de détecter automatiquement toute activité suspecte ou malveillante. L'objectif principal de l'IDS est d'identifier les incidents potentiels pouvant représenter des violations, ou des menaces imminentes de violation, des politiques de sécurité, des règles d'utilisation acceptable ou des bonnes pratiques en matière de cybersécurité.

L'IDS permet ainsi d'automatiser la détection des menaces en alertant les administrateurs de sécurité ou en transmettant les alertes à un système centralisé de gestion des incidents. Il peut être mis en œuvre sous forme de logiciel, de matériel, ou d'une combinaison des deux. Généralement placé à l'intérieur du réseau, l'IDS analyse les flux de données internes ainsi que les journaux d'activité, afin de repérer les tentatives d'intrusion.

Ces dernières peuvent provenir de différentes sources, notamment des malwares (tels que vers ou spywares), des attaques extérieures, ou encore des abus de privilèges par des utilisateurs légitimes.

Fonctionnant selon un modèle passif, l'IDS surveille l'activité du système et réagit aux violations détectées en accord avec les politiques de sécurité définies. Grâce à leur efficacité et leur capacité à repérer les intrusions avant qu'elles ne causent des dommages majeurs, ces systèmes se révèlent aujourd'hui indispensables pour renforcer la protection des infrastructures numériques [27].

1.3.2 Architecture d'un IDS

La conception d'un système de détection d'intrusion repose sur une architecture cohérente et intégrée, composée des éléments physiques et logiques. Cette combinaison assure un suivi continu des activités du réseau, une évaluation précise des événements et une réponse adaptée aux éventuelles menaces.

1.3.2.1 Les composants physiques d'un IDS

Un système de détection d'intrusion (IDS) repose traditionnellement sur trois composants physiques fondamentaux le capteur, l'analyseur et le gestionnaire. Ces éléments interagissent entre eux pour assurer une surveillance efficace du système et une réponse appropriée face aux menaces détectées.

La figure 1.3 illustre les interactions entre ces trois entités fonctionnelles.

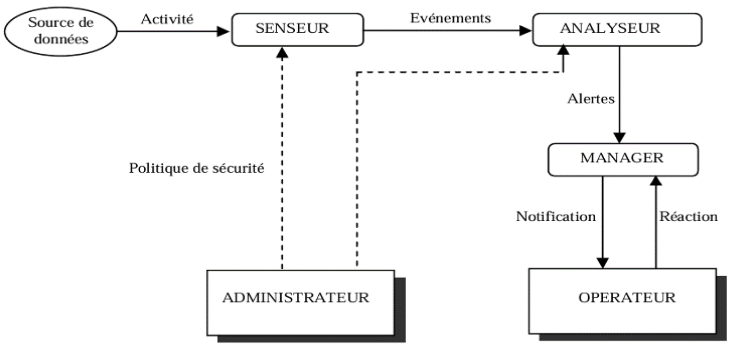

Figure 1.3 - Architecture d'un IDS [28]

- **Le capteur** (ou **senseur**) est chargé de superviser l'activité du système à travers différentes sources de données. Il détecte les changements d'état du système et fournit à l'analyseur une série d'événements qui indiquent la progression de l'état du système.

  Ces événements peuvent être transmis tels quels à l'analyseur, mais dans la plupart des cas, les capteurs sont spécialisés : **les capteurs système**, **les capteurs réseau**, et **les capteurs applicatifs** chacun permettant une surveillance ciblée de leur environnement respectif.

- **L'analyseur** reçoit le flux d'événements transmis par le capteur et a pour mission d'identifier toute activité potentiellement malveillante. Il s'agit ici de filtrer, corréler et interpréter les données collectées, afin de déterminer si celles-ci présentent les caractéristiques d'une attaque ou d'un comportement anormal. Il peut reposer sur des techniques statistiques, comportementales ou basées sur des signatures connues d'attaques. C'est le cœur décisionnel du système de détection.

- **Le gestionnaire** (ou **manager**) joue un rôle clé dans la gestion des alertes générées par le système. Il reçoit les signaux d'alarme de l'analyseur, les structure, les hiérarchise et les présente de manière compréhensible à l'opérateur ou à l'administrateur en charge de la sécurité. Le gestionnaire peut également être responsable de la mise en œuvre d'une réponse appropriée, selon des politiques prédéfinies.

1.3.2.2 Le Fonctionnement logique d'un IDS

Après avoir défini les composants matériels d'un IDS, il est essentiel de comprendre le fonctionnement logique de ces éléments, autrement dit la manière dont ils interagissent pour former un processus complet de détection d'intrusions. Trois étapes principales structurent ce processus : la collecte des données, la détection des intrusions, et la mise en œuvre d'une réponse adaptée. (Voir Figure 1.4)

- **Collecte des données :** La première phase consiste à collecter et prétraiter les données issues du système ou du réseau. Ces données peuvent provenir de journaux système, de paquets réseau ou de bases d'informations de gestion (MIB). Elles sont transformées dans un format commun, stockées si nécessaire, puis transmises au module de détection. Cette étape garantit une base d'information fiable et cohérente pour l'analyse.

- **Détection :** Une fois les données préparées, le module de détection entre en jeu. Il analyse les informations afin d'identifier les tentatives d'intrusion. L'objectif est de repérer, grâce à différents algorithmes, les comportements anormaux ou les signatures d'attaque connues. Les événements considérés comme suspects ou malveillants sont alors transmis au module de réponse.

- **Réponse :** Lorsqu'un événement est classifié comme dangereux, le module de réponse agit selon la politique de sécurité prédéfinie. On distingue généralement deux types de réponses : **1)** Une réponse **passive**, qui se limite à notifier les responsables sans intervenir directement ; **2)** Une réponse **active**, qui cherche à atténuer ou empêcher les dégâts en prenant des mesures concrètes comme blocage d'une adresse IP, coupure temporaire des connexions réseau, etc.

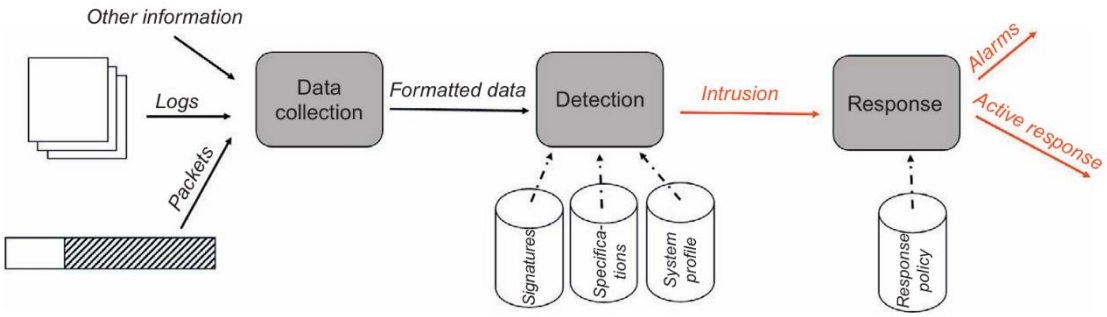

Figure 1.4 - Architecture logique d'IDS [29]

1.3.3 Techniques de détection d'intrusions

1.3.3.1 Les approches de détection

Les systèmes de détection d'intrusion (IDS) reposent sur différentes techniques pour identifier les activités malveillantes au sein d'un système ou d'un réseau. On distingue principalement trois approches de détection :

La détection par signature (misuse-based), la détection par anomalie (anomaly-based), et la détection par spécification (specification-based).

a) Détection par signature (misuse-based)

La détection par signature, également appelée détection basée sur les connaissances, consiste à comparer les activités observées à une base de signatures d'attaques connues. Lorsqu'une correspondance est trouvée, une alerte est déclenchée. Cette méthode est largement utilisée dans les systèmes commerciaux pour son efficacité et son faible taux de faux positifs.

Cependant, elle présente une limite majeure : l'incapacité à détecter des attaques inconnues ou des variantes récentes d'attaques existantes (exploits zero-day). De plus, elle nécessite des mises à jour régulières de la base de signatures pour rester performante.

b) Détection par anomalie (anomaly-based)

La détection par anomalie, aussi appelée détection comportementale, repose sur l'établissement d'un profil du comportement "normal" du système. Elle surveille des éléments comme la fréquence des commandes, l'utilisation des ressources, ou les appels système. Toute déviation significative de ce comportement est interprétée comme une potentielle intrusion.

La détection par anomalie a l'avantage de pouvoir repérer des attaques inconnues. Néanmoins, elle peut générer un taux important de faux positifs et requiert une capacité de calcul élevée pour l'analyse continue des comportements.

Dans la pratique, les systèmes modernes combinent souvent ces approches, en particulier la détection par signature et par anomalie. Cette hybridation permet de profiter de la précision des signatures pour les menaces connues, tout en bénéficiant de la capacité de l'analyse comportementale pour les attaques inédites.

c) Détection par spécification (specification-based)

La détection par spécification repose sur un ensemble de règles ou de contraintes définissant le comportement attendu d'un programme ou d'un protocole. Toute action qui sort de ce cadre est considérée comme suspecte.

Ce type d'IDS offre une capacité de détection efficace aussi bien pour les attaques connues qu'inconnues, tout en maintenant un faible taux de fausses alertes. Il est particulièrement utilisé dans les environnements spécifiques tels que les réseaux ad hoc.

Le tableau 1.1 présente un résumé global des principales caractéristiques des trois méthodologies de détection d'intrusions précédemment abordées. Il met en évidence les principes de fonctionnement de chaque méthode, leurs capacités de détection, leurs taux de fausses alertes ainsi que leurs inconvénients respectifs.

Comme on peut le constater, chaque approche présente des avantages et des limites, ce qui rend difficile le choix d'une solution unique et universelle. Le choix d'un type d'IDS approprié dépend ainsi du contexte, des objectifs et des contraintes du système à protéger.

	Basé sur les signatures	Basé sur l'anomalie	Basé sur les spécifications
Méthode	Identifier des schémas d'attaque connus	Identifier des comportements d'activité inhabituels	Identifier les violations de règles prédéfinies
Taux de détection (DR)	Élevé	Faible	Élevé
Taux de fausses alertes	Faible	Élevé	Faible
Détection des attaques inconnues	Incapable	Capable	Incapable
Inconvénients	Mise à jour des signatures fastidieuse	Le calcul lié à l'apprentissage automatique est lourd	Dépendance au savoir expert pour définir les règles

Tableau 1.1 - Comparaison des types d'IDS selon la méthodologie [30]

1.3.3.2 Mesures d'Evaluation d'un IDS

L'efficacité d'un système de détection d'intrusions repose non seulement sur sa capacité à détecter des attaques, mais également sur sa fiabilité, sa rapidité de réaction, et sa robustesse. Pour cela, plusieurs mesures d'évaluation sont utilisées afin de juger les performances globales d'un IDS.

a) Mesures qualitatives de l'efficacité

Parmi les principales mesures qualitatives de l'efficacité à prendre en compte, on retrouve :

- **La précision** (**accuracy**) : un IDS est considéré comme précis lorsqu'il est capable d'identifier les attaques réelles tout en minimisant les fausses alertes. Il est essentiel d'avoir un IDS précis pour prévenir les alertes erronées qui risqueraient de surcharger les équipes de sécurité avec des événements inappropriés.

- **La performance de traitement** : elle se mesure à la vitesse de traitement des événements. Un bon IDS doit être capable de fonctionner en temps réel, afin de réagir immédiatement aux menaces détectées.

- **La complétude** : L'exhaustivité évalue la faculté d'un système de détection d'intrusions à identifier une vaste gamme de menaces et d'actions malintentionnées. Un IDS exhaustif peut identifier divers types d'attaques, depuis celles fondées sur des signatures spécifiques jusqu'aux attaques non identifiées basées sur des agissements douteux.

- **La tolérance aux pannes** : un IDS doit résister aux attaques ciblant le système lui-même, notamment les attaques par déni de service (DoS), et continuer à fonctionner même en cas de défaillance partielle de l'infrastructure.

- **La rapidité** : la détection et la réponse doivent être suffisamment rapides pour empêcher l'attaquant de compromettre davantage le système ou de masquer ses traces.

b) Mesures quantitatives de la performance

Deux mesures principales sont utilisées pour évaluer quantitativement un IDS :

- **Taux de détection** (Detection Rate, DR) : c'est le rapport entre le nombre d'attaques correctement identifiées (True Positives) et le nombre total d'attaques. Il reflète l'efficacité du système à détecter les activités malveillantes.

- **Taux de fausses alertes** (False Positive Rate, FP) : il représente la proportion d'activités normales que le système identifie à tort comme malveillantes. Un taux élevé de faux positifs peut surcharger les administrateurs et diminuer la confiance envers le système.

Ces indicateurs se basent sur les quatre résultats possibles d'un processus de détection :

- **True Positive (TP)** : attaque détectée correctement.
- **True Negative (TN)** : activité normale correctement identifiée.
- **False Positive (FP)** : activité normale identifiée comme une attaque.
- **False Negative (FN)** : attaque non détectée par l'IDS.

Un IDS avec un taux de détection élevé mais un fort taux de faux positifs peut engendrer une surcharge de travail pour les analystes de sécurité. À l'inverse, un IDS avec peu de faux positifs mais un taux de détection faible risque de laisser passer des attaques dangereuses. Il est donc crucial de trouver un bon compromis entre ces deux aspects.

1.3.4 Typologie des Systèmes de Détection d'Intrusions (IDS)

Les systèmes de détection d'intrusion (IDS) peuvent être classés selon leur emplacement dans l'infrastructure informatique et le type d'activité qu'ils surveillent. Nous identifions trois catégories d'IDS : les IDS basés sur l'hôte (HIDS), les IDS basés sur le réseau (NIDS), et les IDS hybrides ou en environnement cloud.

1.3.4.1 IDS Réseau (NIDS - Network-based IDS)

Les IDS basés sur le réseau (NIDS) sont quant à eux placés à des points stratégiques du réseau, souvent derrière les pare-feux ou aux périmètres du réseau. Leur fonction principale est d'analyser en temps réel le trafic entrant et sortant afin de détecter toute activité suspecte ou malveillante circulant sur le réseau, telles que des scans de ports, des attaques par déni de service ou des tentatives d'accès non autorisé. Un NIDS inspecte les paquets en transit sans interagir directement avec les hôtes.

Ils peuvent également être positionnés à l'intérieur du réseau, par exemple entre différents sous-réseaux, pour détecter les menaces internes ou les mouvements latéraux d'attaquants ayant compromis un système. Cependant, en raison de la quantité élevée de trafic à surveiller, un NIDS peut être confronté à une surcharge qui limite sa capacité à détecter toutes les attaques (voir Figure 1.5).

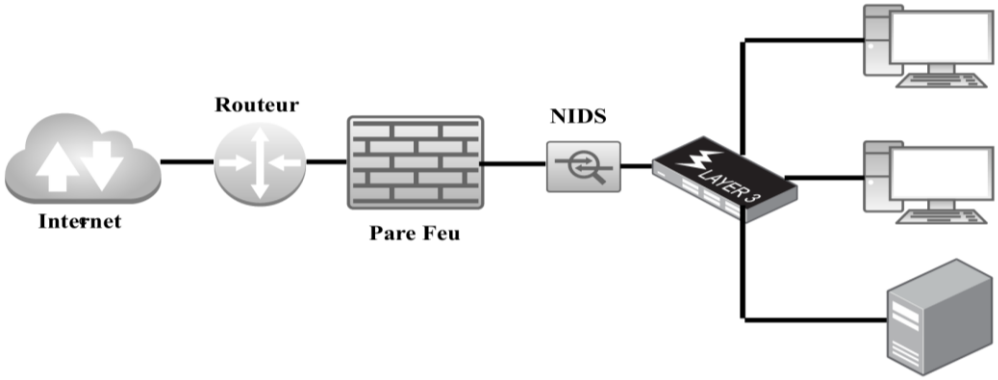

Figure 1.5 - NIDS Typologie

1.3.4.2 IDS Hôte (HIDS - Host-based IDS)

Les systèmes de détection d'intrusion hôte (HIDS) sont conçus pour surveiller l'activité locale d'un terminal spécifique, tel qu'un serveur, un routeur ou un poste de travail.

Pour détecter les comportements suspects ou malveillants, ils analysent l'activité locale de l'hôte sur laquelle ils sont installés, comme les fichiers critiques du système, les journaux d'événements, les registres ou les configurations système.

Bien que leur portée soit restreinte à un seul hôte, les HIDS offrent l'avantage d'accéder à des informations sensibles et chiffrées, souvent inaccessibles aux systèmes basés sur le réseau (voir Figure 1.6).

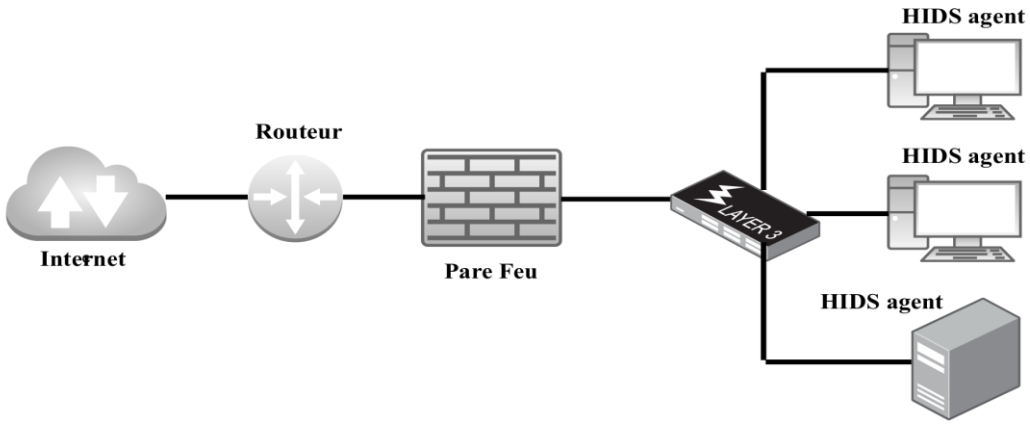

Figure 1.6 - HIDS Typologie

1.3.4.3 IDS Hybride

Les systèmes de détection d'intrusion hybrides associent les approches NIDS et HIDS, permettant une visibilité à la fois sur les activités réseau et sur les hôtes individuels. Ils ont la faculté de surveiller le flux du réseau et les opérations du système sur un hôte, ce qui leur permet d'identifier une gamme plus large de menaces et de fournir des informations plus détaillées sur les attaques.

Une évolution des IDS hybrides réside dans leur adaptation aux environnements cloud, où ils s'intègrent aux différentes couches de l'infrastructure (VMs, API, hyperviseurs, etc...). Ces systèmes profitent de la scalabilité du cloud pour fournir une détection distribuée, avec une authentification sécurisée et une adaptation dynamique aux environnements en constante évolution (voir Figure 1.7).

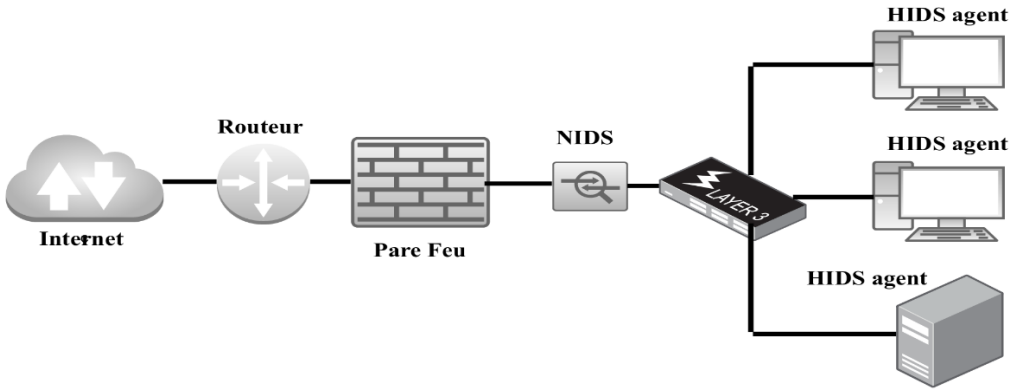

Figure 1.7 - Hybrid IDS Typologie

1.4 Conclusion

Ce premier chapitre a permis de poser les fondations de la sécurité informatique en tant que domaine clé dans la protection des systèmes d'information. Nous y avons défini les objectifs fondamentaux de la cybersécurité ; la confidentialité, l'intégrité, la disponibilité, l'authentification et la non-répudiation qui constituent les piliers indispensables à toute stratégie de défense. Chacun de ces attributs joue un rôle essentiel dans la construction d'une infrastructure résiliente face aux menaces. Toutefois, en dépit de ces principes, des vulnérabilités persistent au sein des systèmes, offrant des opportunités d'exploitation aux attaquants.

Nous avons ensuite analysé les différentes attaques auxquelles un système peut être exposé, telles que les attaques par déni de service (DoS), les attaques par injection, les attaques par brute force, et bien d'autres.

Chaque type d'attaque présente ses propres caractéristiques et impacts, soulignant l'importance d'une défense en profondeur pour parer à ces menaces. Pour se prémunir efficacement contre ces attaques, des contre-mesures telles que les pare-feu, le chiffrement, et les réseaux privés virtuels (VPN) sont indispensables.

Toutefois, parmi l'ensemble des dispositifs de défense, les systèmes de détection d'intrusion (IDS) occupent une place particulière. Nous avons étudié leur rôle stratégique en tant que systèmes d'alerte avancés capables de détecter des comportements anormaux ou malveillants. Qu'ils soient basés sur des signatures, sur des anomalies, ou sur des spécifications, et qu'ils soient déployés au niveau des hôtes (HIDS), du réseau (NIDS) ou de manière hybride, les IDS permettent une surveillance active et ciblée des environnements numériques.

Néanmoins, malgré la diversité des outils et des méthodologies disponibles surs, il est essentiel de rappeler qu'aucune solution ne garantit une protection absolue. La sécurité informatique n'est pas un état figé, mais un processus évolutif, qui repose sur une combinaison cohérente de technologies, de politiques de sécurité robustes, et de pratiques organisationnelles rigoureuses.

En définitive, ce chapitre a permis de comprendre que la mise en place d'un système de sécurité efficace ne dépend pas uniquement des outils utilisés, mais surtout de la manière dont ils sont intégrés dans une politique de défense globale, régulièrement mise à jour, contextualisée, et adaptée aux besoins spécifiques de l'organisation.

C'est dans cette optique que les IDS doivent être envisagés : non comme une solution unique, mais comme une composante essentielle d'une stratégie de cybersécurité proactive et intelligente.

Le chapitre suivant sera consacré à l'introduction des méthodes d'apprentissage par renforcement (RL) et d'apprentissage par renforcement profond (DRL), comme approches innovantes pour renforcer la détection d'intrusions.

Chapitre 2

Apprentissage par renforcement profond

2.1 Introduction

Le Deep Reinforcement Learning (DRL) s'est imposé sur la scène scientifique lorsque DeepMind a réalisé une avancée spectaculaire en atteignant des performances de niveau humain dans les jeux d'arcade Atari. Ce succès, obtenu sans connaissance préalable du jeu et à partir d'images brutes uniquement, a marqué un tournant dans l'intelligence artificielle. Pour la première fois, un agent artificiel apprenait à agir efficacement par simple essai-erreur, sans supervision explicite — une idée fascinante et radicalement différente des approches traditionnelles de l'apprentissage supervisé.

Cette percée a non seulement démontré le potentiel du DRL, mais elle a aussi ravivé l'intérêt pour l'apprentissage par renforcement, en lui ajoutant la puissance des réseaux de neurones profonds. Le DRL s'inscrit dans le cadre plus large de l'apprentissage automatique, aux côtés de l'apprentissage supervisé et non supervisé, chacun se distinguant par sa manière de formuler les problèmes et d'extraire de la connaissance à partir des données. Ce qui rend le DRL unique, c'est sa capacité à apprendre des politiques d'action optimales en interagissant activement avec un environnement, souvent complexe, incertain et non stationnaire.

Au cœur du DRL se trouvent des concepts fondamentaux tels que l'approximation de fonctions, la prise de décision séquentielle, et l'optimisation à long terme des récompenses. Dans ce chapitre, nous explorerons en détail les bases théoriques du DRL, ses principales familles d'algorithmes, ainsi que les défis et perspectives liés à leur mise en œuvre dans des environnements concrets.

2.2 Apprentissage automatique

L'apprentissage automatique (Machine Learning) constitue une branche de l'informatique reposant sur la conception d'algorithmes capables d'apprendre et d'exécuter des tâches spécifiques**.** Depuis plusieurs décennies, les chercheurs nourrissent l'ambition de développer des systèmes intelligents capables de prédire, d'analyser et d'accomplir diverses tâches sans intervention humaine directe. L'apprentissage automatique répond à cette aspiration en élaborant des méthodes algorithmiques permettant aux machines d'apprendre à partir d'expériences passées afin d'optimiser leur prise de décision dans le futur.

Arthur Samuel, l'un des précurseurs du Machine Learning, a défini cette discipline comme étant un « *Field of study that gives computers the ability to learn without being explicitly programmed* » (Arthur Samuel, 1950), soit un domaine d'étude qui confère aux ordinateurs la capacité d'apprendre sans être expressément programmés.

L'apprentissage automatique s'attache à la découverte et à l'élaboration de structures algorithmiques capables d'apprendre directement à partir des données**.** Ces algorithmes construisent des modèles qui, en réceptionnant des entrées (inputs), produisent des prédictions ou des décisions adaptées. Face à l'impossibilité de spécifier a priori toutes les conditions ou règles dans un programme, l'approche algorithmique de l'apprentissage automatique repose sur la capacité d'adaptation et d'auto-amélioration des systèmes.

Il convient de souligner que les notions d'apprentissage automatique et d'intelligence artificielle (IA) sont parfois confondues. Toutefois, elles renvoient à des concepts distincts : l'apprentissage automatique se limite à l'élaboration de logiciels capables d'apprendre de manière autonome à partir de données et d'expériences passées, tandis que l'IA englobe un champ beaucoup plus vaste comprenant également la logique, le raisonnement et la planification.

Dans le domaine de l'apprentissage automatique, on distingue généralement trois grandes catégories (voir Figure 2.1) :

- Apprentissage supervisé
- Apprentissage non supervisé
- Apprentissage par renforcement

2.2.1 Apprentissage supervisé

L'apprentissage supervisé repose sur l'entraînement d'un modèle à partir de données étiquetées sous la supervision d'un instructeur. Lors de la phase d'entraînement, un ensemble de données d'apprentissage comprenant des entrées associées à leurs résultats attendus est fourni au modèle. Ce dernier apprend ainsi à établir des correspondances entre les données et leurs sorties cibles. Après avoir formulé une prédiction, le modèle la compare au résultat réel transmis durant l'apprentissage. Que la prédiction soit correcte ou erronée, le modèle ajuste progressivement ses paramètres internes afin d'améliorer la précision de ses prévisions. Ce processus d'ajustement itératif se poursuit jusqu'à ce qu'un niveau de performance jugé satisfaisant soit atteint. Une fois cette phase d'apprentissage achevée, le modèle est mis à l'épreuve sur de nouvelles données dont les résultats ne lui sont pas communiqués. Il doit alors générer ses propres prédictions de manière autonome.

Cette approche est qualifiée d'apprentissage supervisé en raison de l'intervention d'un superviseur au cours de l'entraînement, ce dernier fournissant les réponses correctes permettant de guider l'apprentissage du modèle.

On peut décomposer l'apprentissage supervisé en deux catégories principales :

- **Classification** : Un problème de classification se présente lorsque nous devons déterminer une catégorie. Par exemple, nous avons accès au dossier médical d'un patient admis à l'hôpital et souhaitons évaluer le nombre de personnes susceptibles de faire une crise cardiaque. Dans ce cadre, la sortie possible sera oui ou non.
- **Régression :** Un problème de régression se présente lorsqu'il s'agit de prévoir un chiffre ou une valeur. Par exemple, quand on prédit une valeur boursière, le résultat est toujours un chiffre réel.

2.2.2 Apprentissage non supervisé

La seconde catégorie concerne l'apprentissage sans supervision. Dans l'apprentissage non supervisé, nous allons mettre à disposition un ensemble de données complet sans réponses préétablies. Par la suite, la machine s'efforcera d'organiser les données en fonction de modèles identifiés. Par imitation, il tentera de modifier légèrement les données afin de détecter des motifs et de les regrouper. Quand le modèle traite de nouveaux ensembles de données, il va essayer de déterminer à quel groupe ils appartiennent. Contrairement à l'apprentissage supervisé, un superviseur n'est jamais présent pour fournir un modèle ou une signification afin de classer les données. C'est le modèle qui fait cette détermination de manière autonome.

Les problèmes d'apprentissage non supervisé se divisent en deux classes :

- **Clustering :** Une problématique de clustering se traduit par l'identification d'un schéma ou la compréhension de la façon de classer à partir des données proposées.
- **Association :** L'association est un enjeu d'apprentissage basé sur les règles où vous identifiez un schéma décrivant une bonne portion des données fournies. Par exemple, dans le cas d'une librairie virtuelle, le système de suggestion propose que les clients ayant acheté le livre A ont également tendance à acheter certains autres ouvrages.

2.2.3 Apprentissage par renforcement

L'apprentissage par renforcement (RL) est une méthode d'apprentissage automatique (ML) qui forme les programmes à faire des choix afin d'atteindre les résultats optimaux. Elle reproduit la méthode d'apprentissage par essais et erreurs utilisée par les humains pour réaliser leurs buts. Les actions qui aident à atteindre votre but sont renforcées, alors que celles qui entravent ce processus sont négligées.

Les algorithmes d'apprentissage par renforcement fonctionnent selon un système de récompense et de sanction pour le traitement des données. Ils tirent des leçons des retours d'expérience de chaque action et découvrent par eux-mêmes les voies optimales de traitement pour parvenir aux résultats finaux. Les algorithmes ont aussi la capacité de différer la gratification. Pour atteindre la meilleure stratégie globale, des concessions à court terme peuvent être nécessaires ; par conséquent, l'approche la plus efficace qu'ils trouvent pourrait comporter des sanctions ou des inversions. L'apprentissage par renforcement est une technique efficace qui permet aux systèmes d'intelligence artificielle (IA) de parvenir à des résultats optimaux dans des contextes non observables.

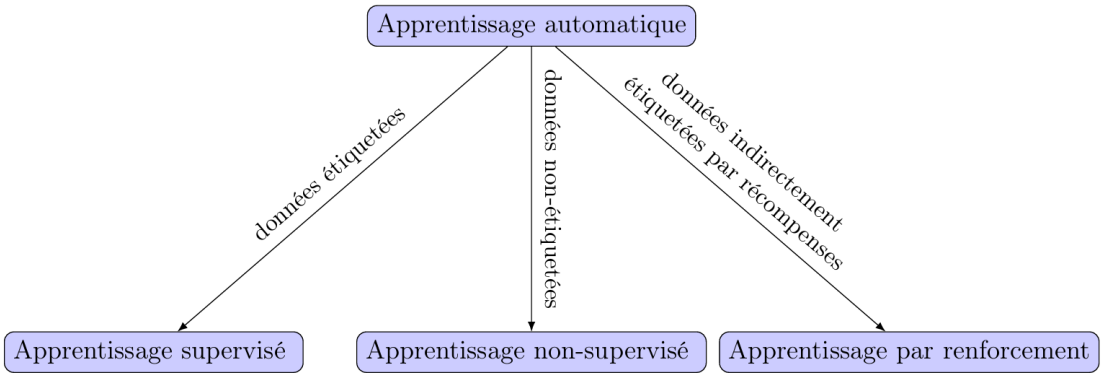

Figure 2.1 - Les trois grandes classes d'apprentissage automatique [31]

2.3 Apprentissage par renforcement (Reinforcement learning)

L'apprentissage par renforcement (RL) est axé sur la résolution de problèmes de prise de décision séquentielle. De nombreux problèmes réels, tels que jouer à des jeux vidéo, les sports, la conduite, l'optimisation des inventaires et le contrôle des robots, peuvent être formulés de cette manière. Ce sont des activités que les humains et les machines accomplissent.

Lors de la résolution de ces problèmes, nous avons un objectif ou un but, tel que gagner le jeu, arriver à destination en toute sécurité ou minimiser le coût de fabrication des produits. Nous effectuons des actions et obtenons des retours du monde sur la proximité de l'atteinte de notre objectif – comme le score actuel, la distance jusqu'à notre destination ou le prix par unité. Atteindre notre but implique généralement de prendre plusieurs actions successives, chaque action modifiant l'environnement autour de nous. Nous observons ces changements dans l'environnement ainsi que les retours reçus avant de décider de la prochaine action à entreprendre en réponse.

Les problèmes de RL peuvent être exprimés comme un système constitué d'un agent et d'un environnement. Un environnement génère des informations décrivant l'état du système. Cela s'appelle un état. Un agent interagit avec un environnement en observant l'état et en utilisant ces informations pour sélectionner une action. L'environnement accepte l'action et passe à l'état suivant. Il retourne ensuite au prochain

état et une récompense à l'agent. Lorsque le cycle (état → action → récompense) est complet, on considère qu'un pas de temps a été effectué. Le cycle se répète jusqu'à ce que l'environnement se termine, par exemple lorsque le problème est résolu. Ce processus complet est décrit par le diagramme de la boucle de contrôle (voir figure 2.2).

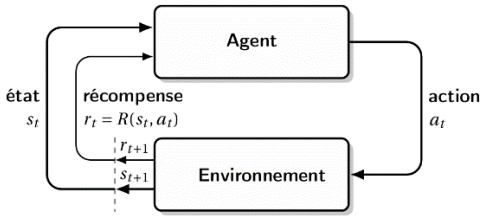

Figure 2.2 - La boucle de contrôle en apprentissage par renforcement

Nous appelons la fonction de production d'action d'un agent une politique. Formellement, une politique est une fonction qui associe des états à des actions. Une action changera l'environnement et affectera ce que l'agent observe et fait ensuite. L'échange entre un agent et un environnement se déroule dans le temps, de sorte qu'il peut être considéré comme un processus de prise de décision séquentielle. Les problèmes de RL ont un objectif, qui est la somme des récompenses reçues par un agent. L'objectif d'un agent est de maximiser cet objectif en sélectionnant de bonnes actions. Il apprend à le faire en interagissant avec l'environnement dans un processus d'essais et d'erreurs, et utilise les signaux de récompense qu'il reçoit pour renforcer les bonnes actions.

L'agent et l'environnement sont définis comme étant mutuellement exclusifs, de sorte que les frontières entre l'échange d'état, d'action et de récompense sont claires. Nous pouvons considérer l'environnement comme tout ce qui n'est pas l'agent. Par exemple, lors de la conduite d'un vélo, nous pouvons avoir plusieurs définitions valides mais équivalentes de l'agent et de l'environnement. Si nous considérons tout notre corps comme étant l'agent qui observe notre environnement et produit des mouvements musculaires comme actions, alors l'environnement est le vélo et la route. Si nous considérons nos processus mentaux comme l'agent, l'environnement devient notre corps physique, le vélo et la route, les actions étant les signaux neuronaux envoyés par notre cerveau aux muscles et les états étant les entrées sensorielles envoyées au cerveau [32].

Essentiellement, un système d'apprentissage par renforcement est une boucle de contrôle de rétroaction où un agent et un environnement interagissent et échangent des signaux, tandis que l'agent tente de maximiser l'objectif. Les signaux échangés sont (s_t, a_t, r_t), qui représentent respectivement l'état, l'action et la récompense, et t désigne le pas de temps auquel ces signaux se produisent. Le triplet (s_t, a_t, r_t) est appelé une expérience. La boucle de contrôle peut se répéter indéfiniment ou se terminer lorsqu'un état terminal est atteint ou lorsque le nombre maximal de pas de temps t = T est atteint. L'horizon temporel, de t = 0 à la terminaison de l'environnement, est appelé un épisode. Une trajectoire est une séquence d'expériences au cours d'un épisode, $\tau = (s_0, a_0, r_0), (s_1, a_1, r_1)$, etc. Un agent a généralement besoin de nombreux épisodes pour apprendre une bonne politique, allant de centaines à des millions, en fonction de la complexité du problème.

Examinons maintenant un exemple d'environnement d'apprentissage par renforcement, illustré à la figure 2.3, ainsi que la manière dont les états, les actions et les récompenses y sont définies. Cet environnement est disponible via la bibliothèque OpenAI Gym [33], qui est une bibliothèque open source fournissant un ensemble standardisé d'environnements.

Atari Breakout (Figure 2.3) est un jeu d'arcade rétro qui se compose d'une balle, d'une raquette située en bas de l'écran contrôlée par un agent, et de briques. L'objectif du jeu est de détruire toutes les briques

en faisant rebondir la balle sur la raquette. Le joueur commence avec cinq vies, et une vie est perdue chaque fois que la balle tombe en bas de l'écran.

1. **Objectif** : Maximiser le score du jeu.
2. **État** : Une image numérique en couleurs (RGB) d'une résolution de 160 × 210 pixels, correspondant à ce qui est affiché à l'écran du jeu.
3. **Action** : Un entier appartenant à l'ensemble {0, 1, 2, 3}, correspondant aux actions du contrôleur de jeu : aucune action, lancer la balle, se déplacer vers la droite, se déplacer vers la gauche.
4. **Récompense** : La différence de score entre deux états consécutifs.
5. **Condition de terminaison** : Lorsque toutes les vies du joueur sont perdues.

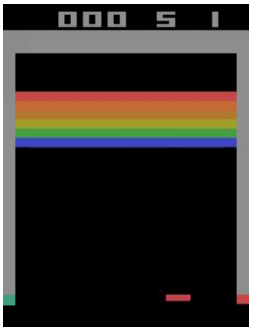

Figure 2.3 - Un exemple d'environnement d'OpenAI Gym.

Voyons maintenant comment décrire formellement les états, les actions et les récompenses.

$s_t \in \mathcal{S}$ est l'état, $\boldsymbol{\mathcal{S}}$ est l'espace des états.

$a_t \in \mathcal{A}$ est l'action, $\boldsymbol{\mathcal{A}}$ est l'espace des actions.

$r_t = (s_t, a_t, s_{t+1})$ est la récompense , $\boldsymbol{\mathcal{R}}$ est la fonction de récompense.

L'espace d'états $\boldsymbol{\mathcal{S}}$ est l'ensemble de tous les états possibles dans un environnement. Selon l'environnement, il peut être défini de différentes manières : entiers, réels, vecteurs, matrices, données structurées ou non structurées. De même, l'espace d'actions $\boldsymbol{\mathcal{A}}$ est l'ensemble de toutes les actions possibles définies par un environnement. Il peut également prendre plusieurs formes, mais est généralement défini comme un scalaire ou un vecteur. La fonction de récompense (s_t, a_t, s_{t+1}) attribue un scalaire positif, négatif ou nul à chaque transition (s_t, a_t, s_{t+1}). L'espace d'états, l'espace d'actions et la fonction de récompense sont spécifiés par l'environnement. Ensemble, ils définissent les triplets (s, a, r), qui sont l'unité de base de l'information décrivant un système d'apprentissage par renforcement.

2.4 Processus de décision markovien (MDP)

À présent, considérons comment un environnement passe d'un état à un autre à l'aide de ce que l'on appelle la fonction de transition. En apprentissage par renforcement, une fonction de transition est formulée comme un processus de décision markovien (MDP), qui constitue un cadre mathématique pour modéliser la prise de décision séquentielle.

Pour comprendre pourquoi les fonctions de transition sont représentées sous forme de MDP, considérons la formulation générale donnée dans l'Équation 2.1.

$$s_{t+1} \sim \mathcal{P}\left(s_{t+1} \mid (s_0, a_0), (s_1, a_1), \ldots, (s_t, a_t)\right) \tag{2.1}$$

L'Équation 2.1 indique qu'au pas de temps t, le prochain état s_{t+1} est échantillonné à partir d'une distribution de probabilité $\boldsymbol{\mathcal{P}}$ conditionnée par l'ensemble de l'historique. La probabilité qu'un environnement

passe de l'état s_t à s_{t+1} dépend de tous les états s et actions a précédents qui se sont produits jusqu'à présent dans un épisode. Il est difficile de modéliser une fonction de transition sous cette forme, notamment lorsque les épisodes durent de nombreux pas de temps.

Afin de rendre la fonction de transition plus praticable, nous la transformons en un MDP en ajoutant l'hypothèse que la transition vers l'état suivant s_{t+1} dépend uniquement de l'état précédent s_t et de l'action a_t. Cela est connu sous le nom de propriété de Markov. Avec cette hypothèse, la nouvelle fonction de transition devient la suivante :

$$s_{t+1} \sim \mathcal{P}\left(s_{t+1} \mid s_t, a_t\right) \tag{2.2}$$

Cette forme est plus simple que la fonction de transition originale. La propriété de Markov implique que l'état actuel et l'action au pas de temps t contiennent suffisamment d'informations pour déterminer complètement la probabilité de transition vers l'état suivant au temps t + 1.

Malgré la simplicité de cette formulation, elle reste extrêmement puissante. De nombreux processus peuvent être exprimés de cette manière, notamment les jeux, le contrôle robotique et la planification. Cela est possible car un état peut être défini de manière à inclure toute l'information nécessaire pour satisfaire la propriété de Markov.

Nous sommes désormais en mesure de présenter la formulation MDP d'un problème d'apprentissage par renforcement. Un MDP est défini par un quadruplet ($\boldsymbol{\mathcal{S}}$, $\boldsymbol{\mathcal{A}}$, $\boldsymbol{\mathcal{P}}$, $\boldsymbol{\mathcal{R}}$), où :

- $\mathcal{S}$ est l'ensemble des états,
- $\mathcal{A}$ est l'ensemble des actions,
- $\mathcal{P}(s_{t+1} \mid s_t, a_t)$ est la fonction de transition d'état de l'environnement,
- $\mathcal{R}(s_t, a_t, s_{t+1})$ est la fonction de récompense de l'environnement.

Une hypothèse importante dans les problèmes d'apprentissage par renforcement est que les agents n'ont pas accès directement à la fonction de transition $\mathcal{P}(s_{t+1} \mid s_t, a_t)$ ni à la fonction de récompense $\mathcal{R}(s_t, a_t, s_{t+1})$. La seule façon pour un agent d'obtenir des informations sur ces fonctions est à travers les états, les actions et les récompenses qu'il expérimente réellement dans l'environnement.

2.4.1 MDP vs POMDP

La fonction de transition du MDP est présentée dans l'Équation 2.2. La fonction de transition possède la propriété de Markov où la transition vers s_{t+1} est entièrement déterminée par l'état et l'action actuels, (s_t, a_t). Il peut y avoir eu de nombreux états antérieurs expérimentés par l'agent dans l'épisode, s_0, s_1, ..., s_{t-1}, mais ceux-ci ne fournissent aucune information supplémentaire sur l'état vers lequel l'environnement va évoluer.

Le concept d'état apparaît à deux endroits. D'abord, il y a l'état produit par un environnement et observé par un agent : c'est l'état observé s_t. Ensuite, il y a l'état utilisé par la fonction de transition : c'est l'état interne de l'environnement, noté s_t^{int}.

Si un environnement est un MDP, il est décrit comme entièrement observable, et $s_t = s_t^{int}$. Par exemple, le problème CartPole est un environnement entièrement observable. Cependant, l'état interne de l'environnement peut être caché à l'agent, c'est-à-dire $s_t \neq s_t^{int}$. Ces types d'environnements sont connus sous le nom de MDPs partiellement observables (POMDPs).

Les POMDPs ont la même fonction de transition que les MDPs. Toutefois, les POMDPs ne fournissent pas l'état interne s_t^{int}. à l'agent ; ils fournissent à la place l'état observé s_t. Cela signifie que l'agent ne connaît plus l'état interne s_t^{int}. et doit l'inférer à partir de tout ou partie des états observés $(s_t, s_{t-1}, \ldots, s_0)$.

Les états internes décrivent entièrement le système qui nous intéresse. Dans un MDP, l'agent observe l'état interne. Cependant, dans les POMDPs, nous devons distinguer l'état observé de l'état interne de l'environnement.

Les POMDPs peuvent être divisés en trois catégories :

- Entièrement observable avec historique partiel
- Entièrement observable avec historique complet
- Jamais entièrement observable

Pour compléter la formulation du problème, nous devons également formaliser le concept d'un objectif que l'agent cherche à maximiser. Définissons d'abord le retour R(τ) en utilisant une trajectoire issue d'un épisode,
$\tau = (s_0, a_0, r_0), \ldots, (sT, aT, rT)$:

$$R(\tau) = r^0 + \gamma r^1 + \gamma^2 r^2 + \cdots + \gamma^{\mathrm{T}} r_T = \sum_{t=0}^{T} \gamma^{\mathrm{t}} r^t \quad (2.3)$$

L'Équation 2.3 définit le retour comme une somme actualisée des récompenses d'une trajectoire, où $\gamma \in [0, 1]$ est le facteur d'actualisation (discount factor).

Ensuite, l'objectif $J(\tau)$ est simplement l'espérance mathématique des retours sur plusieurs trajectoires, comme montré dans l'Équation 2.4 :

$$J(\tau) = \mathbb{E}(\tau \sim \pi)[R(\tau)] = \mathbb{E}(\tau)\left[\sum_{t=0}^{T} \gamma^{\mathrm{t}} r^t\right] \quad (2.4)$$

Le facteur d'actualisation $\gamma \in [0, 1]$ est une variable importante qui modifie la manière dont les récompenses futures sont valorisées. Plus γ est petit, moins les récompenses futures sont pondérées, ce qui rend l'agent plus *"shortsighted"* (myope). Dans le cas extrême où γ = 0, l'objectif ne considère que la récompense initiale r_0, comme illustré dans l'Équation 2.5 :

$$R(\tau)_{\gamma=0} = \sum_{t=0}^{T} \gamma^{\mathrm{t}} r^t = r^0 \quad (2.5)$$

Plus γ est grand, plus les récompenses futures sont prises en compte : l'objectif devient alors plus *"far-sighted"* (clairvoyant). Si γ = 1, les récompenses de chaque instant sont pondérées de manière égale, comme montré dans l'Équation 2.6 :

$$R(\tau)_{\gamma=1} = \sum_{t=0}^{T} \gamma^{\mathrm{t}} r^t = \sum_{t=0}^{T} r^t \quad (2.6)$$

Pour les problèmes avec un horizon temporel infini, nous devons fixer γ < 1 pour empêcher que l'objectif ne devienne non borné. Pour les problèmes avec un horizon temporel fini, γ est un paramètre important, car il peut rendre le problème plus ou moins difficile à résoudre selon la valeur du facteur d'actualisation utilisée.

2.5 Les fonctions fondamentales dans RL

En apprentissage par renforcement, il existe trois fonctions principales à apprendre :

1. **Une politique**, π, qui associe un état à une action : $a \sim \pi(s)$
2. **Une fonction de valeur**, $\boldsymbol{V}(s)$ ou $\boldsymbol{Q}(s, a)$, qui estime le retour attendu $\mathbb{E}[R(\tau)]$
3. **Un modèle de l'environnement**, $\mathbf{P}(s' \mid s, a)$

La politique définit la manière dont un agent choisit ses actions dans l'environnement afin de maximiser l'objectif. Dans la boucle de contrôle de l'apprentissage par renforcement, l'agent doit produire une action à chaque étape de temps après avoir observé un état **s**. La politique est donc fondamentale pour cette boucle de contrôle, car elle génère les actions qui la font fonctionner. Une politique peut être stochastique, C'est-à-dire qu'elle peut produire différentes actions de manière probabiliste pour un même état. On note cela $\pi(a \mid s)$, représentant la probabilité de choisir l'action a étant donné l'état s. Une action échantillonnée depuis une politique est notée $a \sim \pi(s)$.

Les fonctions de valeur fournissent des informations sur l'objectif à atteindre. Elles aident l'agent à évaluer la qualité des états et des actions disponibles en termes de futur retour attendu. Elles existent sous deux formes principales : les fonctions $\boldsymbol{V}(s)$ et $\boldsymbol{Q}(s,a)$

$$V(s) = \mathbb{E}\left[\sum_{t=0}^{T} r_t \mid s_0 = s\right] \quad (2.7)$$

$$Q(s,a) = \mathbb{E}\left[\sum_{t=0}^{T} r_t \mid s_0 = s, a_0 = a\right] \quad (2.8)$$

La fonction de valeur $\boldsymbol{V}$ présentée dans l'équation 2.8 évalue la qualité d'un état. Elle mesure le retour attendu à partir de l'état s, en supposant que l'agent continue d'agir selon sa politique actuelle π. Le retour est mesuré depuis l'état courant s jusqu'à la fin de l'épisode. Il s'agit d'une mesure tournée vers l'avenir, car toutes les récompenses reçues avant l'état s sont ignorées. Cela illustre le caractère prospectif de la fonction de valeur et sa capacité à aider l'agent à distinguer des états qui procureraient la même récompense immédiate. Plus un agent est proche de l'état objectif, plus la valeur de l'état est élevée.

La fonction $\boldsymbol{Q}$ de valeur d'action, présentée dans l'équation 2.9 évalue la qualité d'un couple état-action. $\boldsymbol{Q}$ mesure le retour attendu après avoir pris l'action a dans l'état s, en supposant que l'agent continue ensuite à suivre sa politique π. À l'instar de **V**, ce retour est également mesuré depuis l'état s jusqu'à la fin de l'épisode, en ignorant les récompenses passées.

La fonction de transition $P(s' \mid s,a)$ fournit des informations sur l'environnement. Si un agent apprend cette fonction, il est alors capable de prédire l'état suivant s′ vers lequel l'environnement évoluera après avoir pris l'action a depuis l'état s. En utilisant la fonction de transition apprise, l'agent peut ainsi "imaginer" les conséquences de ses actions sans interagir directement avec l'environnement, ce qui lui permet de planifier de bonnes actions.

2.6 L'apprentissage par renforcement profond (DRL)

L'apprentissage par renforcement profond (Deep Reinforcement Learning, DRL) résulte de la combinaison de l'apprentissage par renforcement (RL) et des techniques d'apprentissage profond (deep learning) (voir Figure 2.4).

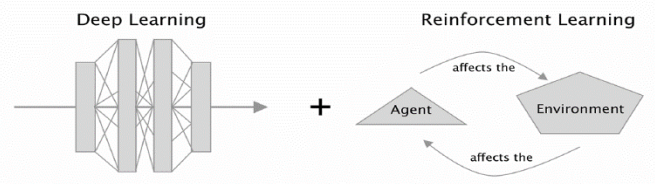

Figure 2.4 - L'apprentissage par renforcement profond

Bien que le RL soit étudié depuis plusieurs décennies et ait démontré son efficacité dans des environnements simples, il a longtemps peiné à passer à l'échelle pour traiter des problèmes plus complexes et de grande dimension. L'émergence de l'apprentissage profond a permis de surmonter cette limite en offrant aux agents la capacité de traiter et d'apprendre directement à partir de données riches et complexes, telles que des images ou des signaux perceptifs bruts. Grâce aux réseaux de neurones profonds, le DRL est

capable d'extraire automatiquement des caractéristiques pertinentes et des niveaux d'abstraction élevés sans nécessiter une ingénierie manuelle des variables, contrairement aux méthodes classiques. Cette approche a ainsi permis de résoudre des tâches de décision séquentielle particulièrement difficiles, avec un besoin réduit de connaissances préalables sur l'environnement. Parmi les réalisations majeures dans ce domaine, on peut citer les travaux de DeepMind, dont les agents basés sur le DRL ont surpassé les performances humaines dans de nombreux jeux vidéo, démontrant ainsi le potentiel considérable de cette approche. L'un des exemples les plus emblématiques est celui d'AlphaGo [35], qui en 2016 a réussi à battre le champion du monde de jeu de Go, illustrant ainsi la capacité du DRL à maîtriser des environnements complexes, hautement stratégiques et longtemps considérés comme inaccessibles à l'intelligence artificielle traditionnelle.

Il existe trois grandes familles d'algorithmes d'apprentissage par renforcement profond : les méthodes basées sur la politique (*policy-based*), les méthodes basées sur la valeur (*value-based*), et les méthodes basées sur le modèle (*model-based*), qui apprennent respectivement des politiques, des fonctions de valeur et des modèles. Il existe également des approches hybrides dans lesquelles les agents apprennent plusieurs de ces fonctions simultanément par exemple, une politique et une fonction de valeur, ou une fonction de valeur et un modèle. La Figure 2.5 présente un aperçu des principaux algorithmes d'apprentissage par renforcement profond dans chaque famille ainsi que leurs relations.

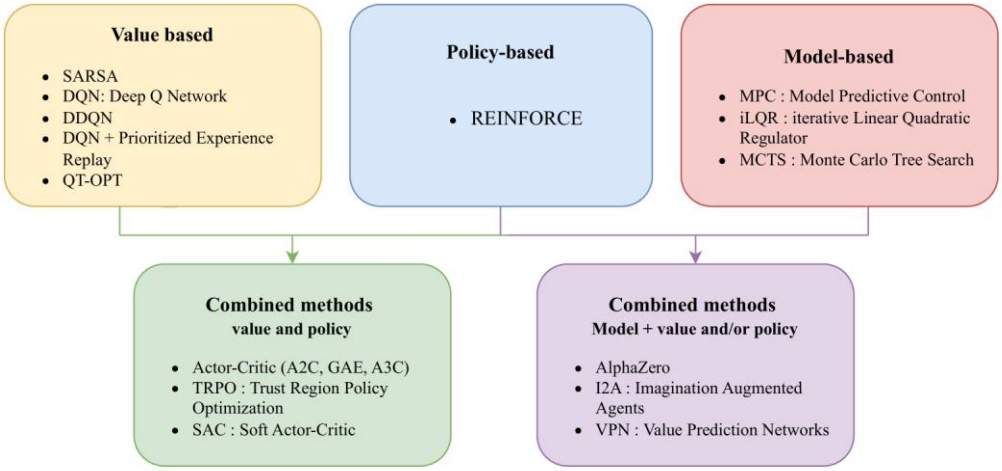

Figure 2.5 - Familles d'algorithmes d'apprentissage par renforcement profond [32]

2.6.1 Les algorithmes basés sur des politiques (Policy-Based Algorithms)

Les algorithmes basés sur des politiques font partie d'une famille d'algorithmes qui apprennent une politique. De bonnes politiques devraient générer des actions qui produisent des trajectoires maximisant l'objectif d'un agent, $J(\pi) = \mathbb{E}[\sum_{t=0}^{T} \gamma^t r_t]$. Cette approche est assez intuitive : si un agent doit agir dans un environnement, il est logique d'apprendre une politique. Ce qui constitue une bonne action à un moment donné dépend de l'état, donc une fonction de politique π prend un état s en entrée pour produire une action $a \sim \pi(s)$. Cela signifie qu'un agent peut prendre de bonnes décisions dans différents contextes. REINFORCE est l'algorithme basé sur des politiques le plus connu et forme la base de nombreux algorithmes ultérieurs.

Un des grands avantages des algorithmes basés sur des politiques est qu'ils représentent une classe d'optimisation très générale. Ils peuvent être appliqués à des problèmes avec tout type d'actions discrètes, continues, ou un mélange (multi-actions). Ils optimisent directement ce que l'agent considère comme étant le plus important l'objectif $J(\pi)$. De plus, cette classe de méthodes est garantie de converger vers

une politique localement optimale. Un inconvénient de ces méthodes est qu'elles présentent une forte variance et sont peu efficaces en termes d'échantillons.

2.6.1.1 REINFORCE

L'algorithme REINFORCE, inventé par Ronald J. Williams en 1992 dans son article *« Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning »* [36], apprend une politique paramétrée qui génère des probabilités d'actions à partir des états. Les agents utilisent directement cette politique pour agir dans un environnement.

L'idée clé est que, durant l'apprentissage, les actions ayant conduit à de bons résultats doivent devenir plus probables ; elles sont ainsi renforcées positivement. À l'inverse, les actions ayant abouti à de mauvais résultats doivent devenir moins probables. Si l'apprentissage est réussi, au fil de nombreuses itérations, les probabilités d'actions produites par la politique évoluent vers une distribution qui permet d'obtenir de bonnes performances dans l'environnement.

Les probabilités d'action sont modifiées en suivant le gradient de la politique ; ainsi, REINFORCE est classé parmi les algorithmes de type *policy gradient* (gradient de politique). L'algorithme repose sur trois composants essentiels : Une politique paramétrée, Un objectif à maximiser (Ce point a déjà été présenté dans la quatrième section du chapitre 2, qui traite des processus de décision markoviens (MDP)) et une méthode pour mettre à jour les paramètres de la politique.

a) Politique

Une politique π est une fonction qui associe les états aux probabilités d'actions, utilisée pour échantillonner une action $a \sim \pi(s)$. Dans l'algorithme REINFORCE, un agent apprend une politique et l'utilise pour agir dans un environnement. Une bonne politique est celle qui maximise la somme cumulative des récompenses actualisées. L'idée centrale de l'algorithme est donc d'apprendre une bonne politique, ce qui implique de réaliser une approximation de fonction.

Les réseaux de neurones sont des approximateurs de fonctions puissants et flexibles ; ainsi, une politique peut être représentée à l'aide d'un réseau de neurones profond comportant des paramètres ajustables θ. Ce type de réseau est souvent appelé réseau de politique (*policy network*). On dit que la politique est paramétrée par ces paramètres.

Chaque ensemble spécifique de valeurs des paramètres du réseau de politique correspond à une politique particulière. Le processus d'apprentissage d'une bonne politique correspond donc à la recherche d'un bon ensemble de valeurs pour ces paramètres θ. Pour cette raison, il est crucial que le réseau de politique soit différentiable.

b) Le Gradient de Politique

Nous avons maintenant défini la politique et l'objectif $J(\theta)$, qui sont deux éléments essentiels pour dériver l'algorithme du gradient de politique. La politique fournit un moyen pour un agent d'agir, et l'objectif définit une cible à maximiser.

Le dernier composant de l'algorithme est le gradient de politique. Formellement, un algorithme de gradient de politique résout le problème suivant :

$$max_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] \quad (2.9)$$

Pour maximiser l'objectif, on effectue une ascension du gradient sur les paramètres de la politique θ. Rappelons, en calcul différentiel, que le gradient pointe dans la direction de la plus forte augmentation. Pour améliorer l'objectif, on calcule le gradient et on l'utilise pour mettre à jour les paramètres, comme illustré dans l'Équation 2.10 :

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta) \quad (2.10)$$

Ici, α est un taux d'apprentissage (ou learning rate), un scalaire qui contrôle l'amplitude de la mise à jour des paramètres. Le terme $\nabla_\theta J(\pi_\theta)$ est appelé gradient de politique, défini par l'Équation 2.11 :

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} R_t(\tau)\nabla_\theta \log \pi_\theta (a_t \mid s_t)] \quad (2.11)$$

Le terme $\pi_\theta(a_t \mid s_t)]$ représente la probabilité de l'action choisie par l'agent à l'instant t, action échantillonnée selon la politique $a \sim \pi_\theta$. Le membre de droite de l'équation indique que le gradient du logarithme de cette probabilité, par rapport à θ, est multiplié par le retour $R_t(\tau)$.

L'Équation 2.12 montre que le gradient de l'objectif est équivalent à la somme espérée des gradients des log-probabilités des actions, pondérés par les retours associés.

Le gradient de politique est donc le mécanisme par lequel les probabilités d'action produites par la politique sont modifiées. Si le retour $R_t(\tau)$> 0, alors la probabilité de l'action $\pi_\theta(a_t \mid s_t)$ est augmentée ; à l'inverse, si $R_t(\tau)$ < 0, cette probabilité est diminuée. Au fil de nombreuses mises à jour (Équation 2.10), la politique apprend ainsi à produire des actions qui génèrent des retours élevés $R_t(\tau)$. L'Équation 2.11 constitue la base de toutes les méthodes de gradient de politique. L'algorithme REINFORCE fut le premier à l'utiliser dans sa forme la plus simple.

c) Méthode du gradient de politique par Monte Carlo

L'algorithme REINFORCE estime numériquement le gradient de politique en utilisant l'échantillonnage Monte Carlo. L'échantillonnage Monte Carlo fait référence à toute méthode qui utilise un échantillonnage aléatoire pour générer des données utilisées pour approximer une fonction.

C'est une approche très simple. L'espérance implique que, à mesure que davantage de trajectoires τ_S sont échantillonnées à l'aide d'une politique $\mathbb{E}_{\tau\sim\pi_\theta}$, cela rapproche le gradient de politique réel et, une fois moyennées, ces trajectoires approchent l'objectif $\nabla_\theta J(\pi_\theta)$. Plutôt que d'échantillonner de nombreuses trajectoires par politique, nous pouvons en échantillonner une seule, comme le montre l'Équation 2.12.

$$\nabla_\theta J(\pi_\theta) \approx \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} R_t(\tau)\nabla_\theta \log \pi_\theta (a_t \mid s_t)] \quad (2.12)$$

C'est ainsi que le gradient de politique est implémenté comme une estimation Monte Carlo sur les trajectoires échantillonnées.

2.6.2 Algorithmes Basés sur la Valeur (Value-Based Algorithms)

Un agent apprend soit V(s), soit $Q(s,a)$. Il utilise la fonction de valeur apprise pour évaluer les paires (s, a) et générer une politique. Par exemple, la politique d'un agent peut consister à toujours choisir l'action a dans l'état s ayant la plus haute valeur estimée $Q(s,a)$. L'apprentissage de $Q(s,a)$ est bien plus courant que celui de V(s) dans les approches purement basées sur la valeur, car il est plus facile à convertir en politique. Cela s'explique par le fait que $Q(s,a)$ contient des informations sur les couples état-action, tandis que V(s) ne contient que des informations sur les états.

SARSA est l'un des anciens algorithmes d'apprentissage par renforcement. Malgré sa simplicité, SARSA intègre de nombreuses idées fondamentales des méthodes basées sur la valeur. Cependant, il est aujourd'hui rarement utilisé en raison de sa forte variance et de son inefficacité en matière d'échantillons lors de l'entraînement. Les Deep Q-Networks (DQN) et leurs variantes, comme Double DQN et DQN avec Rejeu d'Expérience Prioritaire (PER), sont des algorithmes bien plus populaires et efficaces.

Les algorithmes basés sur la valeur sont généralement plus efficaces en termes d'échantillons que les algorithmes basés sur la politique. Cela s'explique par leur plus faible variance et leur meilleure exploitation des données collectées depuis l'environnement. Toutefois, il n'existe aucune garantie que ces algorithmes convergent vers un optimum. Dans leur formulation standard, ils ne s'appliquent également qu'aux environnements avec des espaces d'actions discrets. Cela a longtemps représenté une limite majeure, mais grâce aux avancées récentes, comme QT-OPT, ils peuvent désormais être appliqués efficacement à des environnements avec des espaces d'actions continus.

2.6.2.1 SARSA

SARSA, un algorithme basé sur la valeur, a été inventé par Rummery et Niranjan dans leur article de 1994 intitulé "On-Line Q-Learning Using Connectionist Systems" [37]. Il a été nommé ainsi car « il est nécessaire de connaître l'enchaînement État-Action-Récompense-État-Action avant de procéder à une mise à jour ».

Les algorithmes basés sur la valeur évaluent les paires état-action (s, a) en apprenant l'une des fonctions de valeur V(s) ou Q(s, a) et utilisent ces évaluations pour sélectionner les actions. L'apprentissage des fonctions de valeur s'oppose à celui de REINFORCE, dans lequel un agent apprend directement une politique qui associe les états aux actions. L'algorithme SARSA apprend la fonction Q(s, a), tandis que d'autres algorithmes, comme Actor-Critic, apprennent V(s).

L'algorithme SARSA repose sur deux idées principales. La première est une technique d'apprentissage de la fonction Q appelée apprentissage par différence temporelle (TD learning). Il s'agit d'une alternative à l'échantillonnage Monte Carlo pour estimer les valeurs des états ou des paires état-action à partir des expériences collectées par un agent dans un environnement.

La seconde idée essentielle est une méthode de génération d'actions à partir de la fonction Q. Cela soulève la question de comment les agents découvrent de bonnes actions. L'un des défis fondamentaux en apprentissage par renforcement est de trouver un bon équilibre entre exploiter ce que l'agent sait déjà et explorer l'environnement pour en apprendre davantage. C'est ce qu'on appelle le dilemme exploration-exploitation.

a) Les Fonctions *Q* et V

La fonction Q mesure la valeur des paires état-action (s, a) sous une politique donnée, comme défini dans l'Équation 2.13. La valeur de (s, a) correspond à la récompense cumulative escomptée, actualisée, obtenue à partir de l'action *a* dans l'état *s*, suivie par la poursuite de la politique.

$$V^{\pi}(s) = \mathbb{E}_{s_0=s,\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^{t}r_{t}\right] \tag{2.13}$$

Les fonctions de valeur sont toujours définies relativement à une politique spécifique, ce qui justifie la notation avec un exposant. Pour comprendre cela, supposons que l'on évalue (s, a). $Q^{\pi}(s,a)$ dépend des séquences de récompenses que l'agent peut espérer recevoir après avoir pris l'action *a* dans l'état *s*. Ces récompenses dépendent des états et actions futurs, eux-mêmes conditionnés par une politique donnée. Des politiques différentes peuvent engendrer des séquences d'actions futures distinctes à partir de (s, a), menant ainsi à des récompenses différentes.

La fonction $V^{\pi}(s)$ évalue la valeur d'un état *s* selon une politique donnée et est définie par l'Équation 2.13. La valeur d'un état $V^{\pi}(s)$ est la récompense cumulative escomptée à partir de cet état *s* sous une politique spécifique.

La fonction $Q^{\pi}(s,a)$ est étroitement liée à $V^{\pi}(s)$. En effet, $V^{\pi}(s)$ représente l'espérance des valeurs Q pour toutes les actions *a* possibles dans l'état *s* sous la politique considérée :

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi(s)}[Q^{\pi}(s,a)] \tag{2.14}$$

Quelle fonction est-il préférable pour un agent d'apprendre : $Q^{\pi}(s,a)$ ou $V^{\pi}(s)$? La fonction $Q^{\pi}(s,a)$ présente l'avantage d'offrir à l'agent un moyen direct pour agir. L'agent peut calculer $Q^{\pi}(s,a)$ pour chaque action a disponible dans l'état s, puis sélectionner celle ayant la valeur maximale. Dans le cas optimal, $Q^{\pi}(s,a)$ représente la valeur escomptée optimale de l'action a dans l'état s, notée $Q^{*}(s,a)$. Elle correspond à la meilleure performance possible si l'agent agit de manière optimale par la suite. Connaître $Q^{*}(s,a)$ permet ainsi de dériver une politique optimale.

Cependant, l'apprentissage de $Q^{\pi}(s,a)$ est plus complexe d'un point de vue computationnel et requiert davantage de données que V(s). Estimer correctement $V^{\pi}(s)$ exige une couverture raisonnable de l'espace des états, tandis qu'une estimation fiable de $Q^{\pi}(s,a)$ nécessite que toutes les paires (s, a) soient bien représentées, ce qui est plus exigeant en données, étant donné que l'espace des paires état-action est souvent bien plus vaste que celui des états seuls.

$V^{\pi}(s)$, bien que plus simple à approximer, présente un inconvénient majeur : pour choisir une action via $V^{\pi}(s)$, l'agent doit tester chaque action *a* disponible dans l'état *s*, observer l'état suivant *s'* et la récompense *r*. Ensuite, il peut agir de manière optimale en sélectionnant l'action qui maximise l'espérance $\mathbb{E}[r + V^{\pi}(s')]$. Mais si les transitions d'état sont stochastiques, l'agent devra répéter cette opération plusieurs fois afin d'estimer précisément la valeure escomptée de chaque action, ce qui constitue un processus potentiellement coûteux.

Cette exigence de *regard en avant à un pas* de $V^{\pi}(s)$ peut poser problème. Il se peut, par exemple, qu'il soit impossible de redémarrer l'agent dans n'importe quel état pour tester toutes les actions possibles, ou que cela soit coûteux. $Q^{\pi}(s,a)$ contourne ce problème en apprenant directement la valeur de (s, a), ce qui équivaut à mémoriser le regard en avant pour chaque action dans chaque état. Par conséquent, les algorithmes d'apprentissage par renforcement qui sélectionnent les actions à partir d'une fonction de valeur apprise tendent à approximer $Q^{\pi}(s,a)$.

b) Apprentissage par Différence Temporelle (TD)

Dans cette section, nous présentons la méthode d'apprentissage de la fonction Q par l'approche Temporal Difference (TD). L'idée principale consiste à utiliser un réseau de neurones pour estimer les valeurs Q à partir de paires (s, a). Ce réseau est appelé réseau de valeur (value network).

Le processus d'apprentissage des paramètres de ce réseau de valeur suit les étapes suivantes : on génère des trajectoires τ_s d'états s et on prédit une valeur Q pour chaque paire (s, a). Ensuite, ces trajectoires sont utilisées pour générer des valeurs cibles Q^{π}_{tar}. Enfin, on minimise la distance entre les valeurs prédites Q et les valeurs cibles Q^{π}_{tar} en utilisant une fonction de perte standard de régression telle que l'erreur quadratique moyenne (MSE – Mean Squared Error). Ce processus est répété de nombreuses fois.

L'idée clé de l'apprentissage TD repose sur le fait que les valeurs Q à un instant donné peuvent être définies en fonction des valeurs Q au pas de temps suivant. Autrement dit, Q(s, a) est définie de manière récursive, comme l'illustre l'Équation 2.15 :

$$Q^{\pi}(s,a) = \mathbb{E}_{s'\sim p(s'|s,a),r\sim\mathcal{R}(s,a,s')}\left[r + \gamma\mathbb{E}_{a'\sim\pi(s')}[Q^{\pi}(s',a')]\right] \tag{2.15}$$

Cette équation est connue sous le nom d'équation de Bellman. Si la fonction Q est correcte pour une politique donnée π, alors l'Équation 2.16 est exactement vérifiée. Elle fournit également une méthode pour apprendre les valeurs Q. Nous avons vu précédemment comment l'échantillonnage Monte Carlo peut être utilisé pour apprendre Q(s, a) à partir de nombreuses trajectoires d'expériences. Le même principe peut être appliqué ici, en utilisant l'apprentissage TD pour dériver la valeur cible Q^{π}_{tar} pour chaque paire (s, a).

Supposons que nous disposions d'un réseau de neurones pour représenter la fonction Q. En apprentissage TD, $Q^{\pi}_{tar}(s\ , a)$ est calculée en estimant le membre de droite de l'Équation 2.15 à l'aide de Q. À chaque itération d'entraînement, $Q(s_t, a_t)$ est mis à jour pour se rapprocher de $Q^{\pi}_{tar}(s\ , a)$.

Cependant, deux difficultés se posent lorsqu'on utilise l'Équation 2.15 pour construire $Q^{\pi}_{tar}(s\ , a)$: les deux espérances qu'elle contient.

La première difficulté concerne l'espérance extérieure $\mathbb{E}_{S'\sim p(s'|s,a),r\sim R(s,a,s')}[\dots]$, qui dépend de l'état suivant et de la récompense. Pour illustrer cela, supposons que nous ayons une collection de trajectoires $\tau_1, \tau_2, \dots, \tau_M$. Chaque trajectoire τ_i contient des tuples (s, a, r, s′). Pour chaque tuple, un seul exemple de l'état suivant s′ est disponible, étant donné l'action choisie.

Si l'environnement est déterministe, il est alors correct de ne considérer que l'état suivant effectivement observé pour calculer l'espérance. En revanche, si l'environnement est stochastique, cela pose problème : une action a dans l'état s pourrait mener à plusieurs états suivants différents, mais seul un état est observé à chaque pas. Cette difficulté peut être contournée en ne prenant en compte que l'exemple observé. Cela implique que l'estimation de la valeur Q peut avoir une variance élevée dans un environnement stochastique, mais cela rend l'estimation plus praticable. En n'utilisant qu'un seul exemple pour estimer la distribution des états suivants s′ et des récompenses r, on peut sortir l'espérance extérieure de l'équation, qui devient alors :

$$Q^{\pi}(s,a) = r + \gamma \mathbb{E}_{a'\sim\pi(s')}[Q^{\pi}(s',a')] \tag{2.16}$$

La deuxième difficulté concerne l'espérance intérieure $\mathbb{E}_{a'\sim\pi(s')}[\dots]$, portant sur les actions. Nous avons accès aux estimations de Q pour toutes les actions possibles dans l'état suivant s′, car nous utilisons l'estimation actuelle de la fonction Q pour les calculer. Le problème est que nous ne connaissons pas la distribution de probabilité sur les actions, nécessaire pour calculer l'espérance. Deux approches permettent de résoudre ce problème, chacune correspondant à un algorithme différent : SARSA et DQN.

La solution de SARSA est d'utiliser l'action effectivement prise dans l'état suivant a, comme illustré dans l'Équation 2.17. À travers de nombreux exemples, la proportion des actions sélectionnées devrait approximer la distribution de probabilité sur les actions étant donné les états.

La solution de DQN est d'utiliser la valeur Q maximale parmi toutes les actions possibles dans l'état suivant, comme dans l'Équation 2.18. Cela correspond à une politique implicite consistant à toujours sélectionner l'action avec la valeur Q maximale (politique greedy).

$$\text{SARSA: } Q^{\pi}(s,a) \approx r + \gamma Q^{\pi}(s',a') = Q^{\pi}_{\text{tar:SARSA}}(s,a) \tag{2.17}$$

$$\text{DQN: } Q^{\pi}(s,a) \approx r + \gamma \max_{a'_i} Q^{\pi}(s',a'_i) = Q^{\pi}_{\text{tar:DQN}}(s,a) \tag{2.18}$$

Ainsi, $Q^{\pi}_{tar}(s\ , a)$ peut être calculé à partir du membre de droite de l'équation 2.18 ou 2.19 pour chaque tuple (s, a, r, s′, a′).

c) Exploration et Exploitation

Un problème lié à une politique strictement *gloutonne* (greedy), c'est-à-dire fondée uniquement sur les valeurs Q, est qu'elle est déterministe. Cela signifie qu'un agent risque de ne pas explorer suffisamment l'ensemble de l'espace état-action. Si un agent sélectionne toujours la même action a dans un état s,

alors de nombreuses paires (s, a) risquent de ne jamais être expérimentées. En conséquence, les estimations de la fonction Q pour certaines paires (s, a) peuvent être inexactes puisqu'elles ont été initialisées aléatoirement et qu'aucune expérience réelle n'a permis de les corriger. Le réseau de neurones n'a donc jamais eu l'occasion d'apprendre à propos de cette partie de l'espace, ce qui peut conduire l'agent à prendre des décisions sous-optimales et à rester bloqué dans un minimum local.

Pour atténuer ce problème, il est courant d'utiliser une politique ε-greedy plutôt qu'une politique purement gloutonne. Dans une telle politique, l'agent choisit l'action gloutonne avec une probabilité $1-\varepsilon$, et une action aléatoire avec une probabilité ε. Cette dernière est appelée probabilité d'exploration, car en agissant de manière aléatoire (même à 100 %), l'agent explore mieux l'espace état-action. Toutefois, cette exploration a un coût : une telle politique peut être moins performante qu'une politique gloutonne, car l'agent prend des actions aléatoires avec une probabilité non nulle, au lieu de systématiquement maximiser la valeur Q.

La tension entre l'exploration (pour découvrir de nouveaux états et comportements) et l'exploitation (tirer parti des connaissances actuelles pour maximiser les récompenses) est connue sous le nom de **compromis exploration-exploitation**. L'agent doit-il exploiter ce qu'il sait déjà pour maximiser ses performances immédiates, ou prendre le risque de mal agir pour éventuellement découvrir de meilleures stratégies ? Une autre manière d'exprimer ce dilemme est la suivante : si l'agent avait accès à la fonction Q optimale, il devrait toujours agir de manière gloutonne ; cependant, tant qu'il est en phase d'apprentissage, se limiter à des choix gloutons peut l'empêcher de progresser.

Une méthode classique pour gérer ce compromis consiste à commencer avec une valeur élevée de ε, proche de 1.0, afin que l'agent agisse presque totalement au hasard et explore rapidement l'espace état-action. En effet, au début de l'apprentissage, l'agent ne sait encore rien, donc il n'y a rien à exploiter. Progressivement, ε est diminué, de sorte qu'après un grand nombre d'itérations, la politique se rapproche idéalement de la politique optimale. À mesure que l'agent apprend de meilleures fonctions Q, et donc de meilleures politiques, le besoin d'exploration diminue, et l'agent peut se comporter de manière plus gloutonne.

Dans le meilleur des cas, après un certain temps, l'agent aura découvert la fonction Q optimale, et l'on pourra alors réduire ε à 0. En pratique, en raison du temps limité d'apprentissage, de la nature continue ou de la haute dimensionnalité de l'espace d'états discrets, et de l'utilisation d'approximâtes de fonction non linéaires, la fonction Q apprise ne converge jamais totalement vers la politique optimale. Par conséquent, ε est généralement réduit progressivement (annealed) vers une petite valeur (par exemple entre 0.1 et 0.001), puis fixé, afin de maintenir un faible niveau d'exploration continue.

L'apprentissage TD a été démontré comme convergent vers la fonction Q optimale lorsqu'il est utilisé avec des approximâtes de fonction linéaires. Cependant, les progrès récents en apprentissage par renforcement sont dus à l'introduction d'approximâtes de fonction non linéaires complexes, comme les réseaux de neurones, qui peuvent représenter des fonctions Q bien plus riches. Malheureusement, cette évolution vers des approximations non linéaires s'accompagne de l'absence de garantie de convergence de l'apprentissage TD.

2.6.2.2 Le Réseau Q Profond (DQN)

DQN (*Deep Q-Network*) est un algorithme fondé sur la différence temporelle (TD), qui repose sur l'apprentissage d'une fonction de valeur Q. La fonction Q ainsi apprise est ensuite utilisée par l'agent pour sélectionner des actions. DQN est uniquement applicable aux environnements à espaces d'action discrets**.** Cependant, DQN apprend une fonction Q différente de celle apprise par SARSA : il s'agit de la fonction Q optimale, et non de la fonction Q correspondant à la politique courante. Ce changement, bien que mineur en apparence, est crucial, car il améliore la stabilité et la rapidité de l'apprentissage.

DQN est un algorithme hors-politique (off-policy). Cela signifie que la fonction Q optimale n'est pas dépendante de la politique utilisée pour collecter les données. DQN peut donc apprendre à partir des expériences collectées par n'importe quel agent.

Transformer SARSA en DQN ne requiert qu'un léger ajustement de la mise à jour de la fonction Q, mais cela donne naissance à un algorithme qui surmonte de nombreuses limitations de SARSA. DQN permet notamment de décorréler et réutiliser les expériences grâce à l'échantillonnage aléatoire de mini-lots dans une mémoire de rejouement (experience replay), ce qui autorise de multiples mises à jour des paramètres à partir du même lot. Ces mécanismes renforcent considérablement l'efficacité de DQN par rapport à SARSA.

a) Apprentissage de la fonction Q dans DQN

DQN, comme **SARSA**, apprend la fonction Q en utilisant l'apprentissage par différences temporelles (TD). La différence entre les deux algorithmes réside dans la manière dont est construite la valeur cible $Q_{tar}^{\pi}(s\ ,a)$.

Les équations 2.19 et 2.20 présentent respectivement l'équation de Bellman pour **DQN** et **SARSA** :

$$Q_{\mathrm{DQN}}^{\pi}(s,a) \approx r + \gamma \max_{a'} Q^{\pi}(s',a') \tag{2.19}$$

$$Q_{\mathrm{SARSA}}^{\pi}(s,a) \approx r + \gamma Q^{\pi}(s',a') \tag{2.20}$$

De manière correspondante, les équations 2.21 et 2.22 indiquent comment est construite la valeur cible $Q_{tar}^{\pi}(s\ ,a)$ dans chaque algorithme :

$$Q_{\mathrm{tar:DQN}}^{\pi}(s,a) = r + \gamma \max_{a'} Q^{\pi}(s',a') \tag{2.21}$$

$$Q_{\mathrm{tar:SARSA}}^{\pi}(s,a) = r + \gamma Q^{\pi}(s',a') \tag{2.22}$$

Contrairement à SARSA, qui utilise l'action réellement choisie dans l'état suivant s′ pour estimer $Q_{tar}^{\pi}(s\ ,a)$, DQN utilise la valeur maximale parmi toutes les actions possibles dans cet état. En d'autres termes, dans DQN, la valeur Q utilisée pour l'état suivant s′ ne dépend pas de la politique ayant servi à collecter les expériences.

Étant donné que la récompense r et l'état suivant s′ sont produits par l'environnement à partir de l'état courant s et de l'action a, aucune partie de l'estimation de $Q_{tar}^{\pi}(s\ ,a)$ ne dépend de la politique utilisée pour collecter les données [38]. Cela fait de DQN un algorithme off-policy, car la fonction apprise est indépendante de la politique suivie pour interagir avec l'environnement. À l'inverse, SARSA est un algorithme on-policy, car il utilise l'action a′ effectivement choisie par la politique courante dans l'état s′ pour calculer la valeur Q suivante. L'apprentissage dépend donc directement de la politique utilisée pour générer les expériences.

b) La politique de Boltzmann

Une politique gloutonne ou ε-greedy ne constitue pas la seule option disponible pour un agent utilisant DQN ou SARSA. Dans cette section, nous présentons une troisième option : la politique de Boltzmann, nommée d'après la distribution de probabilité de Boltzmann.

Une bonne politique doit parvenir à équilibrer l'exploration de l'espace état-action et l'exploitation des connaissances acquises par l'agent. Les politiques ε-greedy tentent d'équilibrer exploration et exploitation en réduisant progressivement la probabilité de choisir des actions aléatoires au fur et à mesure de l'entraînement. Ce type de politique favorise donc l'exploration au début de l'apprentissage, et l'exploitation par la suite. Cependant, cette stratégie d'exploration reste naïve : l'agent explore de manière aléatoire sans tenir compte des connaissances déjà acquises sur l'environnement.

La politique de Boltzmann cherche à améliorer ce comportement en sélectionnant les actions selon leur valeur Q relative. Ainsi, l'action a ayant la plus haute valeur Q dans un état s sera choisie le plus souvent, mais les autres actions ayant également des valeurs Q relativement élevées auront aussi une probabilité significative d'être sélectionnées. Inversement, les actions avec des valeurs Q très faibles seront rarement choisies**.** Cela permet de concentrer l'exploration sur des actions prometteuses, plutôt que de répartir les probabilités de façon uniforme comme le fait une exploration aléatoire.

Pour formaliser la politique de Boltzmann, on définit une distribution de probabilité sur les actions a dans un état s en utilisant un paramètre de température τ, comme le montre l'équation suivante :

$$p_{\text{boltzmann}}(a \mid s) = \frac{e^{Q^{\pi}(s,a)/\tau}}{\sum_{a'} e^{Q^{\pi}(s,a')/\tau}} \tag{2.23}$$

Le rôle du paramètre de température τ dans la politique de Boltzmann est analogue à celui de ε dans la politique ε-greedy : il favorise l'exploration de l'espace état-action.

L'avantage principal de la politique de Boltzmann par rapport à la politique ε-greedy réside dans le fait qu'elle explore l'environnement de manière moins aléatoire. À chaque fois que l'agent sélectionne une action, celle-ci est échantillonnée à partir de la distribution de probabilité sur les actions générée par la politique de Boltzmann.

Plutôt que d'agir de façon entièrement aléatoire avec une probabilité ε, l'agent sélectionne une action a avec une probabilité $P_{\text{Boltzmann}}(a|s)$ Ainsi, les actions ayant des valeurs Q plus élevées ont une plus grande probabilité d'être choisies. Même si l'agent ne choisit pas l'action qui maximise la valeur Q, il est plus probable qu'il choisisse la deuxième meilleure action plutôt que la troisième ou la quatrième, selon les estimations de Q(s, a).

En outre, une politique de Boltzmann engendre une relation plus lisse entre les estimations des valeurs Q et les probabilités associées aux actions, comparée à celle produite par une politique ε-greedy.

Les actions devraient être sélectionnées avec une probabilité approximativement égale lorsque leurs valeurs Q sont proches. C'est précisément ce que permet la politique de Boltzmann. En revanche, les politiques ε-greedy engendrent des comportements plus extrêmes : si l'une des valeurs Q est légèrement supérieure à l'autre, la politique ε-greedy attribuera toute la probabilité non aléatoire (1−ε) à cette action. Si, à l'itération suivante, l'autre action obtient une valeur Q légèrement plus élevée, la politique ε-greedy basculera immédiatement pour attribuer toute la probabilité non aléatoire à cette nouvelle action. Ainsi, une politique ε-greedy peut se révéler plus instable qu'une politique de Boltzmann, ce qui peut compliquer l'apprentissage de l'agent.

Cependant, une politique de Boltzmann présente aussi des limites : elle peut conduire un agent à rester bloqué dans un minimum local, si l'estimation de la fonction Q est inexacte dans certaines régions de l'espace des états.

c) Expérience de Replay

Les algorithmes hors politique tels que DQN n'ont pas besoin de jeter les expériences une fois qu'elles ont été utilisées. Long-Ji Lin [39] a fait cette observation pour la première fois en 1992 et a proposé un ajout à Q-learning appelé "expérience de replay". Il a observé que l'apprentissage TD pouvait être lent en raison du mécanisme d'essai-erreur pour la collecte des données inhérent à l'apprentissage par renforcement (RL) et de la nécessité de propager l'information en arrière dans le temps. Accélérer l'apprentissage TD revient à accélérer soit le processus d'affectation des crédits, soit à raccourcir le processus d'essai-erreur [40]. L'expérience de replay se concentre sur ce dernier aspect en facilitant la réutilisation des expériences.

Une mémoire d'expérience de replay stocke les ***k*** expériences les plus récentes qu'un agent a collectées. Si la mémoire est pleine, l'expérience la plus ancienne est supprimée pour faire de la place pour la plus récente. Chaque fois qu'un agent s'entraîne, un ou plusieurs lots (batch) de données sont échantillonnés de manière aléatoire et uniforme depuis la mémoire d'expérience de replay. Chacun de ces lots est utilisé à son tour pour mettre à jour les paramètres du réseau de la fonction Q. La valeur de ***k*** est généralement assez grande, entre 10 000 et 1 000 000, tandis que le nombre d'éléments dans un lot est beaucoup plus petit, typiquement entre 32 et 2048.

La taille de la mémoire doit être suffisamment grande pour contenir de nombreuses époques d'expériences. Chaque lot contiendra généralement des expériences provenant de différentes époques et de différentes politiques, ce qui décorelle les expériences utilisées pour entraîner un agent. Cela réduit la variance des mises à jour des paramètres, ce qui aide à stabiliser l'entraînement. Cependant, la mémoire doit également être suffisamment petite pour qu'il soit probable qu'une expérience soit échantillonnée plus d'une fois avant d'être supprimée, ce qui rend l'apprentissage plus efficace. La suppression des expériences les plus anciennes est également importante. À mesure qu'un agent apprend, la distribution des paires (s, a) qu'il rencontre change. Les expériences plus anciennes deviennent moins utiles car un agent a moins de chances de revenir sur les états plus anciens. Avec un temps et des ressources informatiques limités, il est préférable qu'un agent se concentre sur l'apprentissage à partir des expériences les plus récentes, car elles sont généralement plus pertinentes. Le fait de ne stocker que les ***k*** expériences les plus récentes dans la mémoire met en œuvre cette idée.

Le principal problème avec une représentation tabulaire de la fonction Q est qu'elle n'apprend rien sur les relations entre les états et les actions. En revanche, un réseau de neurones est capable d'extrapoler les valeurs Q pour des paires (s, a) inconnues à partir des paires connues, car il apprend à modéliser les relations entre états et actions. Cette capacité est très utile lorsque l'espace (s, a) est grand ou infini, car elle permet à l'agent d'apprendre une bonne estimation de la fonction Q sans devoir visiter toutes les paires (s, a). Il suffit que l'agent explore un sous-ensemble représentatif de l'espace état-action.

2.6.2.3 Le Double Réseau Q Profond (DDQN)

L'algorithme Double DQN permet de réduire la surestimation des valeurs de Q en apprenant deux estimations différentes de la fonction Q à partir d'expériences distinctes. L'action a qui maximise Q est sélectionnée à l'aide de la première estimation, tandis que la valeur Q utilisée pour calculer $Q^{\pi}_{tar}(s\ ,a)$ est générée par la seconde estimation, en utilisant l'action a choisie par la première.

L'utilisation d'une seconde fonction Q, entraînée à partir d'expériences différentes, élimine le biais positif dans l'estimation. La modification du DQN en Double DQN est relativement simple. Cette modification devient plus claire si l'on réécrit l'expression de $Q^{\pi}_{tar}(s\ ,a)$ comme illustré dans l'Équation 2.24 :

$$\begin{aligned} Q^{\pi}_{\text{tar:DQN}}(s,a) &= r + \gamma \max_{a'} Q^{\pi_\theta}(s',a') \\ &= r + \gamma Q^{\pi_\theta}\left(s', \max_{a'} Q^{\pi_\theta}(s',a')\right) \end{aligned} \tag{2.24}$$

L'algorithme DQN utilise le même réseau pour sélectionner l'action a et pour évaluer la fonction Q correspondant à cette action. À l'inverse, Double DQN utilise deux réseaux différents : le réseau de formation, noté θ, est utilisé pour sélectionner l'action, tandis que le réseau cible, noté φ, est utilisé pour calculer la valeur Q de (s′, a) comme indiqué dans l'Équation 2.25 :

$$Q^{\pi}_{\text{tar:DoubleDQN}}(s,a)r + \gamma Q^{\pi_\varphi}\left(s', \max_{a'} Q^{\pi_\theta}(s',a')\right) \quad (2.25)$$

Depuis l'introduction des réseaux cibles en Section 5.1, nous disposons déjà de deux réseaux : le réseau d'entraînement et le réseau cible φ. Bien que ces réseaux soient entraînés à partir d'expériences partiellement similaires, si l'intervalle de temps entre les mises à jour φ ← θ est suffisamment long, alors ils sont, en pratique, suffisamment différents pour remplir les rôles distincts requis par Double DQN. Le réseau d'entraînement θ est utilisé pour sélectionner l'action, tandis que le réseau cible φ est utilisé pour évaluer cette action. Il est important de souligner que si aucun délai n'est introduit entre θ et φ (c'est-à-dire si φ = θ), alors l'Équation 2.24 se réduit à l'algorithme DQN original.

2.6.3 Algorithmes basés sur un modèle (Model-Based Algorithms)

Les algorithmes de cette famille apprennent un modèle des dynamiques de transition de l'environnement, ou utilisent un modèle connu. Une fois que l'agent dispose d'un modèle de l'environnement, noté $P(s' \mid s, a)$, il peut "imaginer" ce qui va se passer dans le futur en prédisant une trajectoire sur plusieurs étapes. Si l'environnement est dans l'état s, l'agent peut estimer comment cet état évoluera en effectuant une séquence d'actions $a1, a2, \ldots, an$, en appliquant de manière répétée la fonction $P(s' \mid s, a)$, et ce sans réellement interagir avec l'environnement. Ainsi, la trajectoire prédite est simulée mentalement à l'aide du modèle. L'agent peut simuler de nombreuses trajectoires avec différentes séquences d'actions, puis comparer ces options pour choisir l'action optimale à effectuer.

Les approches purement basées sur un modèle sont souvent appliquées à des jeux avec un état cible, comme gagner ou perdre une partie d'échecs, ou des tâches de navigation vers un état objectif s*. Cela s'explique par le fait que les fonctions de transition ne modélisent généralement pas les récompenses. Pour planifier des actions, des informations sur l'objectif de l'agent doivent être encodées dans les états eux-mêmes.

Monte Carlo Tree Search (MCTS) est une méthode bien connue de ce type. Elle peut être utilisée dans des environnements discrets et déterministes, avec des fonctions de transition connues. De nombreux jeux de plateau comme les échecs ou le Go entrent dans cette catégorie. Jusqu'à récemment, MCTS alimentait la plupart des programmes informatiques jouant au Go. Cette méthode ne fait pas appel à l'apprentissage automatique, mais explore les états du jeu et estime leur valeur en effectuant des séquences aléatoires d'actions, appelées rollouts Monte Carlo.

D'autres méthodes comme les régulateurs quadratiques linéaires itératifs (iLQR) ou le contrôle prédictif par modèle (MPC) consistent à apprendre les dynamiques de transition, souvent sous des hypothèses assez restrictives. Pour ce faire, l'agent doit interagir avec l'environnement afin de collecter des exemples de transitions réelles (tuples (s,a,r,s′)).

Les algorithmes basés sur un modèle sont attrayants car un modèle parfait donne à l'agent une forme de prévoyance : il peut simuler des scénarios et comprendre les conséquences de ses actions sans agir réellement dans l'environnement. C'est un avantage majeur dans les cas où l'interaction avec l'environnement est coûteuse ou lente, comme en robotique. Comparés aux méthodes fondées sur la politique ou sur la valeur, ces algorithmes nécessitent souvent beaucoup moins d'échantillons pour apprendre de bonnes politiques, car l'agent peut enrichir ses expériences réelles par des expériences imaginées.

Cependant, dans la plupart des cas, il est difficile d'obtenir un bon modèle. De nombreux environnements sont stochastiques, et leurs dynamiques de transition ne sont pas connues. Dans ces situations, il faut apprendre le modèle, ce qui pose plusieurs défis. Premièrement, un environnement avec un grand espace d'états et d'actions est difficile à modéliser, voire intractable si les transitions sont très complexes. Deuxièmement, les modèles ne sont utiles que s'ils peuvent prédire avec précision les transitions à plusieurs étapes dans le futur. Si le modèle est imparfait, les erreurs de prédiction peuvent s'accumuler à chaque pas de temps, rendant le modèle rapidement inutilisable.

Le manque de bons modèles est aujourd'hui l'une des principales limites à l'applicabilité des approches basées sur un modèle. Pourtant, lorsqu'ils fonctionnent, ces algorithmes peuvent être 10 à 100 fois plus efficaces en échantillons que les méthodes sans modèle.

La distinction entre algorithmes avec modèle et sans modèle sert également à classifier les algorithmes d'apprentissage par renforcement. Un algorithme model-based est simplement un algorithme qui utilise les dynamiques de transition d'un environnement, qu'elles soient apprises ou connues à l'avance. Les algorithmes model-free, en revanche, n'utilisent pas explicitement ces dynamiques.

2.6.4 Méthodes combinées (Combined Methods)

Ces algorithmes apprennent deux ou plusieurs fonctions fondamentales de l'apprentissage par renforcement. Étant donné les forces et faiblesses de chacune des trois méthodes précédentes (basées sur la valeur, la politique, ou un modèle), il est naturel de vouloir les combiner pour tirer parti de leurs avantages respectifs.

Un groupe largement utilisé d'algorithmes apprend à la fois une politique et une fonction de valeur. Ces méthodes sont appelées algorithmes acteur-critique (Actor-Critic) car la politique joue le rôle de l'acteur, tandis que la fonction de valeur critique ses actions. L'idée clé est que pendant l'apprentissage, la fonction de valeur apprise fournit à la politique un signal de retour plus informatif que la simple séquence de récompenses fournies par l'environnement. La politique apprend donc grâce aux informations issues de la fonction de valeur. Ensuite, comme dans les méthodes basées sur la politique, cette politique est utilisée pour générer des actions.

Les algorithmes acteur-critique constituent un domaine de recherche actif, avec de nombreux développements récents, notamment : **Trust Region Policy Optimization (TRPO), Proximal Policy Optimization (PPO)** – actuellement le plus utilisé, **Deep Deterministic Policy Gradient (DDPG)** et **Soft Actor-Critic (SAC)**.

Certains algorithmes utilisent également un modèle de l'environnement en combinaison avec une fonction de valeur et/ou une politique. En 2016, les chercheurs de DeepMind ont développé AlphaGo, qui combinait MCTS, l'apprentissage d'une fonction de valeur V, et une politique, pour maîtriser le jeu de Go [41]. Un autre algorithme connu, Dyna-Q [42], apprend itérativement un modèle à partir des données réelles issues de l'environnement, puis utilise les données simulées par le modèle appris pour améliorer l'apprentissage de la fonction Q.

2.6.4.1 Advantage Actor-Critic (A2C)

Les algorithmes Actor-Critic combinent de manière élégante deux idées clés : le gradient de politique (*policy gradient*) et une fonction de valeur apprise. Dans ces algorithmes, une politique est renforcée par un signal d'apprentissage généré à l'aide d'une fonction de valeur elle-même apprise. Cela contraste avec l'approche REINFORCE, qui utilise une estimation Monte Carlo du retour, connue pour sa forte variance, afin de renforcer la politique. Tous les algorithmes Actor-Critic comportent deux composantes apprises conjointement : **l'acteur** (*actor*), qui apprend une politique paramétrée, **le critique** (*critic*), qui apprend une fonction de valeur pour évaluer les couples état-action.

Le critique fournit un signal de renforcement à l'acteur. La motivation principale derrière ces algorithmes est que ce signal de renforcement appris peut être plus informatif que les récompenses brutes fournies par l'environnement.

L'apprentissage de la politique dépend donc de la qualité de l'estimation de la fonction de valeur, laquelle est également en cours d'apprentissage. Tant que cette fonction ne génère pas de signaux pertinents, il est difficile pour la politique d'apprendre à sélectionner de bonnes actions.

Dans ces méthodes, il est courant d'apprendre la fonction d'avantage, définie par :

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{2.26}$$

Cette fonction évalue dans quelle mesure une action est meilleure ou moins bonne que la moyenne des actions disponibles dans un état donné. Cette approche privilégie une évaluation relative des actions plutôt qu'une évaluation absolue fondée uniquement sur la fonction Q. Les algorithmes Actor-Critic qui exploitent cette fonction sont appelés Advantage Actor-Critic (A2C).

a) L'Acteur

L'acteur apprend une politique paramétrée en utilisant le gradient de politique, comme illustré par les équations suivantes :

$$\textit{Actor-Critic:} \quad \nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_t\left[A_t^{\pi} \nabla_{\theta} \log \pi_{\theta}(a_t \mid s_t)\right] \tag{2.27}$$

$$\textit{REINFORCE:} \quad \nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_t\left[R_t(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t \mid s_t)\right] \tag{2.28}$$

La principale différence ici est que REINFORCE utilise une estimation brute du retour $\mathcal{R}_t(\tau)$ tandis qu'Actor-Critic emploie l'avantage A_t^{π}, qui tend à réduire la variance du gradient tout en conservant un bon biais.

b) Le Critique

Le rôle du critique est d'apprendre à évaluer les paires état-action (s, a), puis d'utiliser ces évaluations pour générer les valeurs d'avantage A^{π}, fournissant ainsi des signaux de renforcement à l'acteur.

c) La Fonction d'Avantage

Intuitivement, la fonction d'avantage $A^{\pi}(s,a)$ indique à quel point une action est meilleure (ou pire) que la moyenne des actions dans un état donné. Elle possède plusieurs propriétés intéressantes.

$$\mathbb{E}_{a \in \mathcal{A}}[A^{\pi}(s_t,a)] = 0 \tag{2.29}$$

Cela signifie que si toutes les actions sont équivalentes, alors leur avantage est nul, et la politique ne sera pas modifiée par l'apprentissage. À l'inverse, un signal de renforcement basé sur des valeurs absolues (comme $V^{\pi}(s)$ ou $Q^{\pi}(s,a)$) pourrait conduire à modifier une politique même si aucune action ne se distingue réellement.

L'avantage est également une mesure relative. Il considère l'effet d'une action particulière par rapport à la valeur moyenne des autres actions disponibles dans le même état. Cela permet d'éviter : de pénaliser une action simplement parce que l'état actuel est défavorable, de récompenser une action simplement parce que l'état est avantageux. L'évaluation reste ainsi centrée sur l'impact futur de l'action, indépendamment de l'historique qui a mené à l'état courant.

Pour estimer $A^{\pi}(s,a)$, il est nécessaire d'estimer $Q^{\pi}(s,a)$ et $V^{\pi}(s)$. Une approche possible consiste à apprendre ces deux fonctions séparément via des réseaux de neurones distincts. Toutefois, cela présente deux inconvénients majeurs : Il faut veiller à ce que les deux estimations soient cohérentes, l'apprentissage est moins efficace et plus coûteux. C'est pourquoi, en pratique, on préfère généralement n'apprendre que $V^{\pi}(s)$, puis estimer $Q^{\pi}(s,a)$ à partir des trajectoires observées et des récompenses cumulées.

2.6.4.2 Asynchrone Advantage Actor-Critic (A3C)

Lorsque la parallélisation est synchrone (ou bloquante), le réseau global attend de recevoir un ensemble de mises à jour provenant de tous les travailleurs (workers) avant de mettre à jour ses paramètres. Réciproquement, après avoir envoyé leurs mises à jour, les travailleurs doivent attendre que le réseau global applique les nouvelles mises à jour de paramètres, afin que tous les travailleurs disposent des mêmes paramètres actualisés.

En revanche, dans le cas d'une parallélisation asynchrone (ou non bloquante), le réseau global met à jour ses paramètres dès qu'il reçoit des données de n'importe quel travailleur. De leur côté, les travailleurs mettent à jour périodiquement leurs propres paramètres en les synchronisant avec ceux du réseau global. Cela signifie que, à tout moment, les différents travailleurs peuvent utiliser des versions légèrement différentes des paramètres du modèle. La parallélisation asynchrone a été introduite pour la première fois dans le domaine de l'apprentissage par renforcement profond par Mnih *et al.* Dans l'article intitulé *"Asynchronous Methods for Deep Reinforcement Learning"*, également connu sous le nom de papier A3C (*Asynchronous Advantage Actor-Critic*) [43]. Cette approche a joué un rôle déterminant dans la démocratisation des méthodes Actor-Critic, en les rendant compétitives par rapport aux méthodes fondées sur la valeur telles que DQN.

Non seulement l'algorithme A3C a atteint des performances de pointe sur les jeux Atari, mais il a également démontré que les algorithmes d'apprentissage par renforcement profond pouvaient être entraînés plus rapidement, et ce, en utilisant des processeurs CPU, beaucoup moins coûteux que les unités de traitement graphique (GPU). L'algorithme A3C peut être vu comme une version asynchrone des algorithmes Advantage Actor-Critic.

La grande efficacité computationnelle de l'algorithme A3C (Asynchronous Advantage Actor-Critic) provient de l'utilisation d'agents apprenants distincts, exécutés en parallèle [43]. Ces agents mettent à jour de manière asynchrone les paramètres d'un réseau neuronal central à l'aide d'une méthode d'optimisation fondée sur la descente de gradient stochastique.

Chaque agent interagit avec une instance distincte de l'environnement, en prenant des décisions à partir d'une copie locale indépendante du modèle central. Lorsqu'un agent termine un épisode dans son environnement, sa copie locale du réseau est réinitialisée avec l'état courant du réseau central. À chaque fin d'épisode, les paramètres du modèle central sont mis à jour en fonction de ceux des copies locales des agents.

En substance, la méthode A3C peut être considérée comme un ensemble d'agents d'apprentissage par renforcement simples, comparables à ceux utilisés dans l'algorithme DQN. C'est l'influence collective de ces agents sur le modèle central qui explique les performances obtenues. Comme les agents apprenants ont la capacité d'explorer leur environnement de manière indépendante, les données d'entraînement deviennent diversifiées, ce qui contribue à améliorer la robustesse et l'efficacité de l'apprentissage [43].

2.6.4.3 Proximal Policy Optimization (PPO)

L'un des principaux défis lors de l'entraînement d'agents à l'aide d'algorithmes basés sur le gradient de politique réside dans leur vulnérabilité à des effondrements de performance, où l'agent se met soudainement à adopter des comportements sous-optimaux. Ce type de dégradation peut être difficile à corriger, car l'agent commence alors à générer de mauvaises trajectoires, qui sont ensuite utilisées pour continuer à entraîner la politique, amplifiant ainsi les erreurs. Nous avons également observé que les algorithmes *on-policy* présentent une inefficacité en matière d'échantillonnage, puisqu'ils ne permettent pas la réutilisation des données issues d'anciennes politiques.

L'algorithme Proximal Policy Optimization (PPO), introduit par Schulman et al. [44], en 2017, est une méthode d'optimisation de politique stochastique largement utilisée en apprentissage par renforcement profond. PPO fait partie de la famille des méthodes policy gradient, c'est-à-dire des méthodes qui apprennent directement une politique en maximisant une fonction objectif $J(\pi_\theta)$, via une procédure de gradient de politique.

PPO s'inspire des fondements théoriques de TRPO (Trust Region Policy Optimization), mais remplace la complexité liée à la contrainte de région de confiance par des heuristiques plus simples et efficaces. L'algorithme PPO peut être utilisé pour étendre des méthodes comme REINFORCE ou Actor-Critic, en remplaçant leur fonction objective initiale par celle modifiée propre à PPO. Ce changement conduit à un apprentissage plus stable et plus efficace sur le plan de l'échantillonnage.

PPO est applicable aux environnements ayant un espace d'actions discret ou continu. Il entraîne une politique stochastique de manière on-policy, ce qui signifie que les mises à jour de la politique sont faites à partir de trajectoires générées par la politique actuelle. Il utilise également la méthode actor-critic : l'« acteur » sélectionne les actions en fonction des observations, tandis que le « critique » (ou critique d'état-action) estime la valeur attendue (fonction de valeur) des états ou des paires état-action.

À chaque époque d'entraînement : 1) Un ensemble de trajectoires (épisodes) est collecté en échantillonnant l'environnement avec la politique courante. 2) Les retours cumulés (rewards-to-go) et les estimations d'avantage sont calculés. 3) La politique est mise à jour via une ascension de gradient stochastique, et la fonction de valeur via une descente de gradient.

La version de base de l'objectif de PPO repose sur l'idée de maximiser une fonction objective de substitution (surrogate objective), notée :

$$J^{\mathrm{CPI}}(\theta) = \mathbb{E}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t \mid s_t)} A_t^{\pi_{\theta_{\mathrm{old}}}}\right] = \mathbb{E}_t[r_t(\theta) A_t] \tag{2.30}$$

Où :

- $\pi_\theta(at \mid st)$: la probabilité que la politique actuelle choisisse l'action a_t dans l'état s_t.
- $\pi_{\theta old}$: la politique avant mise à jour.
- $A_t^{\pi old}$: l'avantage calculé selon l'ancienne politique.

Le rapport de probabilité est noté :

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t|s_t)} \tag{2.31}$$

a) Deux variantes principales du PPO

- **PPO avec pénalité KL adaptative (KL-PENALTY)**

Cette variante reprend l'idée de TRPO en introduisant une pénalité de divergence KL dans la fonction objectif :

$$L^{\text{KLPEN}}(\theta) = \hat{\mathbb{E}}_t \left[r_t(\theta)\hat{A}_t - \beta \text{KL}\left(\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right)\right] \quad (2.32)$$

Où :

- $\boldsymbol{KL}[\boldsymbol{\pi_\theta} \parallel \boldsymbol{\pi_{\theta old}}]$ est la divergence de Kullback-Leibler entre la politique actuelle et l'ancienne,
- $\boldsymbol{\beta}$ est un coefficient adaptatif qui régule l'importance de la pénalité.

Mise à jour adaptative de $\boldsymbol{\beta}$ après chaque mise à jour si la divergence KL observée est trop faible par rapport à une cible δ_{tar} , on réduit β (le modèle n'explore pas assez). Si elle est trop élevée, on augmente β (le modèle fait des sauts de politique trop grands).

- **PPO avec objectif tronqué (Clipped Objective)**

Afin de contourner le calcul coûteux de la divergence KL et sa gestion, une version simplifiée est introduite avec l'objectif tronqué :

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min\left(r_t(\theta)\hat{A}_t, g\left(\varepsilon, \hat{A}_t\right)\right)\right] \quad (2.33)$$

Où g est défini comme :

$$g(\varepsilon, A) = \begin{cases} (1+\varepsilon)A & \text{si } A \geq 0 \\ (1-\varepsilon)A & \text{si } A < 0 \end{cases} \quad (2.34)$$

L'introduction de l'hyperparamètre permet d'atténuer l'attrait des politiques qui s'éloigneraient trop de la politique initiale, ce qui va dans le sens de mises à jour de proximité, typiquement $\varepsilon = 0.1\ ou\ 0.2$.

Principe du clipping si $r_t(\theta)$ reste dans l'intervalle $[1-\varepsilon, 1+\varepsilon]$, on garde $r_t(\theta)A_t$, Sinon, on tronque la mise à jour, limitant ainsi les changements trop drastiques dans la politique. Ce mécanisme impose une borne supérieure à la fonction objective, réduisant les mises à jour risquées de la politique.

2.6.5 Algorithmes On-Policy et Off-Policy

Une distinction fondamentale entre les algorithmes d'apprentissage par renforcement profond réside dans leur nature *on-policy* ou *off-policy*. Cette distinction influence la manière dont les itérations d'apprentissage exploitent les données.

Un algorithme est dit *on-policy* s'il apprend à partir de la politique en cours, c'est-à-dire que l'entraînement ne peut utiliser que les données générées par la politique actuelle. Cela implique que, durant le processus d'apprentissage, chaque itération ne se base que sur la politique en vigueur à ce moment-là

pour générer les données d'entraînement. Par conséquent, toutes les données doivent être abandonnées après chaque phase d'entraînement, car elles deviennent obsolètes. Cette contrainte rend les méthodes *on-policy* peu efficaces en termes d'échantillonnage, puisqu'elles nécessitent davantage de données d'apprentissage. Les méthodes *on-policy* comprennent notamment REINFORCE, SARSA, ainsi que les méthodes hybrides telles que Actor-Critic et PPO (*Proximal Policy Optimization*).

En revanche, un algorithme est dit *off-policy* s'il ne présente pas cette contrainte. Il peut réutiliser les données collectées, indépendamment de la politique qui les a générées. Par conséquent, les méthodes *off-policy* sont généralement plus efficaces du point de vue de l'échantillonnage, bien qu'elles puissent exiger une mémoire plus importante pour le stockage des données. Les méthodes *off-policy* incluent DQN (*Deep Q-Network*) et ses extensions.

2.7 Réseaux de Neurones Artificiels

Les réseaux de neurones artificiels sont des réseaux fortement connectés de processeurs élémentaires fonctionnant en parallèle. Chaque processeur élémentaire calcule une sortie unique à partir des informations qu'il reçoit. Toute structure hiérarchique de réseaux est, par définition, un réseau [45]. Les réseaux de neurones sont une composante de chaque algorithme.

Un réseau de neurones est considéré comme étant hautement structuré et se compose toujours de couches.
La première couche est appelée la couche d'entrée (*input layer*), les couches intermédiaires sont appelées les couches cachées (*hidden layers*), et la dernière couche est appelée la couche de sortie (*output layer*), comme défini dans la Figure 2.6.

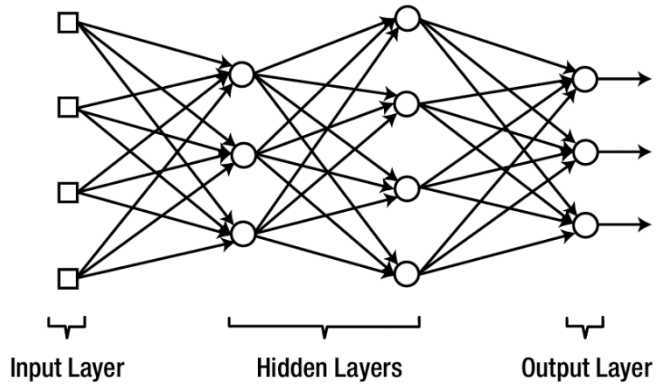

Figure 2.6 – Structure d'un réseau de neurones [46]

La figure 2.7 montre les composants d'un neurone artificiel. Chaque neurone artificiel est un processeur élémentaire. Il reçoit un nombre variable d'entrées en provenance de neurones amont. A chacune de ces entrées est associé un poids w abréviation de weight (poids en anglais) représentatif de la force de la connexion. Chaque processeur élémentaire est doté d'une sortie unique, qui se ramifie ensuite pour alimenter un nombre variable de neurones avals. A chaque connexion est associé un poids.

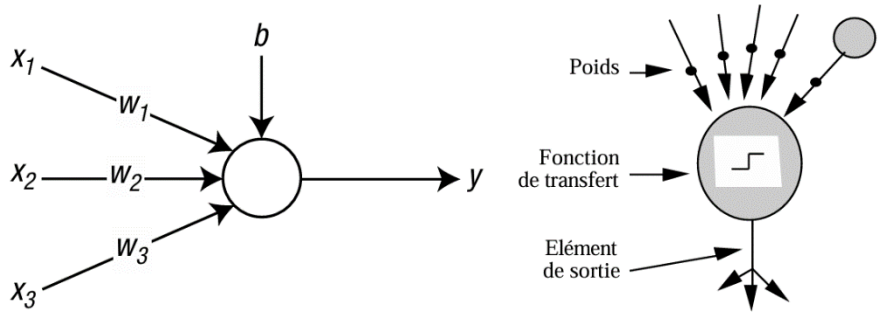

Figure 2.7 – Composants d'un neurone artificiel [46]

2.7.1 Types de Réseaux de Neurones

Les réseaux de neurones peuvent être classés en différentes familles. Chaque famille possède des caractéristiques particulières et se prête à des tâches spécifiques, en fonction du type de données d'entrée à traiter. L'on distingue principalement trois grandes catégories : les perceptrons multicouches (MLP), les réseaux de neurones convolutifs (CNN) et les réseaux de neurones récurrents (RNN). Ces types peuvent également être combinés pour créer des réseaux hybrides, tels que les CNN-RNN.

Chaque famille est caractérisée par les types de couches utilisées et l'organisation du flux de calcul au sein du réseau. Ces structures reflètent également des connaissances a priori différentes sur les données d'entrée, permettant d'exploiter au mieux leurs propriétés pour l'apprentissage. Passons maintenant à une description détaillée des trois types de réseaux, en commençant par le réseau MLP.

2.7.1.1 Perceptrons Multicouches (MLPs)

Les perceptrons multicouches (MLPs) sont le type de réseau de neurones le plus simple et le plus général.
Ils ne sont composés que de couches entièrement connectées (également appelées *couches denses*). Dans une couche entièrement connectée, chaque sortie de la couche précédente est connectée à chaque neurone de la couche actuelle par un poids distinct. Des fonctions d'activation non linéaires sont généralement appliquées aux sorties de chaque couche dense. Ces fonctions d'activation non linéaires donnent aux réseaux de neurones leur expressivité, ce qui leur permet d'approximer des fonctions non linéaires très complexes.

Les MLPs sont à usage général, et les entrées de ces réseaux sont organisées sous la forme d'un vecteur unique comportant n éléments. Les MLPs font très peu d'hypothèses sur la nature de leurs entrées. Par exemple, ils n'intègrent aucune information sur la manière dont les différentes dimensions de l'entrée sont liées entre elles. Cela représente à la fois une force et une limitation. Cela permet aux MLPs d'avoir une vision globale de leurs entrées. Ils peuvent répondre à une structure ou à des motifs globaux en apprenant des caractéristiques issues de la combinaison de tous les éléments d'entrée.

Lorsqu'un grand nombre de paramètres ajustables est combiné à l'inefficacité d'échantillonnage des algorithmes actuels d'apprentissage par renforcement profond (deep RL), un agent peut mettre beaucoup de temps à s'entraîner. Par conséquent, les MLPs sont généralement bien adaptés aux environnements présentant deux caractéristiques : un espace d'états de faible dimension, et des caractéristiques apprises nécessitant tous les éléments de l'état.

Enfin, les MLPs sont sans mémoire. Cela signifie qu'ils ne gardent aucune trace des entrées précédentes. Les entrées des MLPs ne sont pas ordonnées, et chaque entrée est traitée indépendamment des autres.

2.7.1.2 Réseaux de Neurones Convolutifs (CNNs)

Les réseaux de neurones convolutifs (CNNs) excellent dans l'apprentissage à partir d'images. Les CNNs sont spécifiquement conçus pour exploiter la structure spatiale des données d'image, car ils contiennent une ou plusieurs couches convolutives, chacune composée d'un certain nombre de noyaux de convolution (*convolution kernels*).
Les noyaux de convolution sont appliqués de manière répétée à des sous-ensembles de leur entrée pour produire une sortie. On dit alors qu'un noyau est *convolué* avec son entrée ; il s'agit de l'opération de convolution.
Par exemple, un noyau peut être convolué avec une image 2D pour produire une sortie 2D. Dans ce cas, une application unique du noyau correspond à son application sur une petite portion de l'image composée de quelques pixels seulement, par exemple une zone de 3×3 5×5 pixels ; cette opération produit une sortie scalaire. Cette sortie scalaire peut être interprétée comme un indicateur de la présence ou de l'absence d'une caractéristique particulière dans la région locale de l'entrée où le noyau a été appliqué. Par conséquent, un noyau peut être considéré comme un détecteur de caractéristiques locales, car il produit une sortie basée sur un sous-ensemble spatialement contigu des caractéristiques d'entrée.

Le noyau (ou détecteur de caractéristiques) est appliqué localement sur l'ensemble de l'image afin de produire une *carte de caractéristiques* (*feature map*) qui décrit tous les endroits où une caractéristique particulière est présente dans l'image.

Une seule couche convolutive est généralement composée de plusieurs noyaux (entre 8 et 256, en général), chacun pouvant apprendre à détecter une caractéristique différente. Grâce aux convolutions, le réseau apprend « gratuitement » qu'une certaine configuration de valeurs représente la même caractéristique, peu importe où elle se trouve dans l'image. Les couches convolutives sont aussi plus efficaces que les couches entièrement connectées en termes de nombre de paramètres. Un seul noyau peut être utilisé pour identifier une même caractéristique n'importe où dans l'image ; il n'est pas nécessaire d'apprendre un noyau différent pour chaque position possible de cette caractéristique.

Un inconvénient des couches convolutives est leur nature locale : elles ne traitent qu'un sous-ensemble de l'espace d'entrée à la fois et ignorent la structure globale de l'image. Or, la structure globale est souvent importante.
Comme les MLPs, les CNNs sont également sans mémoire (*stateless*). Cependant, contrairement aux MLPs, les CNNs sont parfaitement adaptés à l'apprentissage à partir d'images, car ils supposent que les entrées possèdent une structure spatiale. En réalité, les CNNs sont tellement plus performants pour apprendre à partir d'images que les autres types de réseaux qu'une règle empirique s'applique souvent : si l'état fourni par l'environnement est une image, il faut inclure des couches convolutives dans le réseau.

2.7.1.3 Réseaux de Neurones Récurrents (RNNs)

Les réseaux de neurones récurrents (RNNs) sont spécialisés dans l'apprentissage à partir de données séquentielles. Un seul point de données pour un RNN consiste en une séquence d'éléments, où chaque élément est un vecteur. Contrairement aux MLPs et aux CNNs, les RNNs partent du principe que l'ordre dans lequel les éléments sont reçus a une signification.

Les phrases sont un exemple typique de données que les RNNs traitent efficacement. Chaque élément de la séquence est un mot, et l'ordre des mots influence le sens global de la phrase. Cependant, un point de données peut aussi être une séquence d'états par exemple, une séquence ordonnée d'états vécus par un agent. La caractéristique principale des RNNs est qu'ils sont à mémoire (*stateful*), ce qui signifie qu'ils se souviennent des éléments précédemment rencontrés. Ce mécanisme de mémoire est utile lorsque les informations fournies par l'environnement à l'instant t ne suffisent pas, à elles seules, pour prendre une bonne décision à ce moment-là. C'est le principal avantage des RNNs par rapport aux CNNs ou aux MLPs.

En résumé, les RNNs sont particulièrement adaptés aux problèmes où les données sont représentées sous forme de séquences ordonnées. Dans le contexte de l'apprentissage par renforcement profond (*deep RL*), ils sont généralement utilisés lorsqu'un agent doit se souvenir d'événements passés sur un long horizon temporel afin de prendre de bonnes décisions.

Heureusement, il n'est pas nécessaire de choisir un seul type de réseau de manière exclusive. Il est en effet très courant de créer des architectures hybrides, qui contiennent plusieurs sous-réseaux, chacun pouvant appartenir à une famille différente.

2.7.2 Différences entre MLP, CNN et RNN

Un perceptron multicouche (MLP) est un réseau entièrement connecté (*fully connected*). Dans la littérature, on le trouve souvent sous les termes de *réseau profond à propagation directe* (*deep feed-forward network*) ou de *réseau de neurones à propagation avant* (*feed-forward neural network*). Comprendre ce réseau en lien avec ses applications cibles permet de mieux saisir les raisons sous-jacentes à la conception de modèles d'apprentissage profond plus avancés.

Les MLPs sont couramment utilisés dans des problèmes simples de régression logistique ou linéaire. Cependant, les MLPs ne sont pas optimaux pour traiter des données séquentielles ou multidimensionnelles. Par conception, un MLP a des difficultés à mémoriser des motifs dans des données séquentielles, et nécessite un grand nombre de paramètres pour traiter des données multidimensionnelles.

Pour l'entrée de données séquentielles, les réseaux de neurones récurrents (RNNs) sont populaires car leur architecture interne permet d'identifier des dépendances dans l'historique des données, ce qui est utile pour la prédiction. Pour des données multidimensionnelles comme les images ou les vidéos, les réseaux de neurones convolutifs (CNNs) excellent dans l'extraction de cartes de caractéristiques (*feature maps*) à des fins de classification, de segmentation, de génération ou d'autres tâches en aval. Dans certains cas, un CNN sous forme de convolution 1D peut également être utilisé pour traiter des entrées séquentielles. Toutefois, dans la majorité des modèles d'apprentissage profond, on combine généralement un MLP avec un CNN ou un RNN afin de tirer parti des forces de chaque type de réseau.

Les MLP, CNN et RNN ne suffisent pas à eux seuls à constituer une architecture de réseau profond complète. Il est nécessaire d'identifier une fonction objective (Loss function), un optimiseur et un régulariseur.
L'objectif est de réduire la valeur de la fonction de perte durant l'entraînement, car cette réduction est un bon indicateur que le modèle apprend efficacement.

Pour minimiser cette valeur, le modèle utilise un optimiseur, un algorithme qui détermine comment ajuster les poids et les biais à chaque étape d'entraînement. Un modèle entraîné doit non seulement fonctionner sur les données d'entraînement, mais aussi généraliser à des données jamais vues. Le rôle du régulariseur est de garantir cette capacité de généralisation à de nouvelles données.

2.7.2.1 Fonctions d'Activation

En apprentissage automatique, les fonctions d'activation sont des fonctions mathématiques utilisées dans les neurones d'un réseau de neurones pour introduire de la non-linéarité dans le modèle. Sans fonction d'activation, un réseau de neurones serait simplement une combinaison linéaire de ses entrées, incapable de modéliser des relations complexes entre les données. Grâce à la non-linéarité, le réseau peut apprendre des représentations plus riches et puissantes, comme des motifs, des seuils ou des dépendances conditionnelles.

Parmi les fonctions les plus courantes, la **ReLU** (ou *relu*) est une fonction non linéaire simple. Elle agit comme un filtre : elle laisse passer les valeurs positives sans modification et ramène toutes les autres

(négatives ou nulles) à zéro. Il existe d'autres fonctions non linéaires pouvant être utilisées, telles qu'elu, selu, Softplus, Sigmoïde (*sigmoid*) et tanh. Cependant, ReLU est la fonction la plus couramment utilisée en raison de sa simplicité et de son efficacité computationnelle. Les fonctions sigmoïdes et tanh sont souvent utilisées comme fonctions d'activation dans la couche de sortie.

Le tableau 2.1 présente les fonctions d'activation ainsi que les équations associées à chacune d'elles :

`relu`	relu(x) = max(0, x)
`softplus`	softplus(x) = log(1 + e^x)
`elu`	$elu(x,a) = \begin{cases} x & if\ x \geq 0 \\ a(e^x - 1) & otherwise \end{cases}$ where $a \geq 0$ and is a tunable hyperparameter
`selu`	$selu(x) = k \times elu(x, a)$ where k = 1.0507009873554804934193349852946 and a = 1.6732632423543772848170429916717
`sigmoid`	$sigmoid(x) = \frac{1}{1 + e^{-x}}$
`tanh`	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Tableau 2.1 – Définition des fonctions d'activation non linéaires courante

2.7.2.2 Régularisation

Un réseau de neurones a tendance à mémoriser ses données d'entraînement, surtout lorsqu'il dispose d'une capacité excessive. Dans ce cas, le réseau échoue de manière catastrophique lorsqu'il est confronté aux données de test. C'est un exemple classique d'échec de généralisation.

Pour éviter cette tendance, on introduit une fonction ou une couche de régularisation dans le modèle. Une couche de régularisation courante est la Dropout.

Le principe du *dropout* est simple : étant donné un taux de dropout (par exemple, *dropout = 0,45*), la couche Dropout supprime aléatoirement cette fraction d'unités, les empêchant ainsi de participer à la couche suivante.
Par exemple, si la première couche contient 256 unités, après l'application d'un *dropout* de 0,45, seules *(1 - 0,45) × 256 = 140* unités participeront à la couche suivante.

La couche Dropout rend les réseaux de neurones plus robustes face à des données d'entrée imprévues, car le réseau est entraîné à faire des prédictions correctes même lorsque certaines unités sont absentes. Il est important de noter que le *dropout* n'est pas utilisé dans la couche de sortie et qu'il est actif uniquement pendant l'entraînement. Il n'est pas présent pendant l'inférence ou les prédictions.

D'autres techniques de régularisation existent, comme L1 et L2. Il est possible de régulariser les biais, les poids et les sorties d'activation à chaque couche. Les régularisations L1 et L2 encouragent l'utilisation de valeurs de paramètres plus petites en ajoutant une fonction de pénalité. La régularisation L1 applique une pénalité basée sur la somme des valeurs absolues des paramètres, tandis que la régularisation L2 utilise la somme des carrés de ces valeurs. Autrement dit, la fonction de pénalité pousse l'optimiseur à trouver des valeurs de paramètres plus faibles, ce qui peut améliorer la capacité du réseau à généraliser.

2.7.2.3 Fonctions de Perte & d'activation en sortie

Un type de fonction de perte est l'erreur quadratique moyenne (*mean_squared_error* ou MSE), qui correspond à la moyenne des carrés des écarts entre la cible (ou étiquette) et la prédiction. La fonction categorical_crossentropy.
Elle est définie comme le négatif de la somme des produits entre la cible (ou étiquette) et le logarithme de la prédiction, par catégorie. D'autres fonctions de perte sont disponibles dans Keras, telles que : mean_absolute_error (*erreur absolue moyenne*) et binary_crossentropy (*entropie croisée binaire*).

Le tableau 2.2 résume les fonctions de perte les plus courantes.

Loss Function	Equation
`mean_squared_error`	$\frac{1}{categories}\sum_{i=1}^{categories}\left(y_i^{label}-y_i^{prediction}\right)^2$
`mean_absolute_error`	$\frac{1}{categories}\sum_{i=1}^{categories}\left\|y_i^{label}-y_i^{prediction}\right\|$
`categorical_crossentropy`	$-\sum_{i=1}^{categories} y_i^{label}\log y_i^{prediction}$
`binary_crossentropy`	$-y_1^{label}\log y_1^{prediction}-\left(1-y_1^{label}\right)\log\left(1-y_1^{prediction}\right)$

Tableau 2.2 – Définition des fonctions de pertes courante

L'idée de la fonction softmax est étonnamment simple ; elle réduit les sorties à des probabilités en normalisant les scores (ou logits) en une distribution de probabilité dont la somme vaut 1.

$$softmax(x_i)=\frac{e^{x_i}}{\sum_{j=0}^{N-1}e^{x_j}} \qquad (2.\ 30)$$

Il existe d'autres choix de fonction d'activation pour la couche de sortie, notamment : fonction identité, qui copie simplement son entrée vers la sortie. La fonction sigmoid également appelée sigmoïde logistique, chaque élément de la sortie est indépendamment mappé entre 0 et 1. Contrairement à *softmax,* la somme des éléments n'est pas contrainte à 1,0.

$$sigmoid(x)=\sigma(x)=\frac{1}{1+e^{-x}} \qquad (2.31)$$

La fonction tanh transforme les valeurs d'entrée dans l'intervalle [-1,0 ; 1,0]. Elle est utilisée lorsque la sortie peut prendre des valeurs positives et négatives. Bien qu'elle soit plus couramment employée dans les couches internes des réseaux neuronaux récurrents, elle peut également être utilisée comme fonction d'activation en sortie.

2.7.2.4 Optimisation et métriques de performance

L'objectif de l'optimisation est de minimiser la fonction de perte. L'idée est que si la perte est réduite à un niveau acceptable, le modèle a indirectement appris la fonction qui associe les entrées aux sorties.

Les métriques de performance servent à évaluer si le modèle a bien appris la distribution sous-jacente des données. Dans Keras, la métrique par défaut est la perte (*loss*). Cependant, lors de l'entraînement, de la validation et du test, d'autres métriques comme la précision (*accuracy*) peuvent être ajoutées. La précision correspond au pourcentage (ou à la fraction) de prédictions correctes par rapport aux données réelles (*ground truth*).

En apprentissage profond, il existe de nombreuses autres métriques, mais leur choix dépend fortement de l'application cible du modèle. Dans les publications scientifiques, on rapporte généralement les métriques de performance sur l'ensemble de test pour comparer différents modèles d'apprentissage profond.

Keras propose plusieurs optimiseurs, dont les plus utilisés sont : SGD (*Stochastic Gradient Descent*), Adam (*Adaptive Moment Estimation*), RMSprop (*Root Mean Squared Propagation*). Chaque optimiseur peut être réglé via des paramètres ajustables tels que : le taux d'apprentissage (*learning rate*), le momentum, ou la décroissance (*decay*).

Adam et RMSprop sont des variantes de SGD avec des taux d'apprentissage adaptatifs. Dans le réseau de classification, Adam est utilisé car il offre la meilleure précision sur l'ensemble de test. SGD est considéré comme l'optimiseur le plus fondamental. C'est une version simplifiée de la descente de gradient en calcul différentiel. L'idée de la descente de gradient est d'explorer la courbe d'une fonction vers le bas, comme si l'on descendait une vallée jusqu'à atteindre le fond (le minimum).

2.8 Conclusion

Dans ce chapitre, nous avons exploré en profondeur les fondements théoriques et les principales approches du Deep Reinforcement Learning (DRL), une discipline qui allie l'apprentissage par renforcement classique à la puissance des réseaux de neurones profonds. Cette combinaison permet aux agents d'apprendre efficacement dans des environnements complexes, où l'espace d'états est vaste et les dynamiques sont souvent non linéaires.

Nous avons examiné les composantes essentielles du DRL, telles que la formulation en processus de décision markovien (MDP), la fonction de récompense, la politique d'action, et le dilemme exploration/exploitation. Une attention particulière a été portée à différents types d'algorithmes, notamment ceux basés sur la valeur (DQN, Double DQN), sur la politique (REINFORCE), ainsi que les méthodes hybrides de type Actor-Critic (A2C, PPO).

Ces algorithmes se distinguent par trois caractéristiques fondamentales : leur nature on-policy ou off-policy, le type d'espace d'actions qu'ils peuvent gérer (discret ou continu), et les fonctions qu'ils apprennent (fonction de valeur, politique, ou les deux). Ainsi, REINFORCE, A2C et PPO tous on-policy sont applicables aux environnements à actions discrètes ou continues, tandis que DQN et Double DQN off-policy sont limités aux actions discrètes mais offrent une grande efficacité d'échantillonnage. PPO et Double DQN émergent comme les plus performants et stables de leur catégorie respective.

Enfin, les forces et limites des différentes approches soulignent les défis persistants dans le domaine, notamment la sensibilité aux hyperparamètres, la nécessité de vastes volumes de données, et la stabilité de l'apprentissage. Ces constats orientent les recherches actuelles vers des algorithmes plus robustes, efficaces et généralisables.

Ces éléments théoriques posent les bases nécessaires à la mise en œuvre pratique des algorithmes de DRL. Le chapitre suivant sera ainsi consacré à leur application dans le cadre des systèmes de détection d'intrusions et à l'analyse des résultats obtenus.

Chapitre 3

Implémentation et discussion des résultats

3.1 Introduction

La troisième partie de ce mémoire présente de manière détaillée la méthodologie adoptée pour concevoir, implémenter et évaluer une solution de détection d'intrusions basée sur l'apprentissage par renforcement profond (Deep Reinforcement Learning, DRL). L'objectif principal de cette étude est de renforcer la capacité des systèmes de détection d'intrusion (IDS) à identifier avec précision les comportements malveillants, tout en réduisant le taux de fausses alertes et en assurant une robustesse face à des scénarios d'attaque variés.

Notre approche repose sur l'utilisation conjointe de trois ensembles de données de référence dans le domaine de la cybersécurité : CIC-IDS2017, NSL-KDD et AWID. Nous commencerons par décrire ces datasets, en mettant en évidence leurs caractéristiques spécifiques ainsi que les étapes de prétraitement mises en œuvre pour garantir la qualité et la pertinence des données vis-à-vis de notre problématique.

Nous poursuivrons par la présentation d'algorithmes de DRL sélectionné, le Deep Q-Network (DQN), ainsi que des bibliothèques logicielles mobilisées pour leur implémentation. Le fonctionnement général de ce modèle, leurs paramètres essentiels et leur adaptation au contexte spécifique de la détection d'intrusions seront également abordés.

Enfin, nous détaillerons les métriques d'évaluation choisies pour mesurer la performance de notre approche. Ces indicateurs, à la fois classiques (précision, rappel, F1-score) et spécifiques à l'apprentissage par renforcement (récompense cumulée, stabilité de la politique), permettront d'apprécier la capacité de notre solution à détecter efficacement les intrusions tout en minimisant les fausses alertes.

Ce chapitre joue un rôle fondamental dans la structuration de notre démarche, en exposant les fondements techniques et expérimentaux nécessaires à la validation de notre solution en environnement simulé.

3.2 Datasets de détection d'intrusions

Dans le cadre de ce travail, nous avons retenu trois ensembles de données qui répondent à une série de critères essentiels : (1) des ensembles de données étiquetés, nécessaires à l'entraînement supervisé de modèle, (2) déséquilibrés, mais avec des niveaux d'inégalité variés, ce qui permet d'analyser le comportement d'algorithme dans différentes conditions, (3) une séparation prédéfinie entre les données d'apprentissage et de test, facilitant ainsi la comparaison des résultats avec ceux d'autres travaux, (4) des ensembles de données largement utilisés dans la littérature, ce qui offre un référentiel solide pour la validation expérimentale, (5) l'inclusion à la fois de datasets plus anciens et plus récents**,** dans une optique de généralisation et de diversité et enfin, (6) des données issues de différentes architectures réseau (par exemple, réseaux filaires pour NSL-KDD et CIC-IDS2017, et réseau sans fil pour AWID). Au regard de ces exigences, notre choix s'est porté sur les datasets NSL-KDD**,** AWID et CIC-IDS2017, qui répondent de manière satisfaisante à l'ensemble des critères susmentionnés.

3.2.1 Dataset NSL-KDD

La base de données KDD Cup 99 est l'un des ensembles de données les plus emblématiques utilisés pour l'évaluation des systèmes de détection d'intrusion (IDS). Cette base de données a été préparée par Stolfo et al. [47] en 1999, à partir des données collectées dans le cadre du programme d'évaluation DARPA'98, mené par la Defense Advanced Research Projects Agency (DARPA) et l'Air Force Research Laboratory (AFRL), et traitées par le MIT Lincoln Laboratory. Le corpus initial comprenait environ 4 Go de données brutes (tcpdump) couvrant sept semaines de trafic réseau simulé. Ces données ont été transformées en environ cinq millions d'enregistrements de connexion, chacun représenté par un vecteur de 41 attributs (features), et étiqueté comme normal ou correspondant à un type d'attaque spécifique parmi quatre catégories : Denial of Service (DoS), Remote to Local (R2L), User to Root (U2R) et Probing (voir tableau 3.1).

F#	Nom de Feature	F#	Nom de Feature	F#	Nom de Feature
F1	Duration	F15	Su attempted	F29	Same srv rate
F2	Protocol_type	F16	Num root	F30	Diff srv rate
F3	Service	F17	Num file creation	F31	Srv diff host rate
F4	Flag	F18	Num shells	F32	Dst host count
F5	Src bytes	F19	Num access files	F33	Dst host srv count
F6	Dst bytes	F20	Num outbound cmds	F34	Dst host same srv rate
F7	Land	F21	Is host login	F35	Dst host diff srv rate
F8	Wrong fragment	F22	Is guest login	F36	Dst host same srv port rate
F9	Urgent	F23	Count	F37	Dst host srv diff host rate
F10	Hot	F24	Srv count	F38	Dst host serror rate
F11	Num_failed_logins	F25	Serror rate	F39	Dst host srv serror rate
F12	Logged_in	F26	Srv serror rate	F40	Dst host rerror rate
F13	Num compromised	F27	Rerror rate	F41	Dst host srv rerror rate
F14	Root shell	F28	Srv rerror rate	F42	Class label

Tableau 3.1-Liste des caractéristiques de l'ensemble de données NSL-KDD

Cependant, malgré son usage massif, le dataset KDD Cup 99 présente plusieurs limitations majeures : une redondance importante des enregistrements (particulièrement dans l'ensemble d'apprentissage), un déséquilibre marqué entre les classes, et une faible représentativité des attaques rares, ce qui fausse les performances de nombreux algorithmes d'apprentissage.

Pour remédier à ces problèmes, le dataset NSL-KDD a été proposé en 2010 comme une version améliorée et plus équilibrée de KDD Cup 99. Il conserve la même structure de base (41 attributs et une étiquette de classe), mais élimine les doublons, rééquilibre les classes, et sélectionne les échantillons en fonction de leur niveau de difficulté, permettant une évaluation plus fine et plus représentative des performances des modèles. À ce jour, NSL-KDD demeure une référence incontournable dans les travaux de recherche sur la détection d'intrusion. Il contient cinq classes : une classe normale et quatre classes représentant les mêmes types d'attaques que KDD Cup 99 (voir tableau 3.2).

Categories	Notation	Definitions	# of Samples
Normal	N	Activités normales identifiées à partir des caractéristiques.	148517
DoS	D	L'attaquant tente d'empêcher les utilisateurs d'accéder à un service.	53385
Probe	P	L'attaquant tente de scanner le réseau cible afin de collecter des informations telles que des vulnérabilités.	14077
U2R	U	Des attaquants ayant un accès local à la machine de la victime tentent d'obtenir des privilèges utilisateur.	119
R2L	R	Un attaquant sans compte local essaie d'envoyer des paquets à la machine cible pour en obtenir l'accès.	3882

Tableau 3.2 - Classes des enregistrements de données du jeu de données NSL-KDD

Dans notre démarche expérimentale, nous avons appliqué un prétraitement rigoureux au dataset NSL-KDD pour le rendre exploitable par des modèles d'apprentissage profond. La base de données contient deux types de caractéristiques : 38 attributs numériques (valeurs continues ou discrètes), et 3 attributs catégoriels : protocol_type (tcp, udp, icmp), service (70 services distincts), et flag (11 états de connexion). Ces attributs catégoriels ne peuvent pas être utilisés directement par un modèle de machine learning, qui exige des données numériques. C'est pourquoi nous avons utilisé la méthode d'encodage one-hot, qui transforme chaque valeur catégorielle en un vecteur binaire. Par exemple, la valeur udp de l'attribut protocol_type est convertie en [0, 1, 0]. Chaque catégorie est ainsi représentée par une position unique dans un vecteur, évitant toute interprétation ordinale erronée. Grâce à cet encodage, le nombre total de caractéristiques passe de 41 à 122, ce qui permet une représentation complète et précise des données. Par la suite, nous avons appliqué une normalisation Min-Max, qui consiste à ramener toutes les valeurs dans l'intervalle [0, 1], afin de garantir une échelle uniforme et d'accélérer la convergence des modèles d'apprentissage.

Enfin, le dataset NSL-KDD est divisé en deux sous-ensembles prédéfinis : KDDTrain+ (125 973 échantillons, dont 67 343 normaux et 58 630 malveillants) et KDDTest+ (22 544 échantillons, dont 9 711

normaux et 12 833 malveillants). Cette répartition montre que les données sont relativement équilibrées entre le trafic normal et anormal (voir figure 3.1), ce qui constitue un avantage important pour la stabilité et la fiabilité de l'évaluation des performances des modèles d'intrusion.

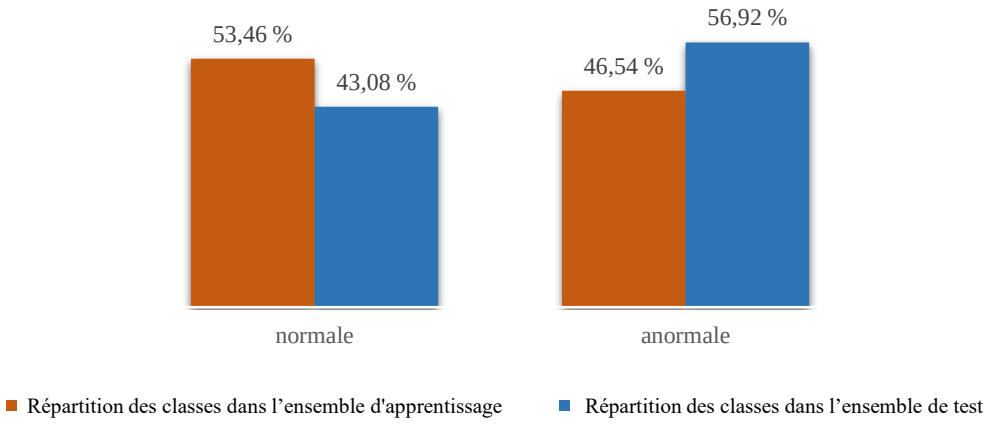

Figure 3.1 – Distribution des intrusions dans les ensembles d'apprentissage et de test (NSL-KDD)

3.2.2 Dataset CIC-IDS-2017

Dans le domaine de la détection d'intrusions, de nombreuses données historiques ont été largement exploitées dans la littérature. Toutefois, certaines d'entre elles présentent des limites majeures : taille réduite, absence de représentativité du trafic réel, ou encore absence de diversité dans les types d'attaques. Pour pallier ces insuffisances, le dataset CICIDS2017, conçu par Sharafaldin et al. (2018) au Canadian Institute for Cybersecurity de l'Université du Nouveau-Brunswick, s'est imposé comme une référence moderne. Il vise à fournir un ensemble complet, étiqueté et représentatif du trafic réseau contemporain, intégrant aussi bien des activités bénignes que des attaques récentes telles que : Brute Force (FTP, SSH), DoS, DDoS, Heartbleed, Web Attacks, Infiltration ou encore Botnet.

Ce dataset couvre une période de cinq jours (du 3 au 7 juillet 2017) et inclut huit captures distinctes (voir tableau 3.3). Chaque capture représente des scénarios de trafic réseau simulé, réalisés selon un profil comportemental de 25 utilisateurs. Les flux réseau ont été extraits via CICFlowMeter, produisant des enregistrements contenant 83 caractéristiques issues du trafic (horodatage, IPs, ports, protocoles, longueurs, flags, etc.) ainsi qu'une étiquette d'attaque ou de trafic normal (voir Tableau 3.4). Dans sa version brute, le dataset contient 3 119 345 enregistrements et 15 classes (1 normale + 14 malveillantes). Toutefois, cette version inclut plus de 288 000 entrées sans étiquette ou comportant des valeurs manquantes. Après nettoyage initial, on obtient une version combinée de 2 830 540 flux exploitables.

Dans le cadre de ce travail, nous avons récupéré et regroupé manuellement les fichiers quotidiens. Un prétraitement approfondi a ensuite été appliqué : remplacement des valeurs infinies par des NaN, suppression des lignes avec valeurs manquantes, et élimination des doublons. Le résultat est un ensemble de 2 657 447 enregistrements et 85 caractéristiques. Cette légère différence par rapport aux 83 caractéristiques mentionnées dans la version brute s'explique par la présence de colonnes supplémentaires générées automatiquement ou issues de métadonnées, telles que Flow ID, Timestamp ou des duplicatas comme Fwd Header Length.1, qui ne sont pas incluses dans la structure initiale produite par CICFlowMeter. Les types de données sont les suivants : float64 (45), int64 (35), object (5). Des colonnes non exploitables pour la détection (telles que les IPs, ports, horodatage, ou certaines mesures redondantes) ont ensuite été supprimées, réduisant l'ensemble à 69 caractéristiques pertinentes.

Il est à noter que certains fichiers n'ont pas été inclus dans notre version consolidée, notamment celui correspondant au jeudi matin : "Thursday-WorkingHoursMorning-WebAttacks.pcap_ISCX.csv". Par conséquent, trois types d'attaques Web (Web Attack – Brute Force, Web Attack – Sql Injection, Web Attack – XSS) ont été omis, avec respectivement 1507, 21 et 652 échantillons. Ces attaques ont été exclues involontairement en raison de l'absence du fichier source dans l'agrégation initiale des données. Cependant, leur faible volume aurait de toute façon compromis l'entraînement efficace d'un modèle supervisé sans recours à des techniques avancées de suréchantillonnage.

Les valeurs de la variable cible Label ont été regroupées en 5 classes : Benign, DoS, DDoS, PortScan et Others attack. La classe Others attack regroupe plusieurs types d'attaques peu représentées individuellement, notamment Bot (1966), FTP-Patator (7938), SSH-Patator (5897), Infiltration (36), et Heartbleed (8), ce qui permet de conserver l'ensemble des attaques pertinentes malgré leur faible fréquence. L'encodage des étiquettes a été réalisé via un LabelEncoder pour permettre leur traitement numérique. Les caractéristiques quantitatives ont ensuite été normalisées via la méthode Min-Max, afin de ramener les valeurs entre 0 et 1 et assurer une homogénéité des données en entrée du modèle.

Le dataset obtenu a ensuite été divisé en deux sous-ensembles : un ensemble d'entraînement de 2 125 957 instances et un ensemble de test de 531 490 instances, selon un ratio de 80/20 avec stratification. Cependant, l'analyse de la distribution des classes a mis en évidence un fort déséquilibre, avec près de 2 100 000 échantillons bénins, contre seulement 15 835 pour les attaques de type "Others". Pour corriger ce biais, nous avons appliqué une stratégie combinant sous-échantillonnage aléatoire (RandomUnderSampler) de la classe majoritaire et sur-échantillonnage synthétique (SMOTE) des classes minoritaires. Ce rééquilibrage a permis d'obtenir un ensemble d'entraînement homogène et représentatif des différents types de trafic.

Nom des fichiers	**Jour d'activité**	**Attaques détectées**
Monday-WorkingHours.pcap_ISCX.csv	Lundi	Benign (activités humaines normales)
Tuesday-WorkingHours.pcap_ISCX.csv	Mardi	Benign, FTP-Patator, SSH-Patator
Wednesday-working-Hours.pcap_ISCX.csv	Mercredi	Benign, DoS GoldenEye, DoS Hulk, DoS Slowhttptest, DoS slowloris, Heartbleed
Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv	Jeudi	Benign, Web Attack – Brute Force, Web Attack – Sql Injection, Web Attack – XSS
Thursday-WorkingHours-Afternoon-Infiltration.pcap_ISCX.csv	Jeudi	Benign, Infiltration
Friday-WorkingHours-Morning.pcap_ISCX.csv	Vendredi	Benign, Bot
Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv	Vendredi	Benign, PortScan
Friday-WorkingHours-Afternoon-DDoS.pcap_ISCX.csv	Vendredi	Benign, DDoS

Tableau 3.3 - Description des fichiers contenant le dataset CICIDS2017

Étiquette de classe	Nombre d'instances
BENIGN	2 359 087
DoS Hulk	231 072
PortScan	158 930
DDoS	41 835
DoS GoldenEye	10 293
FTP-Patator	7 938
SSH-Patator	5 897
DoS slowloris	5 796
DoS Slowhttptest	5 499
Bot	1 966
Web Attack – Brute Force	1 507
Web Attack – XSS	652
Infiltration	36
Web Attack – Sql Injection	21
Heartbleed	11

Tableau 3.4 - Répartition des instances selon les classes (CICIDS2017)

3.2.3 Dataset AWID

Aegean Wi-Fi Intrusion Dataset (AWID), publié en 2016 par Kolias et al, est un corpus public destiné à la détection d'intrusions dans les réseaux sans fil, notamment les réseaux IEEE 802.11. Contrairement à d'anciens ensembles de données comme NSL-KDD, AWID se distingue par sa réalisation à partir de traces réelles d'un réseau Wi-Fi protégé par WEP, reflétant fidèlement le trafic normal et malveillant observé dans des contextes domestiques ou professionnels. L'objectif principal du dataset est de faciliter la conception, l'évaluation et la comparaison d'algorithmes de détection d'intrusions dans des réseaux sans fil modernes. Parmi les différents ensembles proposés par AWID, nous avons exploité AWID-CLS-R, un sous-ensemble réduit contenant 1 795 574 enregistrements pour l'apprentissage et 575 642 pour le test, avec une classification en quatre classes : normal, flooding, injection et impersonation. Le dataset est extrêmement déséquilibré, avec environ 91 % de trafic normal, ce qui pose un défi majeur pour les algorithmes d'apprentissage automatique, souvent biaisés par des classes majoritaires (voir Figure 3.2).

Chaque échantillon est décrit à l'origine par 156 attributs, dont une grande partie est superflue ou bruitée. Cette redondance nuit aux performances de détection et au temps d'apprentissage. Nous avons donc appliqué un processus rigoureux de prétraitement des données, débutant par l'assignation manuelle des colonnes du dataset (awid_columns), suivie d'un nettoyage approfondi. Tous les enregistrements contenaient des valeurs manquantes identifiées par le caractère '?'. En les remplaçant par NaN, nous avons observé que l'ensemble des 2 371 218 lignes en contenait. Ensuite, nous avons supprimé les colonnes avec plus de 60 % de valeurs manquantes (67 colonnes) ainsi que celles sans variation (44 colonnes), ne conservant ainsi que 35 attributs pertinents.

Les caractéristiques catégorielles restantes ont été encodées par fréquence (frequency encoding), une technique qui remplace chaque valeur par sa fréquence d'apparition dans la colonne. Cela permet de transformer les variables catégorielles en valeurs numériques continues, tout en préservant une certaine sémantique statistique et sans générer une explosion dimensionnelle comme le one-hot encoding. Les étiquettes cibles (normal, injection, impersonation, flooding) ont quant à elles été encodées à l'aide de LabelEncoder, ce qui a abouti à une représentation en quatre classes numériques : [0, 1, 2, 3], avec une distribution très déséquilibrée. Le dataset a ensuite été séparé en ensembles d'entraînement et de test selon un ratio de 80/20, en conservant la distribution originale des classes. Cette étape nous a permis de reconstituer le découpage proposé initialement dans AWID, que nous avions temporairement fusionné pour faciliter le prétraitement.

Pour améliorer la pertinence des données, nous avons effectué une sélection de caractéristiques basée sur l'information mutuelle (mutual_info_classif) en ne conservant que 24 attributs les plus informatifs. Les attributs retenus incluent notamment frame.len, radiotap.length, wlan.fc.subtype, wlan.qos.priority, parmi d'autres (voir Tableau 3.5), jugés essentiels à la discrimination entre trafic légitime et attaques. Les données sélectionnées ont été normalisées via MinMaxScaler afin de restreindre toutes les valeurs

numériques à l'intervalle [0, 1], ce qui est crucial pour garantir une convergence stable et rapide dans les modèles d'apprentissage profond.

F#	Nom de Feature	F#	Nom de Feature
F1	frame.len	F13	wlan_mgt.rsn.akms.list
F2	radiotap.length	F14	wlan_mgt.rsn.pmkid.data
F3	wlan.fc.subtype	F15	wlan_mgt.extended_capabilities
F4	wlan.fc.ds	F16	wlan_mgt.vendor_specific.type
F5	wlan.duration	F17	wlan_mgt.rsn.pmkid.count
F6	wlan.da	F18	wlan_mgt.fixed.status_code
F7	wlan.sa	F19	wlan_mgt.fixed.timestamp
F8	wlan.qos.priority	F20	wlan_mgt.country_info.environment
F9	wlan.qos.fps	F21	wlan_mgt.country_info.number_channels
F10	wlan_mgt.fixed.beacon	F22	tcp.ack
F11	wlan_mgt.country_info.max_power	F23	tcp.flags
F12	wlan_mgt.erp_info	F24	tcp.hdr_len

Tableau 3.5 - Features optimales issues de la sélection de caractéristiques

Enfin, compte tenu du fort déséquilibre des classes, nous avons appliqué un sous-échantillonnage (undersampling) de la classe majoritaire (normal) via la méthode RandomUnderSampler. Nous avons réduit le nombre d'échantillons normaux à 500 000 tout en conservant les autres classes dans leurs proportions d'origine, ce qui a abouti à un ensemble d'apprentissage plus équilibré : normal (500 000), injection (82 061), impersonation (68 601), flooding (56 581) rebattu ensuite aléatoirement pour éviter les biais de séquence. Ce traitement permet d'améliorer la performance de classification sur les classes minoritaires, souvent négligées par les modèles entraînés sur des données déséquilibrées.

En somme, AWID est un ensemble de données réaliste, complexe et précieux, bien qu'il présente plusieurs limitations : fort déséquilibre des classes, bruit dans les attributs bruts, et hétérogénéité des attaques. Néanmoins, lorsqu'il est correctement nettoyé, encodé, équilibré et réduit, il représente une ressource incontournable pour l'évaluation des systèmes de détection d'intrusion dans les réseaux Wi-Fi.

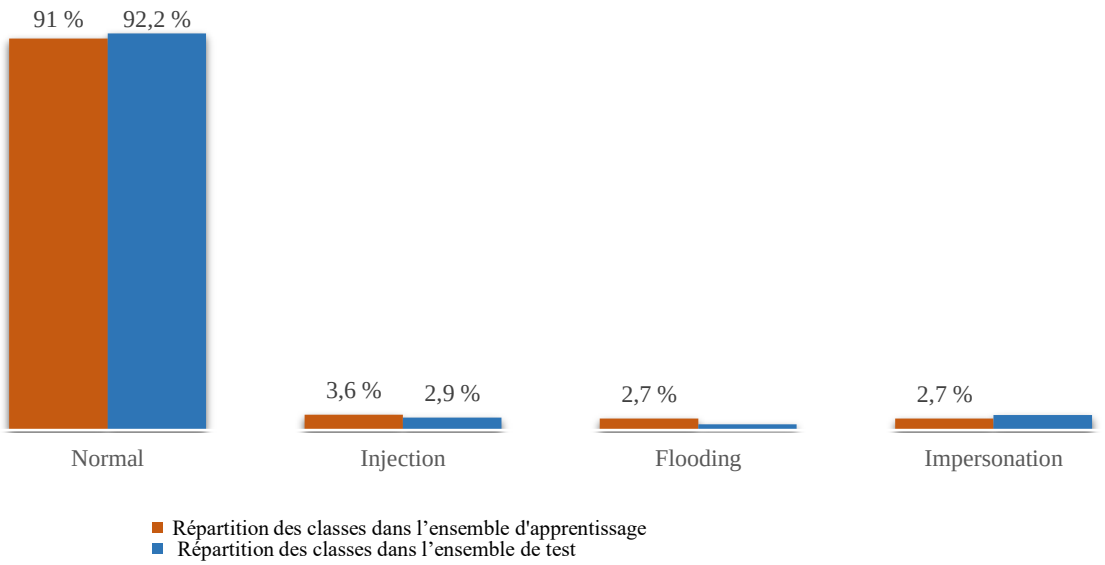

Figure 3.2 – Distribution des intrusions dans les ensembles d'apprentissage et de test (AWID)

3.3 La description du modèle DQN

Dans le cadre de notre système de détection d'intrusions, nous avons développé un modèle d'apprentissage par renforcement profond basé sur l'algorithme Deep Q-Network (DQN), en l'intégrant dans un environnement personnalisé respectant l'interface OpenAI Gym. Cet environnement simule l'interaction entre un agent et des données issues du trafic réseau, représentées sous forme de vecteurs de caractéristiques normalisés. Chaque état correspond ainsi à une observation extraite du dataset, tandis que l'espace d'actions discrètes reflète l'ensemble des classes de trafic possibles (target).

Le rôle de l'agent DQN est d'apprendre une politique optimale permettant de maximiser la récompense cumulative, en s'appuyant sur une fonction d'approximation de la valeur d'action, notée $Q(s, a)$. Un réseau de neurones (NN) est utilisé comme approximation de la fonction Q. Cette fonction Q est implémentée à l'aide d'un réseau de neurones profond entièrement connecté, constitué de trois couches principales : une couche d'entrée prenant en charge un vecteur d'état de dimension fixe, deux couches cachées activées par ReLU, et une couche de sortie linéaire produisant un vecteur de taille égale au nombre d'actions, chaque composante représentant la valeur Q associée à une action donnée. L'apprentissage du réseau est effectué à l'aide d'une fonction de perte de type erreur quadratique moyenne (Mean Squared Error), entre la valeur estimée de Q pour l'état courant ($\hat{q}_t$) et une valeur de référence q_{ref}, calculée en ajoutant la récompense actuelle (r_t) à la valeur de Q pour l'état suivant ($\hat{q}_{t+1}$), pondérée par un facteur d'actualisation (λ).

Une attention particulière a été portée à la définition de la fonction de récompense, cruciale pour guider l'agent vers des décisions pertinentes. Elle est conçue pour encourager fortement la détection correcte des attaques, tout en pénalisant sévèrement les erreurs critiques. Ainsi, une prédiction correcte sur un trafic normal (vrai négatif) donne lieu à une récompense de +1, tandis qu'une détection correcte d'une attaque (vrai positif) est valorisée à +2. À l'inverse, les fausses alertes (faux positifs) entraînent une pénalité de -1, et les faux négatifs – erreurs les plus graves – sont sanctionnées par une pénalité de -2. Cette structure de récompense, orientée vers la classification, favorise un apprentissage équilibré tout en mettant l'accent sur la sécurité. Pour obtenir la valeur prédite pour l'état courant ($\hat{a}_t$), la fonction Q est évaluée pour l'état courant (s_t) en combinaison avec l'ensemble des valeurs possibles du label ($\{a\}$). Cette évaluation est représentée à la Figure 3.3. $Q(s_t, \{a\})$, un vecteur de valeurs indiqué par une flèche épaisse : $Q(s_t, \{a\}) = [Q(s_t, \{a\}_0), Q(s_t, \{a\}_1), \dots, Q(s_t, \{a\}_n)]$, où $\{a\}$ représente l'ensemble des actions possibles et n la cardinalité de cet ensemble. Ensuite, on choisit l'action produisant la valeur de Q maximale issue de cette évaluation : $argmax\ (Q(s_t, \{a\}))$. L'action ainsi sélectionnée est soumise à un algorithme ε-greedy qui choisit cette action avec une probabilité ε ou une action aléatoire avec une probabilité $(1 - \varepsilon)$. Le résultat de cette étape donne l'action prédite ($\hat{a}_t$).

L'action prédite pour l'état suivant ($\hat{a}_{t+1}$) est obtenue de manière similaire. Cette action prédite, combinée avec l'état suivant (s_{t+1}), permet d'estimer la valeur de Q pour cet état ($\hat{q}_{t+1}$), utilisée pour le calcul de $qref$.
La prédiction de la valeur de Q pour l'état suivant peut être simplifiée ainsi : $\hat{q}_{t+1} = \max_a(Q_{s_{t+1}}, a)$.
Une fois l'apprentissage du modèle achevé, le réseau de neurones implémentant la fonction Q est utilisé pour la prédiction. Pour un état donné, la fonction Q fournit une valeur de Q pour chacune des actions possibles. L'action prédite est celle qui maximise cette valeur. Grâce à ce processus d'apprentissage itératif, l'agent parvient à distinguer de manière de plus en plus précise les différents types de trafic, en optimisant ses décisions pour maximiser la récompense cumulative. Il en résulte une politique robuste de classification, capable de s'adapter aux variations dynamiques du trafic réseau ainsi qu'aux comportements d'attaque potentiels.

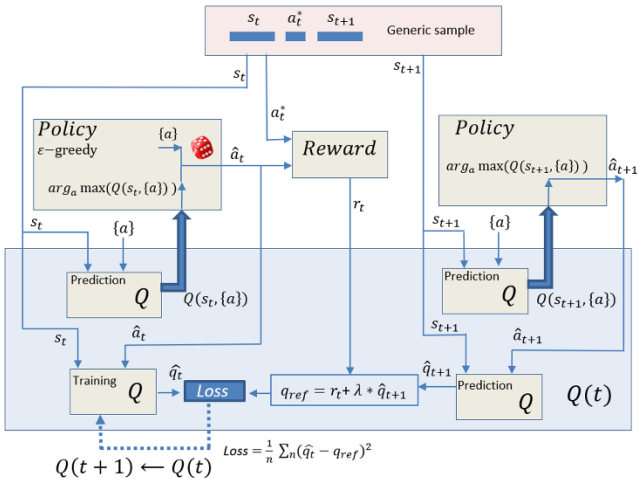

Figure 3.3 - Schéma du modèle DQN pendant l'entraînement [48]

3.4 Évaluation du modèle

3.4.1 Environnement expérimental

L'exécution d'algorithmes d'apprentissage par renforcement profond (Deep Reinforcement Learning, DRL), tels que le Deep Q-Learning (DQL), requiert une infrastructure matérielle puissante et optimisée. En effet, ces méthodes impliquent l'apprentissage de politiques optimales à travers de nombreuses interactions avec l'environnement, la mise à jour fréquente de réseaux de neurones profonds, ainsi que la gestion de grandes quantités de données. Cela rend le processus particulièrement intensif en ressources computationnelles**.**

Contrairement aux approches classiques de classification supervisée, les modèles DRL nécessitent non seulement la rétropropagation sur des réseaux neuronaux profonds, mais aussi la gestion dynamique d'une mémoire d'expérience et de stratégies d'exploration/exploitation. Ces exigences deviennent encore plus critiques lorsqu'on travaille avec des ensembles de données de grande taille tels que CIC-IDS2017 ou AWID. Dans ce contexte, l'utilisation d'un matériel de calcul performant, comme des unités de traitement graphique (GPU) de dernière génération et une quantité de mémoire vive importante, s'avère indispensable pour garantir une formation efficace, réduire les temps d'exécution et permettre la convergence des modèles.

Compte tenu du volume important des données traitées, en particulier lors des phases de prétraitement, nous avons eu recours à Google Colab, une plateforme de développement en ligne gratuite proposée par Google, qui fournit un environnement Python avec un accès à des ressources matérielles dans le cloud, notamment des GPU. Le prétraitement des données a été effectué sur Google Colab à l'aide de différentes bibliothèques Python telles que Pandas, NumPy, Scikit-learn et Matplotlib.

Quant à l'entraînement et à l'évaluation du modèle DQL, ceux-ci ont été réalisés localement sur une station de travail dotée d'un processeur Intel Core i9 de 13e génération, de 32 Go de mémoire RAM, et équipée d'un GPU NVIDIA GeForce RTX 4070 super. L'environnement logiciel comprend Anaconda (version 4.20.0) pour la gestion des environnements virtuels, Jupyter Notebook (version 7.2.2) pour le développement interactif, ainsi que TensorFlow (version 2.13) pour la construction et l'entraînement des réseaux neuronaux. Le système exploite CUDA (version 11.8) et cuDNN (version 8.6) afin d'accélérer les calculs sur GPU. Cette configuration matérielle et logicielle a permis d'assurer une exécution

fluide des algorithmes tout en maintenant une performance d'apprentissage satisfaisante (voir tableau 3.6).

Matérial	Caractéristiques techniques
Processeur	Intel Core i9-13900K, 24 cœurs (8 cœurs Performance et 16 cœurs Efficient)
Mémoire vive (RAM)	32 Go
Cartes graphiques (GPU)	NVIDIA GeForce RTX 4070 super

Tableau 3.6 - Caractéristiques techniques de la station utilisée pour l'implémentation

3.4.2 Métriques de performance

L'évaluation de la performance de notre modèle DQL, appliqué à la détection d'intrusions réseau, repose sur plusieurs métriques bien établies en apprentissage automatique et en cybersécurité. Ces mesures permettent d'analyser la capacité du système à distinguer avec précision le trafic normal du trafic malveillant. Parmi ces indicateurs, nous retenons principalement l'exactitude **(Accuracy)**, la précision **(Precision)**, le rappel **(Recall ou Sensibilité)** et le **score F1**, qui fournissent ensemble une vision complète de l'efficacité du modèle. Bien que l'Accuracy soit largement utilisée pour évaluer les modèles de classification, elle peut s'avérer trompeuse en cas de déséquilibre entre les classes, car elle ne tient pas compte de la nature des erreurs de classification. Pour cette raison, les autres métriques, telles que le Recall et la Precision, permettent de mieux cerner les faiblesses du modèle, notamment en termes de détection des attaques réelles et de limitation des fausses alertes. Le F1-score, en tant que moyenne harmonique du Precision et du Recall, constitue un bon indicateur global de performance lorsque ces deux métriques sont en tension.

Les définitions formelles des métriques utilisées sont présentées ci-dessous, avec leurs équations respectives basées sur les quatre composantes fondamentales : vrais positifs (TP)**,** vrais négatifs (TN)**,** faux positifs (FP) et faux négatifs (FN)**.**

- **Vrai Positif (TP)** : nombre d'intrusions correctement classées comme malveillantes.
- **Vrai Négatif (TN)** : nombre de paquets normaux correctement classés comme bénins.
- **Faux Positif (FP)** : nombre de paquets normaux incorrectement classés comme attaques.
- **Faux Négatif (FN)** : nombre d'intrusions non détectées, classées à tort comme bénignes.

3.4.2.1 Exactitude (Accuracy)

La précision globale mesure la proportion d'échantillons correctement classés par le modèle (qu'ils soient malveillants ou non) par rapport au total des prédictions effectuées.

$$Accuracy = \frac{TP+TN}{TP+FP+TN+TP} \quad (3.1)$$

3.4.2.2 Précision (Precision)

La précision évalue la capacité du modèle à ne classer comme malveillants que les paquets qui le sont effectivement. Elle représente le pourcentage d'intrusions détectées parmi toutes les instances classées comme des attaques.

$$Precision = \frac{TP}{TP+FP} \quad (3.2)$$

3.4.2.3 Rappel (Recall ou Taux de détection)

Le rappel mesure la capacité du modèle à détecter correctement toutes les véritables intrusions présentes dans les données. Il indique le pourcentage d'attaques détectées parmi toutes les attaques réelles.

$$Recall = \frac{TP}{TP+FN} \tag{3.3}$$

3.4.2.4 Score F1

Le score F1 est la moyenne harmonique entre la précision et le rappel. Il fournit une mesure unique de performance, surtout utile en présence de déséquilibre entre les classes.

$$F1\ score = 2 \times \frac{Precision+Recall}{Precision \times Recall} \tag{3.4}$$

Un bon modèle est celui qui atteint à la fois une **précision élevée** (peu de faux positifs), un **rappel élevé** (peu de faux négatifs), et par conséquent, un **score F1 élevé**.

3.5 Les résultats expérimentaux

3.5.1 Les résultats pour CIC-IDS-2017

Dans cette partie, nous avons mis en œuvre un modèle d'apprentissage par renforcement profond (Deep Q-Learning, DQN) pour la détection d'intrusions à partir d'ensemble de données CIC-IDS-2017, qui contient 83 caractéristiques normalisées décrivant le trafic réseau. Ces caractéristiques ont été utilisées comme variables d'état pour l'agent DQN, avec une taille de lot fixée à 500, ce qui signifie que chaque état est constitué de 500 enregistrements réseau. En raison de la complexité de l'espace d'état-action, un réseau neuronal profond (DNN) a été utilisé comme estimateur de fonction au lieu d'une table Q traditionnelle. Ce réseau comprend deux couches cachées entièrement connectées de 256 neurones avec une activation ReLU. L'entraînement a été effectué sur 10 000 épisodes, en utilisant une stratégie epsi-lon-greedy avec un taux d'exploration initial de ε = 0.9 (voir le tableau 3.7).

Paramètres	Description	Valeurs
num-iteration	Nombre d'itérations pour améliorer les valeurs Q dans le DQN	10000
hidden_layers	Nombre de couches cachées : définit les poids et génère les sorties selon la fonction d'activation	2
num_units	Nombre d'unités cachées pour améliorer la qualité de la prédiction et de l'apprentissage	2 × 256
Valeur initiale des poids	Initialisation normale des poids	Défaut
Fonction d'acti-vation	Fonction d'activation non linéaire	ReLU
Gamma γ	Facteur d'actualisation pour la prédiction des cibles	0.001
Taille de lot (bs)	Taille des lots d'enregistrements d'ensemble de données NSL-KDD utilisés pour l'apprentissage	500
Epsilon ε	Degré d'aléa pour l'exécution des actions	0.9

Tableau 3.7 - Paramètres de l'agent DQL et du réseau de neurones.

Un aspect central de l'expérimentation a consisté à évaluer l'impact du facteur d'actualisation γ (voir le tableau 3.8), qui détermine l'importance accordée aux récompenses futures. Trois valeurs ont été testées : 0.001, 0.01 et 0.99. Les résultats montrent qu'un faible facteur d'actualisation (γ = 0.001 et γ = 0.01) conduit à de meilleures performances, avec une précision de 0.93, un rappel de 0.91, un score F1 de 0.92 et une exactitude de 0.97. À l'inverse, un facteur γ élevé (0.99) entraîne une dégradation notable de la performance, avec un F1 tombant à 0.80, ce qui indique que privilégier les récompenses immé-diates favorise un apprentissage plus efficace dans ce contexte.

MERTIQUE	Facteur d'actualisation		
	$\gamma = 0.001$	$\gamma = 0.01$	$\gamma = 0.99$
Précision	0.93	0.94	0.87
Rappel	0.91	0.90	0.80
Score F1	0.92	0.92	0.80
Accuracy	0.97	0.97	0.95

Tableau 3.8 - Évaluation des performances du modèle DQL selon différentes valeurs du facteur d'actualisation.

L'analyse des matrices de confusion confirme ces observations (voir figure 3.4 et 3.5). Pour γ = 0.001, les classifications correctes sont concentrées sur la diagonale principale, avec peu de fausses classifications. Tandis que les taux d'erreur restent faibles (par example seulement 366 DDoS prédites comme BENIGN). En comparaison, avec γ = 0.99, les erreurs sont plus fréquentes et des confusions accrues entre les classes. Ces résultats montrent que l'agent DQN apprend mieux les relations immédiates entre états et récompenses lorsqu'un faible γ est utilisé.

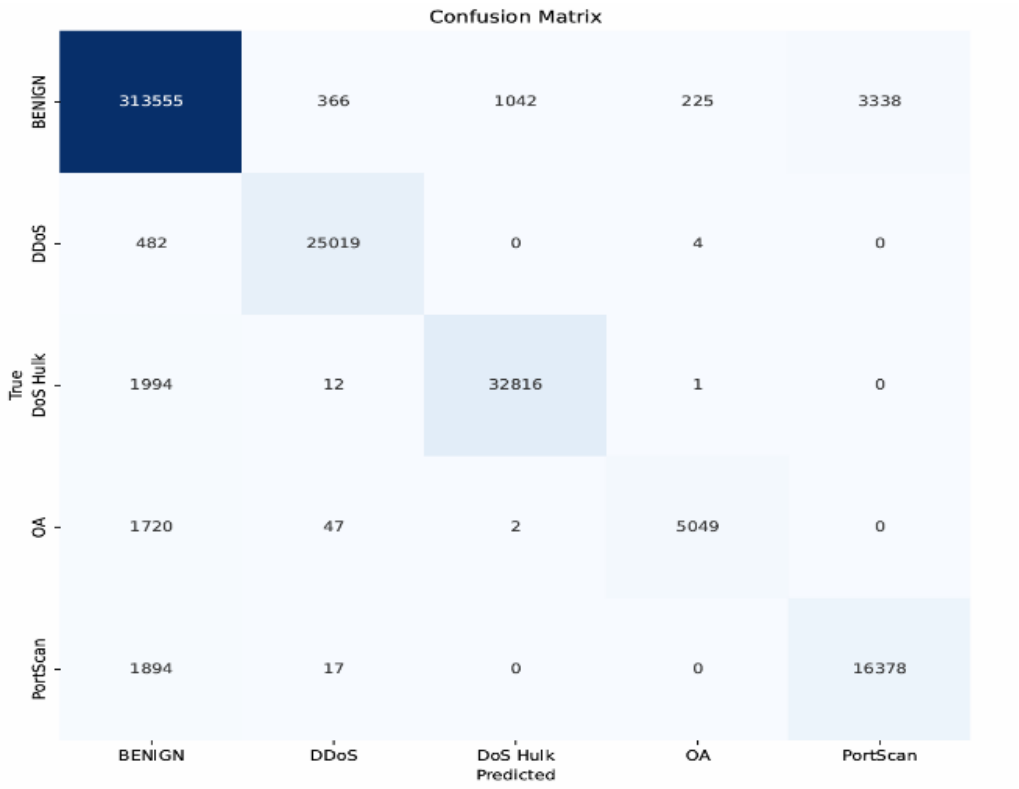

Figure 3.4 - Matrice de confusion basée sur les catégories de classification pour notre modèle DQL selon une valeur du facteur d'actualisation : γ = 0.01

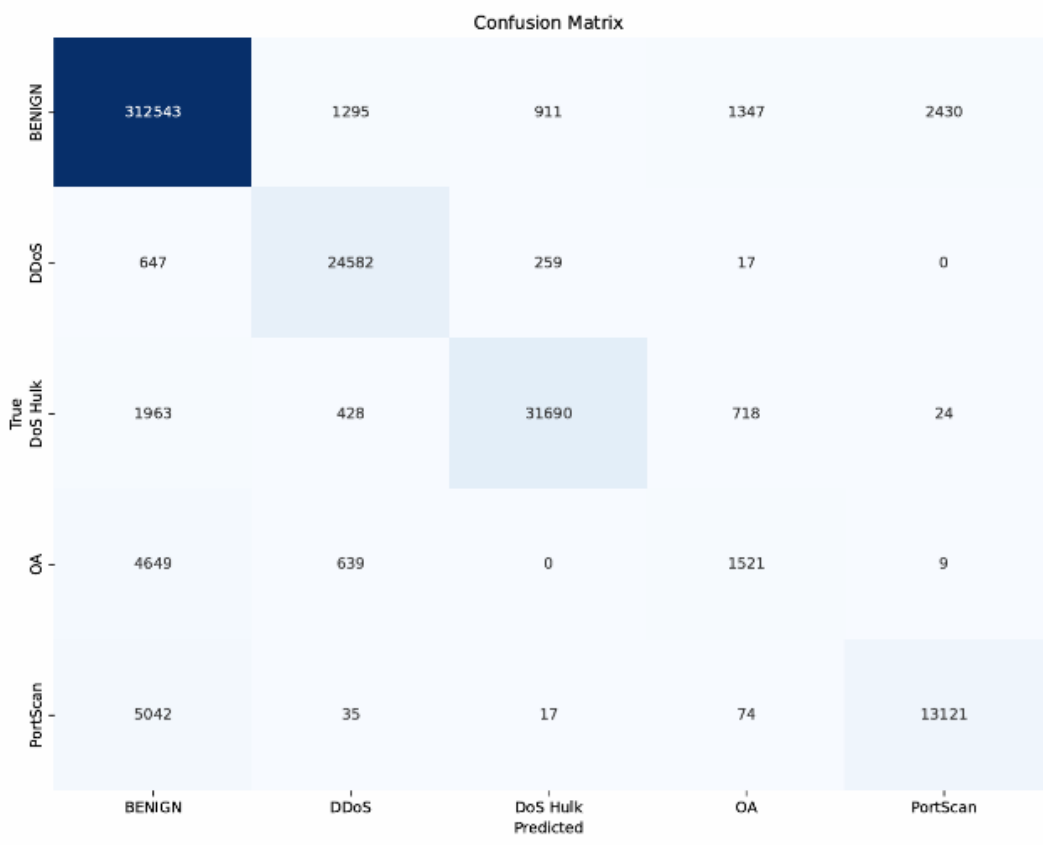

Figure 3.5 - Matrice de confusion basée sur les catégories de classification pour notre modèle DQL selon une valeur du facteur d'actualisation : $\gamma = 0.99$.

En termes de performance par type d'attaque, le modèle affiche d'excellents résultats pour les classes BENIGN (F1 = 0.99), DoS (F1 = 0.96) et DDoS (F1 = 0.99), avec une précision et un rappel particulièrement élevés. En revanche, une baisse des performances est observée pour les classes PortScan (F1 = 0.86) et OA (F1 = 0.84), principalement en raison de rappels plus faibles (respectivement 0.81 et 0.75). Ces résultats moins satisfaisants peuvent s'expliquer par une représentation insuffisante de ces classes dans l'ensemble de données, entraînant un sous-apprentissage du modèle pour ces types d'attaques. De plus, un déséquilibre marqué dans la répartition des classes accentue cette limitation, en défavorisant l'apprentissage des catégories minoritaires. Par ailleurs, la similarité entre les caractéristiques de certaines attaques peut rendre leur différenciation plus difficile, ce qui contribue également à la diminution des performances pour ces classes. Malgré ces écarts, l'exactitude demeure élevée, atteignant 0.98, ce qui témoigne de la robustesse générale du modèle (voir tableau 3.9).

MERTIQUE	Les categories des attaques				
	BENIGN	**DoS**	**DDoS**	**Port Scan**	**Others attack**
Précision	0.98	0.98	0.99	0.91	0.95
Rappel	0.99	0.94	0.99	0.81	0.75
Score F1	0.99	0.96	0.99	0.86	0.84
Accuracy	0.98	0.98	0.98	0.98	0.98

Tableau 3.9 - Métriques d'évaluation pour le modèle DQL selon chaque classe

Une analyse complémentaire a été menée afin d'évaluer l'impact de la taille du batch (batch size) sur les performances du modèle, en testant trois configurations : 32, 256 et 500, avec un facteur d'actualisation (γ) fixé à 0.001. Les résultats révèlent une amélioration progressive des performances globales à mesure que la taille du batch augmente. Les taux d'exactitude (Accuracy) obtenus sont respectivement de 0.97, 0.98 et 0.98, tandis que les scores F1 sont de 0.92, 0.92 et 0.93. L'impact de la taille du batch sur l'apprentissage résulte d'un équilibre à trouver entre la stabilité des gradients et la capacité de généralisation du modèle. Les petites tailles de batch, comme 32, introduisent davantage de bruit dans les mises à jour des poids, ce qui peut favoriser l'exploration de l'espace de recherche et permettre d'échapper à certains minima locaux. Toutefois, cela peut également engendrer une instabilité de l'apprentissage. À l'inverse, une taille de batch plus élevée, comme 500, fournit une estimation plus fiable du gradient global, ce qui permet une convergence plus stable et souvent de meilleures performances sur l'ensemble des classes. Dans notre cas, la taille de 500 semble offrir un bon équilibre entre stabilité, performance et capacité de généralisation.

En conclusion, le modèle DQN, entraîné sur l'ensemble de données CIC-IDS2017 et configuré avec une taille de batch de 500 ainsi qu'un faible facteur d'actualisation ($\gamma = 0.001$), démontre des performances optimales pour la détection d'intrusions dans des environnements réseau complexes. Il parvient à apprendre efficacement à distinguer le trafic légitime du trafic malveillant, en obtenant des résultats particulièrement solides pour les attaques les plus fréquentes, tout en maintenant des performances acceptables pour les catégories d'attaques moins représentées.

3.5.2 Les résultats pour AWID

Le corpus AWID présente une forte disparité dans la distribution des classes, ce qui en fait un cas particulièrement pertinent pour évaluer la robustesse des systèmes de détection d'intrusions. Dans ce cadre, nous avons comparé plusieurs approches : des modèles supervisés classiques (ML), un réseau de neurones MLP (Deep Learning), et un modèle de Deep Q-Network (DQN) inspiré de l'approche de López-Martín et al. (2020) [48].

Les performances globales indiquent une nette supériorité du modèle DQN, qui atteint une précision de 95,41 % et un F1-score de 93,72 %, contre 91,10 % pour Random Forest et 90,48 % pour KNN. À l'inverse, des modèles classiques tels que Naive Bayes ou Gradient Boosting, bien qu'efficaces dans des contextes mieux équilibrés, se montrent ici peu adaptés. Comme l'illustre le tableau 3.10, ces derniers tendent à prédire excessivement la classe dominante, biaisant fortement les résultats.

Il convient de détailler ici les différents modèles d'apprentissage supervisé appliqués dans notre étude comparative. Le modèle Logistic Regression a été entraîné avec une configuration standard de 1000 itérations. Ce classifieur linéaire a été choisi pour sa rapidité d'exécution, bien qu'il reste limité face aux relations non linéaires complexes caractéristiques des intrusions réseau. Le Naive Bayes a été utilisé dans sa forme gaussienne, adaptée aux variables continues, mais reste peu performant dans un contexte de classes déséquilibrées. Le Random Forest, composé de 100 arbres de décision, a démontré une meilleure robustesse grâce à sa capacité à capturer les interactions complexes entre les attributs. Le modèle K-Nearest Neighbors (KNN) a été configuré avec k = 3, offrant des résultats corrects mais sensibles au bruit et aux déséquilibres d'ensemble de données. Nous avons également évalué les performances des modèles d'ensemble Gradient Boosting et AdaBoost, reconnus pour leur capacité à réduire le biais, mais dont l'efficacité reste fortement conditionnée par un bon équilibre des données. Enfin, le réseau de neurones MLP (Multi-Layer Perceptron) a été implémenté avec trois couches entièrement connectées comprenant respectivement 128, 64 neurones et une couche de sortie softmax. Les fonctions d'activation ReLU ont été utilisées pour les couches cachées, et l'entraînement a été réalisé à l'aide de l'optimiseur Adam. Ce modèle a obtenu de bonnes performances globales, atteignant un F1-score de 95,43 %, tout en restant légèrement inférieur aux résultats du DQN.

Il est particulièrement notable que le DQN se distingue par sa performance en recall, une métrique essentielle en détection d'intrusions. Un taux de rappel élevé indique que le modèle identifie efficacement les intrusions réelles, limitant ainsi les faux négatifs. Ce critère est déterminant dans un système de

sécurité, où toute attaque non détectée peut avoir des conséquences graves. Le DQN, à travers cette capacité, démontre donc une adaptation pertinente à des contextes critiques.

		Les résultats de test			
		Accuracy	**Precision**	**Recall**	**F1-score**
Machine Learning Algorithms (Supervised Learning)	Logistic Regression	0.7638	0.9590	0.7638	0.8313
	Naive Bayes	0.6841	0.9629	0.6841	0.7798
	Random Forest	0.9110	0.9770	0.9110	0.9356
	KNN	0.9048	0.9741	0.9048	0.9304
	Gradient Boosting	0.5967	0.9544	0.5967	0.7011
	AdaBoost	0.7326	0.9358	0.7326	0.7923
Deep Learning	MLP	0.9126	0.9440	0.9126	0.9543
Deep Reinforcement Learning	DQN	0.9541	0.9244	0.9541	0.9372

Tableau 3.10 - Scores de performance pour l'ensemble des modèles (dataset AWID)

Il est important de préciser que, pour des raisons de temps de calcul, de limites techniques sur l'infrastructure et de complexité de traitement, nous avons volontairement limité l'apprentissage du modèle DQN à seulement 1 % de datasets AWID. Cette réduction, bien qu'elle implique une perte potentielle d'information, nous a permis de valider la faisabilité du modèle, d'étudier la stabilité de l'apprentissage par renforcement et d'optimiser rapidement les hyperparamètres clés.

Parmi ces paramètres, le facteur de remise γ s'est révélé particulièrement déterminant. Nous avons testé différentes valeurs de γ (0.01 et 0.99) et constaté que les meilleures performances sont obtenues avec une valeur faible (γ = 0.01). Dans le contexte de la détection d'intrusions, où il est crucial de réagir rapidement à des comportements suspects, il est souvent préférable de valoriser davantage les récompenses immédiates plutôt que les gains à long terme. Un gamma faible pousse donc l'agent à prioriser les actions ayant un effet direct et rapide sur la détection, ce qui est essentiel dans un système devant prendre des décisions en temps quasi-réel (voir tableau 3.11).

MERTIQUE	**Facteur d'actualisation**	
	$\gamma = 0.01$	$\gamma = 0.99$
Precision	0.9913	0.9782
Rappel	0.9895	0.9705
Score F1	0.9899	0.9727
Accuracy	0.9895	0.9705

Tableau 3.11 - Évaluation des performances du modèle DQL selon différentes valeurs du facteur d'actualisation.

La table des métriques par classe montre que le DQN atteint une performance quasi parfaite pour les classes injection, flooding et normal, avec des scores F1 dépassant 0.91. La classe impersonation, bien que minoritaire, est détectée avec un F1-score de 0.8966. Cela montre que même en entraînant le modèle sur seulement 1 % des données, le DQN est capable de généraliser efficacement, y compris sur les classes peu représentées (voir tableau 3.12).

Les categories des attaques	**MERTIQUE**		
	Precision	**Rappel**	**Score F1**
flooding	0.8462	1.0000	0.9167
impersonation	0.8125	1.0000	0.8966
injection	1.0000	1.0000	1.0000
normal	1.0000	0.9884	0.9942

Tableau 3.12 - Métriques d'évaluation pour le modèle DQL selon chaque classe

La courbe d'évolution des récompenses totales par épisode, présentée en figure 3.6, témoigne de la capacité du modèle à apprendre rapidement une politique efficace. Une stabilisation est observée dès l'épisode 3, indiquant une convergence rapide malgré l'apprentissage sur un faible volume. Cette propriété est particulièrement adaptée aux systèmes temps réel où la réactivité est un critère majeur.

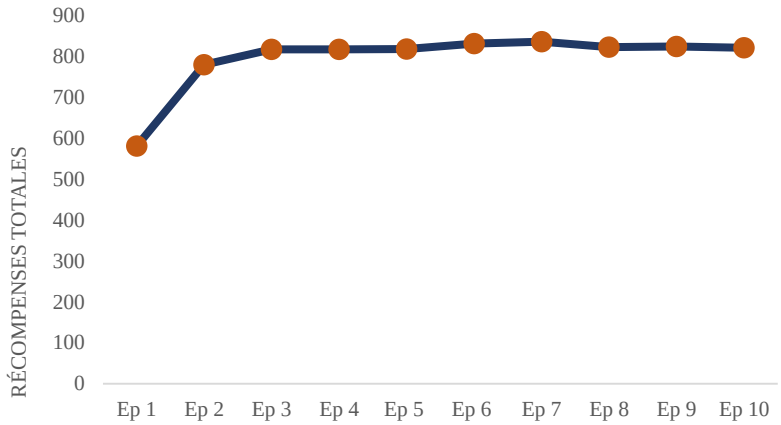

Figure 3.6 - Évolution de la récompense totale par épisode durant l'entraînement

En guise de synthèse, le tableau 3.13 présente l'ensemble des hyperparamètres finaux adoptés pour la configuration de notre modèle DQN, tel qu'utilisé dans notre approche expérimentale.

Paramètres	Description	Valeurs
num-iteration	Nombre d'itérations pour améliorer les valeurs Q dans le DQN	10
hidden_layers	Nombre de couches cachées : définit les poids et génère les sorties selon la fonction d'activation	3
num_units	Nombre d'unités cachées pour améliorer la qualité de la prédiction et de l'apprentissage	256 & 128
Valeur initiale des poids	Initialisation normale des poids	Défaut
Fonction d'activation	Fonction d'activation non linéaire	ReLU
Gamma γ	Facteur d'actualisation pour la prédiction des cibles	0.01
Taille de lot (bs)	Taille des lots d'enregistrements d'ensemble de données NSL-KDD utilisés pour l'apprentissage	32
Epsilon ε	Degré d'aléa pour l'exécution des actions	1.0
epsilon_decay	Vitesse de réduction progressive de l'exploration au cours de l'apprentissage	0.95

Tableau 3.13 - Paramètres de l'agent DQL et du réseau de neurones.

3.5.3 Les résultats pour NSL-KDD

Nous avons appliqué notre approche par apprentissage par renforcement profond (DRL) au corpus NSL-KDD dans l'objectif de l'évaluer dans un environnement aux caractéristiques différentes. Cependant, en raison de contraintes matérielles et de temps d'exécutions importantes, nous n'avons pu tester notre modèle que sur une portion très réduite des données (1%).

Cette proportion étant extrêmement faible, les résultats obtenus ne permettent ni d'évaluer les performances réelles du modèle, ni de tirer des conclusions pertinentes sur son efficacité ou sa capacité de généralisation. Ils n'ont pas non plus permis de mener des tests approfondis sur les différents paramètres du modèle. Ce test limité a néanmoins confirmé que notre architecture DQN fonctionne correctement d'un point de vue technique, en produisant des sorties cohérentes et en s'entraînant sans erreurs majeures. Il s'agit donc d'une validation fonctionnelle, mais insuffisante pour établir une quelconque analyse comparative ou évaluer la qualité de la détection.

Face à l'impossibilité de tester efficacement notre propre architecture DQN sur l'ensemble complet de NSL-KDD, nous avons choisi de nous appuyer sur le modèle DQN proposé par López-Martín et al. (2020) [48], déjà validé dans la littérature. L'objectif était de le confronter à plusieurs algorithmes classiques de machine learning, ainsi qu'à un modèle MLP (Multi-Layer Perceptron) que nous avons conçu, dans le but d'analyser la pertinence du DRL dans un contexte de détection d'intrusions.

Les modèles testés incluent : la régression logistique, Naive Bayes, Random Forest, KNN, Gradient Boosting, AdaBoost, notre MLP, ainsi que le DQN de référence. Tous ces modèles ont été évalués selon quatre métriques standard : accuracy, precision, recall et F1-score. Le tableau 3.14 suivant synthétise les résultats obtenus :

		Les résultats de test			
		Accuracy	**Precision**	**Recall**	**F1-score**
Machine Learning Algorithms (Supervised Learning)	Logistic Regression	0.7502	0.7601	0.7502	0.7066
	Naive Bayes	0.4466	0.6219	0.4466	0.4648
	Random Forest	0.7494	0.8067	0.7494	0.7042
	KNN	0.7506	0.7720	0.7720	0.7129
	Gradient Boosting	0.7704	0.8241	0.7704	0.7381
	AdaBoost	0.7306	0.7947	0.7306	0.6830
Deep Learning	MLP	0.7752	0.5312	0.7752	0.5505
Deep Reinforcement Learning	DQN	0.8787	0.8933	0.8937	0.8935

Tableau 3.14 - Scores de performance pour l'ensemble des modèles (dataset NSL-KDD)

Ce constat rejoint les conclusions présentées par López-Martín et al [48], qui soulignent dans leur étude l'adéquation des modèles de DRL, en particulier DQN, pour la détection d'intrusions réseau. Ces modèles montrent une capacité remarquable à extraire des schémas complexes à partir des données, leur permettant de mieux identifier les comportements anormaux. Le F1-score élevé du DQN indique un bon équilibre entre précision et rappel, ce qui est essentiel pour minimiser les faux négatifs, c'est-à-dire les attaques non détectées.

À l'inverse, certains modèles comme Naive Bayes ou même notre MLP, bien qu'opérationnels, peinent à atteindre des performances comparables. Le faible F1-score du MLP est notamment dû à un déséquilibre entre le recall et la précision, ce qui limite son efficacité en détection. Ces écarts de performance illustrent bien la valeur ajoutée des approches basées sur le renforcement profond, particulièrement dans des contextes où les structures de données sont complexes et dynamiques.

Un aspect clé abordé dans les travaux de López-Martín et al. (2020) [48] concerne l'influence du facteur d'actualisation γ sur les performances des modèles DRL, notamment DQN (voir tableau 3.15). Ce paramètre régule le poids accordé aux récompenses futures par rapport aux récompenses immédiates, ce qui peut profondément affecter le comportement de l'agent d'apprentissage. Dans leur étude, deux configurations extrêmes ont été explorées : γ = 0.01 : qui favorise les récompenses immédiates et γ = 0.99 : qui accorde une importance élevée aux récompenses futures. Les résultats reportés dans la littérature sont les suivants :

MERTIQUE	**Facteur d'actualisation**	
	$\gamma = 0.01$	$\gamma = 0.99$
Score F1	0.8935	0.2800
Accuracy	0.8787	0.4816

Tableau 3.15 - Évaluation des performances du modèle DQL selon différentes valeurs du facteur d'actualisation.

Ces chiffres illustrent clairement que des valeurs faibles de γ sont bien plus efficaces dans le contexte de la détection d'intrusions. En effet, dans ce type d'application, il est crucial que le système réagisse rapidement aux comportements suspects, sans attendre une accumulation de récompenses à long terme. Cela explique pourquoi une configuration comme γ = 0.01, qui renforce les décisions immédiates, permet d'atteindre des performances nettement supérieures. Dans ce cadre, deux métriques sont particulièrement pertinentes pour évaluer les performances accuracy et F1.

Le contraste observé entre les deux valeurs de γ vient confirmer l'idée que les modèles DRL, en particulier DQN, sont très sensibles à ce paramètre, et que le succès de leur application à des systèmes de détection repose en partie sur un réglage adéquat de cette variable. Cette conclusion est en parfaite cohérence avec les observations issues de notre propre étude, et nous permet de mieux comprendre l'importance de concevoir des modèles DRL adaptés aux contraintes temps-réel propres aux environnements de sécurité réseau.

Enfin, bien que notre propre modèle DQN n'ait pas pu être évalué sur l'intégralité de NSL-KDD en raison de limitations matérielles, nous en détaillons néanmoins la configuration complète dans un tableau récapitulatif (voir tableau 3.16).

Paramètres	Description	Valeurs
num-iteration	Nombre d'itérations pour améliorer les valeurs Q dans le DQN	10
hidden_layers	Nombre de couches cachées : définit les poids et génère les sorties selon la fonction d'activation	3
num_units	Nombre d'unités cachées pour améliorer la qualité de la prédiction et de l'apprentissage	256 & 128
Valeur initiale des poids	Initialisation normale des poids	Défaut
Fonction d'activation	Fonction d'activation non linéaire	ReLU
Gamma γ	Facteur d'actualisation pour la prédiction des cibles	0.01
Taille de lot (bs)	Taille des lots d'enregistrements d'ensemble de données NSL-KDD utilisés pour l'apprentissage	32
Epsilon ε	Degré d'aléa pour l'exécution des actions	1.0
epsilon_decay	Vitesse de réduction progressive de l'exploration au cours de l'apprentissage	0.95

Tableau 3.16 - Paramètres de l'agent DQL et du réseau de neurones.

3.6 Conclusion

Les résultats obtenus montrent que le modèle DQN, basé sur le Deep Reinforcement Learning, surpasse les algorithmes classiques de type Random Forest, Decision Tree, SVM ou Naïve Bayes sur l'ensemble des scénarios testés. Il a su maintenir des performances de détection élevées (accuracy, F1-score, precision, recall), et dans plusieurs cas, les a même dépassées.

Parmi les différentes configurations testées, le meilleur score a été enregistré avec DQN sur CIC-IDS2017, atteignant une précision de 0.97, ce qui reflète son efficacité dans des environnements complexes et hétérogènes. Ce très bon résultat peut notamment s'expliquer par le volume important de données disponibles dans ce dataset, offrant une grande diversité de situations et d'attaques, ce qui favorise un apprentissage plus riche et plus complet du modèle. Alors que les modèles classiques varient fortement en performance selon les cas (par exemple, SVM performant sur NSL-KDD et Decision Tree sur AWID), DQN s'est révélé plus constant et robuste, ce qui en fait une solution plus fiable dans des contextes variés. Un point particulièrement notable est la valeur élevée du recall obtenue avec DQN, une métrique essentielle en détection d'intrusions, car elle permet de réduire le nombre de faux négatifs, c'est-à-dire les attaques non détectées.

Bien que l'entraînement d'un modèle DRL comme DQN nécessite un temps de calcul important et des ressources matérielles conséquentes, notamment en raison du caractère itératif et exploratoire de l'apprentissage par renforcement, le modèle final obtenu est un réseau de neurones relativement léger. Une fois entraîné, ce modèle offre des temps de prédiction très courts, ce qui est particulièrement adapté à une exécution en environnement moderne, distribué et en temps réel.

Enfin, les résultats ont mis en évidence l'impact critique du facteur de réduction (γ) dans le processus d'apprentissage. Les meilleures performances ont été atteintes avec des valeurs faibles, ce qui suggère qu'en l'absence d'interaction avec un environnement en ligne (comme c'est le cas ici avec des données labellisées statiques), il est préférable d'adopter une mise à jour plus prudente de la politique. Cette stratégie ralentit la convergence mais la rend plus stable, un effet particulièrement observé avec les modèles DQN dans notre étude.

Conclusion Générale

Ce mémoire s'inscrit dans la problématique de la détection d'intrusions dans un contexte de cybersécurité moderne, caractérisé par une complexité croissante, des menaces évolutives et une nécessité de réactivité en temps réel. Il part d'une question centrale : est-il possible de rendre les systèmes IDS classiques plus autonomes et adaptatifs face à des environnements dynamiques et imprévisibles ? Si l'intelligence artificielle ne peut rivaliser avec l'humain en termes de créativité, elle peut toutefois exceller dans les tâches itératives, répétitives et décisionnelles où la rapidité et la précision sont primordiales.

Dans cette optique, nous avons exploré l'utilisation d'un algorithme de Deep Reinforcement Learning, le DQN, pour développer un système capable d'apprendre automatiquement à distinguer les comportements normaux des comportements malveillants. Le modèle a été implémenté et évalué sur trois ensembles de données représentatifs en cybersécurité (NSL-KDD, AWID et CIC-IDS2017), montrant des performances solides et constantes, avec une précision remarquable de 0.97 obtenue sur CIC-IDS2017, probablement favorisée par la richesse et le volume des données disponibles dans ce cas. En plus de démontrer l'efficacité du modèle en termes de métriques classiques (accuracy, recall, precision, F1-score), l'étude a mis en évidence le rôle critique du facteur de réduction (γ) dans la stabilité de l'apprentissage, et souligné que, malgré un entraînement coûteux en ressources et en temps, le modèle final offre des prédictions rapides, compatibles avec un déploiement en environnement temps réel et distribué.

Ce travail confirme ainsi le potentiel du Deep Reinforcement Learning comme levier d'autonomisation des IDS dans des contextes complexes et évolutifs.

À l'avenir, des travaux complémentaires pourraient explorer d'autres variantes de Deep Reinforcement Learning, telles que Double DQN (DDQN), qui permet de réduire le biais de surestimation en utilisant deux réseaux de neurones distincts : l'un pour sélectionner l'action, l'autre pour évaluer sa valeur. Cette approche pourrait potentiellement améliorer encore les performances observées. Par ailleurs, des méthodes combinées plus avancées comme PPO (Proximal Policy Optimization), réputées pour leur stabilité et efficacité, constituent également des pistes prometteuses dans le domaine de la détection d'intrusions basée sur l'apprentissage par renforcement.

Références

[1] **Astra Security**, "How Many Cyber Attacks Per Day?", *Astra Security Blog*. [En ligne]. Disponible sur : https://www.getastra.com/blog/security-audit/how-many-cyber-attacks-per-day/. [Consulté le : Apr. 2, 2025].

[2] J.J. Tsai and Z. Yu, "Intrusion Detection: A Machine Learning Approach," vol. 3, World Scientific, 2011.

[3] **Le Monde Informatique**, "Trop de faux positifs saturent les équipes sécurité", *Le Monde Informatique*. [En ligne]. Disponible sur : https://www.lemondeinformatique.fr/actualites/lire-trop-de-faux-positifs-saturent-les-equipes-securite-78549.html. [Consulté le : Apr. 2, 2025].

[4] **National Institute of Standards and Technology (NIST)**, "Security Assurance", *NIST Computer Security Resource Center*. [En ligne]. Disponible sur : https://csrc.nist.gov/glossary/term/security_assurance. [Consulté le : Apr. 3, 2025]

[5] **Fortinet**, "Authentication vs. Authorization: What's the Difference?", *Fortinet Cyber Glossary*. [En ligne]. Disponible sur : https://www.fortinet.com/fr/resources/cyberglossary/authentication-vs-authorization. [Consulté le : Apr. 5, 2025].

[6] **OWASP,** Foundation, *Denial of Service (DoS)*, [En ligne]. Disponible sur : https://owasp.org/www-community/attacks/Denial_of_Service. [Consulté le : Apr. 5, 2025].

[7] **Cloudflare,** *What is a DDoS attack?*, [En ligne]. Disponible sur : https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/ [Consulté le : Apr. 5, 2025].

[8] **Acunetix**, "What are Injection Attacks?", *Acunetix Blog*. [En ligne]. Disponible sur : https://www.acunetix.com/blog/articles/injection-attacks/. [Consulté le : Apr. 5, 2025].

[9] **Cloudflare**, "What is SQL Injection?", *Cloudflare Learning Center*. [En ligne]. Disponible sur : https://www.cloudflare.com/learning/security/threats/sql-injection/. [Consulté le : Apr. 5, 2025].

[10] **CyberUniversity**, "Attaques informatique: Tout savoir sur les différentes menaces", *CyberUniversity*. [En ligne]. Disponible sur: https://www.cyberuniversity.com/post/attaque-informatique-en-quoi-ca-consiste. [Consulté le : Apr. 5, 2025].

[11] **CrowdStrike,** "Common Cyberattacks," *CrowdStrike*, [En ligne]. Disponible sur : https://www.crowdstrike.com/en-us/cybersecurity-101/cyberattacks/common-cyberattacks/. [Consulté le : Apr. 8, 2025].

[12] **IT-Connect**, "Attaques par brute force: principes et mesures de protection", *IT-Connect*. [En ligne]. Disponible sur : https://www.it-connect.fr/attaques-brute-force-principes-et-mesures-de-protection/. [Consulté le : Apr. 5, 2025].

[13] **OWASP**, "Brute-force attack", *OWASP*. [En ligne]. Disponible sur : https://owasp.org/www-community/attacks/Brute_force_attack. [Consulté le : Apr. 5, 2025].

[14] **Linux Console**, "Attaques par force brute", *Linux Console*. [En ligne]. Disponible sur : https://fr.linuxconsole.net/?p=22697. [Consulté le : Apr. 5, 2025].

[15] **IBM,** *Understanding Man-in-the-Middle (MitM) Attacks*, [En ligne]. Disponible sur : https://www.ibm.com/think/topics/man-in-the-middle [Consulté le : 7 avr. 2025].

[16] **LockSelf,** *Attaque par homme du milieu : qu'est-ce que c'est ?*, [En ligne]. Disponible sur : https://blog.lockself.com/attaque-par-homme-du-milieu [Consulté le : 7 avr. 2025].

[17] **Rapid7,** *Man-in-the-Middle (MitM) Attacks*, [En ligne]. Disponible sur sur : https://www.rapid7.com/fundamentals/man-in-the-middle-attacks/ [Consulté le : 7 avr. 2025].

[18] **Y. Shen and M. Tehranipoor,** "A Study of Probing Attacks on Secure ICs," *IEEE Design & Test*, vol. 34, no. 5, pp. 63–71, Oct. 2017. [En ligne]. Disponible sur : https://tehranipoor.ece.ufl.edu/wp-content/uploads/2021/07/2017-DT-Probe.pdf

[19] **Kaspersky,** "Qu'est-ce qu'un botnet ?", *Kaspersky*, [En ligne]. Disponible sur sur : https://www.kaspersky.fr/resource-center/threats/botnet-attacks. [Consulté le : 8 avril 2025].

[20] **OWASP,** "Buffer Overflow," *OWASP Foundation*, [En ligne]. Disponible sur : https://owasp.org/www-community/vulnerabilities/Buffer_Overflow. [Consulté le : Apr. 8, 2025].

[21] **Abid, M. and Beneddine,** M.M.A., 2021. Conception d'un IDS basé sur le Deep Learning et RBN (Doctoral dissertation, Université Ibn Khaldoun-Tiaret-).

[22] Cisco, "What is a Firewall?" *Cisco*, [En ligne]. Disponible sur: https://www-cloud.cisco.com/site/uk/en/learn/topics/security/what-is-a-firewall.html#tabs-8649922ce2-item-ae9878695b-tab. [Consulté le : 11-Apr-2025].

[23] Palo Alto Networks, "Types of Firewalls," *Palo Alto Networks*, [En ligne]. Disponible sur : https://www.paloaltonetworks.com/cyberpedia/types-of-firewalls. [Consulté le : 11-Apr-2025].

[24] Cloudflare, "Qu'est-ce qu'un VPN ?", *Cloudflare*, [En ligne]. Disponible sur : https://www.cloudflare.com/fr-fr/learning/access-management/what-is-a-vpn/. [Consulté le 11 avril 2025].

[25] Cisco, "What is a Virtual Private Network (VPN)?", *Cisco*, [En ligne]. Disponible sur : https://www.cisco.com/site/us/en/learn/topics/security/what-is-a-virtual-private-network-vpn.html#tabs-35d568e0ff-item-4bd7dc8124-tab. [Consulté le 11 avril 2025].
[26] Kaspersky, "What is Cryptography?," *Kaspersky Resource Center*, [En ligne]. Disponible sur : https://www.kaspersky.fr/resource-center/definitions/what-is-cryptography. [Consulté le : Apr. 11, 2025].
[27] Hamouda, D., 2020. Un système de détection d'intrusion pour la cybersécurité.
[28] Farhaoui, Y., 2012. Evaluation des Systèmes de Détection et de Prévention des Intrusions et la Conception d'un BiIDS
[29] Sen, S., 2015. A survey of intrusion detection systems using evolutionary computation. In *Bio-inspired computation in telecommunications* (pp. 73-94). Morgan Kaufmann.
[30] Kim, K., Aminanto, M.E. and Tanuwidjaja, H.C., 2018. Network intrusion detection using deep learning: a feature learning approach. Springer.
[31] Zimmer, M., 2018. *Apprentissage par renforcement développemental* (Doctoral dissertation, Université de Lorraine).
[32] Learning, D.R., 2020. Deep Reinforcement Learning: Fundamentals, Research and Applications.
[33] OpenAI, « CartPole Environment — Gym Documentation », en ligne. Disponible sur : https://www.gymlibrary.dev/environments/classic_control/cart_pole/. [Consulté le 27-avr.-2025].
[34] Barto, A.G., Sutton, R.S. and Anderson, C.W., 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5), pp.834-846.
[35] DeepMind, « AlphaGo », *DeepMind*, [En ligne]. Disponible sur : https://deepmind.google/research/breakthroughs/alphago/. [Consulté le : 28-avr.-2025].
[36] Williams, R.J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, *8*, pp.229-256.
[37] Rummery, G.A. and Niranjan, M., 1994. On-line Q-learning using connectionist systems (Vol. 37, p. 14). Cambridge, UK: University of Cambridge, Department of Engineering.
[38] Sutton, R.S. and Barto, A.G., 1998. Reinforcement learning: An introduction (Vol. 1, No. 1, pp. 9-11). Cambridge: MIT press.
[39] Lin, L.J., 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, *8*, pp.293-321.
[40] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. and Dieleman, S., 2016. Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), pp.484-489.
[41] Sutton, R.S., 1991. Dyna, an integrated architecture for learning, planning, and reacting. ACM Sigart Bulletin, 2(4), pp.160-163.
[42] Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. and Kavukcuoglu, K., 2016, June. Asynchronous methods for deep reinforcement learning. In International conference on machine learning (pp. 1928-1937). PmLR.
[43] Bergdahl, J., 2017. Asynchronous advantage actor-critic with adam optimization and a layer normalized recurrent network.
[44] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.
[45] Touzet, C., 1992. les réseaux de neurones artificiels, introduction au connexionnisme. Ec2.
[46] Kim, P., 2017. Matlab deep learning. *With machine learning, neural networks and artificial intelligence*, *130*(21), p.151.
[47] Stolfo, S.J., Fan, W., Lee, W., Prodromidis, A. and Chan, P.K., 2000, January. Cost-based modeling for fraud and intrusion detection: Results from the JAM project. In Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00 (Vol. 2, pp. 130-144). IEEE.
[48] Lopez-Martin, M., Carro, B. and Sanchez-Esguevillas, A., 2020. Application of deep reinforcement learning to intrusion detection for supervised problems. Expert Systems with Applications, 141, p.112963.