Représentation à base radiale pour l'apprentissage par renforcement visuel

Julien Hautot

ÉCOLE DOCTORALE
DES SCIENCES POUR L'INGÉNIEUR
Université Clermont Auvergne

Représentation à base radiale pour l'apprentissage par renforcement visuel

---

Thèse présentée par **Julien HAUTOT**

Pour obtenir le grade de :
Docteur de l'Université Clermont Auvergne

Spécialité : **Image, système de perception, robotique**

Soutenue publiquement le **23 octobre 2024** devant le jury composé de

David FILLIAT	Rapporteur	Professeur des Universités ENSTA Paris
François SEPTIER	Rapporteur	Professeur des Universités Université Bretagne Sud
Thierry CHATEAU	Président du jury	Professeur des Universités Université Clermont Auvergne
Tomoko MATSUI	Examinatrice	Professeur des Universités The Institute of Statistical Mathematics, Tokyo
Nourddine AZZAOUI	Directeur	Professeur des Universités, HDR Université Clermont Auvergne
Céline TEULIERE	Encadrante	Maîtresse de Conférences Université Clermont Auvergne

Remerciement

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué, de près ou de loin, à la réalisation de cette thèse.

Tout d'abord, je remercie chaleureusement mes encadrants, Céline Teulière et Azzaoui Nourddine, pour leur accompagnement, leurs conseils avisés et leur soutien constant tout au long de ce projet. Leur expertise et leur vision m'ont permis de surmonter les nombreux défis rencontrés au cours de ces années de recherche. Leur patience et leur encouragement ont été une source d'inspiration précieuse.

Je tiens également à remercier les membres de mon jury pour l'honneur qu'ils me font d'évaluer mon travail. Un grand merci à David Filliat, Professeur des Universités à ENSTA Paris, et François Septier, Professeur des Universités à l'Université Bretagne Sud, pour avoir accepté le rôle de rapporteurs de cette thèse. Je suis très reconnaissant pour le temps qu'ils ont consacré à la lecture et à l'évaluation de mon manuscrit, ainsi que pour leurs retours constructifs.

Je remercie aussi sincèrement Thierry Chateau, Professeur des Universités à l'Université Clermont Auvergne, et Tomoko Matsui, Professeur des Universités à The Institute of Statistical Mathematics, Tokyo, pour avoir accepté de faire partie de mon jury et pour l'intérêt qu'ils portent à mon travail.

Un remerciement particulier à mes collègues et amis, pour leur soutien, leurs échanges scientifiques et leurs moments de camaraderie. Vos encouragements ont été essentiels dans les moments difficiles et votre bonne humeur a souvent illuminé mes journées de travail.

Je n'oublie pas ma famille, dont l'amour inconditionnel et la patience m'ont permis de mener à bien cette aventure. Vous avez toujours cru en moi et vous avez été là à chaque étape de ce chemin. À mes parents, merci pour votre soutien sans faille et pour avoir été ma source de motivation.

Enfin, je tiens à remercier toutes les personnes que je n'ai pas citées mais qui, à leur manière, ont contribué à la réussite de cette thèse.

À tous, merci.

"Your soul needs exploration and growth, and the only way you will get it is by forcing yourself to be uncomfortable."
Mandragora - Center Of The Universe

Résumé

Ce travail de thèse s'inscrit dans le contexte de l'apprentissage par renforcement (*Renforcement Learning - RL*) à partir de données image. Contrairement à l'apprentissage supervisé qui permet d'effectuer différentes tâches telles que la classification, la régression ou encore la segmentation à partir d'une base de données annotée, le *RL* permet d'apprendre, sans base de données, via des interactions avec un environnement. En effet, dans ces méthodes, un agent tel qu'un robot va effectuer différentes actions afin d'explorer son environnement et de récupérer les données d'entraînement. L'entraînement de ce type d'agent s'effectue par essais et erreurs ; lorsque l'agent échoue dans sa tâche, il est pénalisé, tandis que lorsqu'il réussit, il est récompensé. Le but pour l'agent est d'améliorer son comportement pour obtenir le plus de récompenses à long terme.

Nous nous intéressons aux extractions visuelles dans des scénarios de *RL* utilisant des images vues à la première personne. L'utilisation de données visuelles fait souvent appel à des réseaux de convolution profonds permettant de travailler directement sur des images. Cependant, ces réseaux présentent une complexité calculatoire importante, manquent d'explicabilité et souffrent parfois d'instabilité. Pour surmonter ces difficultés, nous avons investigué le développement d'un réseau basé sur des fonctions à base radiales qui permettent des activations éparses et localisées dans l'espace d'entrée. Les réseaux à base radiale (*RBFN*) ont connu leur apogée dans les années 90, puis ont été supplantés par les réseaux de convolution car ils étaient jugés difficilement utilisables sur des images en raison de leur coût en calcul. Dans cette thèse, nous avons développé un extracteur de caractéristiques visuelles inspiré des RBFN en simplifiant le coût calculatoire sur les images. Nous avons utilisé notre réseau pour la résolution de tâches visuelles à la première personne et nous avons comparé ses résultats avec différentes méthodes de l'état de l'art; en particulier, des méthodes d'apprentissage de bout-en-bout, des méthodes utilisant l'apprentissage de représentation d'état et des méthodes d'apprentissage machine extrême. Différents scénarios ont été testés issus du simulateur VizDoom, ainsi que du simulateur physique de robotique Pybullet. Outre la comparaison des récompenses obtenues après l'apprentissage, nous avons aussi effectué différents tests sur la robustesse au bruit, la génération des paramètres de notre réseau et le transfert d'une tâche dans la réalité.

Le réseau proposé obtient les meilleures performances lors d'apprentissage par renforcement sur les scénarios testés, tout en étant plus simple d'utilisation et d'interprétation. De plus, notre réseau est robuste face à différents bruits, ce qui ouvre la voie à un transfert efficace des connaissances acquises en simulation à la réalité.

Mots-Clés : RBFN, RL visuel, Extraction de représentations, Observabilité partielle, CNN, ResNet, Auto-Encodeur, EML, Reality Gap, VizDoom, Pybullet, Robotique.

Sommaire

Liste des figures

Liste des tableaux

Acronymes

Les acronymes utilisés dans ce document sont reportés dans le tableau suivant :

Symbole	*Définition*
AE	Autoencoder - Auto-encodeur
Adam	Adaptive Moment Estimation - Estimation adaptative des moments
A3C	Asynchronous Advantage Actor-critique - Acteur-Critique Asynchrone avec Avantages
BA	Environnement Basic de VizDoom
CL	Contrastive Learning - Apprentissage contrastif
CNN	Convolutional Neural Network - Réseaux neuronal convolutionnel
DCNN	DeConvolutional Neural Network - Réseau neuronal déconvolutionnel
DDPG	Deep Deterministic Policy Gradient - Gradient de politique déterministe profonde
DDQN	Double Deep Q-Network - Double réseau Q profond
DQN	Deep Q-Network - Réseau Q profond
DRL	Deep Reinforcement Learning - Apprentissage par renforcement profond
DRQN	Deep Recurrent Q-Network - Réseau Q récurrent profond
DL	Deep Learning - Apprentissage profond
DTC	Environnement Defend the Center de VizDoom
ELM	Extreme Learning Machine - Machine à apprentissage extrême
FCL	Fully Connected Layer - Couche entièrement connectée
FD	Forward Dynamics - Dynamique directe
FF	Feedforward - Propagation avant
FFNN	Feedforward Neural Network - Réseau neuronal à propagation avant
GA	Genetic Algorithm - Algorithme génétique
GPU	Graphics Processing Unit - Unité de traitement graphique
HG	Environnement Health gathering de VizDoom
HGS	Environnement Health Gathering Supreme de VizDoom
IA	Artificial Intelligence - Intelligence artificielle
KL	Kullback-Leibler Divergence - Divergence de Kullback-Leibler
LSTM	Long Short-Term Memory - Mémoire à court et long terme

MDP	Markov Decision Process - Processus de décision de Markov
MLP	Multilayer Perceptron - Perceptron multicouche
MSE	Mean Squared Error - Erreur quadratique moyenne
MWH	Environnement My Way Home de VizDoom
PCA	Principal Component Analysis - Analyse en composantes principales
POMDP	Partially Observable Markov Decision Process - Processus de décision de Markov partiellement observable
PPO	Proximal Policy Optimization - Optimisation de politique proximale
PSO	Particle Swarm Optimization - Optimisation par essaim de particules
RBF	Radial Basis Function - Fonction à base radiale
RBFN	Radial Basis Function Network - Réseau de fonctions à base radiale
ReLU	Rectified Linear Unit - Unité linéaire rectifiée
RGB	Red Green Blue - Rouge Vert Bleu
RL	Reinforcement Learning - Apprentissage par renforcement
RNN	Recurrent Neural Network - Réseau neuronal récurrent
SAC	Soft Actor critique
SGD	Stochastic Gradient Descent - Descente de gradient stochastique
SRL	State Representation Learning - Apprentissage de représentation d'état
SVM	Support Vector Machine - Machine à vecteurs de support
TSNE	t-Distributed Stochastic Neighbor Embedding
VAE	Variational Autoencoder - Auto-encodeur variationnel
VRBFN	Visual Radial Basis Function Network - Réseau visuel de fonctions à base radiale

Chapitre 1

Introduction Générale

1.1 Contexte et motivation

L'apprentissage par renforcement (*Reinforcement Learning* - *RL*) constitue une méthode fondamentale de l'apprentissage automatique. L'objectif du *RL* est de développer des algorithmes capables d'apprendre à prendre des décisions optimales dans un environnement dynamique. Les mécanismes d'apprentissage inhérents à ces méthodes s'inspirent de ceux des animaux. En effet, l'optimisation du système, appelé en *RL* un agent, repose généralement sur un processus d'essai-erreur : l'agent est gratifié lorsqu'il accomplit une tâche avec succès et en cas d'échec, il est pénalisé ou ne reçoit pas de récompenses positives. Contrairement aux méthodes d'apprentissage supervisées, qui requièrent un ensemble de données préalablement collectées et annotées, l'apprentissage par renforcement ne dépend pas d'un jeu de données étiqueté préexistant. En effet, l'agent collecte lui-même les données nécessaires à son apprentissage en interagissant avec l'environnement. Les algorithmes de *RL* permettent de trouver une fonction optimale reliant l'état à l'action, appelée politique ou stratégie optimale, afin de maximiser l'accumulation des récompenses au fil du temps.

Une des motivations des recherches en *RL* est de créer des robots capables d'apprendre via l'exploration de leur environnement à partir de leurs propres capteurs. Dans cette thèse, on considèrera un capteur visuel mobile, fournissant des vues à la première

personne. Entraîner directement un robot à effectuer une tâche dans un cadre réel est difficile dans ce domaine. Le comportement aléatoire initial d'un agent peut s'avérer destructeur dans le monde réel, et les coûts associés aux expériences réelles sont prohibitifs. De plus, l'apprentissage en *RL* peut nécessiter un temps considérable, certaines tâches exigeant des années d'entraînement avec des méthodes basiques de *RL*. Ainsi, la collecte des données nécessaires à l'apprentissage est souvent effectuée en environnement simulé, bien que cela puisse présenter des limitations, notamment les différences entre la simulation et le monde réel, communément appelées le *reality gap*.

Le domaine de l'apprentissage automatique sur des données image a connu d'énormes avancées grace à la disponibilité de bases de données conséquentes et au développement des GPUs, et suite aux travaux sur les réseaux de convolution (*Convolutional Neural network - CNN*), permettant le traitement direct des images dans les processus d'apprentissage. Depuis, de nombreux réseaux de neurones ont été développés, utilisant très souvent des couches de convolution. Cette révolution a largement impacté le domaine du *RL*. En effet, depuis leur succès dans la résolution de jeux Atari [1] avec un entraînement de bout en bout (*end-to-end*), les réseaux de convolution se sont dès lors imposés comme une approche incontournable pour la résolution de tâches de *RL* visuel. L'entraînement de bout en bout est une approche où l'on entraine un modèle à effectuer une tâche à partir de données brutes sans étape intermédiaire ou séparation entre la perception et la commande. De nombreux travaux ont été publiés pour améliorer l'apprentissage par renforcement visuel, alimentant une course à la performance dominée par des réseaux de neurones de plus en plus complexes et coûteux à optimiser, par exemple avec l'apparition des réseaux transformer [2]. Cette observation est également vraie pour l'ensemble du domaine scientifique de l'apprentissage automatique. On parle alors d'apprentissage automatique profond (*Deep Learning - DL*) et d'apprentissage par renforcement profond (*Deep Reinforcement Learning - DRL*).

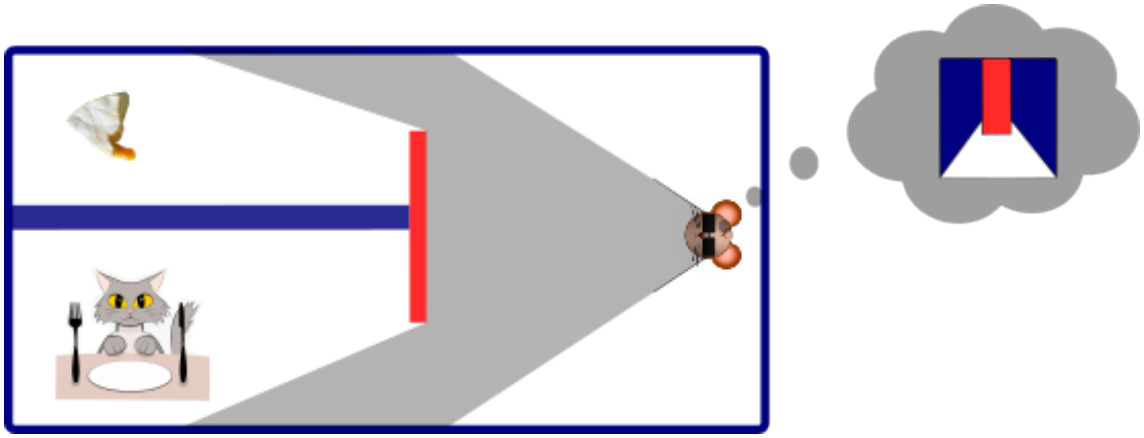


Figure 1.1: Illustration d'un scénario d'apprentissage par renforcement visuel à la première personne.

Motivée par le domaine robotique, dans lequel un agent peut être amené à explorer un environnement inconnu, cette thèse se focalise plus particulièrement sur des tâches de *RL* à partir de données visuelles en vue à la première personne. Dans ces environnements, l'agent perçoit son environnement à travers le prisme de sa vision, limitée à une certaine zone, ce qui le rend non omniscient. Par exemple, dans l'illustration 1.1, l'observation visuelle de la souris ne permet pas d'obtenir toutes les informations de l'état actuel de l'environnement. En effet, avec son observation, elle peut définir sa position par rapport au mur rouge, mais ne peut pas savoir où se trouve le fromage ou le chat, ni même s'il y a un fromage ou un chat si c'est sa première fois dans cet environnement. Ce type d'environnement représente un réel défi pour les méthodes de *RL end-to-end*. Bien que de nombreux travaux aient abordé ce problème en fournissant à l'agent des informations additionnelles telles que la position de l'agent, du fromage ou du chat, ou indirectes en fournissant par exemple une carte de l'environnement ou en donnant un sens supplémentaire à l'agent, dans ces travaux nous voulons résoudre ce type de scénario en utilisant uniquement les observations visuelles. L'objectif est d'explorer l'importance de la représentation visuelle utilisée pour la résolution de ce type de tâches.

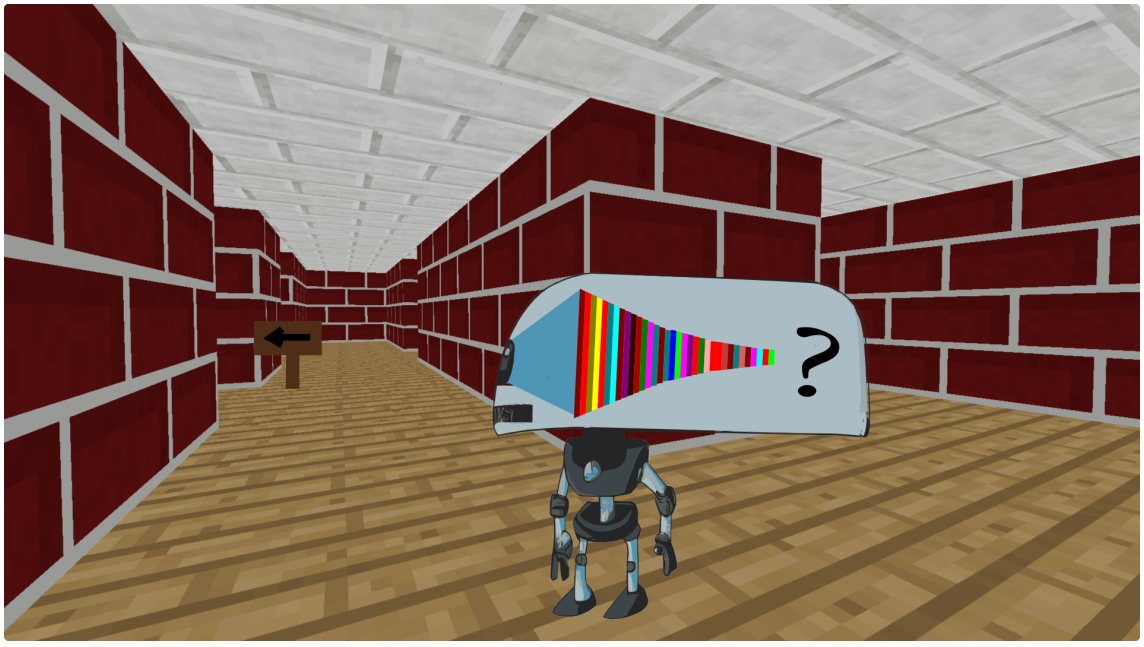


Figure 1.2: Caricature du paradoxe entre l'utilisation de très grands modèles dans la résolution de tâches a priori simples visuellement.

L'objectif de cette thèse est de développer une méthode efficace et facile à utiliser pour résoudre des tâches visuelles à la première personne, contrairement à la tendance de la recherche qui privilégie des méthodes profondes et coûteuses à optimiser. En effet, les modèles dans le domaine du DRL sont souvent inspirés des modèles développés dans d'autres domaines tels que la classification ou la segmentation. Cependant, la résolution d'un problème de navigation par *RL* par exemple ne requiert pas

nécessairement la même compréhension de l'image qu'une classification de multiples objets de la vie courante. L'illustration 1.2 caricature ce paradoxe.

Nos travaux interrogent la représentation visuelle nécessaire à l'apprentissage par renforcement de différentes tâches visuelles. Ils se situent ainsi au croisement entre l'apprentissage des représentations d'état (*State Representation Learning - SRL*) pour le *RL* , l'extrême machine learning (*ELM*) et les approches anciennes d'approximation par fonction à base radiale. L'*ELM* implique l'utilisation de réseaux de neurones non entraînés pour prétraiter les données brutes en vue d'un apprentissage ultérieur. Le *SRL* se concentre sur l'acquisition d'une représentation compacte des états dérivée des actions de l'agent et évoluant au cours de l'apprentissage [3]. Ces représentations facilitent l'apprentissage de la résolution des tâches en fournissant des informations organisées et structurées extraites des observations visuelles. Contrairement aux approches de bout en bout, où l'apprentissage de la perception et la décision sont fusionnées, le *SRL* les sépare. Initialement, l'accent est mis sur la compréhension de l'environnement et l'extraction de données pertinentes pour la tâche à partir des observations de l'agent, puis ces données extraites sont utilisées pour apprendre la tâche elle-même. Outre la réduction de la taille des données d'apprentissage, le *SRL* offre un contrôle accru sur l'extraction, ce qui est avantageux lors de changements de domaines. Par exemple, en apprenant un espace de caractéristiques commun à deux domaines, il devient plus facile d'adapter un agent à un nouvel environnement. De plus, en réduisant la dimension du problème, le *SRL* diminue les problèmes d'interférences contraîrement aux approches de bout en bout qui sont sujettes aux interférences lors de l'entraînement des réseaux de neurones. En apprentissage automatique traditionnel, les interférences se produisent lorsque la distribution des données d'entraînement est modifiée, ce qui conduit le réseau à s'ajuster à la nouvelle distribution et à "oublier" l'apprentissage sur les données initiales. En *RL* l'agent explore l'environnement et apprend simultanément, ce qui rend le phénomène d'interférence plus prononcé.

La méthode développée dans cette thèse est inspirée des travaux sur la perception des enfants de Westermann [4]. Dans ces travaux, les modèles et résultats établis par Fantz sur la perception des enfants [5] sont approximés par un réseau utilisant des fonctions à base radiale. Ces fonctions permettent une activation localisée sur certaines caractéristiques des données. Dans cette thèse, nous avons étudié l'utilisation des fonctions à base radiale [6] pour la perception de problèmes de *RL* visuel. Ces réseaux ne sont pas naturellement conçus pour être utilisés sur des données de grande dimension. Nous avons proposé un extracteur de caractéristiques visuelles inspiré du fonctionnement des fonctions à base radiale, le *Visual Radial Basis Function Network* (*VRBFN*). Différentes tâches de difficulté croissante ont été testées et comparées aux méthodes de l'état de l'art. Nous avons démontré l'efficacité de notre méthode par rapport à cinq autres méthodes de pointe dans sept environnements différents. De plus, nous avons proposé des analyses sur les caractéristiques ainsi que la robustesse de notre méthode.

1.2 Contributions

Visual Radial Basis Function Network – *partiellement publié dans* [7] Nous proposons un extracteur visuel sans apprentissage basé sur les Réseaux de Fonctions à Base Radiale, appliqué directement sur des images à l'aide de zones d'attention. Nous mettons en avant les avantages de notre extracteur, tels que la parcimonie des activations et leur localité.

Étude comparative – *partiellement publié dans* [7] Afin de démontrer l'efficacité des caractéristiques extraites, nous les utilisons dans un processus d'apprentissage par renforcement en utilisant le Q-learning et la Proximal Policy Optimization. Nous comparons les récompenses obtenues par notre agent avec cinq méthodes de l'état de l'art dans différents scénarios partiellement observables. Nous montrons qu'il est possible de résoudre des tâches visuelles basiques à la première personne en utilisant uniquement une couche entièrement connectée. Pour des tâches plus complexes, deux couches ont été suffisantes. Notre réseau est léger et simple d'utilisation, robuste au bruit, et les résultats obtenus sont souvent meilleurs que ceux des méthodes de l'état de l'art.

Spécialisation et Adaptabilité Afin de démontrer la modularité de notre méthode, nous avons mené des études sur les changements dans la distribution des zones d'attention et examiné l'apprentissage des paramètres gaussiens. Un réseau adapté à une tâche de fixation, optimisé en simulation, a été transféré avec succès dans le monde réel, des résultats qualitatif sont présentés.

1.3 Structure du manuscrit

Dans un premier temps, nous introduirons les notions d'apprentissage machine nécessaires à la compréhension des travaux présentés dans ce manuscrit. Ensuite, nous présenterons les problématiques liées à l'apprentissage par renforcement sur des images, ainsi que les méthodes de l'état de l'art utilisées pour ces problématiques. Puis, nous développerons le fonctionnement de notre méthode, le *VRBFN*, et sa mise en place dans des algorithmes d'apprentissage par renforcement. Des résultats sur l'apprentissage de différentes tâches visuelles seront présentés ainsi qu'une analyse des caractéristiques de notre méthode. Enfin, nous présenterons des résultats préliminaires sur le paramétrage de notre méthode et sur la transférabilité des connaissances apprises par notre réseau dans le monde réel. Plus spécifiquement, le manuscrit est organisé comme suit :

Chapitre 2 : Ce chapitre introduit les notions fondamentales sur lesquelles se basent nos travaux, notamment le fonctionnement des réseaux de neurones utilisés dans cette thèse, dont les réseaux à base radiale. De plus, les principes fondamentaux de l'apprentissage par renforcement seront introduits.

Chapitre 3 : Dans ce chapitre, nous dresserons un état de l'art des différentes approches utilisées en matière de représentations visuelles pour le *RL*, telles que l'apprentissage de bout-en-bout, les méthodes de *SRL*, et les méthodes moins conventionnelles qui utilisent, par exemple, les fonctions à base radiale en apprentissage par renforcement.

Chapter 4 : Nous présenterons le *VRBFN*, en détaillant son inspiration, son fonctionnement, son implémentation, ainsi que les propriétés des caractéristiques obtenues. Nous effectuerons également des tests de notre extracteur dans un processus d'apprentissage par renforcement à travers sept différents scénarios partiellement observables. L'objectif est de quantifier l'utilité des extractions dans l'apprentissage de tâches complexes. Nous proposerons une comparaison des récompenses obtenues par le *VRBFN* avec cinq méthodes différentes. Ensuite, nous discuterons des possibilités offertes par notre méthode en ce qui concerne les particularités de nos activations éparses et locales.

Chapter 5 : Une discussion sur la génération de nos paramètres sera menée afin d'envisager la possibilité de spécialiser la perception de l'agent avec des distributions non aléatoires ou avec de l'apprentissage des paramètres gaussiens. De plus, nous aborderons les capacités de notre méthode à s'adapter à différents domaines et présenterons des résultats qualitatifs sur le transfert des connaissances dans le monde réel.

Chapter 6 : Nous conclurons par un récapitulatif général des travaux présentés dans cette thèse, et nous discuterons des perspectives d'amélioration et d'application de notre méthode.

Chapitre 2

Fondamentaux

Introduction

Ce chapitre introduit les différentes notions d'apprentissage machine exploitées dans ce manuscrit. Particulièrement nous utiliserons le *DRL* qui combine les réseaux neuronaux avec le *RL*. Il est essentiel d'introduire les différentes notions utiles à la conception de nos méthodes qui requièrent l'utilisation d'un réseau de neurones, d'un agent, d'un environnement et de méthodes d'optimisation.

Dans un premier temps, nous rappellerons le fonctionnement et la construction des réseaux de neurones. Ensuite, nous présenterons l'apprentissage par renforcement et les équations impliquées dans la résolution des problèmes de *RL*. Nous présenterons brièvement les algorithmes du *RL* et nous insisterons plus particulièrement sur les

deux algorithmes d'apprentissage par renforcement utilisés dans cette thèse : le *Deep Q-Network* (*DQN*) et le *Proximal Policy Optimization* (*PPO*). Nous expliquerons en détail leur fonctionnement, leurs avantages et leurs domaines d'application.

2.1 Réseaux de neurones artificiels

En apprentissage profond par renforcement, l'utilisation de réseaux de neurones en tant qu'approximateurs de fonctions est courante. Cette partie mettra en lumière les différents types de réseaux de neurones utilisés dans la thèse, ainsi que les méthodes d'optimisation nécessaires pour leur utilisation en tant qu'approximateurs de fonctions.

2.1.1 Le neurone artificiel

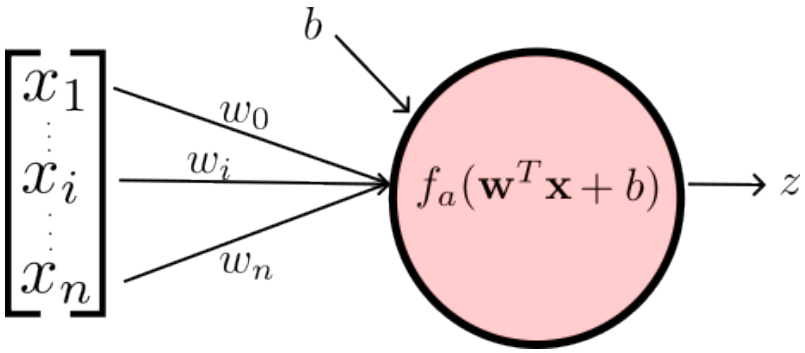


Figure 2.1: Schéma d'un neurone formel

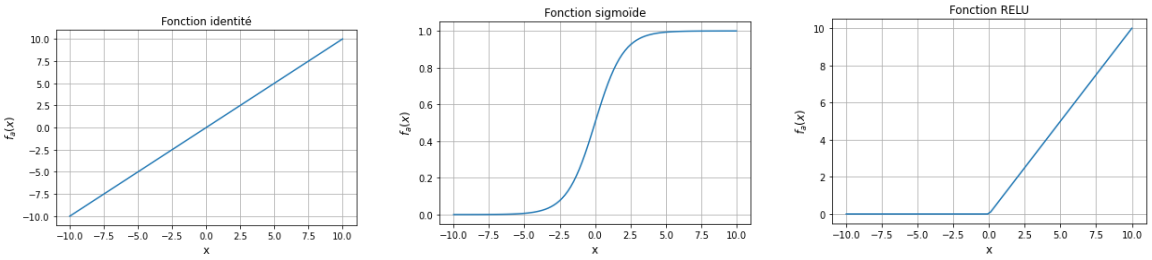
Un neurone artificiel ou formel, introduit par [8] en 1943, est une représentation mathématique et informatique d'un neurone biologique, schématisée par la figure 2.1. Il est modélisé par une fonction paramétrable qui prend en entrée un vecteur $\mathbf{x} \in \mathbb{R}^n$ et calcule une valeur $z \in \mathbb{R}$. Cette fonction est paramétrée par un vecteur de poids $\mathbf{w} \in \mathbb{R}^n$, un biais $b \in \mathbb{R}$ et une fonction d'activation $f_a(.)$. La valeur z est calculée comme suit :

$$z = f_a\left(\sum_{i=1}^{n} w_i x_i + b\right) = f_a(\mathbf{w}^\top \mathbf{x} + b) = f_a(\eta) \tag{2.1}$$

2.1.2 Fonctions d'activations

Les fonctions d'activation sont essentielles dans les réseaux de neurones car elles introduisent la non-linéarité, permettant ainsi aux réseaux de modéliser des re-

lations complexes entre les variables. Chaque fonction d'activation présente des caractéristiques spécifiques et peut être plus ou moins adaptée à certaines tâches ou problèmes. Dans cette thèse nous utiliserons principalement les fonctions *Identité*, *ReLU*(*Rectified Linear Unit*), *Sigmoide* et les fonctions à bases radiales présenté plus loin dans ce document.



(a) (b) (c)

Figure 2.2: Exemples de fonctions d'activations : (a) identité, (b) sigmoïde, (c) ReLU.

Identité : La fonction *Identité* $f_a(\eta) = \eta$ est une fonction linéaire simple. Elle est souvent utilisée en sortie de réseau pour la prédiction de valeurs lorsqu'on ne souhaite pas introduire de non-linéarité.

Sigmoïde : La fonction *Sigmoïde*, définie comme $f_a(\eta) = \frac{1}{1+\exp(-\eta)}$, est utilisée pour introduire de la non-linéarité dans un réseau neuronal. Son image est définie dans l'intervalle [0, 1]. Elle est couramment utilisée pour modéliser des probabilités ou des valeurs de sortie normalisées. Cependant, la fonction sigmoïde présente le problème de la disparition du gradient, plus connu sous le nom de *vanishing gradient*. Lors de l'apprentissage, on utilise le gradient de la fonction sigmoïde pour mettre à jour les poids du réseau. Le gradient de la sigmoïde est proche de zéro lorsque la valeur d'entrée η tend vers l'infini, ce qui a pour conséquence de réduire considérablement la propagation du gradient à travers les couches du réseau. Cela peut entraîner des difficultés d'apprentissage, en particulier dans les réseaux de grande profondeur.

ReLU : La fonction *ReLU*, $f_a(\eta) = \max(0, \eta)$ est une fonction non linéaire qui permet des activations éparses. Elle est très utilisée dans les réseaux de neurones profonds en raison de son efficacité de calcul et de sa capacité à éviter le problème du *vanishing gradient*. Cependant, elle peut entraîner des neurones *morts* qui ne sont jamais activés et ne contribuent pas à l'apprentissage.

Il existe de nombreuses autres fonctions d'activation utilisées dans les réseaux de neurones, chacune avec ses avantages et ses limites [9]. Le choix de la fonction d'activation dépend en général de la nature de la tâche, des caractéristiques des données et des propriétés souhaitées du modèle. Dans certains cas, des fonctions

d'activation personnalisées peuvent également être utilisées en fonction des besoins spécifiques du problème.

2.1.3 Les réseaux de neurones

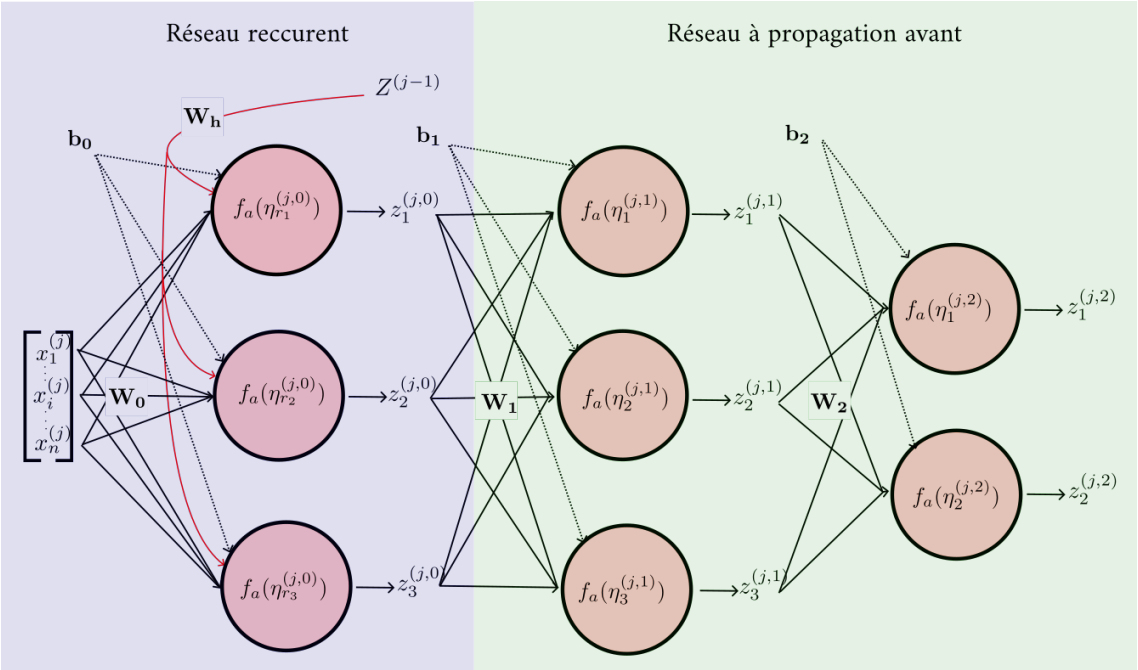


Figure 2.3: Schéma d'une combinaison de réseaux de neurones récurrents (partie bleue) et de perceptrons multicouches (partie verte). $z_{n_l}^{(j,l)} \in \mathbb{R}$ est la sortie du neurone n_l de la couche l pour la donnée $\mathbf{x^{(j)}}$ d'une séquence. $\mathbf{W_l}, \mathbf{W_h}$ sont des matrices de poids et $\mathbf{b_l}$ est un vecteur de biais.

Un réseau de neurones artificiels est composé de couches d'entrée, de sortie et de différentes couches cachées. Dans l'exemple 2.3, on peut voir que les couches cachées sont réparties en deux parties : une partie récurrente et une partie à propagation avant. D'autres types de couches de neurones peuvent être utilisés. Dans cette partie, nous allons présenter les différents types de réseaux et de couches utilisés dans cette thèse.

2.1.3.1 Réseaux de neurones à propagation avant

Les réseaux de neurones à propagation avant, plus connus sous le nom de *Feed Forward Neural Network* (*FFNN*), sont des réseaux constitués de plusieurs couches de neurones artificiels, comme schématisé dans la partie verte de la figure 2.3. Ils sont dits de *propagation avant* car il n'y a pas, entre les couches, de connexion vers *l'arrière*, ce qui signifie que l'information se propage uniquement de l'entrée

vers la sortie, sans rétroaction. Chaque neurone d'une couche reçoit des valeurs en entrée, effectue un calcul et transmet sa sortie aux neurones de la couche suivante. Un réseau de neurones formel complètement connectés est classiquement appelé un Perceptron multi-couches (*Multi-Layer Perceptron - MLP*). Ce terme n'est pas tout à fait correct car le Perceptron historique comprend une fonction d'activation de type échelon (*heaviside*) qui ne permet pas d'utiliser les méthodes d'optimisation d'aujourd'hui. Les poids et les biais du réseau sont des paramètres qui sont appris lors de l'entraînement. Il est également possible d'ajouter des couches spéciales telles que des couches de normalisation, de régularisation ou d'agrégation pour améliorer les performances du réseau et prévenir le surapprentissage.

Les *FFNN* sont capables d'apprendre des représentations hiérarchiques des données, en extrayant des caractéristiques de plus en plus abstraites à mesure que l'information se propage à travers les différentes couches. Ils sont utilisés dans de nombreux domaines pour résoudre des problèmes de classification, de régression et d'autres tâches d'apprentissage automatique. Grâce à leur capacité à apprendre des représentations complexes à partir des données, ces réseaux ont révolutionné de nombreuses applications telles que la reconnaissance vocale, la vision par ordinateur, la reconnaissance d'écriture et la génération de données. Pour les images, on utilise souvent les *CNN*, un autre type de *FFNN*. Ces réseaux ont la particularité de traiter efficacement les images, là où les *MLP* peuvent rencontrer des difficultés dues à la grande dimensionnalité des images.

Les réseaux convolutifs

Un réseau convolutif, schématisé dans la figure 2.4, introduit par [10] en 1998 est composé de neurones qui effectuent des convolutions sur les images. Une couche de convolution reçoit en entrée un tenseur de valeurs $\mathbf{X} \in \mathbb{R}^{w \times h \times c}$, où w, h et c sont des entiers et calcule en sortie $\mathbf{Z} \in \mathbb{R}^{\hat{w} \times \hat{h} \times \hat{c}}$. Le calcul de la sortie est fait grâce à plusieurs filtres de convolution $\mathbf{W} \in \mathbb{R}^{k_w \times k_h \times c}$ appelés noyaux, un biais $b \in \mathbb{R}$ et une fonction d'activation f_a. La sortie $\mathbf{Z}$ est obtenu en appliquant la fonction d'activation à un produit de convolution entre les données $\mathbf{X}$ et les poids $\mathbf{W}$, et en ajoutant un biais b :

$$\mathbf{Z} = f_a(\mathbf{W} * \mathbf{X} + b) \tag{2.2}$$

Les Auto-Encodeurs

Les *Auto-Encodeurs* (*AE*) à propagation avant (*FFNN*), schématisés dans l'illustration 2.5, ont pour objectif principal d'apprendre une représentation compacte d'une donnée. L'architecture d'un autoencodeur comprend un encodeur E et un décodeur D. Dans le cas de données vectorielles, E et D sont souvent composés de couches

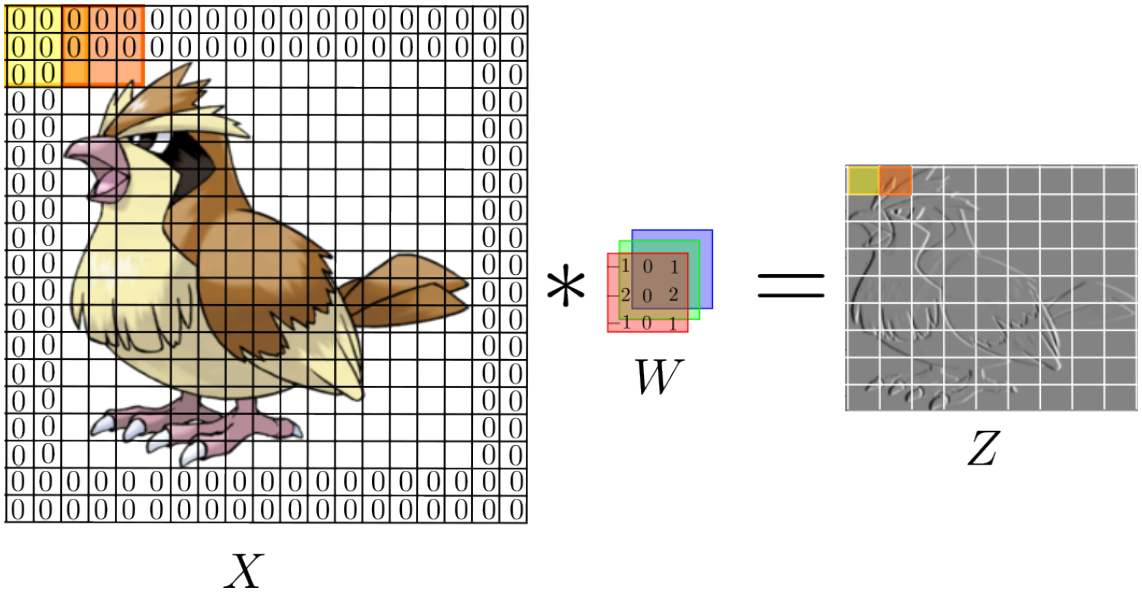


Figure 2.4: Schéma d'une convolution avec un noyau de taille 3×3, un décalage de 2, et un remplissage de 2 colonnes et lignes de zéros (*zeros padding*). Le filtre utilisé est la partie horizontale du filtre de Sobel appliqué sur les trois canaux de couleur. Une fonction *Sigmoid* et utilisé pour l'affichage de **Z**

.

de neurones simples, tandis que pour le traîtement d'images, des couches de neurones convolutionnels sont plus couramment utilisées. Dans le cas d'*AE* vectorielle, l'encoder E prend en entrée une donnée $\mathbf{x} \in \mathbb{R}^{\mathbf{n}}$ et la réduit à une représentation de dimension inférieure $\mathbf{z} \in \mathbb{R}^{\mathbf{m}}$, où $m < n$ dans le but de compresser la donnée.

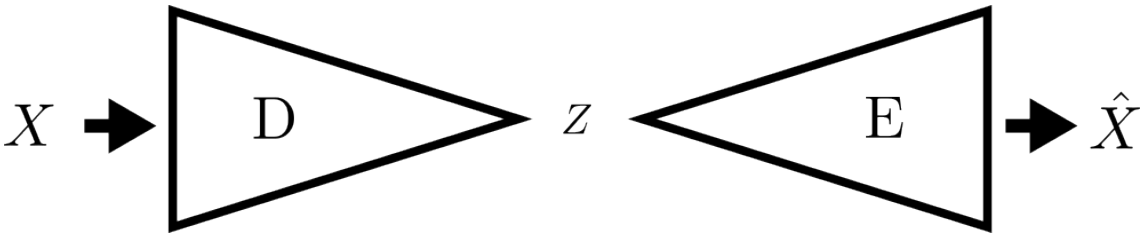


Figure 2.5: Schéma d'un *Auto-Encodeur*. E correspond à l'encodeur, D au décodeur et Z est la représentation latente.

Le décodeur D reconstruit l'entrée $\hat{\mathbf{x}} \in \mathbb{R}^{\mathbf{n}}$ à partir de cette donnée compactée $\mathbf{z}$. Cette relation est généralement formulée comme suit :

$$\hat{\mathbf{x}} = D(E(\mathbf{x})) = D(\mathbf{z}) \tag{2.3}$$

Pour obtenir une représentation compacte $\mathbf{z}$ qui capture les aspects les plus importants et les plus pertinents de l'entrée $\mathbf{x}$, les poids du modèle sont obtenus en minimisant l'erreur entre l'entrée et la sortie. Cette erreur est souvent mesurée à l'aide de l'erreur quadratique moyenne (*Mean Squarre error - MSE*) :

$$MSE(\hat{\mathbf{x}}, \mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{x_i} - x_i)^2 \tag{2.4}$$

Une fois qu'un autoencodeur a minimisé la fonction d'erreur (2.4), un espace latent est obtenu, dans lequel les données $\mathbf{x}$ sont représentées de manière compacte. Cet espace latent peut être structuré de différentes manières en fonction des objectifs. Par exemple, on peut contraindre les variables latentes à suivre une distribution particulière. L'encodeur variationnel [11] (*Variational Autoencoder - VAE*) est une variante de l'autoencodeur qui impose à l'espace latent de se rapprocher de distributions gaussiennes multivariées : $\mathbf{z} \sim \mathcal{N}(\mu, \sigma^2)$. Pour ce faire, on ajoute à l'erreur de reconstruction une composante liée à la divergence de *Kullback-Leibler* (*KL*) entre la distribution de l'espace latent $\mathbf{q}(\mathbf{z}|\mathbf{x})$ et une distribution gaussienne de référence, telle que $\mathbf{p}(\mathbf{z}) \sim \mathcal{N}(0, 1)$. La divergence *KL* mesure la dissimilarité entre deux distributions de probabilité et est formulée comme suit :

$$D_{KL}(\mathbf{q}, \mathbf{p}) = \sum_{\mathbf{x} \in X} \mathbf{p}(E(\mathbf{x})) \log \frac{\mathbf{p}(E(\mathbf{x}))}{\mathbf{q}(E(\mathbf{x}), \mathbf{x})} \tag{2.5}$$

L'ajout de cette composante *KL* à la fonction à minimiser, incite l'encodeur à établir une représentation gaussienne compacte des données d'entrée dans l'espace latent, ce qui permet une meilleure génération de données nouvelles [12].

2.1.3.2 Réseaux Neuronaux Récurrents

Les réseaux neuronaux récurrents (*Recurrent Neural Networks - RNN*) sont spécialement conçus pour traiter des données séquentielles avec des dépendances temporelles. Contrairement aux *FFNN*, les *RNN* peuvent prendre en compte les informations contextuelles précédentes lors du traitement d'une nouvelle donnée de la séquence. Cela leur permet d'effectuer des prédictions ou des classifications basées sur l'historique des données. On utilise souvent un réseau récurrent à une ou plusieurs couches combiné avec des réseaux plus traditionnels comme illustré par la partie bleue dans le schéma 2.3.

Un réseau est dit récurrent lorsqu'il comporte au moins une connexion récurrente. Par exemple le neurone récurrent illustré dans le schéma 2.6 prend en entrée $\mathbf{x}^{(j)} \in \mathbb{R}^n$,

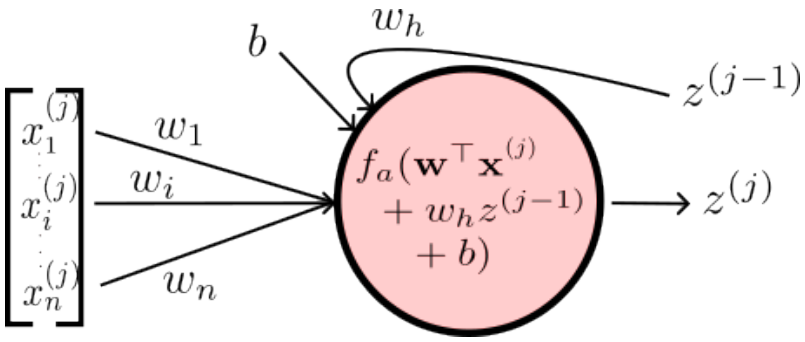


Figure 2.6: Schéma d'un neurone artificiel récurrent

la donnée j d'une séquence, $z^{(j-1)}$ la sortie de la donnée précédente à j et calcule $z^{(j)} \in \mathbb{R}$. Comme pour le neurone formel le calcul de la sortie est pondéré par des poids $\mathbf{w} \in \mathbb{R}^{\mathbf{n}}$ pour l'entrée associée à j, un poids $h_w \in \mathbb{R}$ pour la valeur $z^{(j-1)}$ et une fonction d'activation f_a. Plus le poids w_h est grand, plus le calcul de $z^{(j)}$ sera influencé par la valeur précédente $z^{(j-1)}$. La formule de calcul est la suivante :

$$z^{(j)} = f_a\left(\sum_{i=1}^{n} w_i x_i^{(j)} + h_w z^{(j-1)} + b\right) = f_a(\mathbf{w}^\top \mathbf{x}^{(j)} + h_w z^{(j-1)} + b) = f_a(\eta_r^{(j)}) \quad (2.6)$$

Un *RNN* peut être utilisé pour traiter des séquences de longueurs variables. Les *RNN* sont utilisés dans de nombreuses applications, telles que la traduction automatique, la génération de texte, la reconnaissance vocale, la modélisation de séquences, la prédiction de séries temporelles, etc. Leur capacité à capturer les dépendances séquentielles en fait des outils puissants pour le traitement de données dynamiques et l'apprentissage de modèles prédictifs basés sur des séquences. Cependant, les *RNN* peuvent être sensibles aux problèmes de disparition du gradient, qui peuvent limiter leur capacité à capturer des dépendances à long terme. Pour surmonter ces problèmes, des variantes de *RNN* telles que le *Long Short-Term Memory* (*LSTM*), [13]) ont été développées.

Long Short-Term Memory

Le réseau *LSTM* [13] est l'un des réseaux récurrents les plus utilisés pour intégrer la notion de séquence temporelle aux réseaux de neurones. Un réseau *LSTM* est composé d'une mémoire $\mathbf{c_t} \in \mathbb{R}^m, m \in \mathbb{N}$ et de trois portes : la porte d'entrée, la porte d'oubli et la porte de sortie, qui permettent la mise à jour de la mémoire. Pour une donnée temporelle $\mathbf{x_t} \in \mathbb{R}^n, n \in \mathbb{N}$, la porte d'entrée permet de stocker en mémoire une information de $\mathbf{x_t}$, la porte d'oubli permet d'effacer une information contenue en mémoire et la porte de sortie permet de réguler l'influence de la mémoire sur la sortie du neurone $\mathbf{h_t} \in \mathbb{R}^m$. Les équations des différents passages de porte représentées dans le schéma 2.7 sont les suivantes :

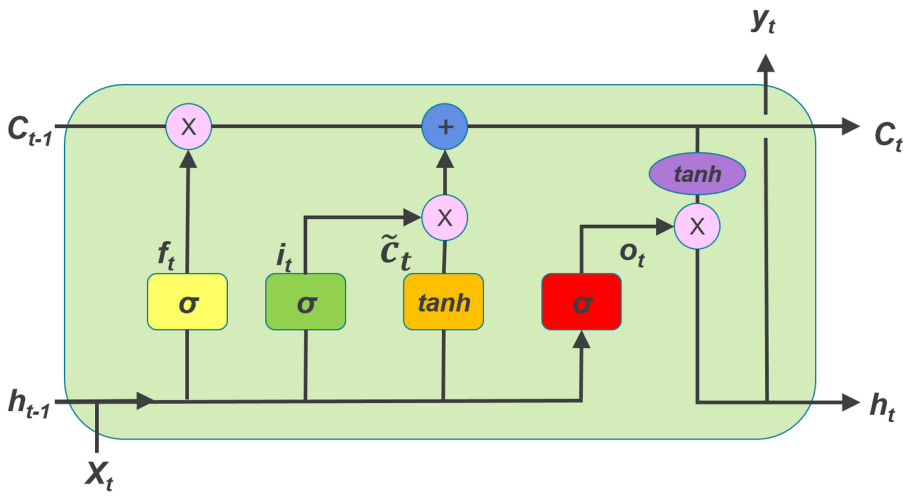


Figure 2.7: Schéma d'une couche *LSTM*

- La porte d'entrée en vert (*input*) :

$$\mathbf{i_t} = \sigma(\mathbf{W_i x_t} + \hat{\mathbf{W}}_\mathbf{i}\mathbf{h_{t-1}} + \mathbf{b_i}) \tag{2.7}$$

- La porte d'oubli en jaune (*forget*):

$$\mathbf{f_t} = \sigma(\mathbf{W_f x_t} + \hat{\mathbf{W}}_\mathbf{f}\mathbf{h_{t-1}} + \mathbf{b_f}) \tag{2.8}$$

- La porte de sortie en rouge (*output*):

$$\mathbf{o_t} = \sigma(\mathbf{W_o x_t} + \hat{\mathbf{W}}_\mathbf{o}\mathbf{h_{t-1}} + \mathbf{b_o}) \tag{2.9}$$

Les poids associés à chacune des portes $\mathbf{W_{i,f,o}}$ sont des matrices de $\mathbb{R}^{m\times n}$ et ceux liés aux entrées récurrentes $\hat{\mathbf{W}}_{\mathbf{i,f,o}} \in \mathbb{R}^{\mathbf{m}\times\mathbf{m}}$. Les $\mathbf{b_{i,f,o}} \in \mathbb{R}^{\mathbf{m}}$ sont les biais des portes. σ correspond à la fonction d'activation *Sigmoid*. La mémoire $\mathbf{c_t}$ est mise à jour en additionnant la composante d'oubli multipliée par l'ancienne mémoire à $t-1$ et la composante d'entrée multipliée par l'information à ajouter à la mémoire représenté en orange dans le schéma 2.7 et calculée comme suit :

$$\mathbf{c_t} = \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot \hat{\mathbf{c}}_\mathbf{t}, \text{ avec } \hat{\mathbf{c}}_\mathbf{t} = \tanh(\mathbf{W_c x_t} + \hat{\mathbf{W}}_\mathbf{c}\mathbf{h_{t-1}} + \mathbf{b_c}) \tag{2.10}$$

La sortie $\mathbf{h_t}$ du module *LSTM* est obtenue en multipliant la porte de sortie avec la mémoire, en violet dans le schéma 2.7 :

$$\mathbf{h_t} = \mathbf{o_t} \odot \tanh(\mathbf{c_t}) \tag{2.11}$$

Dans ces équations, $\odot$ symbolise le produit d'Hadamard, qui est la multiplication terme à terme des vecteurs. Le module *LSTM* permet de mieux gérer la mémoire à court et à long terme en ajoutant ou en oubliant des informations à chaque passage d'une donnée.

2.1.3.3 Réseaux de fonctions à base radiale

Les méthodes numériques utilisant les fonctions à base radiale (méthodes de noyau) ont été introduites par [6] et ont connu un essor dans les années 1980 pour résoudre des problèmes d'interpolation, de modélisation de données multivariables, d'approximation de fonctions multidimensionnelles et de résolution d'équations aux dérivées partielles. Inspirés par le succès de ces techniques, [14] a défini en 1988 la structure standard des réseaux de fonctions à base radiale. Des contributions majeures sur l'entraînement et la résolution de tâches de classification et de régression avec les *RBFN* (Radial Basis Function Networks) ont été apportées dans [15], [16].

Ces réseaux sont comme les *FFNN*, avec activation *sigmoide* ou *ReLU*, des approximateurs universels [17], [18]. Fondamentalement, les réseaux neuronaux de fonctions à base radiale sont similaires aux *MLP* avec une couche cachée dont chaque neurone possède une fonction d'activation à base radiale φ définie par un centre $\boldsymbol{\mu} \in \mathbb{R}^N$ de même dimension que la donnée d'entrée $\mathbf{x} \in \mathbb{R}^N$. Pour une entrée $\mathbf{x}$ la sortie z d'un neurone est calculée comme suit :

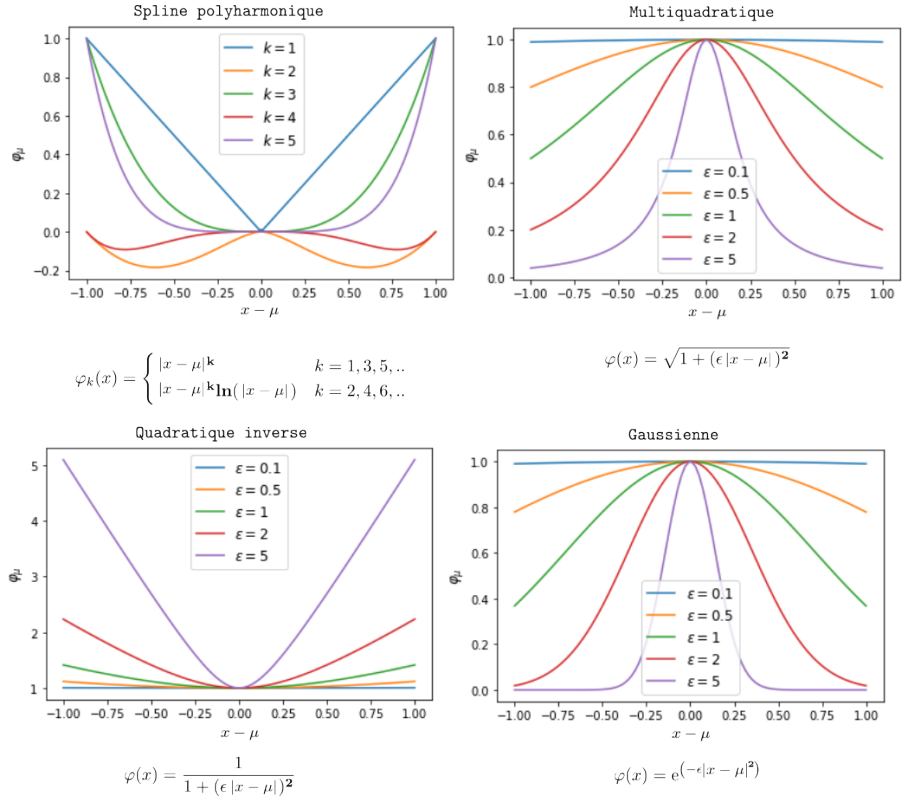
$$z(x) = \varphi(||\mathbf{x} - \boldsymbol{\mu}||) = \varphi\left(\sqrt{\sum_{i=1}^{N}(x_i - \mu_i)^2}\right) \tag{2.12}$$

Les fonctions qui ne dépendent que de la distance par rapport au centre $\boldsymbol{\mu}$ sont radialement symétriques par rapport à ce vecteur, d'où le nom de fonction à base radiale. Différents types de fonctions φ peuvent servir de noyaux. La figure 2.8 illustre certaines des plus courantes. Dans les expressions des graphiques de la figure 2.8, x est la valeur d'entrée du neurone, μ est le centre et ϵ le rayon du noyau. Dans le cas de la fonction gaussienne, $\epsilon = \frac{1}{2\sigma^2}$.

La fonction gaussienne, que nous utiliserons dans nos travaux, permet une activation locale définie par le paramètre σ. Plus la donnée est proche du centre, plus l'activation sera forte. Certains éléments du corps humain réagissent de manière similaire, comme les cellules ciliées du système auditif, qui répondent fortement à certaines bandes de fréquence, les cônes et bâtonnets dans le système visuel ou encore les cellules du cortex somatosensoriel qui ont leur réponse liée à la localisation spatiale de la stimulation physique (pression, toucher, température) [16].

L'architecture classique d'un réseau de neurones gaussiens comporte trois couches comme dans l'illustration 2.9. De l'espace d'entrée à la sortie de la couche cachée, il se produit une transformation non linéaire à travers M noyaux gaussiens. De

Figure 2.8: Différentes activations à base radiale en dimension 1



la couche cachée à la sortie, on effectue une combinaison linéaire des produits des sorties et des poids que l'on additionne à un biais $b \in \mathbb{R}$:

$$y_j = \sum_{i=1}^{M} W_{j,i}\varphi_i(\mathbf{x}) + b \tag{2.13}$$

L'utilisation d'un RBFN nécessite donc de sélectionner les centres et rayons des noyaux et d'apprendre les poids de la couche de sortie. Deux types d'approches principales ont été proposées pour l'initialisation des paramètres du réseau : les *RBFN* régularisés et les *RBFN* généralisés [19]. Les *RBFN* dit *régularisés* sont ajustés aux données fournies, avec un nombre de neurones dans la couche cachée, M, égal au nombre de données à interpoler. Les seuls paramètres inconnus à estimer sont les poids de la couche de sortie, car chaque noyau gaussien a pour centre un vecteur $\boldsymbol{\mu}$ de l'espace d'entrée. Cette approche n'est pas adaptée à des bases de données de grande dimension pour lesquelles le coût en calcul deviendrait prohibitif. De plus, bien qu'il produise des solutions optimales pour des entrées connues, il est surentraîné sur ces dernières et ne permet pas une bonne généralisation. Pour ces raisons l'approche la plus courante consiste à utiliser des *RBF généralisés*, pour lesquels le nombre de neurones de la couche cachée est inférieur à la taille de l'ensemble de données et qui nécessitent ainsi moins de ressources de calcul. Cependant, dans ce cas, il

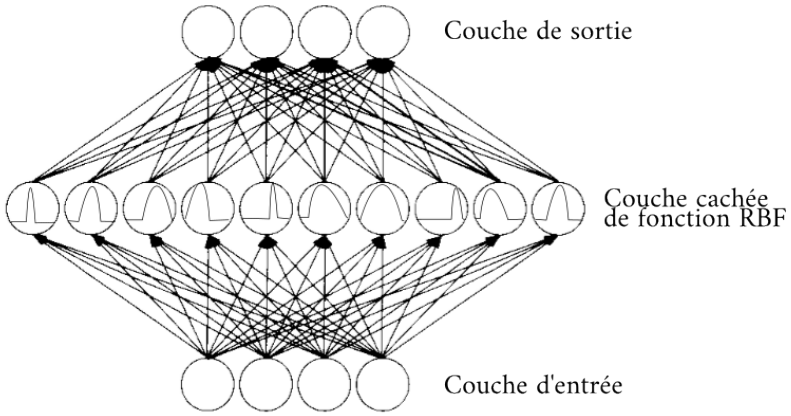


Figure 2.9: Schéma d'un réseau de neurones gaussiens à base radiale issu de [4]

est nécessaire de déterminer les centres $\boldsymbol{\mu}$ dans l'espace d'entrée $\mathbf{X}$ et d'ajuster les rayons $\boldsymbol{\sigma}$ de manière à obtenir un ensemble de noyaux permettant d'inclure et de représenter l'ensemble des données.

Les paramètres des *RBFN* sont souvent obtenus en deux étapes principales :

1. Choix des centres ($\boldsymbol{\mu}$) et largeurs ($\boldsymbol{\sigma}$) de manière non supervisée : Les méthodes couramment utilisées incluent l'initialisation aléatoire et les techniques de clustering telles que *k-means* [16], [20]. Les méthodes de clustering permettent de déterminer les centres des neurones gaussiens adaptés à un type de données spécifique. On trouve d'autres méthodes moins classiques dans la littérature, telles que l'utilisation d'algorithmes de moyenne floue (*fuzzy mean algorithms*) [21].

2. Entraînement des poids (W) : Une fois les centres et les largeurs fixés, les poids de la couche de sortie sont entraînés. Les méthodes couramment utilisées pour cela incluent la régression des moindres carrés (*least-square*) pour une estimation directe et la descente de gradient stochastique (*stochastic gradient descent - SGD*) pour un ajustement plus fin des poids. Des améliorations de ces méthodes ont été fournies au fil des années, comme la descente de gradient restreinte [22].

Une revue des différentes méthodes d'apprentissage des poids et paramètres des *RBFN* et des applications liées à ce type de réseaux est disponible dans [23].

RBFN pour les images

Une des difficultés des RBFN est liée au passage à l'échelle pour des données de grande dimension telles que les images. Il est intéressant de noter que dès 1987, à l'époque où les RBFN sont formulés, Poggio [19] mentionne un type de modèle à

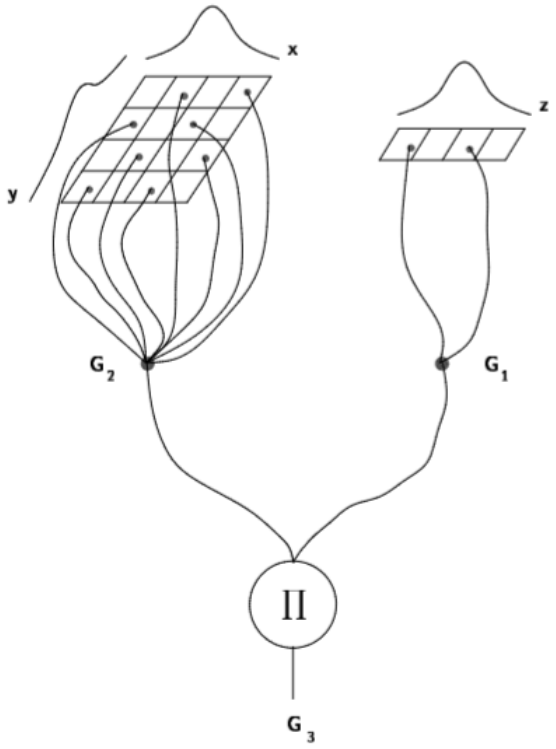


Figure 2.10: Le schéma extrait de [19] illustre la mise en œuvre d'une gaussienne radiale en multipliant des champs récepteurs gaussiens en deux dimensions et en une dimension. Ces deux dernières fonctions sont synthétisées directement par des connexions pondérées à partir des pixels d'une image. Il y a une transformation de la position implicite des pixels et de leur intensité en une valeur.

base radiale pour le traitement d'images, présenté comme une hypothèse dans la section "Gaussian GRBFs and science-fiction neurobiology". Il montre dans la figure 2.10 comment ces réseaux pourraient fonctionner. Il explique que les fonctions de base radiale, comme la gaussienne, peuvent être utilisées pour créer des champs récepteurs sur une image en reliant les pixels de manière pondérée, un peu comme les filtres de convolution utilisés pour détecter des caractéristiques dans une image. Cela simplifierait la représentation des caractéristiques de l'image en deux dimensions. Le schéma 2.10 montre comment un type particulier de représentation pourrait être créé en multipliant des champs récepteurs gaussiens en deux dimensions (agissant sur l'espace spatial de l'image) et en une dimension (agissant sur l'intensité des pixels), fabriqués en reliant les pixels de l'image de différentes manières. Ce schéma explique comment les gaussiennes pourraient transformer la position des éléments dans une image en un nombre, fournissant ainsi une représentation simplifiée de l'image.

Nous verrons dans la suite que le modèle étudié dans cette thèse est proche de cette idée.

2.1.3.4 Modèles de fondation pour le prétraitement des données visuelles

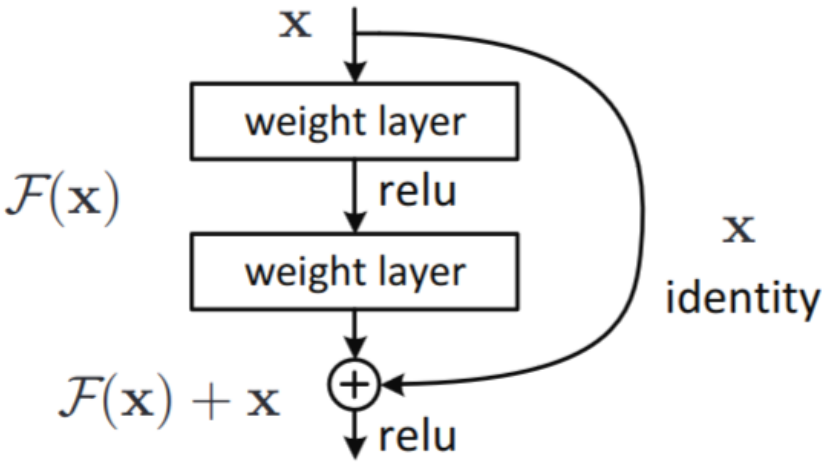


Figure 2.11: Ce schéma illustre un bloc résiduel, élément fondamental des réseaux de neurones profonds utilisés dans les architectures de type *ResNet* (Residual Networks).

Le prétraitement des données visuelles et l'extraction de caractéristiques sont essentiels pour les performances des systèmes de vision par ordinateur. Les modèles de fondation sont entraînés sur de grandes bases de données (visuelles pour ce qui nous concerne) afin de fournir des représentations pertinentes le plus générique possible. Les principaux modèles utilisés à cette fin et considérés aujourd'hui comme des modèles de fondation incluent les *CNN*et les *Vision Transformers* (*ViT*). Les *CNN*s, tels qu'*AlexNet* [24], *VGGNet* [25] et *Inception* [26], utilisent des couches convolutives pour extraire des motifs locaux à différentes échelles. Ces architectures ont montré une grande efficacité pour l'extraction de caractéristiques visuelles. Le *ResNet* [27], utilisé dans nos travaux introduit des connexions résiduelles, présentées dans l'illustration 2.11, permettant l'entraînement de réseaux très profonds sans dégradation des performances, grâce à des blocs résiduels qui facilitent la propagation des gradients lors de l'entrainement et permettent de capturer des caractéristiques plus abstraites. Le bloc résiduel prend une entrée x et applique une série de transformations $F(.)$, chacune suivie d'une activation ReLU. Le résultat de ces transformations $F(x)$ est ensuite additionné à l'entrée d'origine x grâce à une connexion identitaire (*skip connection*). L'addition est suivie d'une autre activation ReLU, produisant ainsi la sortie du bloc résiduel. *DenseNet* [28] utilise des connexions denses entre toutes les couches, améliorant l'efficacité de l'apprentissage en réutilisant les caractéristiques extraites. Cela permet une meilleure exploitation des informations avec moins de paramètres. Les *ViT* [29] appliquent les mécanismes d'attention des *Transformers* aux images en les découpant en patches. Les mécanismes d'attention multi-têtes capturent les relations globales, offrant des caractéristiques globales pertinentes. Le prétraitement des images inclut le redimensionnement, la normalisation et l'augmentation des données. L'extraction de caractéristiques se fait par propagation avant à travers les

couches du modèle, utilisant souvent des couches d'agrégation (pooling) pour réduire la dimensionnalité tout en conservant les informations essentielles. Ces modèles de fondation sont utilisés dans diverses applications telles que la classification d'images, la détection d'objets et la segmentation sémantique et plus récemment l'apprentissage par renforcement. Ils permettent une analyse approfondie et précise des données visuelles, contribuant ainsi aux avancées dans de nombreux domaines. Leur capacité à apprendre des représentations riches et discriminantes est cruciale pour le succès des systèmes de vision par ordinateur, et leur évolution continue promet de nouvelles avancées significatives dans le domaine.

2.1.4 Entraînement des réseaux de neurones

L'entraînement d'un réseau de neurones F_θ vise à optimiser les poids θ du réseau pour minimiser une fonction de coût ou de perte $\mathcal{L}$ (*loss* en anglais). Les méthodes d'optimisation utilisées en apprentissage automatique sont généralement dérivées de la descente de gradient. La plus courante, la descente de gradient stochastique (*SGD- Stochastic Gradient Descent*), permet de minimiser de manière itérative une fonction de coût décrite comme une somme de fonctions différentiables. En pratique, l'entraînement d'un réseau de neurones se déroule en plusieurs phases :

- Inférence : $F_\theta(X) = \hat{Y}$, où X sont les données d'entrée et $\hat{Y}$ la prédiction du réseau de neurones F paramétré par θ.
- Calcul du gradient : rétropropagation de l'erreur $\mathcal{L}(\hat{Y}, Y)$, le gradient de l'erreur est propagé dans le réseau, de la sortie vers l'entrée, grâce à la règle de composition des gradients.
- Mise à jour : La fonction de mise à jour des poids θ du réseau dépend de la méthode utilisée, elle est toujours liée à un taux d'apprentissage α. Pour la *SGD*, on aura :

$$\theta = \theta - \alpha \cdot \frac{\partial \mathcal{L}(\hat{Y}, Y)}{\partial \theta} \tag{2.14}$$

Le choix du taux d'apprentissage α est crucial, car il influence la vitesse de convergence et la qualité de l'apprentissage. Un taux d'apprentissage trop élevé peut entraîner une divergence du processus d'optimisation, tandis qu'un taux d'apprentissage trop faible peut ralentir la convergence.

En plus de la *SGD*, il existe de nombreuses autres variantes de descente de gradient. Dans cette thèse, nous utiliserons exclusivement l'algorithme *Adam* (*Adaptive Moment Estimation*) [30]. *Adam* est souvent privilégié à la *SGD* en raison de sa capacité à s'adapter dynamiquement aux différentes caractéristiques de la fonction de coût, à converger plus rapidement et à être moins sensible aux hyperparamètres.

2.2 Apprentissage par renforcement

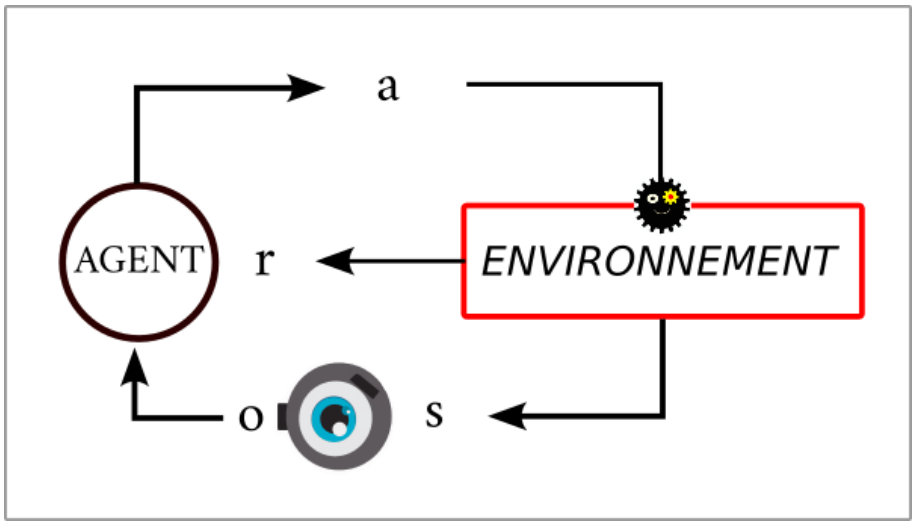


Figure 2.12: Schéma conceptualisant l'apprentissage par renforcement pour un agent muni d'un capteur visuel. L'environnement envoie un état s à l'agent qui reçoit de ses capteurs une observation o. L'agent traite cette observation afin de choisir une action. Cette action est exécutée dans l'environnement, résultant en un nouvel état et une récompense r.

L'apprentissage par renforcement s'inspire du concept de conditionnement opérant [31], qui étudie l'apprentissage basé sur les conséquences des actions. Dans ce type d'apprentissage, les actions souhaitées sont récompensées afin de favoriser leur répétition, tandis que les mauvaises actions sont souvent punies. Les études sur le comportement et l'apprentissage réalisées sur des animaux, notamment dans la cage de Skinner [32], ont enrichi les connaissances sur le conditionnement opérant des animaux.

Le concept du *RL* s'inscrit dans cette lignée et est en quelque sorte une version informatisée de ce processus. Les bases du *RL* que nous utilisons aujourd'hui ont été synthétisées en 1998 par [33]. À l'origine, le *RL* a été développé suite aux travaux de [34] en 1957 sur les processus de décision markoviens, qui formalisent les problèmes du RL. Comme illustré dans la figure 2.12, un agent observe l'état d'un environnement, choisit une action à effectuer, cette action est exécutée par l'environnement, ce qui génère un nouvel état et une récompense qui sont ensuite renvoyés à l'agent. De nos jours, les agents sont souvent modélisés par des réseaux de neurones, et l'environnement peut être un logiciel de simulation, un jeu ou encore une salle d'expérimentation avec un robot.

2.2.1 Modélisation du RL

L'objectif de l'apprentissage par renforcement est d'entrainer un agent à effectuer une tâche en maximisant sa récompense à long terme. Cela se fait à travers un processus itératif d'essais et d'erreurs. Formellement, ce processus est naïvement modélisé par un Processus de Décision Markovien (Markovian Decision Process - *MDP*), représenté par le tuple $< S, A, \mathbb{T}, R >$ où S et A représentent respectivement les ensembles d'états et d'actions. R est la fonction de récompense et $\mathbb{T}$ est la fonction de transition vers l'état s_{t+1} définie par la probabilité de transition vers s_{t+1} sachant que l'état actuel est s_t et que l'action choisie est a_t, c'est-à-dire $\mathbb{T}(s_t, a_t, s_{t+1}) = \mathbb{P}(s_{t+1}|s_t, a_t)$. Pour apprendre à un agent une tâche définie avec ce modèle, l'hypothèse markovienne selon laquelle l'état futur ne dépend que de l'état courant et de l'action doit être vraie.

Dans la réalité, un humain ou un robot ne reçoit jamais l'état réel de son environnement qui est le monde, mais seulement une observation partielle de son état obtenue via ses capteurs, et qui ne permet pas la validation de l'hypothèse markovienne. C'est pourquoi une modélisation plus générale a été introduite pour le *RL*, les Processus de Décision Markovien Partiellement Observables (Partially Observable Markov Decision Processes - *POMDP*), représentés par le tuple $< S, O, \mathbb{O}, A, \mathbb{T}, R >$. À la différence des *MDP*, ces processus prennent en compte l'observabilité d'un état. L'agent reçoit une observation $o \in \mathbb{O}$ définie par la fonction d'observation $O(s_{t+1}, a_t, o_t) = \mathbb{P}(o_t|s_{t+1}, a_t)$. Une illustration d'une trajectoire d'un *POMDP* est donnée dans la figure 2.13.

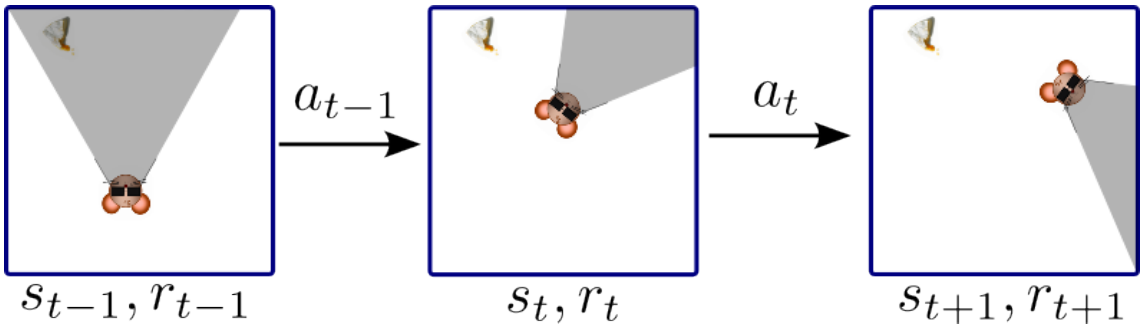


Figure 2.13: Trois itérations d'un *POMDP* dans lesquelles notre agent, Skinny la Souris, est enfermé dans une pièce carrée et a pour but de trouver sa part de camembert. La zone grise représente le récepteur visuel de Skinny lui permettant d'observer l'état.

Les différentes fonctions et variables du *POMDP* sont définies ci-dessous :

Les états

Les états $s \in S$ où S est l'ensemble des états possibles. Ils sont générés par l'environnement. Dans l'exemple de la figure 2.13, un état pourrait être représenté

par toutes les variables permettant de décrire la simulation (position du fromage, de la souris, des murs, vitesse de déplacement, de rotation, etc.).

Les observations

Les observations $o \in \mathbb{O}$ obtenues par O, une fonction sensorielle de S. Dans cette thèse, nous utiliserons uniquement un capteur visuel, une caméra RGB. Skinny dispose d'un capteur visuel qui observe la partie grise de l'environnement dans la figure 2.13. Dans la plupart des environnements visuels à la première personne, les observations sont différentes des états, on dit que l'environnement est partiellement observable. Cependant, on peut imaginer un environnement dans lequel l'observation est équivalente à l'état, et dans ce cas, l'environnement est entièrement observable, c'est-à-dire que $o = s$.

Les actions

Les actions $a \in A$ où A est l'ensemble des actions que l'agent peut effectuer. Les actions peuvent être discrètes ou continues. Par exemple, la souris peut se déplacer et tourner, donc son ensemble d'actions pourrait être $A = \{x, y, \phi\}$, avec x et y représentant les déplacements de Skinny et ϕ une rotation sur l'axe de la caméra. Un ensemble d'actions continues serait représenté par $\mathbb{R}^3$. Si l'ensemble d'actions est discret et fini, il pourrait, par exemple, être composé de toutes les combinaisons possibles de $[c_x, c_y, c_\phi]$ avec c_x, c_y, c_ϕ étant des constantes d'incrémentation prenant des valeurs dans $[+1, -1, 0]$. Dans ce cas, on aurait $|A| = 27$.

La politique

Une politique ou stratégie de l'agent, et la fonction reliant l'état à l'action, elle peut être déterministe $a = \pi(o)$ ou stochastique $a \sim \pi(\cdot|o)$. Dans le domaine du *DRL* elle est modélisée par un réseau de neurones π_θ. Les poids entraînables du réseau de neurones sont représentés par θ. La politique permet de choisir une action en fonction de l'observation de l'état effectuée par l'agent. Dans ces travaux, nous utilisons uniquement des réseaux de neurones pour modéliser la politique, nous omettrons θ lorsque cela sera possible, dans un souci de lisibilité. Dans le cas de Skinny, une bonne politique serait un réseau de neurones qui prend une image en entrée et génère l'action permettant d'aller vers le fromage.

Les récompenses

Les récompenses $r \in \mathbb{R}$ sont des scalaires obtenue après avoir effectué une action dans l'environment. On les distingue souvent en fonction de leur fréquence d'apparition : récompenses rares et récompenses denses. Dans le cas de Skinny, une récompense rare pourrait être obtenue uniquement lorsque Skinny mange le fromage, tandis qu'une récompense dense pourrait être l'inverse de la distance entre Skinny et le fromage. Les récompenses denses peuvent parfois être difficiles et coûteuses en termes de temps d'ingénierie à mettre en place, mais elles peuvent s'avérer utiles. Certaines récompenses dites intrinsèques sont calculées grâce aux données propres de l'agent et non de l'environnement.

La fonction de récompense ou retour

La fonction $R(\tau) \to \mathbb{R}$ représente la récompense à long terme que l'agent doit maximiser. Elle est calculée pour une séquence d'états et d'actions appelée trajectoire $\tau = (s_0, a_0, r_0, s_1, a_1, r_1 \ldots)$ qui a engendré une séquence de récompenses r_τ. La fonction de récompense à long terme que nous utilisons est définie comme suit : $R(\tau) = \sum_{t=0}^{T} \gamma^t r_t$, où T est appelé horizon, et γ est un terme de dévaluation qui permet de diminuer l'importance des récompenses lointaines par rapport aux récompenses à court terme. Si T est fini et $\gamma = 1$, alors R est la fonction de récompense à horizon fini non dévaluée. En revanche, lorsque T est infini ($T \to \infty$), $0 < \gamma < 1$, R est la fonction de récompense à horizon infini dévaluée. On utilise plus souvent la fonction de récompense dévaluée avec γ généralement compris entre 0,95 et 0,99. Plus γ est proche de 0, plus l'agent se focalisera sur les récompenses immédiates.

Apprentissage épisodique

L'apprentissage par renforcement peut se dérouler de manière continue ou dans le cadre d'un apprentissage épisodique. On s'intéressera ici à un cadre épisodique dans lequel lorsque l'agent atteint un état final, l'environnement sera réinitialisé et un nouvel épisode pourra débuter. Dans ce contexte du *RL* avec des environnements épisodiques, le booléen *done* (D) est une variable clé qui indique si un épisode est terminé ou non. Lorsque *done* est vrai, cela signifie que l'agent a atteint un état terminal dans l'environnement, généralement associé à une condition de fin d'épisode telle qu'atteindre un objectif ou dépasser un certain nombre d'étapes. Cette variable est cruciale pour le cycle d'apprentissage de l'agent, car elle permet de déclencher des actions telles que la mise à jour des valeurs de retour et la réinitialisation de l'environnement pour commencer un nouvel épisode. En résumé, le booléen "done" est un signal important qui guide le processus d'apprentissage de l'agent en

déterminant le moment où un épisode se termine et une nouvelle séquence d'actions doit commencer.

2.2.2 Environnements de simulation

L'entraînement complet d'un agent dans le monde réel n'est souvent pas la solution privilégiée. En effet, lors de l'apprentissage d'un agent pour accomplir une tâche dans un environnement donné, ses premières actions sont souvent aléatoires, ce qui, dans le cadre d'un apprentissage sur un robot réel, pourrait endommager certaines parties du robot. De plus, certaines tâches nécessitent plusieurs jours d'entraînement et donc de surveillance constante, ce qui rend l'entraînement dans le monde réel encore plus contraignant, sans parler de la création de l'environnement qui peut s'avérer complexe. C'est pourquoi le domaine de l'apprentissage recourt à des simulateurs pour entraîner les agents. Différents types de simulations sont utilisés dans divers contextes :

- Jeux vidéo : ils sont souvent utilisés dans le cadre théorique du RL, car ils offrent un accès rapide à des environnements virtuels. De nombreux jeux ont déjà été adaptés pour être utilisés dans le *RL*, tels que Atari [35], Doom [36], Minecraft [37] et Gym [38]. L'utilisation de jeux vidéo permet d'avoir une base d'environnements variés, ce qui facilite les comparaisons entre les algorithmes.

- Environnements réalistes : ce type de simulateur prend en compte la physique du monde réel et permet l'intégration de textures photoréalistes. Ces simulations sont souvent utilisées dans le contexte du transfert de connaissances vers des robots. Cependant, la création de tels environnements peut être complexe et demande des ressources importantes. Les comparaisons dans ce domaine se basent généralement sur l'efficacité des méthodes de transfert et la capacité des robots à s'adapter à des situations réelles.

L'idéal pour tester nos algorithmes serait de créer un environnement réel avec un robot équipé de caméras afin de résoudre diverses tâches, telles que l'apprentissage du déplacement ou de la saisie d'objets. Cependant, l'application directe du *RL* sur un agent réel est très compliquée pour les raisons citées plus haut. C'est pourquoi, comme la plupart des travaux dans ce domaine, nous avons décidé d'utiliser un simulateur afin d'étudier et de comparer les capacités d'apprentissage de notre méthode. Il existe différents types de simulateurs d'environnements à la première personne. Les plus connus lorsque mes travaux ont commencé étaient *DeepMind Lab* [39], *ViZDoom* [36], et *Malmo* [37]. Plus tard, d'autres environnements d'étude ont été développés, comme le projet *Habitat* apparu en 2019, qui continue d'être développé [40], ainsi que la version *OpenAI* de Minecraft [41]. Le choix de l'environnement s'est porté assez

rapidement sur *ViZDoom*, qui permet l'apprentissage de tâches visuelles requérant différentes compétences, telles que la précision, l'exploration et la collecte. De plus, *ViZDoom* est la plateforme de simulation pour le *RL* qui comptait le plus de citations et une communauté très active, avec des compétitions d'*IA* portées par la nostalgie du vieux jeu Doom sur lequel est basé le simulateur. En outre, nous avons utilisé le simulateur *PyBullet*, une implémentation *Python* open-source de simulation basée sur le simulateur physique *BulletPhysics* [42]. *PyBullet* est un moteur de physique qui permet de simuler des scènes réalistes pour l'apprentissage de robots. Une comparaison des simulateurs physiques peut être trouvée dans [43]. L'utilisation de ces simulateurs offre un environnement sûr, flexible et efficace pour l'apprentissage par renforcement.

2.2.3 L'optimisation en RL

En *RL*, l'objectif est de trouver une politique optimale π^* qui maximise l'espérance de la fonction de récompense $J(\pi) = \mathbb{E}_{\tau\sim\pi}[R(\tau)]$:

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{2.15}$$

2.2.3.1 Les fonctions du RL

Certaines fonctions ont été introduites dans le cadre du *RL* pour optimiser la politique. Ces fonctions sont les suivantes :

La fonction $Q : S \times A \to \mathbb{R}$: elle évalue la valeur d'une action dans un état pour une politique donnée :

$$Q^\pi(s,a) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a]. \tag{2.16}$$

La fonction $V : S \to \mathbb{R}$: elle évalue la valeur d'un état pour une politique donnée :

$$V^\pi(s) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s]. \tag{2.17}$$

On peut également l'exprimer en fonction de la fonction Q :

$$V^\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi}[Q^\pi(s,a)]. \tag{2.18}$$

Après avoir défini ces deux fonctions, on peut introduire les équations de Bellman [34] qui régissent ces fonctions :

$$V^\pi(s_t) = \mathop{\mathbb{E}}_{\tau\sim\pi}[r_t + \gamma V^\pi(s_{t+1})], \tag{2.19}$$

$$Q^\pi(s_t,a_t) = \mathop{\mathbb{E}}_{\tau\sim\pi}[r_t + \gamma \mathop{\mathbb{E}}_{a_{t+1}\sim\pi} Q^\pi(s_{t+1},a_{t+1})]. \tag{2.20}$$

Pour une politique optimale π^* on a l'égalité suivante :

$$V^{\pi^*}(s) = \max_a Q^{\pi^*}(s, a). \tag{2.21}$$

Ces équations montrent que les valeurs Q et V à un instant t dépendent de la récompense instantanée ainsi que des valeurs de tous les prochains états que l'on peut atteindre en suivant la politique π. Grâce à ces fonctions, nous pouvons également définir la fonction d'avantage, qui mesure l'avantage moyen de choisir une action par rapport aux autres :

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \tag{2.22}$$

Si $A > 0$, cela signifie que l'action a_t est susceptible d'apporter une récompense future plus élevée que les autres actions, et inversement, si $A < 0$, cela signifie qu'une autre action a'_t pourrait permettre d'obtenir une récompense plus élevée à long terme.

2.2.3.2 Exploration vs Exploitation

En *RL*, l'exploration et l'exploitation sont deux concepts fondamentaux qui déterminent la manière dont un agent interagit avec son environnement pour maximiser sa récompense cumulée. L'exploitation consiste à choisir les actions qui semblent être les meilleures en fonction des connaissances actuelles de l'agent, tandis que l'exploration consiste à essayer de nouvelles actions pour découvrir de meilleures stratégies potentielles. Un exemple courant de compromis entre exploration et exploitation est l'algorithme *epsilon-greedy*. Avec cet algorithme, l'agent choisit l'action optimale la plupart du temps (exploitation), mais avec une probabilité epsilon, il explore en choisissant une action aléatoire. Cette probabilité epsilon décroit au cours des épisode. Un autre exemple est l'approche purement gloutonne (*greedy*), où l'agent choisit toujours l'action qui semble être la meilleure selon ses connaissances actuelles, privilégiant ainsi l'exploitation au détriment de l'exploration. Enfin, l'exploration purement aléatoire consiste à choisir des actions de manière entièrement aléatoire, sans tenir compte des connaissances de l'agent. Ces différentes approches permettent de trouver un équilibre entre l'exploration nécessaire pour découvrir de nouvelles opportunités et l'exploitation des connaissances acquises pour maximiser les récompenses à long terme.

2.2.4 Les algorithmes de DRL

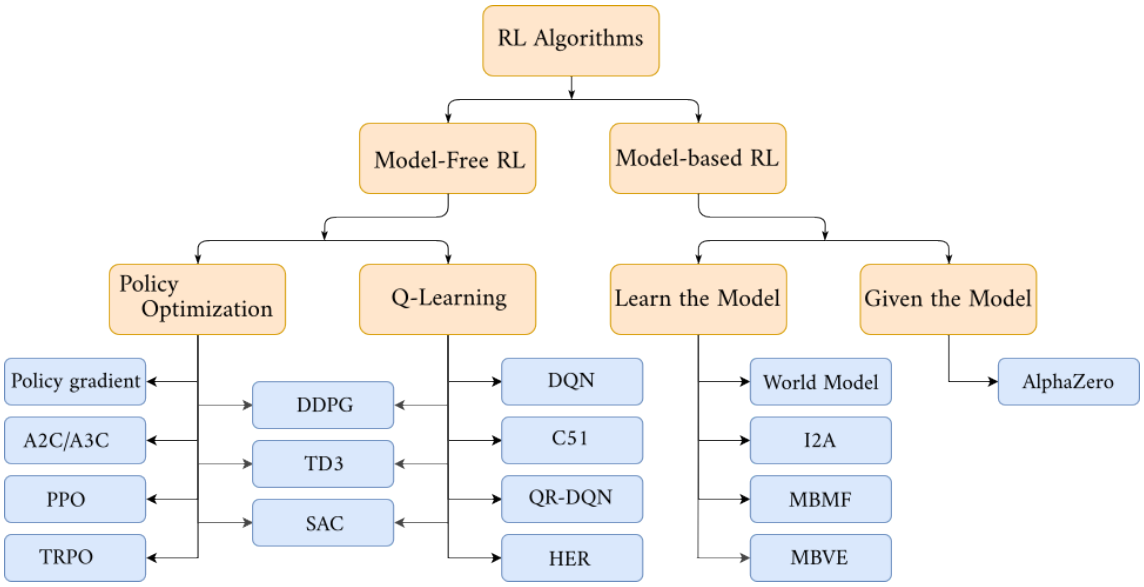


Figure 2.14: Taxonomie des méthodes les plus courantes dans l'état de l'art du *RL*

Les algorithmes de *RL* diffèrent dans leur approche de résolution du problème d'optimisation, que ce soit en termes de données ou de méthodes. Une taxonomie des méthodes issue de *OpenAI* 2.14 permet d'illustrer l'arrangement de ces méthodes. Les algorithmes d'apprentissage par renforcement peuvent être classés en deux catégories : les algorithmes *model-free* et les algorithmes *model-based*. Dans les algorithmes *model-based*, l'agent utilise un modèle des fonctions de récompense et de transition (qui peut être appris). Une fois ces fonctions connues, le problème d'optimisation devient plus simple. Dans les algorithmes *model-free*, l'agent apprend directement à optimiser la récompense. Il est difficile d'apprendre un modèle de l'environnement dans un espace de grande dimension, comme les images c'est pourquoi dans cette thèse, nous nous concentrerons sur les approches *model-free*.

Pour optimiser la politique, on peut distinguer deux catégories d'algorithmes : ceux qui estiment directement la politique et ceux qui estiment plutôt des fonctions de valeurs Q,V,A. De plus, la méthode d'optimisation peut aussi être *on-policy* ou *off-policy*. Ces méthodes définissent la façon dont les données d'apprentissage sont collectées et utilisées. Dans les méthodes *on-policy*, les données sont collectées en suivant la politique actuelle de l'agent, ce qui signifie qu'elles sont utilisées pour mettre à jour cette même politique. En revanche, dans les méthodes *off-policy*, les données peuvent provenir de n'importe quelle politique, ce qui permet une utilisation plus flexible des données d'apprentissage pour mettre à jour la politique de l'agent. Dans la suite nous présentons brièvement les principales catégories d'approches de *DRL*, puis nous détaillerons celles qui ont été utilisées dans cette thèse.

2.2.4.1 Algorithmes basés sur l'estimation de la politique

Ces algorithmes, également connus sous le nom d'algorithmes *policy-based*, cherchent à apprendre directement une politique qui associe les états aux actions. L'idée est d'optimiser la politique π afin de maximiser la récompense cumulée.

Policy Gradient C'est une méthode courante où la politique est paramétrée par un ensemble de paramètres θ. L'agent utilise une approche gradient ascent pour ajuster θ afin de maximiser l'espérance de la récompense future. Un algorithme spécifique est le *REINFORCE*, qui met à jour les paramètres de la politique en utilisant les gradients des récompenses obtenues.

Proximal Policy Optimization (*PPO*) Il s'agit d'un autre algorithme populaire qui utilise une approche basée sur les gradients. Il améliore la stabilité et la performance de la mise à jour de la politique en limitant l'ampleur des changements de la politique entre les étapes d'apprentissage.

2.2.4.2 Algorithmes basés sur l'estimation des fonctions de valeurs

Ces algorithmes, souvent appelés *value-based*, se concentrent sur l'estimation des fonctions de valeur, telles que la fonction de valeur d'action $Q(s, a)$, la fonction de valeur $V(s)$ ou la fonction d'avantage $A(s, a)$. L'objectif est de trouver une politique optimale en utilisant ces fonctions.

Q-learning C'est un algorithme où l'agent apprend une fonction de valeur d'action $Q(s, a)$ qui représente l'espérance de la récompense cumulée en suivant la politique optimale à partir de l'état s en prenant l'action a. L'agent met à jour les valeurs Q en utilisant l'équation de Bellman :

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$$

où α est le taux d'apprentissage, r la récompense, γ le facteur de dévaluation, et s' l'état suivant.

Deep Q-Network (*DQN*) Le *DQN* est une extension du Q-learning qui utilise des réseaux de neurones profonds pour approximer la fonction Q. Cette approche permet de gérer des espaces d'états de grande dimension, comme les images, en utilisant un réseau de neurones convolutifs pour traiter les entrées visuelles.

En résumé, les algorithmes basés sur l'estimation de la politique optimisent directement la politique pour maximiser les récompenses, tandis que les algorithmes basés

sur l'estimation des fonctions de valeurs se concentrent sur l'apprentissage des valeurs d'état ou d'action pour dériver une politique optimale. Les exemples de Policy Gradient et *PPO* illustrent des méthodes *policy-based*, tandis que le Q-learning et *DQN* illustrent des méthodes *value-based*.

2.2.4.3 Algorithmes hybrides (acteur-critique)

Les algorithmes hybrides combinent des éléments des approches *policy-based* et *value-based* pour tirer parti des avantages de chaque méthode. Ces algorithmes cherchent à améliorer l'efficacité et la stabilité de l'apprentissage en utilisant à la fois des politiques et des fonctions de valeurs.

Asynchronous Advantage Actor-critique (*A3C*) L'*A3C* est un algorithme hybride qui combine les approches *policy-based* et *value-based*. Dans cet algorithme, plusieurs agents travaillent en parallèle et de manière asynchrone pour explorer l'environnement et collecter des expériences. Chaque agent apprend une politique (l'acteur) et une fonction de valeur (le critique). La fonction d'avantage $A(s, a)$ est utilisée pour réduire la variance de l'estimation du gradient de la politique. L'acteur met à jour la politique en utilisant les gradients basés sur l'avantage, tandis que le critique met à jour la fonction de valeur pour estimer les récompenses futures. La mise à jour asynchrone permet de stabiliser l'apprentissage et d'améliorer la convergence.

Deep Deterministic Policy Gradient (*DDPG*) Le *DDPG* est un autre algorithme hybride qui combine des éléments du Q-learning et des Policy Gradients. Il est particulièrement adapté pour les environnements avec des espaces d'actions continus. Le *DDPG* utilise un réseau d'acteur pour déterminer les actions et un réseau de critique pour évaluer les actions en fonction de la fonction Q. Le réseau de critique est mis à jour en utilisant l'équation de Bellman, tandis que le réseau d'acteur est mis à jour en utilisant les gradients de la politique déterministe. Pour stabiliser l'apprentissage, le *DDPG* utilise des techniques telles que les réseaux cibles et la mémoire de répétition.

Soft Actor-critique (*SAC*) Le *SAC* est un algorithme hybride qui maximise une version entropique de la récompense cumulée, ce qui favorise des politiques plus exploratrices. Comme le *DDPG*, le *SAC* utilise des réseaux d'acteur et de critique. Cependant, il incorpore une régularisation entropique dans la fonction de valeur, ce qui encourage l'agent à explorer plus de possibilités d'actions. Cela permet de trouver des politiques plus robustes et stables, en particulier dans des environnements complexes.

En résumé, les algorithmes hybrides comme *A3C*, *DDPG* et *SAC* combinent les approches *policy-based* et *value-based* pour optimiser les performances et la stabilité de l'apprentissage. Ces méthodes utilisent à la fois des politiques et des fonctions de valeurs pour tirer parti des avantages de chaque approche et surmonter leurs limitations respectives.

2.2.4.4 Choix de l'algorithme

Table 2.1: Tableaux des méthodes d'apprentissage de bout en bout par renforcement. HD correspond aux observations de grande dimension (image) et LD à celles de petite dimension (vecteur d'état). Les environnements de ce tableau sont Atari (At), MuJoCo (MJC), Torcs (To) et Deepmind Lab (DmL). Pour plus de clarté, nous avons abrégé les titres des colonnes : algo correspond à algorithmes, obs à observation et Env à environnement.

années	papier	algo	Obs	Env	citations	continue
2015	[1]	DQN	HD	At	28202	non
2015	[44]	DDPG	HD,LD	MJC,To	14986	oui
2016	[45]	A3C	HD,LD	At,To,DmL	10486	oui
2017	[46]	PPO	HD	At,MJC	16034	oui
2018	[47]	Rainbow DQN	HD	At	2049	non
2018	[48]	SAC	LD	MJC	7358	oui

Le tableau 2.1 regroupe les algorithmes les plus utilisés dans le domaine de l'apprentissage par renforcement visuel. Le choix des algorithmes utilisés dans ces travaux a été basé sur la renommée, les environnements testés et les résultats des entraînements. De plus, nous avons voulu utiliser deux types d'optimisation, *on-policy* et *off-policy*. Le *DQN* étant l'algorithme off-policy le plus utilisé dans la recherche sur le *RL*, nous avons décidé de l'inclure dans notre étude. Nous n'avons pas considéré l'amélioration *Rainbow* du *DQN* [47] car cela aurait engendré des temps d'entraînement plus longs et un ajout d'hyperparamètres. Pour le choix de l'algorithme *on-policy*, nous avons opté pour *PPO* en raison de sa notoriété et de ses résultats supérieurs aux autres algorithmes pour les scénarios de *VizDoom* [49]. De plus, dans le cas de scénarios continus, *PPO* est le plus performant [50].

2.2.4.5 Deep Q-learning

Le *Deep Q-learning* [1] est un algorithme d'apprentissage *off policy* pour les environnement avec des espaces d'actions discrets. On dit que c'est un algorithme d'itération sur les valeurs, son objectif est d'approximer la fonction Q^* qui représente la valeur maximale pour chaque paire état-action. On utilise un réseau de neurones paramétré par θ pour approximer cette fonction.

La politique optimale $\pi^*(s)$ est définie comme l'action ayant la plus grande valeur $Q*$:

$$\pi^*(s) = \underset{a}{\text{argmax}} Q^*(s, a). \tag{2.23}$$

Pour cela, nous utilisons une version optimale de l'équation de Bellman :

$$Q^*(s_t, a_t) = \underset{s_{t+1} \sim \mathbb{T}}{\mathbb{E}} \left[r + \gamma \underset{a_{t+1}}{\text{max}} Q^*(s_{t+1}, a_{t+1}) \right], \tag{2.24}$$

où r est la récompense, s_{t+1} est l'état suivant, $\mathbb{T}$ est la probabilité de transition, et γ est le facteur de réduction.

Nous cherchons à minimiser la différence entre notre estimateur Q^θ et Q^* en utilisant l'erreur quadratique moyenne (*MSE*) :

$$J(\theta) = \mathbb{E}\left[(Q^\theta - Q^*)^2 \right]. \tag{2.25}$$

Pour atteindre cet objectif, nous minimisons cette erreur sur m lots de transitions $< s_t, a_t, r_t, s_{t+1}, D_t >$, obtenus dans l'environnement en suivant différents types de politiques (gloutonne, ϵ-gloutonne, aléatoire). Ainsi, nous avons :

$$J(\theta) = \frac{1}{m} \sum_{t=0}^{m} \left(Q^\theta(s_t, a_t) - (r_t + \gamma \underset{a_{t+1}}{\text{max}} Q^*(s_{t+1}, a_{t+1})) \right)^2. \tag{2.26}$$

Le problème ici réside dans l'approximation de la valeur Q^*. En pratique, il est impossible de la précalculer lorsque l'espace d'observation est vaste, ce qui est le cas avec des images. Une technique appelée *bootstrapping* consiste à optimiser la valeur $Q^\theta(s, a)$ avec sa propre estimation $Q^\theta(s', a')$:

$$J(\theta) = \frac{1}{m} \sum_{\tau=0}^{m} \left(Q^\theta(s_t, a_t) - (r_t + \gamma \underset{a_{t+1}}{\text{max}} Q^\theta(s_{t+1}, a_{t+1})) \right)^2. \tag{2.27}$$

Cette approche nécessite une bonne approximation de Q^*, ce qui n'est pas le cas au début de l'apprentissage. Pour remédier à ce problème, il est nécessaire de commencer l'apprentissage en *RL* lorsque l'agent a exploré suffisamment de trajectoires différentes pour obtenir une bonne estimation de Q^*. De plus, après chaque mise à jour, l'approximation de Q^* peut changer dans une mauvaise direction, ce qui peut entraîner une instabilité et empêcher la convergence. Pour stabiliser l'entraînement, on utilise un réseau cible avec des paramètres ϕ, qui est initialement identique à θ, et qui est mis à jour régulièrement (par exemple, $\phi \leftarrow \theta$ à intervalles réguliers t_ϕ). Ainsi, nous avons maintenant :

$$J(\theta) = \frac{1}{m_\tau} \sum_{\tau=0}^{m_\tau} \left(Q^\theta(s_\tau, a_\tau) - (r_\tau + \gamma \max_{a'} Q^\phi(s'_\tau, a')) \right)^2 . \tag{2.28}$$

L'idée est d'évaluer la politique actuelle de l'agent par rapport à sa version précédente afin de réduire cette différence et maximiser la récompense.

Algorithm 1 Deep Q-learning

Input : environnement (env), agent (Q^θ), réseaux cible ($Q^\phi, \phi = \theta$), taux d'apprentissage (α), mémoire tampon (Rb), MaxEpisode, taille du batch (b), intervalle de mise à jour du réseau cible (T), le facteur de réduction (γ)
Out : Q^θ Episode = 0
while Episode<MaxEpisode **do**
    s = env.init
    while 1 **do**
        action aléatoire a avec une probabilité ϵ sinon $a = \underset{a}{argmax} Q^\theta(s, a)$
        s', r, D = env(a)
        Rb $\leftarrow \tau =< o, a, o', r, D >$
        if $D == True$ (episode terminal) **then**
            break
        else
            $o = o'$
        end if
        if Rb.size $> b$ **then**
            b_t = Rb.sample(b)
            $target = r_t + D_t \times \gamma \times \underset{a_{t+1}}{max} Q^\phi(s_{t+1}, a_{t+1})$
            $J_\theta = MSE(Q^\theta(s_t, a_t), target)$
            $\theta = \theta - \alpha \frac{\partial J_\theta}{\partial \theta}$
            à chaque T itération : $\phi = \theta$
        end if
    end while Episode += 1
end while

Le pseudo-code de l'algorithme 1 décrit les étapes principales de l'algorithme *Deep Q-Learning*, où les actions sont choisies en utilisant la politique définie par les valeurs Q, et les paramètres du réseau de neurones sont mis à jour par descente de gradient pour minimiser la *MSE* entre les valeurs Q estimées et les valeurs Q optimales.

2.2.4.6 Proximal Policy Optimization (PPO)

Le Proximal Policy Optimization (*PPO*) [46] est un algorithme d'apprentissage *on-policy* visant à optimiser une approximation du gradient de la politique, qui se concentre sur l'itération de la politique. Contrairement au *Q-learning*, où l'optimisation repose sur une politique déterministe $\pi(s)^* = \underset{a}{argmax} Q^*(s,a)$, le *PPO* adopte une approche avec une politique stochastique paramétrée par θ, permettant l'échantillonnage d'actions selon une distribution de probabilité $\pi_\theta(a|s) = P(a|s;\theta)$.

Dans ce type d'algorithmes, l'agent commence par collecter m trajectoires $\tau = (s_0, a_0, r_0, D_0, s_1, a_1, r_1, D_1, ..., s_m)$ de données en utilisant la politique actuelle π_θ; c'est l'évaluation de la politique. Ensuite, il utilise ces données pour mettre à jour les poids θ; c'est l'amélioration de la politique. Les trajectoires, influencées par la politique, ont une probabilité $P(\tau)$ et une récompense $R(\tau)$. *PPO* a pour but de maximiser l'espérance de la récompense totale en suivant π_θ :

$$\underset{\theta}{max} J(\theta) = \underset{\theta}{max} \underset{\tau\sim\pi_\theta}{\mathbb{E}} [R(\tau)] = \underset{\theta}{max} \sum_\tau P(\tau;\theta) R(\tau) \tag{2.29}$$

Pour ce faire, *PPO* s'inspire de l'algorithme de *RL*, *REINFORCE* [51], qui utilise le théorème du gradient de la politique. Pour calculer la probabilité des trajectoires, $P(\tau;\theta)$, on pourrait, dans un cadre totalement déterministe, déterminer chaque probabilité d'obtenir une trajectoire en suivant une politique π_θ. Cependant, en *RL* visuel, la plupart des environnements admettent une part de stochasticité, et cette probabilité de trajectoire doit être approximée de façon stochastique. Afin de maximiser $J(\theta)$, nous allons calculer son gradient par rapport aux poids θ, $\nabla_\theta J(\theta)$, en utilisant l'astuce de la dérivée du logarithme :

$$\begin{aligned} \nabla_\theta J(\theta) &= \textstyle\sum_\tau \nabla_\theta P(\tau;\theta) R(\tau) \\ &= \textstyle\sum_\tau \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta) R(\tau) \\ &= \textstyle\sum_\tau P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) R(\tau) \\ &= \underset{\tau\sim\pi_\theta}{\mathbb{E}} \nabla_\theta \log P(\tau;\theta) R(\tau) \end{aligned} \tag{2.30}$$

Cette expression rend la dérivation plus facile, en effet $P(\tau;\theta)$ peut être réecris sous la forme $P(\tau;\theta) = \prod_{t=0}^{H} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t)$ or le logarithme est additif et

$P(s_{t+1}|s_t, a_t)$ ne dépends pas de θ, on a donc :

$$\begin{aligned} \nabla_\theta \log P(\tau;\theta) &= \nabla_\theta \log \left[\prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t)\right] \\ &= \nabla_\theta \left[\sum_{t=0}^{T} \log P(s_{t+1}|s_t, a_t) + \sum_{t=0}^{T} \log \pi(a_t|s_t)\right] \\ &= \sum_{t=0}^{T} \nabla_\theta \log \pi(a_t|s_t) \end{aligned} \tag{2.31}$$

On a donc:

$$\nabla J(\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t=0}^{T} \nabla_\theta \log \pi(a_t|s_t) R(\tau) \tag{2.32}$$

L'optimisation du réseau se fait sur l'ensemble des trajectoires récupérées par l'agent durant la phase d'évaluation de la politique, le gradient sera donc approximé par :

$$\nabla J(\theta) \approx \frac{1}{m} \sum_{i=0}^{m} \sum_{t=0}^{T} \nabla_\theta \log \pi(a_t^i|s_t^i) R(\tau^i) \tag{2.33}$$

Les poids θ sont mis à jour via un algorithme de rétropropagation du gradient dans le réseau neuronal :

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{2.34}$$

L'algorithme *REINFORCE* souffre d'inconsistance dans l'amélioration de la politique. En effet, la sélection du taux d'apprentissage est cruciale pour éviter des variations trop grandes ou trop petites de la politique, ce qui entraîne des changements trop violents dans le comportement de l'agent. De plus, l'utilisation de la fonction R crée des variations trop importantes entre les trajectoires, ce qui entrave la convergence de l'agent.

Afin de pallier ces problèmes, il existe des méthodes permettant de quantifier l'évolution de la politique après une optimisation afin de la limiter pour éviter qu'elle ne devienne trop importante. De plus, d'autres fonctions de récompense peuvent être utilisées à la place de R, telles que les fonctions de valeur Q, V, A, et elles peuvent être approximées par un réseau de neurones pour avoir une évolution conjointe entre l'approximation des récompenses et l'optimisation de la politique.

L'algorithme *PPO* utilise des bornes pour restreindre l'évolution de la politique. Avec le théorème du gradient de la politique, on optimise la politique en bornant son changement afin de rester proche de l'ancienne politique. De plus, au lieu d'utiliser R comme fonction de récompense, *PPO* utilise une approximation $\hat{A}$ de la fonction Avantage. La fonction de perte associée à la politique est la suivante:

$$L^{PG}(\theta) = \mathop{\mathbb{E}}_{t} \left[\log \pi_\theta(a_t|s_t) \hat{A}_t\right] \tag{2.35}$$

Afin d'estimer l'amélioration de la politique après une optimisation sur de multiples trajectoires, on introduit $r_t(\theta)$ qui est le ratio de probabilité $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, avec $r(\theta_{old}) = 1$, et on définit la fonction objectif comme suit :

$$L^{\text{CLIP}}(\theta) = \mathop{\mathbb{E}}_{t} \left[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right], \tag{2.36}$$

avec ϵ un paramètre souvent choisi autour de 0.2, et *clip* une fonction permettant de borner $r_t(\theta)$ entre $1-\epsilon$ et $1+\epsilon$. Cette formulation permet dans un premier temps de ne pas propager le gradient lorsque r_t est en dehors de $[1-\epsilon, 1+\epsilon]$. En effet, le gradient sera nul car les bornes ne dépendent pas de θ. Ensuite, le minimum entre l'objectif borné et non-borné permet d'obtenir une borne inférieure de $r_t(\theta)\hat{A}_t$ [46].

Pour le calcul de la fonction avantage, *PPO* utilise un réseau de neurones appelé le critique qui approxime la fonction $V(s) = r_t + \gamma V(s') = r_t + \gamma \underset{a}{max} Q(s', a')$. De ce fait, on peut aisément calculer une estimation de la fonction avantage, comme suggéré dans l'article de référence [45], par une méthode de *n-step return* :

$$Q(s_t, a) = r_t + \gamma r_{t+1} + \ ... \ + \gamma^{n-1} r_{t+n-1} + \gamma^n \underset{a}{max} Q(s_{t+n}, a) \tag{2.37}$$

Avec $A_t = Q_t - V_t$ et $t \in [0, T]$ nous avons:

$$\begin{aligned} A_t \ &= r_t + \gamma r_{t+1} + \ ... \ + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} \underset{a}{max} Q(s_T, a) - V_t \\ &= -V_t + r_t + \gamma r_{t+1} + \ ... \ + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T) \end{aligned} \tag{2.38}$$

Une version tronquée de l'avantage peut aussi être utilisée, comme dans l'article de référence [52]. A est approximée grâce au réseau critique. En effet, les trajectoires récoltées par l'agent nous permettent de calculer $Q_{\pi_\theta}(s, a) = \underset{a' \sim \pi_\theta}{\mathbb{E}} [r + \gamma Q(s', a') | s_0 = s, a_0 = a]$ de façon itérative en partant du dernier état T. On peut ensuite optimiser le réseau critique en minimisant la *MSE* entre $Q_{\pi_\theta}(s, a)$ et $V_\phi(s)$.

$$L_t^{\text{VF}}(\phi) = MSE(V_\phi(s), Q_{\pi_\theta}(s, a)) \tag{2.39}$$

On peut voir à travers ces différentes optimisations que les deux réseaux, $\pi(s)$ et $V(s)$, sont intimement liés. En effet, la politique est optimisée avec la fonction avantage, qui est approximée par un réseau de neurones. Au début du processus, la fonction avantage peut donc être loin de la réalité. Ces phénomènes de *bootstrapping* engendrent des problèmes d'exploration, car l'agent peut rapidement rester dans un minimum local.

Pour réduire au mieux les problèmes d'avortement de l'exploration liés aux mauvaises estimations des fonctions de valeur, on peut ajouter à la fonction objectif un bonus d'entropie. L'entropie en *RL* permet de quantifier la rareté de l'action choisie a en suivant une certaine politique π.

$$\mathcal{H}(\pi(., s_t)) = \underset{a \sim \pi(.|s_t)}{\mathbb{E}} [\log \pi(a|s_t)]. \tag{2.40}$$

On peut donc former la fonction objectif à maximiser de *PPO*:

$$L^{\text{PPO}}(\theta) = \mathbb{E}_t \left[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\theta) + c_2 \mathcal{H}[\pi_\theta](s_t) \right] \tag{2.41}$$

Avec c_1 et c_2 des constantes d'ajustement, c_2 est souvent proche de 0.01, et c_1 permet d'ajuster l'erreur du critique par rapport à celle de l'acteur. Le pseudocode de *PPO* 2 est présenté dans la page suivante.

Algorithm 2 Proximal Policy Optimisation

Input : environment (env), agent (π_θ,V_ϕ), poids du réseaux θ taux d'apprentissage (α), MaxLoop, la borne ϵ_{clip}, l'horizon H, une mémoire Rb de taille H, le nombre d'optimisation K, les poids des fonction d'erreur c_1, c_1
Out : π^θ
for $k = 1, 2, \ldots,$ MaxLoop **do**
    while Iteration$<H$ **do**
        Rb.reset
        s = env.init
        while 1 **do**
            $a = \pi_{\theta_k}(s_t, a_t)$
            $s_{t+1}, r_t, D_t = \text{env(a)}$
            $Rb \leftarrow < s_t, a_t, o_{t+1}, r_t, D_t >$
            if $D_t == 1$ **then**
                break
            else
                $s = s_{t+1}$
            end if
        end while
    end while
    for $i = 1, \ldots, K$ **do**
        Calcul de la fonction Advantage pour chaque épisode dans Rb
        $L^{\text{PPO}}(\theta_k) = \mathbb{E}\left[L^{\text{CLIP}}(\theta_k) - c_1 L^{\text{VF}}(\theta_k) + c_2 \mathcal{H}[\pi_{\theta_k}]\right]$
        $\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_{\theta_k} L^{\text{PPO}}(\theta_k)$
    end for
end for

Conclusion

En conclusion de ce chapitre, nous avons passé en revue les fondamentaux des différentes méthodes et algorithmes utilisés dans cette thèse. Dans le chapitre suivant, nous établirons l'état de l'art des méthodes permettant la résolution de problèmes de *RL* à l'aide d'extractions visuelles. Nous présenterons des travaux utilisant des approches traditionnelles, telles que l'apprentissage de bout en bout avec des *CNN*. Nous introduirons l'apprentissage de représentation d'état, le *SRL*, ainsi que ses méthodes découpant l'apprentissage en perception et commande. Nous examinerons également des méthodes moins conventionnelles, comme l'utilisation de *RBFN* dans la résolution de tâches de *RL*. Cette exploration nous permettra de mettre en évidence les avantages et les limitations de chaque approche.

Chapitre 3

Présentation des méthodes de RL pour les tâches visuelles

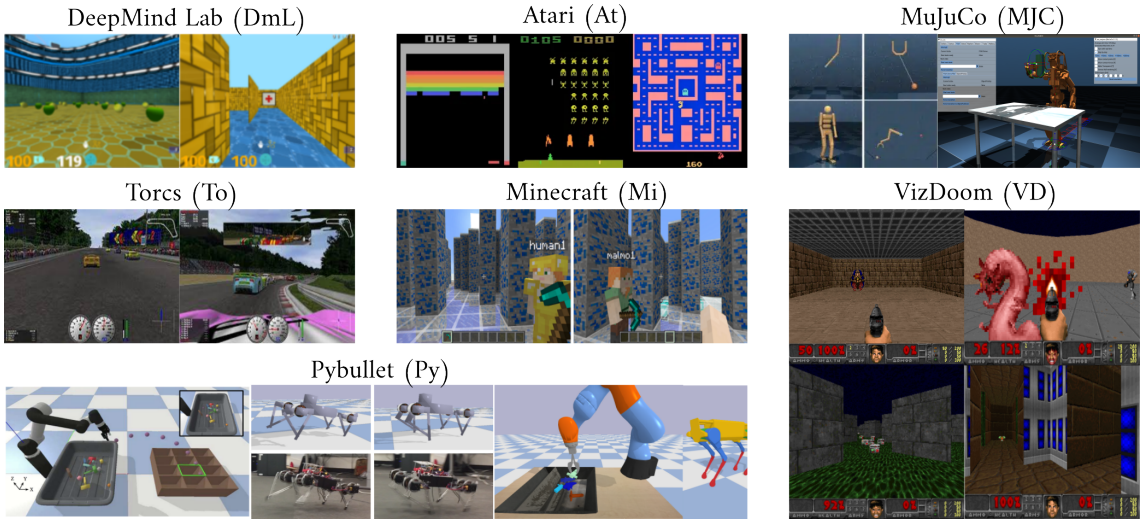


Figure 3.1: Représentation des différents environnements évoqués dans cet état de l'art. Les abréviations sont utilisées dans les tableaux de ce chapitre.

Introduction

Dans le chapitre précédent nous avons introduit les concepts clés de l'apprentissage par renforcement et les principaux algorithmes. Dans le présent chapitre nous nous proposons d'analyser l'état de l'art concernant le *DRL* appliqué à des scénarios visuels, notamment à la première personne. Nous discuterons en particulier des différentes représentations visuelles utilisées pour la résolution de ce type de tâches.

L'utilisation de données images comme entrées de méthodes de *RL* a longtemps été un défi du fait de la grande dimension de l'espace des images. Les approches les plus courantes étaient généralement décomposées en deux parties distinctes : une pour la perception visuelle et une autre pour la prise de décision. La perception reposait souvent sur des méthodes utilisant des extracteurs de caractéristiques de type *handcrafted*, tels que l'utilisation d'extracteurs de type *scale-invariant feature transform SIFT* [53], ou des combinaisons de différentes méthodes d'extraction [54].

Les réseaux de convolution, capables d'ingérer des images de taille conséquente et d'extraire des caractéristiques utiles à différentes tâches de prédiction [55], ont été utilisés dans un premier temps pour la perception [56]. Puis les méthodes de bout en bout avec des *CNN*se sont imposées comme fondation de l'apprentissage à partir de données visuelles, dominant toutes les méthodes précédemment proposées sur les jeux vidéo Atari [1].

Dans cette thèse, notre intérêt se concentre principalement sur les représentations visuelles utilisées dans des tâches de *RL* en vue à la première personne. Nous verrons que ces environnements représentent un réel défi dans le domaine de l'apprentissage par renforcement. Ces environnements impliquent souvent une observabilité partielle, où l'agent est restreint à son propre champ de vision. La résolution de ces problèmes complexes échoue souvent lorsque l'on utilise des apprentissages de bout en bout avec les algorithmes précédemment introduits (*PPO*, *DQN*, *A3C*...). En effet, les variations importantes de ces environnements peuvent causer des interférences significatives lors de la rétropropagation. Ce problème survient lorsque les poids déjà optimisés pour une certaine entrée sont modifiés, ce qui engendre un changement de comportement de l'agent pouvant s'avérer catastrophique pour résoudre une tâche [57], [58].

Dans ce chapitre, nous commencerons par présenter les méthodes de bout en bout pour l'apprentissage par renforcement visuel à la première personne. Ensuite, nous présenterons les différents aspects de l'apprentissage de représentation d'états. Enfin, nous conclurons en présentant les méthodes moins conventionnelles qui ont inspiré ces travaux. Pour faciliter la compréhension des différents tableaux de ce chapitre, nous utiliserons les abréviations du tableau 3.1 pour décrire les méthodes et pour les différents environnements disponibles dans la figure 3.1.

Table 3.1: Tableau des différentes significations des acronymes utilisés dans les tableaux de ce chapitre

Abréviations	Significations
HD-LD : (High-Low Dimensional)	Environnement visuel et non visuel.
SF (Stack Frame)	La méthode concatène plusieurs images comme une seule observation.
Po (Partially Observable)	L'environnement est partiellement observable. Il est souvent accompagné de l'abréviation d'un environnement. Po-At correspond à un jeu Atari comprenant de l'observabilité partielle.

3.1 Apprentissage de bout en bout

Dans [36], Kempka et al. ont montré qu'il est possible de résoudre certains scénarios de la plateforme de simulation VizDoom avec un *CNN* entraîné de bout en bout utilisant un algorithme de type *DQN*. Les scénarios abordés n'impliquaient pas une observabilité très partielle. Cependant, pour des scénarios plus complexes, intégrant une observabilité partielle comme des labyrinthes ou des scénarios de survie avec des monstres attaquant l'agent, l'entraînement classique de bout en bout ne permet pas d'obtenir un comportement optimal de l'agent. En effet, différents algorithmes classiques (*DQN*, *PPO*, *A2C*) ont été testés sur ces scénarios de VizDoom [49], et les résultats montrent que les méthodes d'optimisation de bout en bout sont difficiles à faire converger et sont sensibles à l'hyperparamétrisation du modèle dans les scénarios partiellement observables.

Comme indiqué dans le tableau 3.2, plusieurs méthodes ont été proposées pour gérer les problèmes d'observabilité partielle dans les environnements visuels à la première personne avec l'idée d'enrichir l'information disponible sur l'état. Ces méthodes peuvent être grossièrement séparées en deux catégories : celles utilisant l'empilement d'état (*Stack Frame - SF*) et celles utilisant des modules de mémoire tels que le *LSTM*.

3.1.1 Méthodes utilisant l'empilement d'images

L'empilement d'images (*Stack Frame - SF*) consiste à empiler plusieurs images consécutives pour fournir une meilleure représentation de l'état au réseau de neurones. Cette technique est particulièrement utile pour les environnements où l'agent dispose

d'un champ de vision limité. Dans le contexte de VizDoom, [36], [59] ont démontré que l'utilisation de *SF* permet d'obtenir des performances acceptables dans les scénarios avec une observabilité partielle moindre. Cependant, pour des scénarios plus complexes, cette méthode seule ne suffit pas. Les travaux [49], [60] confirment que les *SF* peuvent améliorer la stabilité de l'apprentissage, mais restent limités dans leur capacité à gérer des environnements hautement dynamiques.

3.1.2 Méthodes utilisant des modules de mémoire

Pour surmonter les limitations des *SF*, des modules de mémoire tels que les *LSTM* (Long Short-Term Memory) ont été intégrés aux architectures de réseaux neuronaux. Ces modules permettent de conserver des informations sur plusieurs pas de temps, améliorant ainsi la capacité de l'agent à inférer l'état de l'environnement à partir d'observations partielles. Toutefois, plusieurs études ont révélé que l'utilisation de *LSTM* pour les tâches visuelles reste complexe et souvent inefficace. Les travaux [61]–[65] ont comparé les performances des algorithmes avec et sans *LSTM* et ont trouvé que les gains en performance étaient souvent limités et sensibles à l'hyperparamétrisation.

[66] ont identifié des facteurs critiques pour le bon fonctionnement des *LSTM* dans les environnements visuels : la séparation des réseaux acteur et critique, et l'inclusion de l'action et/ou de la récompense passées en entrée du *LSTM*. [67] ont également montré que certains algorithmes, comme l'*A3C*, bénéficient davantage de l'intégration de modules de mémoire, mais que cela ajoute une complexité significative à l'entraînement. De plus, leurs travaux confirment la nécessité d'ajouter des informations comme l'action pour le bon fonctionnement du *LSTM*.

Un autre exemple est le concours de *RL* sur les environnements VizDoom mené dans [68], où la méthode la plus performante utilisant un *LSTM* avec une méthode de bout en bout a utilisé des récompenses additionnelles pour mieux guider l'agent [69]. L'entraînement du réseau avec le *LSTM* a pris plus de 27 heures pour 30 millions d'itérations, démontrant les défis liés à cette approche.

Étant donné la difficulté d'implémentation des *LSTM* et l'instabilité de cette méthode, [70] ont introduit l'*AMRL* (*Aggregated Memory for Reinforcement Learning*). Leurs modèles utilisent un module de mémoire standard pour résumer le contexte à court terme, puis agrègent tous les états précédents sans respecter l'ordre. Cette méthode s'est avérée plus performante que le *LSTM* en termes de convergence de l'agent vers un comportement optimal. Cependant, cette approche ajoute un coût de calcul accru. [71] ont montré que l'utilisation de *LSTM* dans des environnements non visuels reste prometteuse, soulignant les différences de performance selon le type de données d'entrée.

En résumé, bien que les modules de mémoire comme le *LSTM* et les techniques de stack frames offrent des solutions pour améliorer la représentation des états dans des environnements visuels partiellement observables, ils présentent chacun leurs propres défis. Les empilements d'images sont simples à implémenter mais limités, tandis que les *LSTM* offrent des avantages potentiels en termes de mémoire mais ajoutent une complexité significative à l'entraînement.

Table 3.2: Tableau des méthodes d'apprentissage de bout en bout par renforcement.

Année	Référence	Algorithmes	État	Environnement	Méthodes
2015	[61]	DRQN	HD	Po-At	LSTM, SF
2016	[36]	DQN	HD	VD	SF
2017	[69]	A3C	HD	VD	LSTM
2017	[59]	DQN/DRQN	HD	VD	SF
2017	[60]	A3C	HD	VD	SF
2018	[63]	DRQN	HD	VD	LSTM
2018	[67]	A-DRQN	HD	Po-At	LSTM, SF
2019	[64]	D(R)QN/C51/D4RQN	HD	VD	LSTM, SF
2019	[62]	DRQN/DQN	HD	Po-Mi	LSTM, SF
2020	[65]	D(R)QN/DD(R)QN	HD	VD	LSTM, SF
2020	[70]	PPO	HD	Po-Mi	AMRL, LSTM
2021	[71]	AC	LD	Po-Py	LSTM
2022	[49]	DQN/A2C/PPO	HD	VD	SF

3.2 Extraction de représentation

L'extraction de représentations est un domaine clé de l'apprentissage machine qui vise à transformer les données brutes vers un espace généralement de plus faible dimension, et qui facilite les tâches visées, qu'il s'agisse de classification, de régression, de clustering etc. Cette transformation est souvent cruciale pour la performance des modèles. Dans cette partie nous aborderons les premiers travaux dans cette direction, telles que les méthodes classiques comme l'analyse en composantes principales (*PCA*) et le codage par tuiles (tile coding), ainsi que les développements plus récents tels que l'apprentissage de représentations d'état (*SRL*) et les réseaux pré-entrainés.

3.2.1 Travaux Préliminaires

PCA (Analyse en Composantes Principales)

L'analyse en composantes principales (*PCA*) [72] est une technique classique d'extraction de représentations. Elle consiste à projeter les données sur un espace de plus faible dimension tout en conservant autant que possible la variance des données originales. *PCA* identifie les directions principales (composantes principales) selon lesquelles les données varient le plus. Cependant la méthode de *PCA* ne capte pas les relations non linéaires des données, de plus cette méthode est sensible au bruit et ne gère pas les données aberrantes [73]. L'application de la méthode *PCA* sur des images est généralement peu conseillée en raison du temps de calcul nécessaire pour son utilisation sur des données de grande dimension. Cependant, lorsque l'espace d'états est de petite dimension, comme des images de petite taille, les méthodes *PCA* peuvent s'avérer utilisables. Par exemple, dans [74], la *PCA* est utilisée pour représenter l'état dans un scénario de *RL* dans le jeu Mario. Les états dans cet environnement sont des vecteurs à une dimension représentant la possibilité de sauter (0 ou 1), s'il est au sol (0 ou 1), s'il peut lancer des boules de feu (0 ou 1), sa direction actuelle (0 à 8), la présence d'ennemis proches et à moyenne disctance dans 8 directions (sous forme de grille), la présence d'obstacles dans les quatre cellules verticales devant lui et la position de l'ennemi le plus proche dans une grille de 21×21 autour de lui. Les résultats obtenus sont supérieurs à l'utilisation d'un *MLP*. De même, dans [75], la *PCA* est utilisée comme extracteur de représentation d'état sur de petites images transformées en vecteurs pour résoudre un problème de *RL*. Les résultats obtenus sont supérieurs à l'utilisation d'une méthode de gradient de politique avec un *MLP*.

Codage par Tuiles

Le codage par tuiles ou *tile coding* [76] est une méthode utilisée principalement dans l'apprentissage par renforcement pour discrétiser l'espace d'états [33]. Elle divise l'espace d'états en régions ou "tuiles" et utilise ces tuiles pour créer des représentations binaires des états. Cette méthode est simple à implémenter et s'avère efficace dans le cas de données de petite dimension [57]. En revanche cette méthode est difficile à mettre en place dans le cas de données de grande dimension. De plus, la précision du découpage peut influencer les performances du codage par tuiles.

Réseaux de fonctions à Base Radiales (*RBF*)

Les réseaux de fonctions à base radiales (*RBF*) sont des outils puissants en apprentissage machine, exploitant des fonctions de base non linéaires telles que les gaussiennes pour projeter les données d'entrée dans un espace de caractéristiques de dimension supérieure [17]. Cette approche permet de capturer des relations non linéaires complexes dans les données. Les *RBF* jouissent d'une grande capacité de généralisation et sont considérées comme des approximations universelles de fonction [77]. Leur polyvalence se reflète dans leur utilisation variée tant dans l'extraction de caractéristique ou plus récemment dans l'apprentissage par renforcement en tant qu'approximateur des fonctions de valeurs.

Utilisation des *RBFN* sur des images

Dans la littérature, on trouve beaucoup de travaux utilisant les *RBFN* sur des données images. Cependant, en raison du coût de calcul prohibitif de l'application directe des *RBFN* sur des images, ces derniers sont souvent utilisés sur des espaces de dimension réduite. Ainsi, dans les années 90, ces réseaux ont été utilisés pour la reconnaissance faciale sur des différences de gaussiennes, des filtres de Gabor extraits d'images [78], ou encore des extractions de type *PCA* [79]. Pour la segmentation d'images médicales, [80] utilise des extractions *hand-crafted* issues des intensités de chaque pixel et de leur voisinage à l'aide de champs récepteurs, pour ensuite utiliser ces extractions en entrée d'un *VRBFN*. Plus récemment, [81], [82] vectorisent directement l'image pour appliquer les *RBFN* dans le cadre de la reconstruction d'images. Ces travaux montrent l'efficacité de ces réseaux pour la segmentation et la classification de pixels. Cependant, le *RBFN* n'est utilisé qu'en second temps après un prétraitement des images. Les *RBFN* ont aussi été utilisés à la suite d'extractions par *CNN* dans le domaine de la vision par ordinateur [83]. Les résultats expérimentaux montrent que l'intégration des *RBF* après les couches de *CNN* atteint des performances compétitives sur des ensembles de données de référence en combinant apprentissage supervisé et non supervisé. L'utilisation de l'architecture *RBF* avec les *CNN* introduit deux opportunités uniques et spécifiques au réseau : l'apprentissage d'une métrique

de distance de similarité et une interprétation plus détaillée du processus de prise de décision. Les images similaires et différentes trouvées en utilisant une métrique de distance de similarité entraînée par les *RBF* sont interprétables par les humains.

L'utilisation des *RBFN* pour l'extraction de caractéristiques visuelles prenant en compte l'aspect spatial des images n'est, à ma connaissance, pas traitée dans l'état de l'art à cause de problèmes liés à la grande dimension des images. Cependant, des travaux utilisant le *RBFN* pour extraire des caractéristiques sur des nuages de points ont été menés dans [84]. Dans ces travaux, la distribution spatiale des points est modélisée explicitement grâce à la non-linéarité localisée des noyaux RBF, permettant de distinguer différents motifs de distribution. Cette approche améliore les performances de reconnaissance des nuages de points et réduit significativement le nombre de paramètres du réseau, accélérant ainsi l'entraînement. Cette approche montre que les *RBFN* peuvent être intéressants dans l'extraction de caractéristiques.

RBFN en apprentissage par renforcement

Dans le cadre de l'apprentissage par renforcement (*RL*), plusieurs travaux ont exploré l'utilisation des *RBFN* pour l'approximation de la fonction de valeurs. Dans [85], un *RBFN* est utilisé dans un environnement de *RL* basé sur l'évitement de collision avec un robot mobile. Les auteurs proposent une approche où les centres des fonctions radiales sont ajustés de manière adaptative en fonction des données d'entrée, améliorant ainsi la précision de l'approximation de la fonction Q. Dans la même lignée, [22] propose un algorithme basé sur la descente de gradient pour entraîner un *RBFN* à approximer la fonction Q dans quatre environnements différents (labyrinthe, *mountain car*, *pole balancing* et *acrobot*). En 2015, Jonschkowski [86] propose d'utiliser le *RBFN* comme approximateur des fonctions de valeurs dans un algorithme d'itérations sur la valeur Q. Ils utilisent des a priori sur des tâches de navigation (simulées et réelles) et une course de voiture pour apprendre une représentation d'état permettant l'utilisation d'un *RBFN* pour l'approximation de la fonction de valeurs. De manière générale, les *RBFN* en *RL* ne sont jamais utilisés directement sur les données d'entrée. En effet, dans la littérature sur le *RL*, les *RBFN* jouent le rôle de contrôleur et sont souvent utilisés comme approximateurs des fonctions de valeurs [87]. De plus, [88] montre l'efficacité de cet approximateur dans des scénarios avec des espaces d'action continus.

En conclusion, les *RBF* sont des outils polyvalents et puissants utilisés dans divers domaines de l'apprentissage automatique, y compris la reconnaissance de formes, la segmentation d'image, la reconstruction d'image et l'apprentissage par renforcement. Leur capacité à capturer des relations non linéaires complexes dans les données en fait des candidats attrayants pour une large gamme de tâches. Cependant, leur utilisation efficace nécessite souvent une expertise dans le choix des paramètres et des architectures adaptées à chaque application spécifique.

3.2.2 Apprentissage de Représentations d'État

Les algorithmes de *RL* sont souvent utilisés en considérant les observations comme des états complets, car pour résoudre un *MDP*, l'environnement doit satisfaire la propriété de Markov. Cette propriété stipule que les probabilités conditionnelles des états futurs dépendent uniquement de l'état présent et non des états passés. Cependant, dans les environnements visuels à la première personne, l'agent doit se contenter de son observation $o = O(s)$, qui correspond à une partie visible de l'état. Dans ces environnements partiellement observables, on cherche souvent à compléter ou améliorer l'observation afin d'obtenir plus d'informations sur l'état général de l'environnement pour utiliser les algorithmes définis dans le cadre des *MDP*.

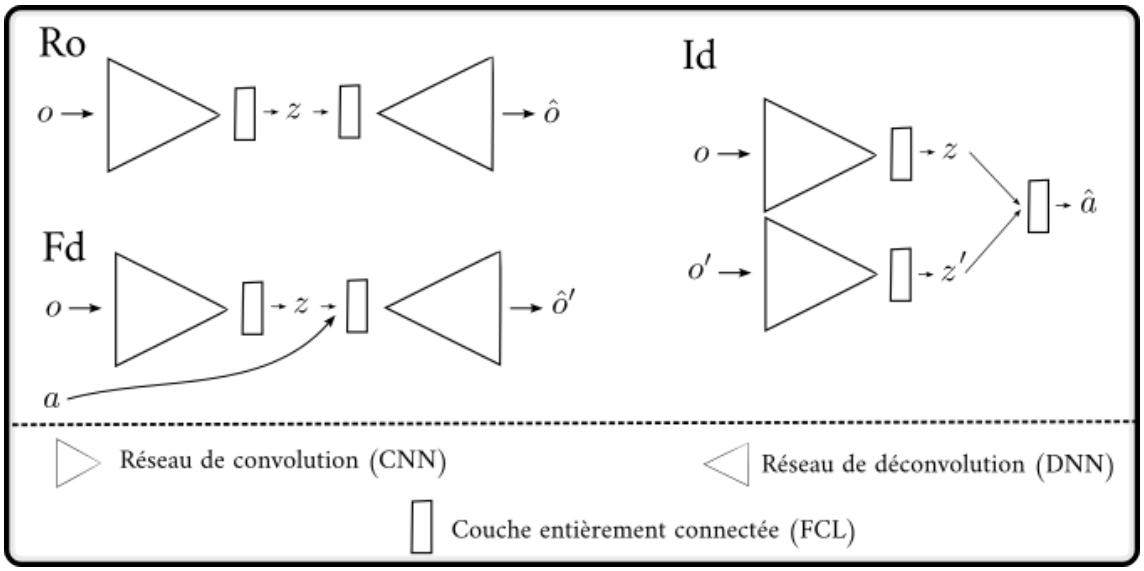


Figure 3.2: Schéma conceptualisant trois méthodes de *SRL*. o, a, et z correspondent aux observations, actions et espaces latents. Lorsqu'ils sont accompagnés d'un chapeau, ce sont les reconstructions du réseau ; s'ils ont une apostrophe, ce sont les états suivants.

La recherche sur l'apprentissage de représentations d'états (*State Representation Learning* - *SRL*) vise à rendre l'apprentissage sur des images en *RL* plus efficace. Cette approche repose sur l'idée de créer des représentations compactes des observations, ce qui peut accélérer l'apprentissage et améliorer les performances. Un état de l'art complet des différentes méthodes de *SRL* a été proposé en 2018 dans [3]. Nous nous concentrons ici sur les principales approches qui utilisent généralement un *AE* composé d'un *CNN* et d'un *DCNN*, comme illustré dans le schéma 3.2, où quatre grandes classes de *SRL* sont représentées :

- **Reconstruction de l'observation (Ro)**
  Cette méthode non supervisée réduit la taille de l'observation en utilisant une approche d'encodage-décodage. L'image est encodée à l'aide d'un réseau de convolution, parfois accompagné de couches linéaires, afin d'obtenir un

vecteur z. Ce vecteur représente l'image de manière compacte. Ensuite, ce vecteur est utilisé pour reconstruire l'image observée par l'agent. Le réseau est optimisé en minimisant l'erreur quadratique entre l'observation originale et sa reconstruction.

- **Apprentissage du modèle de transition (*Forward dynamic* - (Fd))**
  Cette approche utilise les mêmes outils que la méthode précédente pour prédire la prochaine observation en connaissant l'observation actuelle et l'action effectuée. Cela permet d'obtenir une représentation de l'image qui tient compte de la dynamique de l'environnement.

- **Apprentissage du modèle inverse (*Inverse dynamic* - (Id))**
  Cette méthode utilise l'observation actuelle et la prochaine observation pour prédire l'action effectuée. Cela permet d'avoir un encodeur qui caractérise l'image en prenant en compte les actions.

Ces méthodes peuvent être combinées en fonction des besoins. De plus, pour améliorer leur application en *RL*, des contraintes de reconstruction peuvent être ajoutées. Par exemple, on peut utiliser la méthode (Ro) en ajoutant des variables d'état à reconstruire, ou appliquer des contraintes sur l'espace latent. Le domaine du *RL* visuel bénéficie de ces méthodes qui permettent l'apprentissage d'une représentation de l'environnement, laissant ainsi l'algorithme de *RL* traiter uniquement l'aspect contrôle à partir des représentations extraites grâce à la méthode de *SRL*. Ces méthodes, en plus de séparer les tâches de perception et de commande, permettent aussi de réduire l'espace d'entrée du réseau de commande, ce qui permet un apprentissage plus rapide et moins coûteux. Cependant, ces méthodes requièrent des données d'entraînement issues de l'environnement et un pré-entraînement.

Dans le tableau 3.3, nous proposons une liste représentative de l'ensemble des méthodes de *RL* visuel utilisant le *SRL*. Ces méthodes permettent de pallier le manque d'informations sur l'état complet et facilitent l'utilisation des algorithmes de résolution de *MDP* en compactant l'espace d'entrée du *RL*.

Reconstruction de l'Observation (Ro)

Précurseur de l'apprentissage de représentations d'états, S. Lange a montré, avant l'utilisation de méthodes de bout en bout avec les *CNN*, qu'il était possible d'utiliser un auto-encodeur pour résoudre des problèmes de *RL* visuel uniquement avec des couches entièrement connectées [89]. Les extractions obtenues sont comparées avec une méthode de *PCA* à travers un exercice de classification et de *RL*; les résultats obtenus sont favorables à l'auto-encodeur lorsqu'il y a suffisamment de données d'entraînement. Les résultats d'une deuxième publication avec une tâche réelle sur circuit électrique [56] ont montré l'efficacité de la méthode d'auto-encodeur

comme modèle de perception pour un problème de *RL* visuel. Le problème en question consistait à contrôler l'accélération d'une mini voiture électrique sur un circuit uniquement à l'aide d'une vue du dessus du circuit.

L'utilisation de modèles d'auto-encodeurs pour réduire la dimension d'un espace est devenue la méthode de prédilection pour l'apprentissage de représentation d'état. Dans [90], la méthode (Ro) est utilisée dans deux environnements différents, Cart Pole (Cp) et le jeu de pierre-feuille-ciseaux dans une configuration réaliste avec des photos de main jouant au jeu. Une fois l'entraînement de l'*AE* terminé, la partie déconvolution est retirée et l'espace latent est utilisé dans un réseau entièrement connecté (*MLP*) pour estimer les valeurs Q.

Certains travaux améliorent les résultats du *SRL* en utilisant des prédictions basées sur les connaissances de l'environnement, par exemple [91] qui utilise un auto-encodeur pour l'apprentissage de tâches robotiques. Il a ajouté des points d'intérêt dans l'image de reconstruction associés à la configuration des objets dans l'image. De ce fait, l'extraction de l'auto-encodeur contient des informations permettant la reconstruction de ces points d'intérêt. Les extractions sont ensuite utilisées dans un algorithme linéaire en combinaison avec l'état des articulations du bras robotique, leurs dérivées temporelles et la position de la main. L'utilisation de leur modèle de perception améliore les résultats obtenus sans images et permet un apprentissage de la tâche en une quinzaine de minutes d'utilisation du robot.

Dans l'environnement VizDoom, [92] apprend les états du jeu (vie, munitions, ennemis tués, etc.) à partir des images afin de trouver les actions permettant d'arriver à certains états. D'autres travaux sur l'application du *SRL* montrent que l'utilisation d'*AE* limite la qualité de la reconstruction dans une configuration (Ro), en l'occurrence l'*AE* parvient à bien reconstruire le décor mais la reconstruction de petits éléments dans l'image s'avère limitée [93].

Apprentissage du Modèle Dynamique (Fd)

Différents travaux liés au domaine du RL ont étudié la capacité d'un auto-encodeur, et de ses variantes comme le *VAE*, à prédire les observations futures à partir de l'observation actuelle et de l'action effectuée [94]. Ces travaux mettent en avant l'efficacité de la méthode à prédire la prochaine observation à partir de l'action, mais analysent aussi l'influence de l'ajout d'autres variables à prédire comme la récompense ou la fin de l'épisode [95], [96]. De plus, l'utilisation de contraintes, de linéarité dynamique par exemple, sur l'espace latent, notamment grâce à la méthode du *VAE*, permet d'améliorer les résultats de reconstruction [12], [97].

Ces travaux permettent aussi de mettre en avant certains défauts de ces méthodes. En effet, les bases de données nécessaires à l'apprentissage de l'*AE* sont souvent

récupérées avec des explorations aléatoires ou issues d'entraînements de *RL*. De ce fait, les bases de données recouvrent rarement l'ensemble de l'environnement et donc l'extraction n'est plus aussi fiable aux endroits peu rencontrés lors de l'apprentissage. Ce phénomène est amplifié pour les scénarios partiellement observables [98].

Après avoir établi l'utilité d'un auto-encodeur à prédire des variables d'état d'un environnement de *RL*, plusieurs publications ont validé l'utilisation de l'auto-encodeur en *SRL* pour résoudre des tâches de *RL* [99]–[101]. Dans ces recherches, on retrouve le problème d'exploration énoncé ci-dessus. VanHoof [102], qui utilise un *VAE*, montre qu'un réentraînement dit "dynamique" du *VAE* (réentraînement avec l'entraînement de la politique) permet de pallier ce problème d'exploration et améliore les résultats. Cependant, les travaux sur l'intégration de l'*AE* dans le *RL* menés par [103] montrent qu'il est plus efficace d'alterner entre l'optimisation de l'extraction et l'optimisation de la commande plutôt que d'utiliser le pré-entraînement ou d'entraîner les deux objectifs en même temps en utilisant l'erreur de prédiction comme régularisation.

Apprentissage du Modèle Inverse (Id) et Fonction Auxiliaire (Aux)

L'apprentissage de modèles inverses est la méthode de *SRL* la moins représentée dans l'état de l'art. Pourtant, elle est intéressante car elle permet de se passer de la reconstruction d'une image complète, qui peut s'avérer difficile étant donné la grande dimensionnalité des images. Cette méthode est utilisée dans [104], qui apprennent en même temps un modèle direct et un modèle inverse. Le *SRL* peut aussi être utilisé pour extraire des primitives importantes à l'apprentissage d'une tâche, comme par exemple dans [105], où le *SRL* est utilisé pour apprendre la position de l'objet manipulé à partir d'images. Cet apprentissage permet de faciliter l'apprentissage d'une tâche de manipulation à partir des primitives extraites des images par le *SRL*.

En général, le *SRL* bénéficie de l'ajout de fonctions d'erreur annexes afin d'avoir des extractions qui respectent au mieux les caractéristiques importantes d'une représentation d'état [106], à savoir le respect de la propriété de Markov, la compacité ainsi que la généralisation. En plus de ces caractéristiques importantes, certains autres aspects des extractions sont à prendre en considération en *RL*, comme la parcimonie des activations, qui peut être obtenue à l'aide de fonctions auxiliaires sur l'apprentissage de la représentation d'état. En *RL*, l'extraction d'informations éparses semble offrir un avantage [107]. En effet, en activant seulement certains neurones pour des entrées précises, la détection de schémas d'activation liés à certaines caractéristiques de l'environnement est facilitée. Cela permet une meilleure prise de décision de la part de l'agent et réduit considérablement les interférences lors de l'optimisation, car les chemins de rétropropagation diffèrent selon les observations.

SRL et Mémoire

L'utilisation de l'*AE* en apprentissage par renforcement a permis également une meilleure stabilité des approches utilisant des modules de mémoire comme le *LSTM*. En effet, l'utilisation de représentations compactes basées sur l'action (Fd) ou encore sur la récompense ou des a priori est importante pour le bon fonctionnement du *LSTM*. Dans ses travaux sur les *World Model*, [108] met en avant le fait d'apprendre des représentations basées sur la récompense qui permettent à l'agent d'avoir des informations pertinentes sur son objectif et de ce fait d'améliorer l'utilisation d'un LSTM. Dans [105], les données fournies au *LSTM* sont extraites d'un apprentissage de fonction auxiliaire permettant le bon fonctionnement du *LSTM*. De ce fait, le *LSTM* et le *SRL* sont souvent utilisés conjointement pour résoudre des scénarios complexes, car les caractéristiques extraites avec le *SRL* sont plus facilement utilisables par un *LSTM*.

Table 3.3: Tableau des méthodes d'apprentissage de représentations d'états

Année	Référence	Observation	Environnement	SRL
2012	[56]	HD	Circuit voiture	Ro, FC-AE
2015	[109]	HD+LD	CP	Fd, CNN-VAE
2016	[102]	HD	Pendulum	Fd, FC-VAE
2016	[91]	HD+LD	Robot - Visuomotor	Ro-Aux, CNN-AE
2017	[92]	HD+LD	VD	Ro-Aux
2017	[93]	HD+LD	VD	Ro
2017	[104]	HD+LD	VD	Fd, Id, LSTM
2018	[90]	HD	At, Real paper-rock-scissors	Ro, CNN-AE
2018	[105]	HD+LD	Robot - Visuomotor	AUX
2019	[99]	HD	At	Fd, CNN-AE, LSTM
2020	[100]	HD	At, Mu	Ro, VAE
2020	[101]	HD	At	Fd-Aux, VAE

3.2.3 Extreme machine learning (*ELM*) et Réseaux Pré-entrainés

Au delà de l'apprentissage de représentation, certaines approches proposent d'utiliser des représentations extraites via des réseaux de neurones à visée générique, pré-entrainés ou initialisés aléatoirement.

Extreme machine learning (*ELM*)

Les méthodes d'apprentissage machine dites "extrêmes" utilisent des réseaux de convolution ou des *MLP* générés aléatoirement et non entraînés afin de réduire la dimension des données d'entrée [110]. La plupart de travaux utilisant ces approches en *RL* concernent cependant des environnements avec des espaces d'état de petite dimension comme CartPole [111] ou encore un problème de contrôle de bateaux [112]. Cependant, on trouve des travaux où l'*ELM* s'est avéré efficace dans sur des données image comme pour la détection active d'objets, réduisant les coûts de calcul et stabilisant le processus d'optimisation de leurs méthodes de gradient de la politique [113].

Réseaux pré-entraînés

Le développement du machine learning a permis la création de gros réseaux de neurones pré-entraînés sur différents ensembles de données. Ces réseaux peuvent être utilisés comme extracteurs de caractéristiques d'images pour entraîner un agent d'apprentissage par renforcement. Le réseau *ResNet* est utilisé pour extraire des représentations visuelles dans des tâches complexes de manipulation [114], [115].

Dans une étude sur plusieurs réseaux pré-entraînés [116], le *CNN* avec des poids initiaux figés obtient des performances équivalentes à l'état de l'art, remettant en question la pertinence des méthodes entrainées de bout-en-bout en *RL*. La simplicité des caractéristiques extraites par un *CNN* aléatoire semble être favorable à l'apprentissage de tâches de *RL*. Cette observation a contribué à orienter nos recherches vers l'étude d'extracteurs de caractéristiques qui ne nécessitent pas ou peu d'apprentissage préalable et qui extraient des caractéristiques éparses.

Conclusion

Dans ce chapitre, nous avons examiné différentes approches de résolution de tâches d'apprentissage par renforcement à partir de données image. Nous avons observé que les techniques couramment utilisées dans l'état de l'art pour traiter les environnements

partiellement observables reposent souvent sur l'apprentissage de représentations d'état. Ces représentations sont généralement obtenues en utilisant des réseaux de convolution profonds, ce qui peut être coûteux en termes de calcul et instable lors de l'entraînement. Nous avons également exploré des méthodes alternatives permettant d'extraire des caractéristiques, telles que l'*ELM* ou les *RBFN*. Nous avons vu que les *RBFN* ont des capacités intéressantes pour le *RL*, qu'on ne retrouve pas implicitement dans les méthodes utilisant les *CNN*, comme la localité et la rareté des activations, ou encore leur simplicité d'utilisation et l'interprétation de leurs activations. Ces capacités semblent intéressantes dans l'application de méthodes de *RL*.

Dans le prochain chapitre, nous présenterons notre modèle d'extraction de caractéristiques, le *VRBFN*. Nous décrirons en détail son architecture, son fonctionnement, ainsi qu'une étude comparative de ses performances par rapport à d'autres méthodes d'extraction de représentation.

Chapitre 4

Représentation visuelle à base radiale pour du RL

Introduction

Dans ce chapitre, nous introduisons le *Visual Radial Basis Function Network* (*VRBFN*) ainsi que les résultats des différentes études menées sur ce réseau. Dans le chapitre précédent, nous avons montré l'efficacité des méthodes de *SRL* appliquées au *RL*, cependant nous avons aussi établi certaines limites dues au pré-entraînement. D'un autre côté nous avons vu que des extractions aléatoires, issues d'un *CNN* initialisé aléatoirement par exemple, peuvent être utilisées pour le *RL*. Les résultats de l'initialisation aléatoire remettent en question la nécessité de l'entraînement des extracteurs en *SRL*. Notre *VRBFN* est en partie basé sur cette idée. Nous pensons que des extractions aléatoires de bas niveau permettraient un apprentissage plus efficace de la politique dans les scénarios visuels à la première personne. De plus, compte tenu des avantages importants de l'activation éparse dans le domaine du *RL*, nous avons décidé d'orienter notre recherche vers un extracteur doté de fonctions à activations éparses. Ainsi, nous avons opté pour les fonctions radiales qui permettent d'avoir des activations de bas niveau localisées dans l'espace d'entrée.

La première partie de ce chapitre sera consacrée à une présentation détaillée du modèle, mettant en évidence ses propriétés. Puis nous évaluerons notre modèle en comparaison avec différentes méthodes de l'état de l'art dans différentes tâches d'apprentissage par renforcement.

4.1 Présentation du modèle

Le modèle d'extraction de représentation visuelle proposé est inspiré des RBFN présentés précédemment. Il est ainsi constitué d'une couche d'entrée, d'une couche cachée de neurones à base radiale et d'une (ou plusieurs) couches de neurones complètement connectées comme illustré dans le schéma 4.1. On considère de plus qu'à la différence des *RBFN* classiques, chaque neurone de la couche cachée a ici une activation localisée sur l'image, sous la forme d'une zone d'attention gaussienne. L'extracteur visuel est ainsi composé de multiples neurones à base radiale localisés sur des zones de l'espace de l'image.

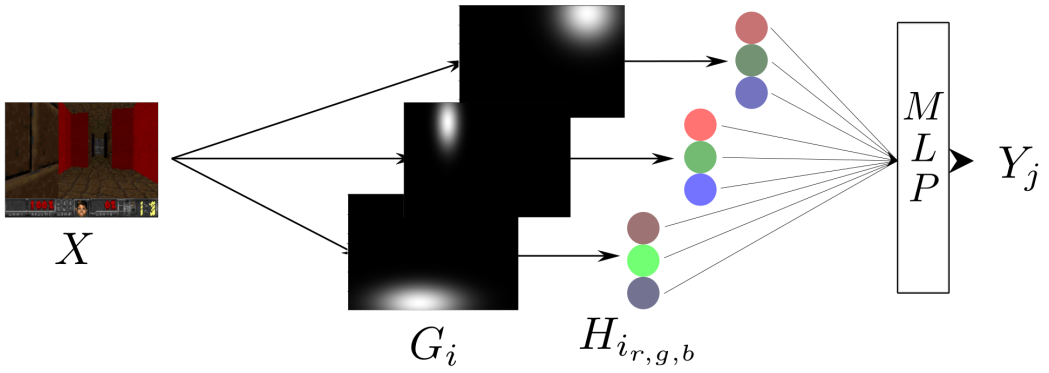


Figure 4.1: Le réseau *VRBFN* fonctionne de la manière suivante : une image $\mathbf{X}$ est multipliée terme à terme par différents filtres gaussiens $\mathbf{G_i}$ qui jouent le rôle de zones d'attention. Une fonction radiale $H_{i_{r,g,b}}$ est appliquée sur les différents canaux de couleurs pour chaque image pondérée. Les sorties sont prédites par un *MLP* à partir des stimulis ainsi créés.

Plus formellement, pour une image $\mathbf{X} \in \mathbb{R}^{w \times h \times c}$, la fonction permettant le calcul du stimulus $h_i(.)$ d'un neurone i est donnée par :

$$h_i(\mathbf{X}) = \exp\left(-\frac{\sum_{x,y=0}^{w-1,h-1}((\mathbf{X}(x,y) - \boldsymbol{\mu}_{i,z})^2 \odot \tilde{\mathbf{G}}_\mathbf{i}(x,y))}{2 \times \boldsymbol{\sigma}_z^2}\right) \quad \tilde{\mathbf{G}}_\mathbf{i}(x,y) = \frac{\mathbf{G_i}(x,y)}{\sum_{x,y=0}^{w-1,h-1} \mathbf{G_i}(x,y)} \tag{4.1}$$

L'équation $h_i : \mathbb{R}^{w \times h \times c} \to [0,1]^c$ correspond au calcul du stimulus pour le neurone i. L'extraction est proche de 0 si l'intensité pondérée de la zone observée ne correspond pas au centre d'intensité, et proche de 1 si la correspondance est parfaite. Dans cette équation $\boldsymbol{\mu}_{i,z} \in \mathbb{R}^c$ sont les moyennes et $\sigma_z \in \mathbb{R}$ l'écart type de la fonction d'activation à base radiale du neurone i. L'opérateur $\odot$ représente le produit d'*Hadamard*, c'est-à-dire une multiplication terme à terme entre deux matrices. Le noyau gaussien $\mathbf{G}_i$ permet de pondérer la fonction h_i sur une partie de l'image et ainsi localiser spatialement les activations. Il est défini pour chaque pixel de coordonnées $x \in [0, w-1]$ et $y \in [0, h-1]$ tel que :

$$\mathbf{G}_i(x,y) = \exp\left(-\left(\frac{(\tilde{x} - \mu_{i,x})^2}{2 \times \sigma_{i,x}^2} + \frac{(\tilde{y} - \mu_{i,y})^2}{2 \times \sigma_{i,y}^2}\right)\right) \tag{4.2}$$

Dans cette équation, $\mu_{i,x}, \mu_{i,y} \in [0,1]$ représentent les centres, et $\sigma_{i,x}, \sigma_{i,y} \in [0,1]$ les écart-types de la gaussienne. On a $\tilde{x} = x/w$ et $\tilde{y} = y/h$ les coordonnées normalisées des pixels. Une simplification a été apportée à la modélisation des gaussiennes en

fixant la covariance à 0, ce qui produit des gaussiennes uniquement verticales ou horizontales. On notera également que ces noyaux gaussiens sont normalisés.

Par ailleurs nous utilisons ici la même valeur d'écart-type σ_z pour chaque neurone afin d'assurer une homogénéité dans les résultats des neurones. Si la sortie de deux neurones est la même, cela signifie qu'il y a la même distance entre le centre d'activation et les pixels sous la zone d'attention. Cette valeur correspond à la tolérance de l'activation du neurone : plus elle est proche de 1, plus l'activation sera forte et souvent saturée, même s'il y a peu de pixels proches du centre. À l'inverse, une valeur proche de 0 aura pour conséquence d'activer le neurone uniquement lorsque les pixels sous la zone d'attention correspondent parfaitement au centre. Cette valeur est empiriquement fixée à 0.1 dans nos expériences.

4.2 Evaluations préliminaires avec le Deep Q-learning

Après avoir présenté le réseau *VRBFN*, nous allons maintenant étudier ses différentes caractéristiques et son comportement dans des scénarios de *RL*. Dans un premier temps nous avons choisi d'utiliser le Q-learning comme algorithme de *RL* pour sa simplicité. De plus, nous avons décidé d'utiliser les deux scénarios présentant le moins de complexité en termes d'observabilité partielle dans l'environnement *ViZDoom*. Nous commencerons par présenter les caractéristiques des scénarios étudiés, puis nous examinerons la paramétrisation et le comportement de notre réseau dans la résolution des tâches de RL. Nous conclurons cette première étude par une évaluation de l'apprentissage des tâches avec notre méthode, en la comparant à deux méthodes de l'état de l'art : un *CNN* et une méthode classique de *SRL*, la reconstruction de l'observation (Ro) avec un auto-encodeur.

4.2.1 Les scénarios

4.2.1.1 VizDoom Basic

Il s'agit du scénario le plus simple de *VizDoom*. L'agent se trouve dans une pièce rectangulaire, avec un monstre en face de lui, au fond de la pièce. Le but de l'agent est de tirer sur le monstre. L'agent apparaît toujours au même endroit, représenté par le point vert en bas de l'illustration 4.2, tandis que le monstre apparaît de manière aléatoire sur la ligne rouge. Dans le scénario original, l'agent peut uniquement effectuer les actions de se déplacer à droite, à gauche et de tirer, ce qui l'oblige également à rester sur sa ligne. Afin d'augmenter la difficulté du scénario, nous avons

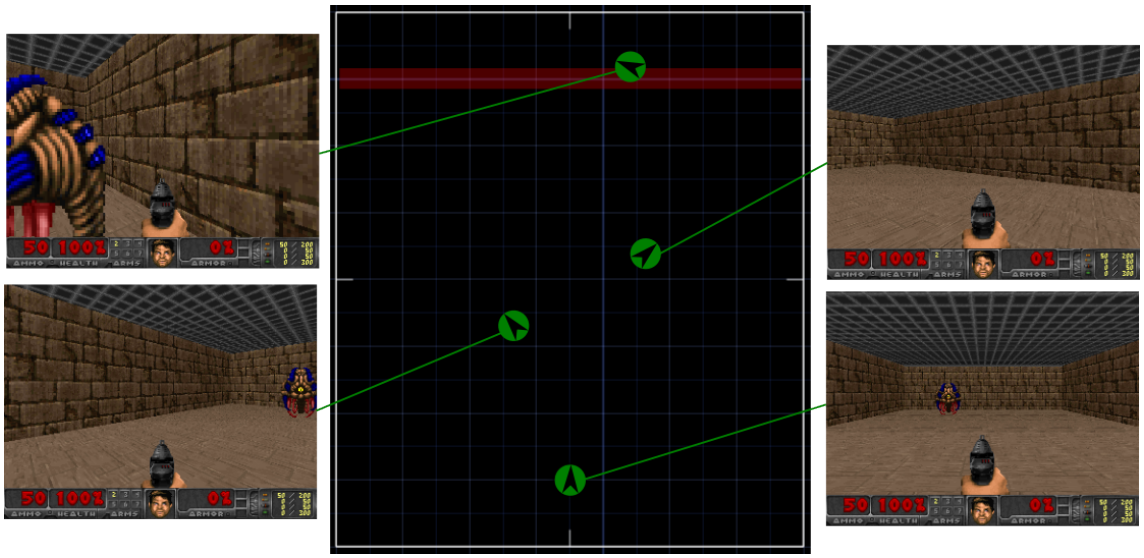


Figure 4.2: Environnement *Basic* de *VizDoom*. La carte du scénario est au centre de l'image, chaque point vert représente une position d'agent regardant dans la direction de la flèche au centre du cercle, les images résultantes sont données sur les côtés.

ajouté les actions d'avancer, de reculer et de tourner (à droite/gauche). Cela permet d'avoir des observations où le monstre n'est pas visible et donc une observabilité partielle. Dans ce scénario, les récompenses sont distribuées comme suit : -5 pour un tir manqué, $+101$ pour un tir réussi et -1 pour toute autre action exécutée. L'épisode se termine après 200 itérations ou lorsque l'agent a tué le monstre.

4.2.1.2 Health Gathering

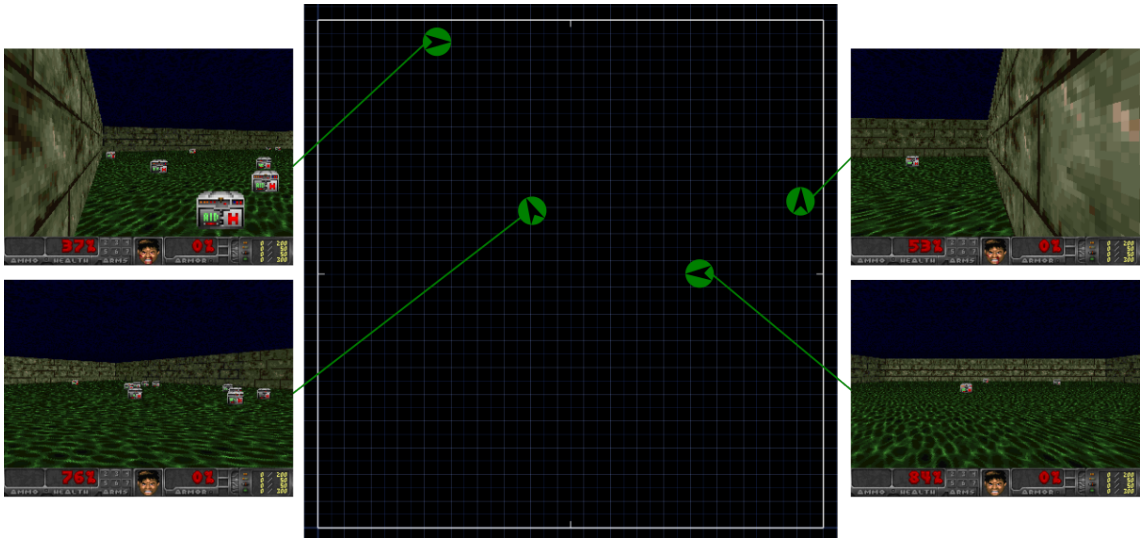


Figure 4.3: Environnement *health gathering* de *VizDoom*.

Dans ce scénario l'agent doit récupérer des packs de soin pour survivre. Il apparaît au centre d'une pièce carrée avec 100 points de vie, et le sol brûlant fait perdre des points de vie à l'agent. Les packs de soin visibles sur les images de l'illustration 4.3 apparaissent de manière aléatoire. L'agent peut avancer, reculer, tourner à droite ou à gauche. La récompense initiale du scénario est basée sur le nombre d'itérations que l'agent a survécues, et cette récompense n'est obtenue qu'au moment de la mort de l'agent ou après avoir atteint 2100 actions consécutives. Ce type de récompense très éparse et qui, de plus, ne peut pas être prédite (car l'agent n'a pas l'information sur le nombre d'actions exécutées) est difficile à optimiser. Les méthodes de l'état de l'art essayant de résoudre ce type de scénario ajoutent à l'observation la vie de l'agent ou redéfinissent la récompense en attribuant la différence de vie de l'agent avant et après l'action : $r = life_t - life_{t-1}$ [68] . Ainsi, si l'agent prend un pack de soin, il recevra une récompense positive, sinon il sera pénalisé, nous avons aussi choisi cette récompense pour ce scénario.

4.2.2 Choix des paramètres

Pour résoudre un scénario de *RL* nous devons définir certain paramètres du Qlearning et de notre *VRBFN*. Dans le contexte d'une étude préliminaire nous avons décidé d'utiliser uniquement une couche entièrement connectée à Ng neurone gaussiens, ce qui nous permet d'approximer linéairement la fonction Q :

$$Q(s,a) = \sum_{i=0}^{Ng-1} h_i(s) \cdot w_{ai}, \tag{4.3}$$

où $h_i(s)$ est l'activation du neurone i et w_{ai} est le poids entraînable reliant la valeur Q de l'action a au stimulus du neurone i.

Le réseau peut être exprimé comme une fonction paramétrée par $[Ng, \boldsymbol{\mu}_x, \boldsymbol{\mu}_y, \boldsymbol{\mu}_z, \boldsymbol{\sigma}_x, \boldsymbol{\sigma}_y, \sigma_z, \mathbf{W}]$. Seuls les paramètres du *MLP* $\mathbf{W}$ seront optimisés par le *Q-learning*. Les centres sont uniformément tirés entre 0 et 1. Afin d'obtenir le maximum d'informations localisées, il est nécessaire de couvrir le plus de zones possible. En effet, si nous n'avons pas assez de neurones ou si les zones d'attentions sont trop petites, l'image ne sera pas entièrement recouverte, ce qui entraînera une perte d'information excessive comme pour les premières lignes et colonnes de la figure 4.4 où différents paramétrages ont été testés et la répartition spatiale des récepteurs a été modélisée. Pour la suite nous avons choisi une répartition uniforme des centres $\sigma_{x,y}$ entre 0.02 et 0.2.

Nous avons proposé d'apprendre à résoudre la tâches du scénario *basic* avec une couche linéaire. Nous avons donc effectué plusieurs tests de 50000 itération d'apprentissage avec le *Q-learning* sans stratégies d'exploration et sans réseau cible. Nous avons testé différents paramètres : le nombre de neurones gaussiens N, le taux d'apprentissage

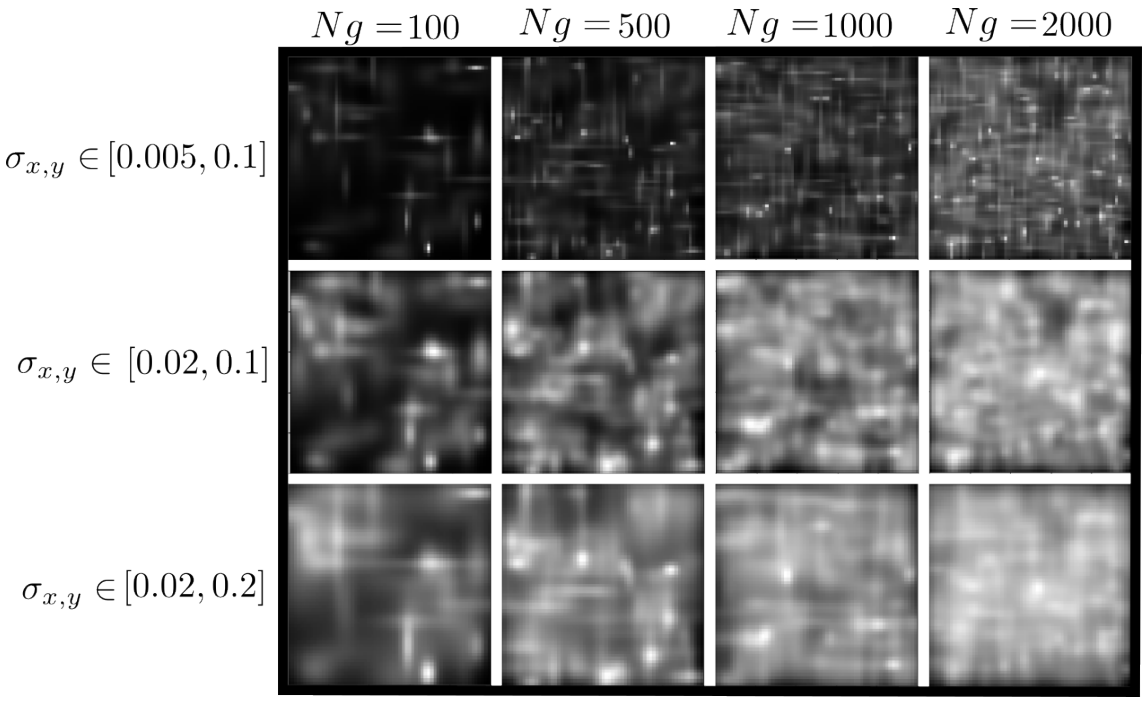


Figure 4.4: Distribution aléatoire des noyaux gaussiens pour différentes valeurs de $\sigma_{x,y}$ et de N. Les images représentent la somme des G_i, plus un pixel est blanc plus il est recouvert par des noyaux gaussiens.

α et la taille du batch bs. Les résultats en termes de récompense sur le scénario *basic* sont donnés dans l'illustration du tableau de résultats de la figure 4.5. Les résultats positifs sont représentés en vert. Quel que soit le nombre de neurones, les meilleurs résultats sont obtenus avec un taux d'apprentissage élevé et une taille de batch importante. Il est important de noter que ces résultats ne sont pas généralisables, car il faudrait effectuer plus de tests avec différentes initialisations. Cependant, ils indiquent une direction d'entraînement. Ces résultats sont assez évidents, car nous n'utilisons pas de réseau cible. Ainsi, afin d'obtenir une bonne première approximation, il est préférable d'optimiser avec une grande taille de batch.

Ces premiers résultats sur l'apprentissage d'une tâches à partir d'extraction du *VRBFN* valident l'utilité des extractions. En effet les scores maximum obtenus étant proche du comportement parfait (score entre 90 et 100 en fonction de la position du montre) nous pouvons considérer que les extractions du *VRBFN* permettent une bonne approximation de la fonction Q avec un taux d'apprentissage élevé et de grandes tailles de batch. Pour les expérimentations des parties suivantes nous avons choisi σ_z à 0.1, $\boldsymbol{\sigma}_x, \boldsymbol{\sigma}_y \in [0.02, 0.2]$. Nous avons fixé le nombre de neurones à 2001 pour les expérimentation en noir et blanc (1 canal) et à 667 pour celles avec des images *RGB* (3 canaux) ce qui permet d'avoir le même nombre de stimulis dans les deux cas d'études.

500 Ng

bs\α	0.001	0.005	0.01	0.05
8	-244 ± 184	-195 ± 203	-311 ± 61	-92 ± 177
64	-199 ± 232	-153 ± 207	-253 ± 180	-134 ± 195
256	-300 ± 1	-136 ± 203	-17 ± 153	-131 ± 183
512	-156 ± 213	-300 ± 0	-178 ± 183	61 ± 74
1024	-225 ± 219	-146 ± 210	-61 ± 151	18 ± 95

1000 Ng

bs\α	0.001	0.005	0.01	0.05
8	-208 ± 213	-144 ± 203	-126 ± 199	-16 ± 152
64	-173 ± 209	-173 ± 206	-114 ± 200	35 ± 35
256	-122 ± 208	-300 ± 1	48 ± 64	38 ± 76
512	-187 ± 190	18 ± 120	36 ± 114	66 ± 42
1024	-107 ± 191	-34 ± 181	26 ± 85	77 ± 18

3000 Ng

bs\α	0.001	0.005	0.01	0.05
8	-129 ± 191	-140 ± 199	-43 ± 161	-113 ± 174
64	-93 ± 198	-84 ± 207	47 ± 101	77 ± 19
256	-105 ± 190	25 ± 126	80 ± 15	56 ± 64
512	-123 ± 186	72 ± 34	80 ± 14	86 ± 7
1024	-9 ± 153	61 ± 72	49 ± 24	86 ± 7

Figure 4.5: Résultats pour différents paramétrages d'agents sur le scénario *basic*, les récompenses moyennes positives sont surlignées en vert, celles négatives en rouge. Ng, bs, α correspondent respectivement au nombre de neurones gaussiens, à la taille du batch et au taux d'apprentissage.

4.2.3 Analyse qualitative des activations

Nous avons analysé les activations de notre extracteur dans les scénarios de l'étude avec les paramètres donnés dans la partie précédente. Tout d'abord, nous avons étudié la localité puis l'activité des neurones, et enfin leur interprétabilité. Dans ces études, nous distinguons les neurones actifs et inactifs. Nous avons estimé qu'un neurone apportait une information si son activation était supérieure à $0,01$ pour au moins un état du scénario. Pour ce faire, nous avons étudié les activations sur 10000 états obtenus par un comportement aléatoire de l'agent.

Localité des extractions

Étant donné la localité de l'extracteur dans l'espace de l'image, tant en termes de position que d'intensité, nous avons étudié la différence d'activation des neurones entre deux images du même scénario et entre les deux scénarios. Sur la figure

4.6, à gauche (A), nous avons les états S et S' sur lesquels nous avons appliqué notre extraction, et à droite (C), les histogrammes des différences d'activation entre les neurones actifs (aN) et les neurones inactifs (iN) des extractions de S et S'. Nous avons calculé la distance $\Delta_N = |\Delta_N(VRBFN(S)) - \Delta_N(VRBFN(S'))|$, où N correspond soit aux neurones actifs (aN), soit aux neurones inactifs (iN). La colonne du milieu (B) montre la différence entre les images, à laquelle nous avons ajouté les zones d'attention des neurones qui diffèrent; l'intensité des zones d'activation correspond aux valeurs de Δ_N.

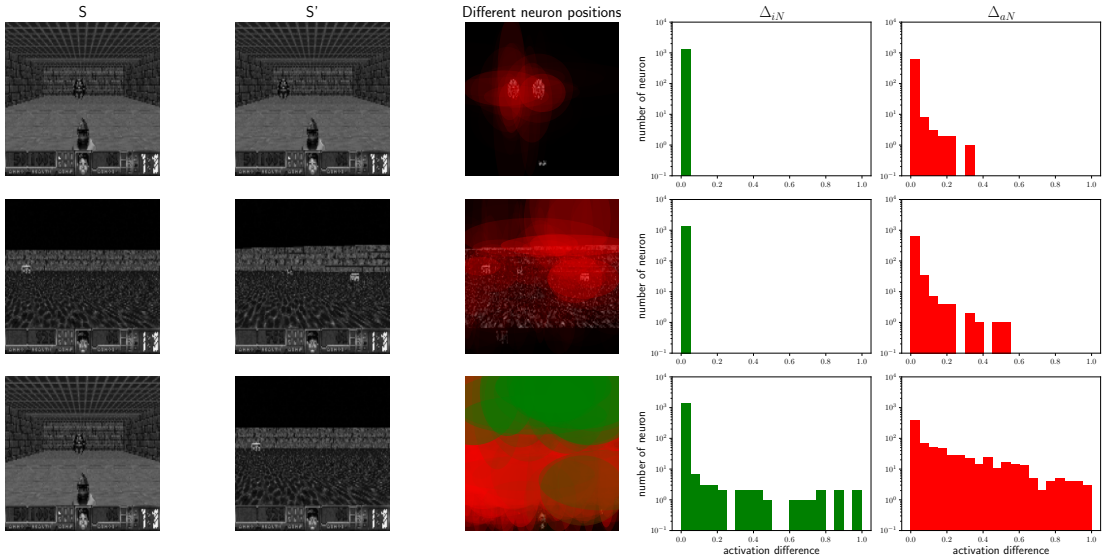


Figure 4.6: Différences d'activation entre deux images différentes S et S'. La colonne du milieu représente les différences localisées sur l'image, les histogrammes montrent la répartition des différences pour les neurones actifs (en rouge) et inactifs (en vert).

Sur cette illustration 4.6, nous pouvons observer deux choses. Premièrement, les différences d'activation entre l'image S et S' sont bien localisées. En effet les représentations (B) montrent que pour le scénario *basic* sur la première ligne, les neurones qui diffèrent se trouvent sur le monstre, tandis que pour le scénario *health gathering* de la deuxièmes ligne, ils se situent sur les packs de soins et le mur à l'arrière-plan. Deuxièmement, sur la derniere ligne, la comparaison des activation entre les deux scénario montre que les neurones inactifs (histogramme en vert) d'un environnement ne sont pas les mêmes que ceux d'un autre environnement, et les neurones actifs (histogramme en rouge) subissent davantage de changements. Ainsi, nous aurons des motifs d'activation différents entre différents environnements et des intensités d'activation différentes pour les mêmes scénarios. Cette localité des extractions est encourageante, car elle montre que le réseau a des activations spécifiques à chaque scénario et les représente de manière distincte.

Activité

Nous avons quantifié les activité des neurones en moyennant la quantité de neurones actifs (Na) sur 1000 images provenant des deux scénarios. Cette moyenne a été calculée sur 20 initialisations différentes du *VRBFN*. Les résultats sont présentés dans le tableau 4.1, où l'on peut observer que le nombre de Na est stable entre les scénarios et avec différentes initialisations.

	Basic		Health gathering	
	gray	RGB	gray	RGB
Neurones actifs (Na)	$32 \pm 1\%$	$31 \pm 1\%$	$29 \pm 6\%$	$32 \pm 5\%$

Table 4.1: Moyenne sur mille images sur vingt initialisations de réseaux des neurones actifs par scénario.

Ces résultats montrent que seulement un tiers des neurones du réseau sont activés sur les différents scénarios. De plus, avec les résultats sur la localité, nous pouvons affirmer que ce ne sont pas les mêmes neurones qui s'activent entre les différents scénarios. Cette activité éparse (activation spécifique aux scénarios) des neurones est importante, car elle permet au réseau d'avoir des motifs d'activation différents pour différentes images.

Interprétation des extractions

Dans l'illustration 4.7, nous avons étudié un neurone actif pour chaque scénario, avec *health gathering* en haut et *basic* en bas. Les deux neurones présentent des schémas d'activation différents. En reliant leurs activations à la position et à l'orientation de l'agent (flèche directionnelle bleu), nous avons remarqué que dans *health gathering*, le neurone à une activation supérieur au seuil orange lorsque l'agent n'a pas de mur devant lui et lorsque le sol se trouve sous sa zone d'attention. En revanche, dans *basic*, l'activité du neurone est supérieure au seuil lorsque l'agent a un mur à sa gauche et lorsque le mur est situé sous sa zone d'attention.

On peut constater que ces neurones fournissent individuellement des indications sur la position de l'agent. De plus, un neurone qui s'active uniquement en présence d'un mur sera désactivé en présence du monstre, ce qui indique un changement pouvant être utile pour résoudre la tâche. On peut donc imaginer qu'une combinaison linéaire de ces activations pourrait permettre à l'agent de connaître sa position dans l'environnement et ainsi de choisir les bonnes actions pour accomplir la tâche.

Cette analyse qualitative met en évidence la capacité du réseau à extraire des informations spatiales et directionnelles pertinentes pour l'accomplissement de la

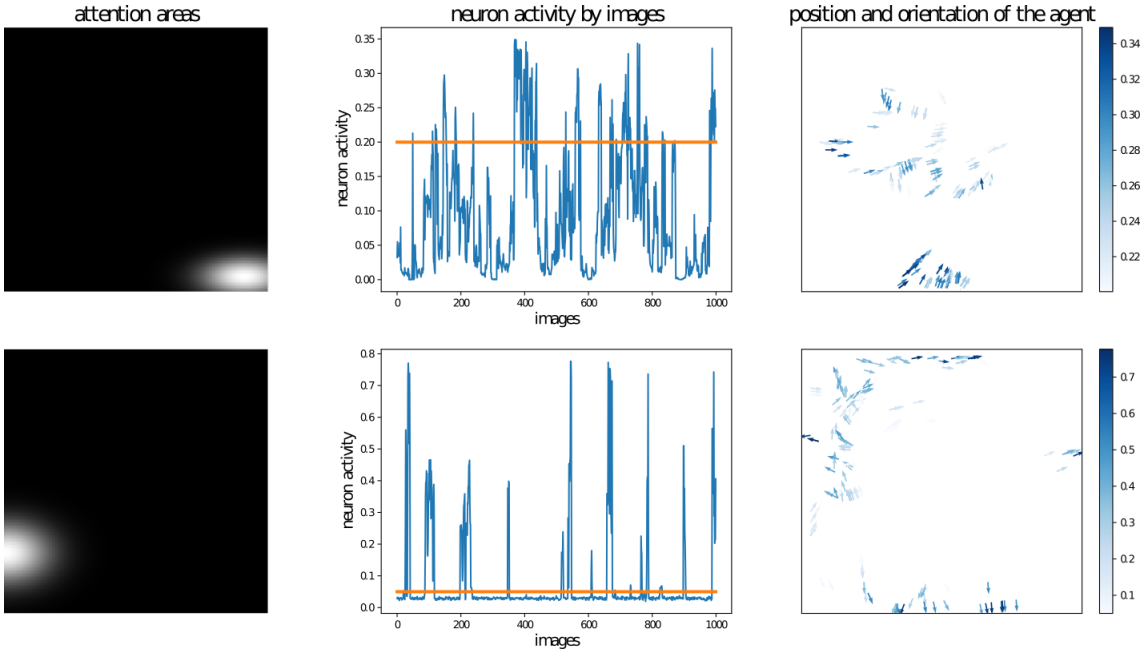


Figure 4.7: Analyse des activations sur différentes images de deux neurones dans les scénarios *health gathering* en haut et *basic* en bas. La barre orange des images de la deuxième colonne représente un seuil. Les positions et orientations des troisièmes images sont celles des images qui activent le neurone au-dessus du seuil.

tâche, en utilisant des neurones spécifiques à chaque scénario. Ces résultats peuvent être étendus aux autres neurones actifs avec des interpretations différentes, par exemple des activations sur le monstre ou le plafond ou les packs de soin.

4.2.4 Apprentissage des tâches de *RL*

Nous voulons à présent évaluer les capacités de notre représentation visuelle par *VRBFN* dans des tâches de *RL* simples, avec l'algorithme de *Q-learning*. Nous comparons notre réseau à deux méthodes de l'état de l'art : un entraînement de bout-en-bout avec un *CNN* et une méthode de *SRL* de type reconstruction de l'observation avec un autoencodeur. Chaque méthode sera entraînée avec des images en noir et blanc, $\mathbf{X} \in \mathbb{R}$, ainsi qu'en couleur ($RGB$) afin d'évaluer les capacités de passage à l'échelle.

Protocole expérimental

L'architecture des différentes méthodes utilisées est décrite ci-dessous :

- *CNN* : Le réseau se compose de trois couches de convolution et d'un *MLP* avec deux couches entièrement connectées. Le réseau contient $142k$ poids entraînables.
- *AE* : Nous entraînons un autoencodeur (AE) pour reconstruire l'observation afin d'obtenir une représentation compacte. Cette représentation est ensuite utilisée pour approximer une fonction à l'aide d'un *MLP* à deux couches. L'AE est entraîné sur $200k$ états du scénario obtenus avec des actions répétées aléatoirement entre 1 et 10 fois pendant $100k$ époques (1 époque correspond à un entraînement sur les 200k données). Après l'entraînement de l'autoencodeur, un *MLP* est entraîné à résoudre la tâche à partir de l'espace latent de l'AE. Le modèle comprend $131k$ paramètres.
- *VRBFN* : Nous utilisons un *MLP* pour l'entraînement de l'agent à partir des extractions à base radiale. Les paramètres du *VRBFN* sont ceux décrits dans la partie 4.2.2. Le réseau compte $148k$ paramètres.

Les différentes méthodes ont été entraînées pendant $100,000$ itérations d'optimisation sur 20 initialisations différentes, en utilisant des images de taille 120×160. De plus, nous avons répété l'action choisie 6 fois pour accélérer l'entraînement. Le *Deep Q-learning* requiert un réseau cible pour réduire les effets liés au *bootstrapping* 2.2.4.5, ainsi qu'une politique d'exploration ϵ-*greedy* pour découvrir l'environnement au début de l'entraînement. Pour les réseaux *CNN* et *AE*, nous avons utilisé des paramètres généraux qui fonctionnent dans de nombreux autres environnements, disponibles dans le tableau 4.4.

Paramètres	Optimiseur	π	α	bs	it_{up}
CNN et *AE*	*Adam*	$\epsilon-$greedy	0.00025	64	$10,000$
VRBFN	*Adam*	greedy	0.01	256	$\emptyset$

Table 4.2: Paramètres pour l'algorithme de *Deep Q-Learning*. π correspond au type de politique choisie, α au taux d'apprentissage, bs à la taille du batch et it_{up} au nombre d'itérations avant la mise à jour du réseau cible.

Dans cette étude, nous prenons $\epsilon_{start} = 1$, $\epsilon_{min} = 0.1$, $\epsilon_{size} = 10,000$, où ϵ_{size} définit le nombre d'itérations avant d'atteindre ϵ_{min}. Pour le *VRBFN*, nous n'utilisons pas de réseau cible et la politique est *greedy*, c'est-à-dire que l'agent choisit la meilleure Q-value.

Scenario	basic		health gathering	
	gray	rgb	gray	rgb
basic-DQN	**86 ± 6**	**86 ± 6**	1519 ± 751	1092 ± 716
Autoencoder-DQN	86 ± 7	85 ± 7	601 ± 357	438 ± 384
VRBFN	85 ± 8	85 ± 9	**1809 ± 543**	**1778 ± 573**
VRBFN (aN)	85 ± 8	84 ± 9	1799 ± 571	1690 ± 635

Table 4.3: Moyenne et écart type des récompenses obtenues par les différentes méthodes pour les deux scénarios testés avec et sans couleur. La dernière ligne correspond aux réseaux *VRBQN* auxquels nous avons supprimé les neurones inactifs par scénario.

Résultats des apprentissages

L'apprentissage des trois différentes méthodes est illustré à la Figure 4.8 avec les courbes de récompense par épisode obtenues dans les deux scénarios. Sur ces courbes, nous pouvons observer que le scénario *basic* n'a posé aucun problème aux trois méthodes, contrairement au scénario *health gathering* qui est plus difficile et partiellement observable.

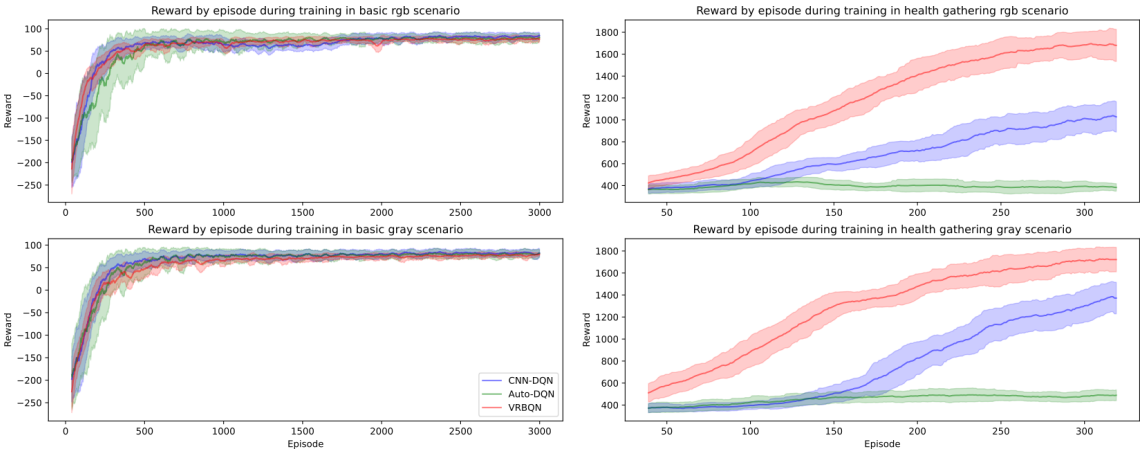


Figure 4.8: Courbes d'apprentissage des trois méthodes, *CNN-DQN*, Auto-*DQN* et *VRBQN*, sur les deux scénarios avec et sans couleurs. [7]

Le deuxième scénario, *health gathering*, est plus difficile à résoudre pour le *CNN* et l'*AE*, qui n'ont pas convergé dans le cadre de notre étude. Les deux méthodes convergent vers des comportements qui trouvent des packs de soins, mais qui se retrouvent bloqués dans les coins de la carte. Notre agent ne rencontre pas ce problème car certains neurones s'activent uniquement lorsque l'agent est contre un mur ou dans un coin, ce qui permet d'optimiser ces neurones pour des actions spécifiques permettant de s'écarter du mur, sans pour autant modifier les poids des autres neurones responsables de la localisation des packs de soins.

Après avoir effectué les entraînements, nous avons testé les réseaux optimisés sur 1000 épisodes. Les résultats des tests sont présentés dans le tableau 4.3. En termes de convergence, nous constatons une nette supériorité de notre méthode sur le scénario *health gathering*. De plus, nous remarquons que le passage à l'échelle, du noir et blanc à la couleur, ne perturbe pas les résultats de notre méthode contrairement aux *CNN* et à l'*AE* pour *health gathering*. Lors du passage à l'échelle, notre méthode garde le même nombre de stimuli mais présente trois fois moins de noyaux gaussiens dans le cas de l'image *RGB*. Les autres méthodes, lors du passage en *RGB*, multiplient par trois le nombre de poids de leurs premières couches convolutives, augmentant ainsi le nombre de paramètres à optimiser, ce qui prend plus de temps et peut expliquer la différence de résultats.

Pour finir, lors des tests, nous avons évalué le *VRBQN* avec uniquement les neurones actifs (*VRBQN*(aN)), et nous avons observé que les résultats n'étaient pas trop dégradés, malgré une réduction du réseau par trois en termes de nombre de neurones. Ce résultat indique que les neurones inactifs ne sont pas nécessaires à la résolution d'une tâche dans un scénario spécifique. Cependant, il est indispensable de garder à l'esprit que ces neurones inactifs pourraient être actifs dans un autre scénario ou lors d'une modification du scénario, permettant ainsi d'ajouter de nouvelles informations sur les nouveaux éléments de l'environnement.

Ces premiers résultats nous ont poussés à aller plus loin dans l'étude du *VRBFN*. En effet, les scénarios choisis ne reflétaient pas suffisamment la difficulté des environnements partiellement observables, mais nous ont cependant permis de valider la conception de notre méthode. De plus, l'algorithme choisi, le deep *Q-learning*, ne permettait pas d'affirmer que nos extractions pouvaient être utilisées par d'autres méthodes, par exemple celles issues de la branche *on-policy*. Enfin, nous n'avons pas réussi à résoudre les autres scénarios présentés au début de ce chapitre avec le *Q-learning*. Nous avons donc réalisé une étude comparative plus poussée avec la méthode *PPO* et comparé notre extracteur à d'autres méthodes de l'état de l'art, comme présenté dans la partie suivante.

4.3 Apprentissage de scénarios partiellement observables avec *PPO*

Dans cette partie, nous présentons une étude comparative de différents modèles de représentation visuelle sur des scénarios d'apprentissage par renforcement plus complexes que *basic* et *health gathering*, en utilisant l'algorithme *PPO*. Les environnements choisis sont plus complexes en termes d'observabilité partielle. De plus, ces scénarios sont difficilement résolus avec les méthodes *vanilla*.

Nous commencerons par présenter les différents scénarios et réseaux utilisés ainsi que les paramètres de *PPO*. Les résultats des apprentissages seront ensuite illustrés et commentés. À la fin de cette partie, nous présenterons des travaux sur la robustesse au bruit de nos activations ainsi que sur leurs activations partielles.

4.3.1 Scénarios de l'étude

4.3.1.1 Defend the center (*DTC*)

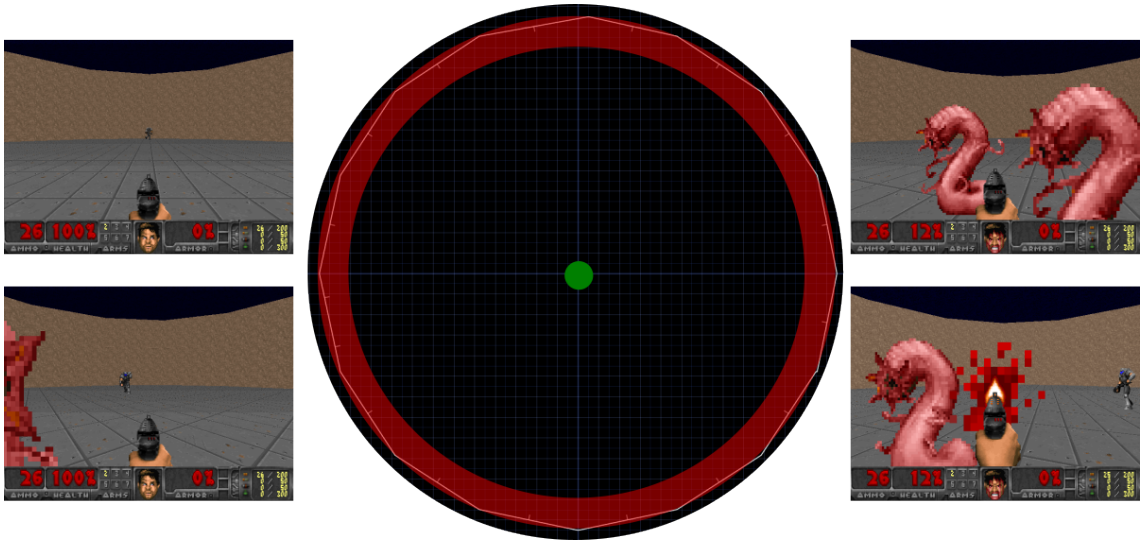


Figure 4.9: Illustration du scénario *defend the center*.

Dans ce scénario, un agent est placé au centre d'une pièce circulaire, représentée par un cercle vert dans le schéma 4.9. Des ennemis apparaissent aux extrémités de la pièce, représentées par les zones rouges. Le but de l'agent est d'éliminer les ennemis à l'aide de son arme. Il dispose de 26 munitions. Les monstres attaquent l'agent de deux manières : les monstres roses attaquent au corps à corps, comme on peut le voir sur les images de gauche, tandis que les tireurs en gris avancent vers l'agent pour lui tirer dessus à courte portée. L'épisode prend fin lorsque l'agent meurt, ou lorsqu'il manque de munitions.

Dans cet environnement, la partie non observable des états est très importante, car l'agent peut être attaqué par un monstre venant de derrière. La seule information visuelle disponible pour savoir quand l'agent est attaqué est le nombre de points de vie en bas à gauche. Cependant, il est très difficile pour un agent en *RL* de récupérer des informations écrites sur une image sans apriori. De plus, le nombre de munitions, écrit à côté de la vie, étant limité, l'agent ne doit pas gaspiller de munitions. Les récompenses sont distribuées de la manière suivante : $+1/20$ pour chaque ennemi tué, -1 pour la mort de l'agent. Le maximum de récompense qu'un agent peut obtenir dans ce scénario est de $0,30$.

4.3.1.2 Health gathering supreme (*HGS*)

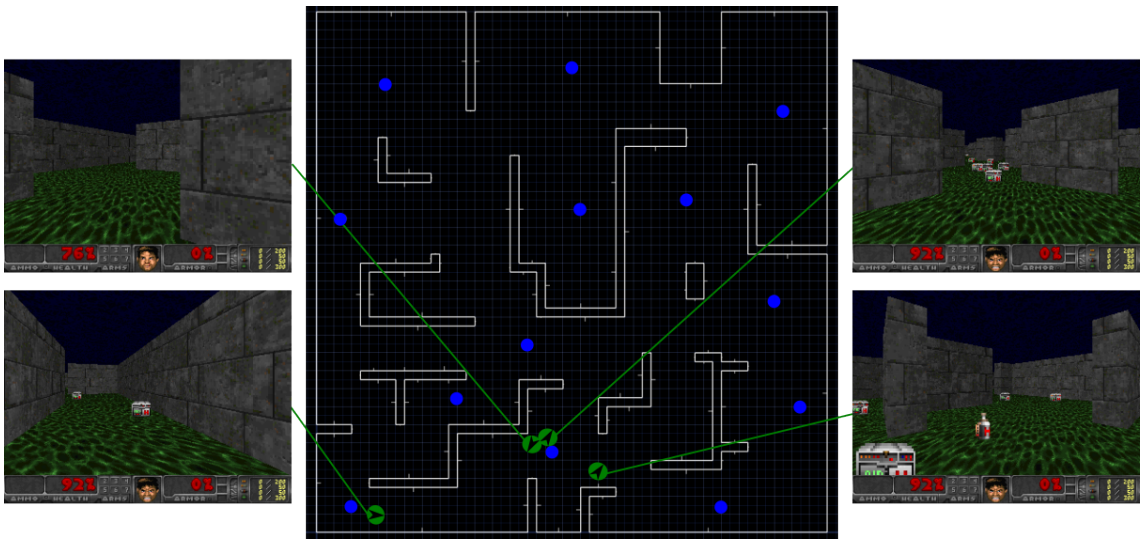


Figure 4.10: Illustration du scénario *health gathering supreme*.

Ce scénario est une version plus difficile de *health gathering*, dans lequel la pièce est un labyrinthe, comme schématisé dans l'illustration 4.10. L'agent apparaît aléatoirement sur l'un des points bleus avec une orientation aléatoire, et doit, comme dans *HG*, récupérer des packs de soin. De plus, il y a des fioles de poison qui infligent des dégâts à l'agent. La difficulté de ce scénario réside dans le déplacement de l'agent. Comme on peut le voir sur les images, il peut se retrouver dans des endroits sans pack de soins, dans des impasses, et les fioles doivent être différenciées des packs de soins.

De plus, l'agent n'a pas directement ses points de vie affichés, ceux-ci sont écrits à l'écran, ce qui rend difficile pour l'agent de différencier les fioles des packs de soins en termes de récompense à long terme. Les récompenses sont les suivantes : $+1/500$ pour chaque itération et -1 si l'agent meurt. L'épisode se termine lorsque l'agent meurt ou effectue 2100 actions.

4.3.1.3 My way home (*MWH*)

L'agent se trouve dans un labyrinthe où chaque pièce a des couleurs et des textures différentes, comme illustré dans l'image 4.11. Le but est de trouver l'armure représentée en jaune et visible dans l'image en haut à gauche. L'agent apparaît aléatoirement sur les points bleus avec une orientation aléatoire. L'épisode se termine après 2100 actions. L'agent reçoit une récompense de $+1$ s'il trouve l'armure. Sinon, pour chaque action effectuée, l'agent subit une pénalité de -1. Ce scénario est un défi pour l'exploration et la mémoire, car l'agent doit apprendre le chemin vers l'armure quel que soit son point de départ. L'observabilité partielle de ce scénario réside dans

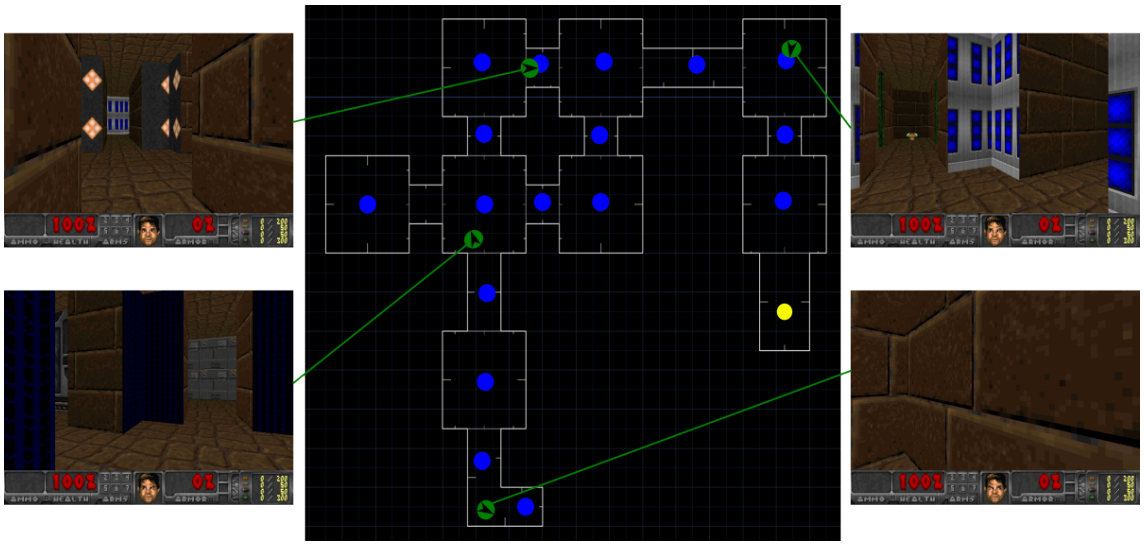


Figure 4.11: Illustration du scénario *My way home*.

l'observation qui n'informe pas l'agent sur sa position. Même si chaque pièce est différente et permet de construire une bonne estimation, les transitions entre les pièces sont les mêmes, comme le mur marron visible dans l'image en bas à gauche, ce qui rend l'apprentissage difficile.

4.3.2 Méthodes de comparaison

Dans cette étude, nous avons comparé six méthodes d'extraction de caractéristiques basées sur les quatre réseaux représentés dans l'illustration 4.12. De plus, un agent aléatoire sera également intégré à l'étude afin de vérifier que les méthodes sont plus intelligentes qu'un comportement aléatoire. Voici une description détaillée des méthodes que nous allons comparer avec notre $VRBFN$(IV) :

- *CNN* end-to-end (CNN) (I) : Ce réseau se compose de trois couches de convolution et d'un MLP (Multilayer Perceptron) comprenant deux couches entièrement connectées. Il totalise $142k$ poids entraînables et son temps d'entraînement moyen sur cinq seeds pour un million d'itérations dans le scénario *defend the center* est de $1h22$.

- *CNN* aléatoire (CNN_{rdm}) (I) : Cette méthode, inspirée du concept du Deep *ELM*, utilise une initialisation aléatoire des poids des couches de convolution. Seuls les $131k$ poids du *MLP* sont entraînables pour un temps d'entraînement moyen de $1h02$.

- *CNN* $\mathbb{L}_{rec}$ (CNN_{auto}) (II) : Dans cette approche, nous utilisons une perte de reconstruction $\mathbb{L}_{auto} = MSE(\hat{S}, S)$, similaire à celle de l'auto-encodeur, mais optimisons cette fonction d'erreur simultanément avec l'algorithme d'apprentissage

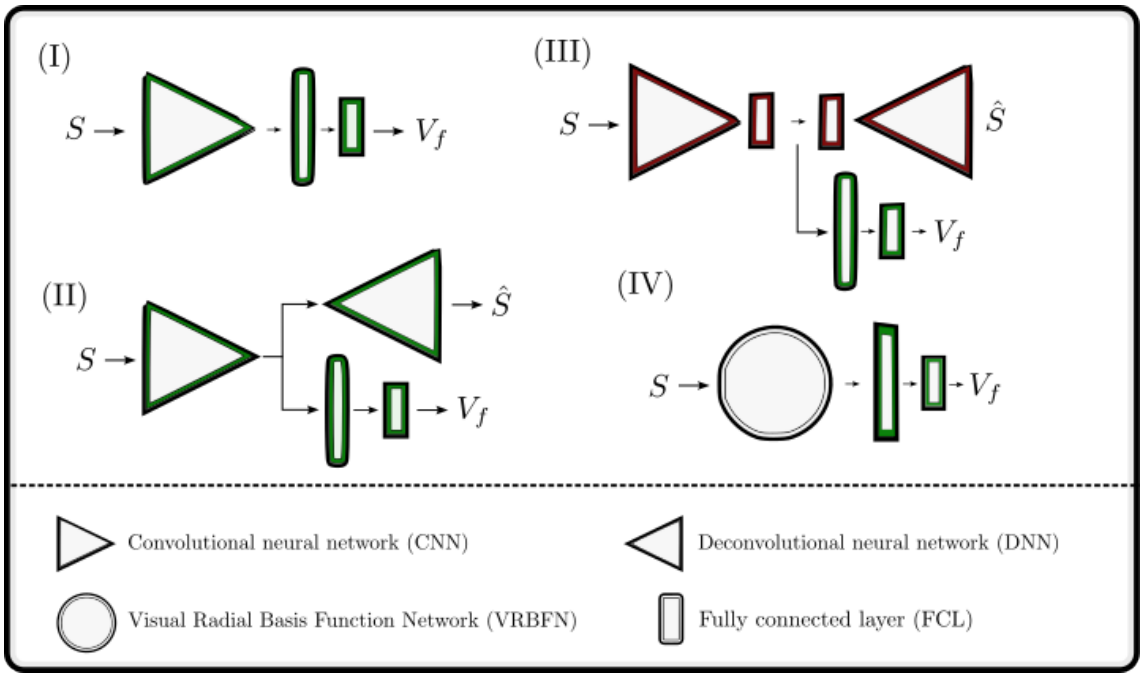


Figure 4.12: Schéma des différents réseaux utilisés dans ce chapitre. Les réseaux en vert sont des réseaux entraînés de bout en bout, ceux en rouge sont pré-entraînés. Les modules sans couleur représentent un réseau non entraîné. S correspond à l'état et V_f à une fonction de valeur.

par renforcement. Le réseau compte $209k$ poids entraînables pour un temps moyen de $1h22$.

- Auto-encodeur (*Auto*) (III) : Nous entraînons un *AE* à reconstruire l'observation afin d'obtenir une représentation compacte. Cette représentation est ensuite utilisée pour approximer une fonction à l'aide d'un *MLP* à deux couches. L'*AE* est entraîné sur $200k$ états du scénario obtenus avec des actions répétées aléatoirement entre 1 et 10 fois pendant $100k$ époques (1 époque correspond à un entraînement sur les $200k$ données). Après l'entraînement de l'autoencoder, un *MLP* est entraîné à résoudre la tâche à partir de l'espace latent de l'*AE*. Le modèle comprend $131k$ poids entraînables pour un temps moyen de $1h36$.

- *ResNet* (*RESnet*) (III) : Nous utilisons l'encodeur du *ResNet18* [27] pré-entraîné sur le dataset *ImageNet* comme extracteur pour résoudre les tâches de renforcement. Cette méthode est similaire à celle de l'auto-encodeur, mais le pré-entraînement n'a pas été réalisé sur *VizDoom*. Le *ResNet* a été entraîné sur des images courantes et réalistes pour la classification. Nous utilisons un *MLP* à deux couches pour résoudre le problème de renforcement qui compte $131k$ paramètres entraînables en $1h52$ en moyenne.

- Visual Radial Basis Function Network ($VRBFN$) (IV) : Nous utilisons un MLP pour l'entraînement de l'agent à partir des extractions à base radiale. Le réseau compte $148k$ paramètres entraînables pour un temps moyen de $1h02$.

Les CNN utilisés dans les modèles I, II et III ont des couches de convolution de tailles $(32, 64, 64)$, avec des noyaux et des pas de tailles $((8, 4), (3, 2), (3, 2))$. Le décodeur de l'AE est une symétrie de l'encodeur utilisant des couches de convolution transposée afin de reconstruire une image de la même taille que celle donnée en entrée. Chaque couche des réseaux est suivie d'une activation $ReLU$, sauf pour les couches de sortie et pour l'espace latent des AE. Nous avons choisi d'utiliser deux réseaux distincts pour l'acteur et le critique. Dans toutes les expériences présentées dans la suite, le $VRBFN$ contient 1002 neurones avec $\sigma_{x,y} \in [0.005, 0.1]$ et $\sigma_z = 0.1$.

Nous avons utilisé l'algorithme PPO pour chacune de ces méthodes. PPO requiert une approximation de la politique avec l'acteur et de la fonction valeur avec le critique, il est donc préférable d'utiliser deux réseaux distincts. Ainsi, pour la partie commande, chaque méthode entraîne deux réseaux en parallèle, l'acteur et le critique.

Pour chaque réseau, nous avons utilisé les mêmes paramètres pour l'algorithme PPO disponibles dans le tableau 4.4, de plus chaque méthode a été testée avec 5 initialisations différentes pour chaque scénario. Entraîner 5 *seeds* pour chaque méthode sur le scénario *defend the center* représente un temps d'entraînement total de $41h20$, ce qui explique pourquoi nous avons restreint les entraînements à cinq *seeds* et à un million d'itérations ($124h$ pour les trois scénarios de *VizDoom*). Les entraînements ont été effectués sur le Mésocentre de l'Université de Clermont Auvergne avec un GPU NVIDIA V100.

`paramètres`	lr_a	lr_c	mbs	rb	K	ϵ_{clip}
`PPO`	0.0001	0.001	100	$10,000$	40	0.2

Table 4.4: Paramètres pour l'algorithme de PPO. lr_a est le taux d'apprentissage pour le réseau de l'acteur, lr_c pour le critique, mbs représente la taille des mini-batchs, rb la taille de la mémoire, K le nombre de boucles d'apprentissage et ϵ_{clip} est un paramètre spécifique à PPO.

4.3.3 Résultats comparatifs des entraînements avec *PPO*

Nous présenterons les courbes d'apprentissage par scénario, puis les résultats des tests pour les trois scénarios. Sur les courbes présentées dans ce chapitre, le nombre d'épisodes diffère entre les différentes méthodes car chaque entraînement a été effectué sur un million d'itérations, et donc en fonction du succès de l'agent, le

nombre d'épisodes atteints pendant un entraînement varie. Par exemple, pour *DTC*, un agent qui a fait plus d'épisodes est moins bon qu'un autre agent avec moins d'épisodes pour le même nombre d'itérations, car un agent fera beaucoup d'épisodes s'il meurt rapidement. Ce raisonnement peut varier en fonction des scénarios : dans *HGS* et *DTC*, moins il y a d'épisodes, meilleur est l'agent, et dans *MWH*, c'est l'inverse. Les courbes des récompenses obtenues pendant les apprentissages pour chaque méthode sont présentées dans les figures 4.13, 4.14, et 4.15 pour les scénarios *DTC*, *HGS*, et *MWH*, respectivement.

Résultats comparatifs sur le scénario *DTC*

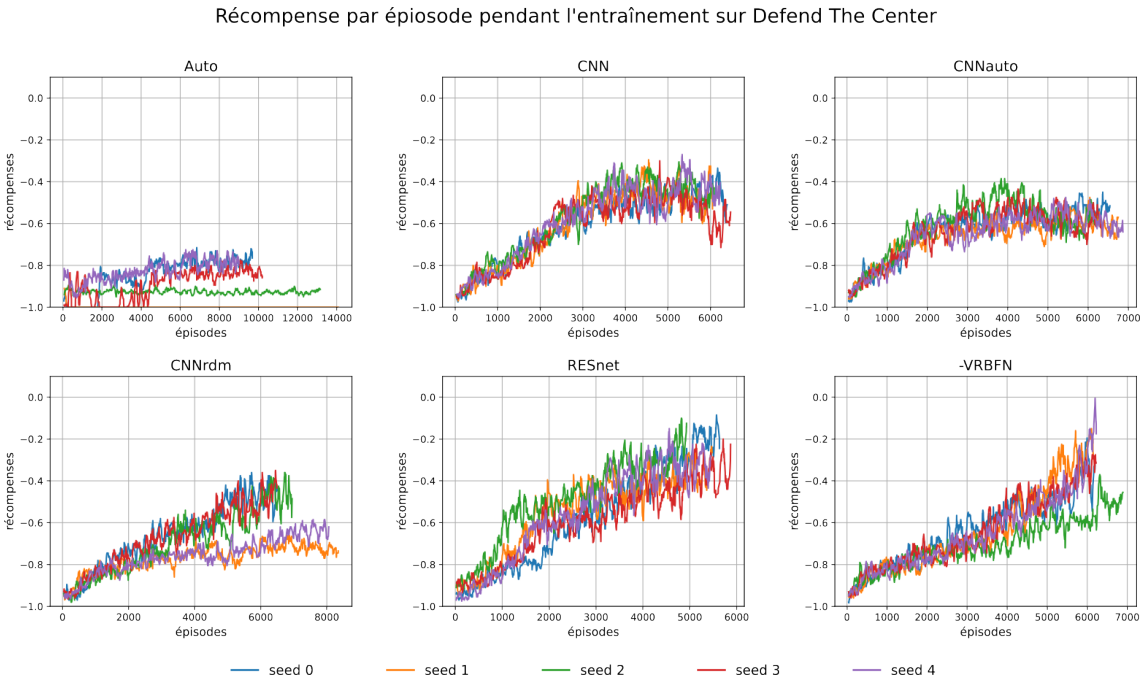


Figure 4.13: Courbes d'apprentissage par seed du scénario *defend the center*.

Dans ce scénario, seul l'agent *Auto* n'arrive pas à converger vers un comportement optimal, comme vous pouvez le voir dans les courbes d'entraînement de la figure 4.13. Les agents CNN_{auto} et CNN_{rdm} ont des récompenses similaires, à l'exception de deux initialisations du CNN_{rdm} qui n'atteignent pas les -0,4. Les autres initialisations du CNN_{rdm} semblent s'améliorer durant l'entraînement, tandis que le CNN et le CNN_{auto} stagnent aux alentours des trois mille épisodes. Les deux méthodes qui dominent ce scénario sont le $VRBFN$ et le *RESnet*. Le *RESnet* converge un peu plus rapidement et de manière plus régulière, tandis que le $VRBFN$ met plus de temps avant d'améliorer ses récompenses, mais le fait de manière plus rapide (la pente est plus forte). Dans notre méthode, les activations sont locales, et donc certaines initialisations peuvent favoriser un type de comportement. Par exemple, dans ce scénario, une bonne répartition des gaussiennes sur l'axe horizontal au milieu

de l'image serait plus favorable pour détecter les monstres au loin et proche. Nous montrerons dans le chapitre suivant comment la distribution des noyaux gaussiens de notre $VRBFN$ peut améliorer l'apprentissage de l'agent.

Résultats comparatifs sur le scénario *HGS*

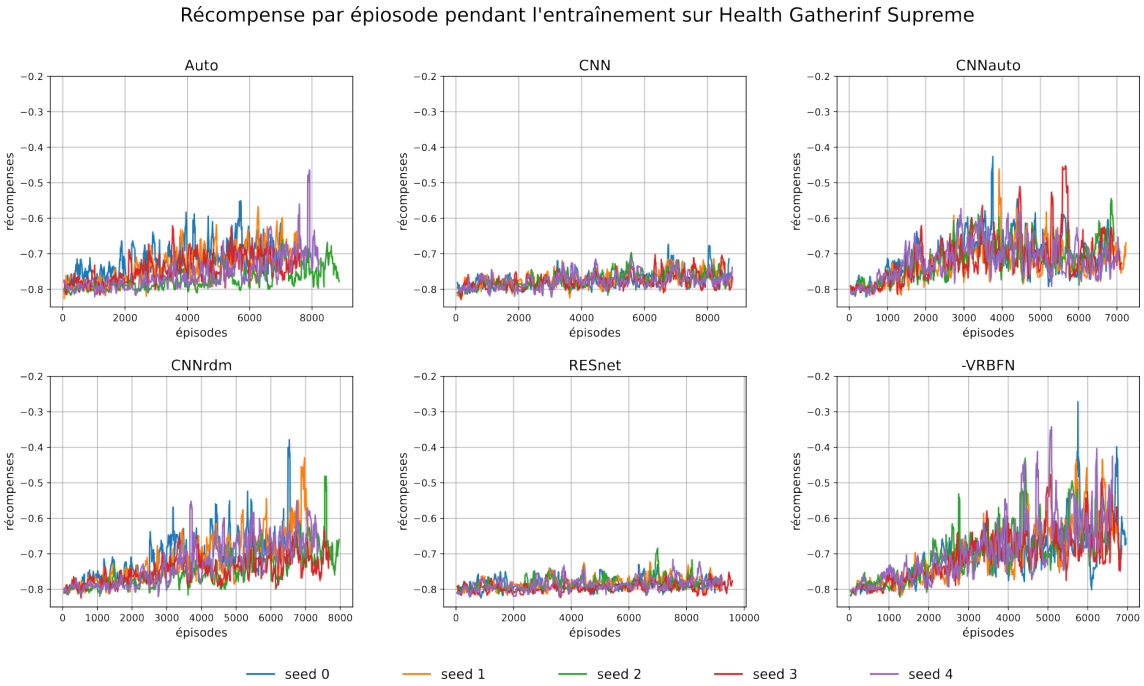


Figure 4.14: Courbes d'apprentissage par seed du scénario *health gathering supreme*.

La difficulté de ce scénario, en plus de l'espace d'observation qui est partiellement observable, réside dans la distribution des récompenses. En effet, l'agent reçoit des récompenses positives pour chaque action et une récompense négative lorsqu'il meurt. Ce type de récompense est difficile à optimiser en *RL*. Par exemple, si l'agent prend une fiole, la récompense instantanée sera identique à celle obtenue lorsqu'il trouve un pack de soin. La seule différence entre la fiole et le pack de soin est la vie après avoir effectué l'action, ce qui n'est pas donné à l'agent. Pour optimiser les résultats de ce scénario, certains ont choisi d'intégrer la vie soit comme observation, soit comme récompense. Dans notre cas, nous voulions tester les différents réseaux dans les conditions initiales du scénario, sans ajout d'information. Ce scénario n'a pas été résolu parfaitement par les agents, comme l'indiquent les courbes d'apprentissage de la figure 4.14. Cependant, les agents CNN_{rdm} et $VRBFN$ parviennent tout de même à atteindre des scores satisfaisants. Sur certaines initialisations, le $VRBFN$parvient même à obtenir la meilleure récompense (2100) sur la plupart des épisodes. Les agents CNN et $RESnet$ ont quant à eux les pires entraînements.

Résultats comparatifs sur le scénario *MWH*

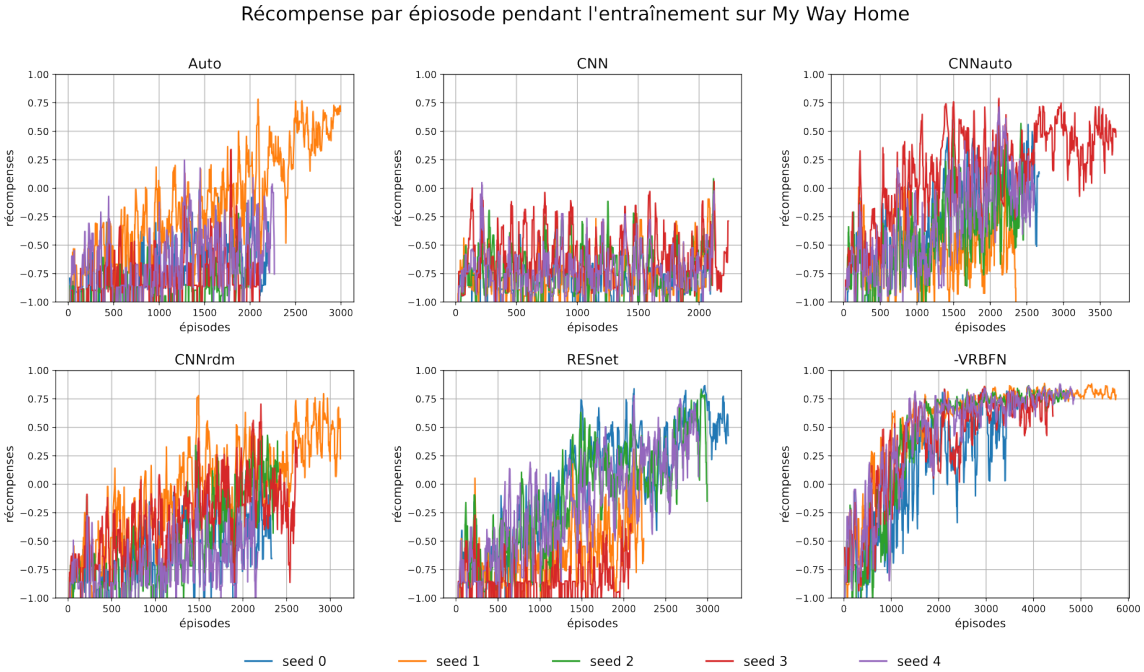


Figure 4.15: Courbes d'apprentissage par seed du scénario *my way home*.

Les courbes présentées dans la figure 4.15 montrent les performances de différents modèles d'apprentissage par renforcement sur *MWH*. Le modèle $VRBFN$ se distingue par ses performances élevées et rapides, atteignant des récompenses stables dès environ 1000 épisodes. CNN_{auto} et $RESnet$ montrent également de bonnes performances, mais nécessitent plus d'épisodes pour atteindre une stabilisation. Les modèles *Auto* et CNN_{rdm} améliorent leurs performances de manière plus progressive, avec une grande variabilité entre les différentes initialisations. En revanche, le modèle CNN semble avoir des difficultés, avec des récompenses majoritairement négatives ou proches de zéro tout au long de l'entraînement.

Résumé des entraînements

Les courbes d'apprentissage peuvent être résumées par le graphique 4.16, qui indique la moyenne des récompenses obtenues par épisode sur les différents ensembles de données. Ces résultats d'entraînement montrent l'efficacité de notre méthode dans différents scénarios, notamment dans ceux où l'agent doit se déplacer.

4.3.4 Analyse des performances de test avec *PPO*

Une fois les entraînements effectués, les réseaux optimisés ont été testés sur mille épisodes pour chaque scénario. Les résultats sont répertoriés dans le tableau 4.5.

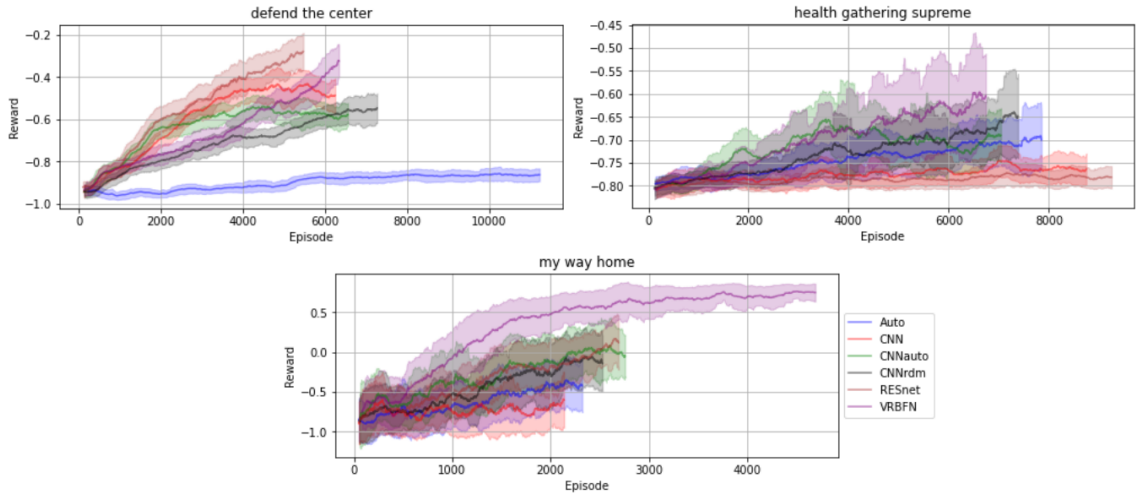


Figure 4.16: Moyenne des courbes d'apprentissage par seed avec la déviation standard représentée par la zone plus claire pour les six méthodes sur les trois scénarios.

Les scores obtenus par les différents agents lors des phases de test montrent que

	`defend the center`	`health gathering supreme`	`my way home`
CNN	10.5 ± 4.8	351 ± 119	-0.07 ± 0.37
CNN_{auto}	9 ± 3.2	392 ± 166	0.19 ± 0.55
CNN_{rdm}	10 ± 5.4	510 ± 310	0.23 ± 0.57
$RESnet$	16.5 ± 3.7	312 ± 65	0.23 ± 0.57
$Auto$	2 ± 2.4	414 ± 199	-0.03 ± 0.43
$VRBFN$	$\mathbf{18.7 \pm 3.7}$	$\mathbf{744 \pm 435}$	$\mathbf{0.75 \pm 0.47}$
Rdm	0 ± 0.93	308 ± 62	-0.07 ± 0.38

Table 4.5: Résultats moyens pour cinq initialisations différentes des tests effectués sur mille épisodes pour chacune des méthodes.

le $VRBFN$ obtient les meilleures performances en termes de récompenses. Les autres méthodes présentent des performances variables en fonction des scénarios. Les résultats du CNN_{rdm} sont supérieurs ou similaires à ceux du CNN et du CNN_{auto}, ce qui montre que l'encodeur convolutif utilisé dans le CNN et dans l'$Auto$ est instable pendant la formation et que le guider pour reconstruire l'état ne suffit pas à stabiliser l'extraction. De cette observation, on peut dire que les méthodes CNN et CNN_{auto} n'ont pas assez de temps pour apprendre une bonne représentation et une politique en un million d'itérations. Cela est mis en évidence par les résultats du $RESnet$, qui a eu plus de temps d'apprentissage pour la partie encodeur sur des données générales, ce qui permet à l'extraction d'être plus efficace et générique que celle obtenue avec le CNN, le CNN_{auto} et l'$Auto$. Le $VRBFN$ obtient les

meilleurs résultats dans les trois scénarios testés. Une des raisons de la supériorité du *VRBFN* pourrait être sa capacité à localiser des éléments dans les images ; ses activations locales et éparses aident à différencier les différents éléments du scénario pour une meilleure prise de décision.

Comportement des agents dans *DTC*

Les comportements des agents dans ce scénario varient en fonction des méthodes utilisées, mais on observe un schéma général qui consiste à tourner dans une direction et à tirer lorsque qu'un monstre est au centre de l'écran. Les différences résident dans la distance à laquelle les agents tirent. Par exemple, les agents CNN et CNN_{auto} tentent de tirer sur les monstres dès leur apparition, mais ce comportement nécessite une grande précision et les tirs sont souvent manqués. Les meilleures initialisations de ces deux agents adoptent plutôt un tir à courte et moyenne distance, ce qui semble plus efficace pour obtenir des récompenses.

L'agent CNN_{rdm} attend que les monstres se rapprochent le plus près de lui pour les attaquer. Bien que cette approche semble intéressante car elle réduit le risque de rater la cible et donc de gaspiller des munitions limitées, l'agent attend parfois trop longtemps, se fait déborder par les monstres et finit par mourir.

Les deux agents les plus performants, le *RESnet* et le $VRBFN$, tirent sur les monstres à des distances proches et moyennes, évitant ainsi les problèmes rencontrés par le CNN et le CNN_{rdm}. En revanche, l'agent utilisant l'autoencodeur ne parvient pas à converger vers un comportement correct. Il tourne toujours dans la même direction en tirant au hasard, et dans certains cas, il ne tourne même pas du tout.

Comportement des agents dans *HGS*

Les agents $VRBFN$ et CNN_{rdm} montrent une capacité à sortir d'un couloir sans issue, à se retourner lorsqu'ils sont dans un coin et parfois à éviter les fioles de poison. Les agents CNN et *RESnet* parviennent à collecter quelques packs de soin, ce qui montre qu'ils ont compris la tâche, mais ils finissent souvent dans un couloir sans issue ou dans un coin sans pouvoir en sortir. En revanche, les deux derniers agents, *Auto* et CNN_{auto}, parviennent à récupérer des packs de soin, mais lorsqu'ils sont confrontés à plusieurs groupes de packs de soin ou à différents couloirs, ils perdent trop de temps à osciller entre deux actions, ce qui entraîne leur mort.

Comportement des agents dans *MWH*

Dans ce scénario, les agents CNN et *Auto* rencontrent des difficultés à se déplacer dans le labyrinthe ; ils finissent souvent coincés dans une pièce ou sur un mur. De plus, lorsque l'agent démarre l'épisode près de la récompense, ces agents parviennent rarement à l'obtenir, ce qui montre qu'ils ont du mal à comprendre la tâche. En revanche, pour CNN_{rdm}, *RESnet* et CNN_{auto}, l'agent est moins souvent bloqué et parvient à récupérer quelques récompenses, ce qui lui permet de mieux appréhender la tâche. Cependant, aucun de ces agents ne parvient à trouver le chemin lorsque l'agent apparaît au point le plus éloigné du scénario.

La seule méthode qui réussit la tâche pour la plupart des initialisations est le $VRBFN$. Pour ses meilleures initialisations, l'agent ne reste pas coincé dans les murs, peut facilement retrouver l'armure et se remet d'avoir pris une mauvaise direction.

4.3.5 Robustesse au bruit

Lorsqu'on travaille avec des images, tester l'adaptabilité des agents au bruit est une approche importante. On peut ajouter du bruit théorique aux observations, comme le bruit gaussien ou le bruit poivre et sel, mais dans nos travaux, nous avons examiné la résistance des agents au bruit induit par les changements de taille d'images. Des tests sur des bruits classiques ont été réalisés, disponibles dans l'annexe A.2, montrant la robustesse de notre méthode. Ce type de bruit est généré par la transformation bilinéaire utilisée pour changer la taille des images, ce qui peut être un problème récurrent lorsqu'on tente de transférer les apprentissages sur différents capteurs ou simulateurs. Par exemple, dans Vizdoom, les observations visuelles peuvent avoir des dimensions différentes d'un autre simulateur. Pour l'apprentissage sur *Vizdoom*, nous avons choisi des observations de taille 160×120 que nous avons redimensionnées en 64×64 pour accélérer l'apprentissage.

Dans un premier temps, nous avons comparé les résultats des méthodes sur des images deux fois plus grandes, de taille 320×240, que nous avons ensuite redimensionnées en 64×64. Dans ces conditions, lorsque nous comparons les observations obtenues lors d'un épisode de DTCpar exemple, nous obtenons une *MSE* de 0.01 entre les deux types d'images. Pour les méthodes d'extraction, nous pouvons également regarder la MSE des extractions sur les deux types d'images. Pour l'auto-encodeur, elle est de 7.53, pour le *RESnet* on obtient 0.52, pour le CNN_{rdm} elle est de 0.04, et pour le $VRBFN$ elle est de 0.003. On constate rapidement que le $VRBFN$ est le plus robuste au bruit causé par le changement de taille d'image, tandis que les méthodes d'apprentissage de représentation souffrent de ce bruit.

Afin de quantifier l'influence du bruit, nous avons testé les méthodes sur 100 épisodes avec des observations bruitées pour chacun des trois scénarios. Les résultats de ce test

sont disponibles dans le tableau 4.6. Le $VRBFN$ est nettement plus résistant à ce type de bruit que les autres méthodes. Dans *MWH*, par exemple, où il n'y a pas trop de petits détails, la *MSE* des images est d'environ 0.006, et donc les performances du $VRBFN$ restent inchangées. En revanche, pour les autres méthodes, même un bruit insignifiant entraîne une baisse significative des performances.

methodes	defend the center	health gathering supreme	my way home
CNN	1.01 ± 1.75 (-90.38%)	348 ± 103 Ø	-0.10 ± 0.31 Ø
CNNauto	0.33 ± 1.03 (-96.33%)	342 ± 101 Ø	0.06 ± 0.49 (-68%)
CNNrdm	5.03 ± 3.19 (-49.7%)	501 ± 279 (-1.76%)	0.1 ± 0.51 (-56.52%)
RESnet	6.19 ± 5.83 (-62.48%)	338 ± 101 Ø	0.09 ± 0.51 (-60.87%)
Auto	-0.07 ± 0.91 (-103.5%)	311 ± 80 (-24.88%)	-0.21 Ø
VRBFN	$\mathbf{14.69 \pm 6.79}$ (-21.44%)	$\mathbf{698 \pm 417}$ (-6.18%)	$\mathbf{0.78 \pm 0.43}$ ($+4\%$)

Table 4.6: Résultats des tests sur cent épisodes dans les trois scénarios pour les différentes méthodes avec des images bruitées par un changement de taille. Les scores représentent la moyenne et l'écart-type sur les différentes initialisations. De plus, pour chaque score, un pourcentage représentant la perte par rapport au test original est calculé. Pour les scores qui ne sont pas significativement différents de la méthode aléatoire, le pourcentage n'est pas calculé car il serait peu informatif ; il est remplacé par Ø.

4.3.6 Passage à l'échelle

Nous allons à présent tester le *RESnet* et le $VRBFN$ face à une augmentation de la taille des images d'entrée. Nous avons choisi de comparer ces deux méthodes car elles ne nécessitent aucune modification architecturale pour passer à l'échelle. En effet, une méthode utilisée dans le *RESnet* consiste à réduire la taille de l'espace latent à une dimension fixe à l'aide de couches de pooling, tandis que le $VRBFN$ est par définition extensible à différentes tailles d'images.

Dans le tableau 4.7, nous constatons que la méthode de réduction de l'espace latent utilisée par le *RESnet* ne permet pas de maintenir les mêmes performances lors

d'un passage à l'échelle, contrairement à notre réseau $VRBFN$, qui s'adapte aux différentes tailles d'images sans trop dégrader les performances de l'agent.

taille d'image	methods	defend the center	health gathering supreme	my way home
64×64	$VRBFN$	$\mathbf{18.7 \pm 3.7}$	$\mathbf{744 \pm 435}$	$\mathbf{0.75 \pm 0.47}$
	$RESnet$	16.5 ± 3.7	312 ± 65	0.23 ± 0.57
128×128	$VRBFN$	$\mathbf{17.16 \pm 5.33}$	$\mathbf{777 \pm 444}$	$\mathbf{0.69 \pm 0.50}$
	$RESnet$	-0.12 ± 1.10	300 ± 50	-0.16 ± 0.21
256×256	$VRBFN$	$\mathbf{16.62 \pm 3.18}$	$\mathbf{710 \pm 431}$	$\mathbf{0.51 \pm 0.57}$
	$RESnet$	-0.71 ± 0.65	306 ± 57	-0.18 ± 0.17

Table 4.7: Résultats des tests sur cent épisodes dans les trois scénarios pour les méthodes $VRBFN$ et $RESnet$ avec des images de taille supérieure aux images d'entraînement.

4.3.7 Activation partielle

Nous avons constaté dans la section 4.2.3, sur les scénarios Basic et HG, que jusqu'à 30% des neurones ne s'activaient jamais. Comme dans l'étude précédente, nous avons testé le $VRBFN$ en supprimant ces neurones inactifs. Les résultats sont répertoriés dans le tableau 4.8. Dans les scénarios DTC et HGS, nous observons approximativement le même pourcentage de neurones inactifs, car ces scénarios ont des couleurs bien définies et relativement restreintes. En revanche, dans le scénario MWH, il y a moins de neurones inactifs, car ce scénario présente plus de variations de coloration et de texture.

Dans les deux premiers scénarios, les performances sont améliorées lorsque les neurones inactifs sont supprimés. Même si ces neurones inactifs ont de petites valeurs d'activation qui ne semblent pas influencer individuellement les résultats, leur présence ensemble peut modifier la prise de décision. On peut considérer que ces neurones inactifs introduisent des biais dans le calcul de l'action à effectuer. La présence de neurones inactifs offre une opportunité pour optimiser le réseau en modifiant ces neurones pour obtenir plus d'informations ou en les supprimant pour alléger le réseau. Cette optimisation pourrait permettre d'améliorer l'efficacité de la représentation des données et donc les performances globales de l'agent.

τ	*None*	0.01	0.1	0.2
% aN pour *DTC*	100%	$36 \pm 1\%$	$51 \pm 1\%$	$60 \pm 1\%$
récompenses	18.7 ± 4.8	17.9 ± 3.9	$\mathbf{18.97 \pm 3.18}$	16.91 ± 4.58
% aN pour *HGS*	100%	$42 \pm 0.5\%$	$52 \pm 1\%$	$57 \pm 1\%$
récompenses	744 ± 435	$\mathbf{773 \pm 465}$	706 ± 429	712 ± 441
% aN pour *MWH*	100%	$13 \pm 1\%$	$21 \pm 2\%$	$26 \pm 3\%$
récompenses	$\mathbf{0.75 \pm 0.47}$	0.73 ± 0.48	0.72 ± 0.48	0.72 ± 0.49

Table 4.8: Résultats des tests sur cent épisodes dans les trois scénarios pour le $VRBFN$ avec uniquement des neurones actif (aN). Différents seuils d'activation ont été testés.

4.3.8 Conclusion

En conclusion, dans cette partie dédiée à l'algorithme *PPO*, nous avons étudié l'impact de différentes méthodes d'extraction de caractéristiques sur les performances des agents dans différents scénarios de l'environnement VizDoom. Nous avons comparé plusieurs architectures de réseaux de neurones, notamment le *CNN*, le *ResNet* et le *VRBFN*, ainsi que différentes approches d'apprentissage de représentations avec l'autoencodeur. Nous avons également examiné la résistance de ces méthodes au bruit induit par les changements de taille d'images.

Nous avons constaté que le *VRBFN* se démarque comme une méthode robuste et adaptable, capable de maintenir de bonnes performances même avec des observations bruitées ou à des échelles différentes.

4.4 Etude des performances sur les tâches robotiques simulées

Dans les expérimentations précédentes, nous avons utilisé la plateforme de simulation *VizDoom*. Les simulateurs de jeux vidéo sont très utiles pour tester des algorithmes, mais il est difficile d'appliquer les connaissances acquises dans un jeu à la réalité. Nous avons souhaité nous rapprocher d'un entraînement reproductible dans la réalité afin d'étendre les résultats de notre *VRBFN*. C'est pourquoi nous avons décidé d'étudier deux tâches robotiques visuo-motrices qui ont été créées dans le simulateur *PyBullet*. Elles permettent de renforcer la diversité des environnements en termes de dynamique et ont le potentiel de mener à une étude de transférabilité dans le monde réel. En effet, nous avons créé une tâche de fixation à l'aide d'une caméra et une

tâche d'atteinte avec un bras robotique *Kuka IWA*. Nous avons décidé de créer nos propres environnements afin d'avoir un contrôle total sur les différents éléments qui les composent. Nous les avons rendus le plus génériques possible afin de faciliter la transférabilité. Dans ces environnement les images sont en *RGB* de taille 64 pixels. Une étude comparative des entraînements et des résultats des différentes méthodes 4.9 sera présentée après avoir introduit les deux tâches robotiques.

4.4.1 Présentation des tâches robotiques

Tâche de fixation (cam)

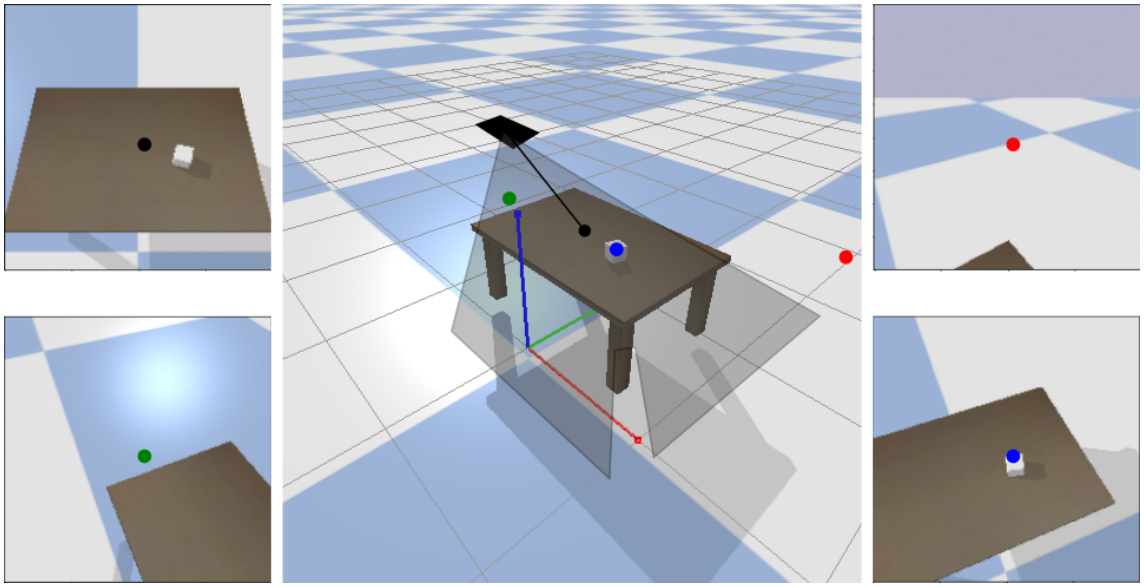


Figure 4.17: Illustration du scénario de fixation du simulateur *PyBullet*. La caméra est représentée par le rectangle noir. Chaque point correspond à une cible de visée que l'on retrouve dans les images résultantes à gauche et à droite.

La tâche de fixation, illustrée dans l'image 4.17, consiste à trouver un cube blanc sur une table. L'agent est une caméra placée en hauteur devant la table (représentée par le carré noir sur l'image). L'image obtenue par la caméra est définie par un point de visée (représenté par les ronds de couleur) dans le repère x, y, z de l'environnement (représenté dans l'image centrale avec l'axe z en bleu, x en rouge, et y en vert). Pour faciliter l'apprentissage, nous avons fixé la dimension z du point de visée à la hauteur de la table. Les actions de l'agent consistent donc à déplacer en x et y le point de visée sur le plan de la table. L'espace des actions est donc l'ensemble : $(-d, 0, d), (-d, 0, d)$, avec $d = 0.1$ dans nos expérimentations.

L'épisode commence par une initialisation aléatoire de la position du cube sur la table, tandis que l'agent commence toujours avec le même point de visée (représenté par le cercle noir en haut à gauche de l'image). L'objectif de l'agent est de réaliser

des actions pour pointer le cube avec la caméra. Une récompense de 1 est obtenue lorsque la distance entre le centre de la caméra est celui du cube est inférieur à 0.1, définie comme suit :

$$r = \begin{cases} +1 & \text{si } \sqrt{(x_{\text{cam}} - x_{\text{cube}})^2 + (y_{\text{cam}} - y_{\text{cube}})^2} < 0.1, \\ -0.1 & \text{sinon.} \end{cases} \tag{4.4}$$

Dans ces équations, $x_{\text{cam}}, y_{\text{cam}}$ sont les coordonnées de visée de la caméra et $x_{\text{cube}}, y_{\text{cube}}$ sont les coordonnées du cube. L'épisode se termine après que l'agent a effectué 100 actions ou qu'une récompense de 1 à été obtenue.

Ce scénario est similaire à celui de *DTC* dans une certaine mesure, car il requiert des extractions précises sur un petit élément du scénario, en l'occurrence le cube. La différence ici est que le cube ne se rapproche pas de la caméra comme les monstres dans *DTC*. Par conséquent, un agent doté d'extractions imprécises ne peut pas attendre que les éléments liés aux récompenses se rapprochent pour agir. L'état est observable au début de l'épisode car l'agent aura toujours le cube dans son champ de vision. Cependant, il est possible qu'avec une séquence d'actions, l'agent se retrouve dans une situation où l'observation ne permet pas de déterminer la position du cube, comme illustré dans l'image en haut à droite, créant ainsi des observations où l'état n'est pas observable.

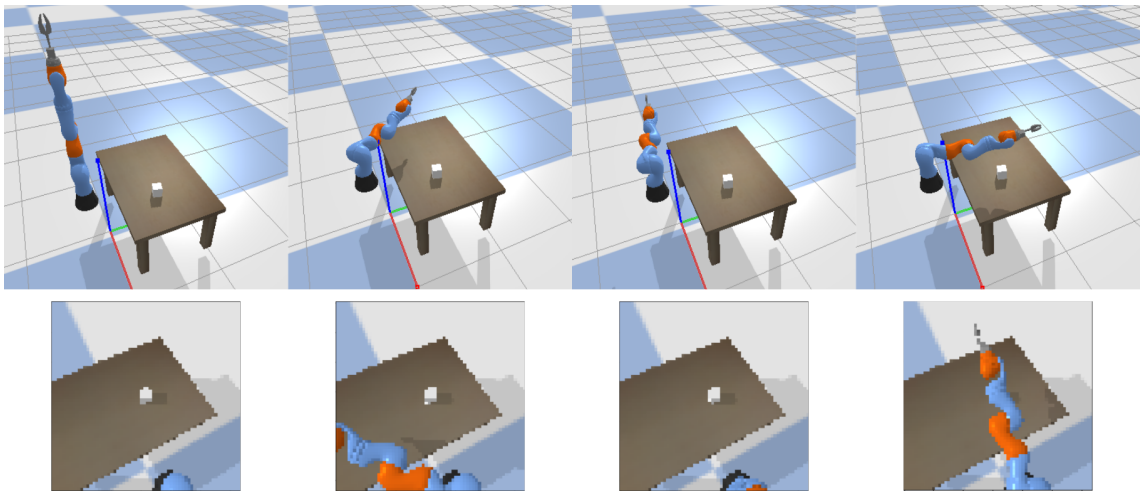


Figure 4.18: Illustration du scénario d'atteinte sur le simulateur *PyBullet*. Les images du haut montrent l'état de l'agent d'un point de vue extérieur à l'agent. Pour chaque image, l'observation de l'agent est donnée en dessous.

Tâche d'atteinte (kuka)

Dans cet environnement l'objectif de l'agent est d'atteindre un cube posé sur une table avec la pince du bras *Kuka* uniquement à l'aide de la vision. La difficulté de ce scénario réside dans l'espace d'observation, car l'agent contrôle le bras robotique

Kuka uniquement à l'aide d'informations visuelles, les observations sont donc très différentes de l'état, l'observabilité de l'agent est partielle. La position de visé de la caméra est initialisée sur le cube et figée durant tout l'épisode, le cube est au centre de l'image. Un bras *Kuka* avec 12 degrés de liberté est instancié, comme illustré dans la première image de la figure 4.18. Nous avons choisit d'utiliser des actions discret pour des raison de simplicité. L'agent peut incrémenter une articulation du bras par action ($\pm d$) ou ne rien faire, ce qui donne un total de 25 actions possibles. L'agent commence sans voir son bras, et en effectuant des actions, il doit atteindre le cube avec sa main, ce qui lui rapporte une récompense de $+1$. Sinon, il reçoit une pénalité de -0.05. L'agent doit trouver une action qui lui permet de déplacer le bras devant la caméra pour avoir un retour visuel afin d'apprendre à toucher le cube. De plus, en fonction de la position du cube sur la table, les mouvements du bras ne seront pas perçus de la même manière par l'agent. En effet, la caméra est orientée de manière à viser le cube, ce qui signifie qu'une caméra orientée vers la droite donnera des images différentes des mouvements du bras par rapport à une caméra orientée vers la gauche.

4.4.2 Entraînement des agents robotiques

Tâche de fixation

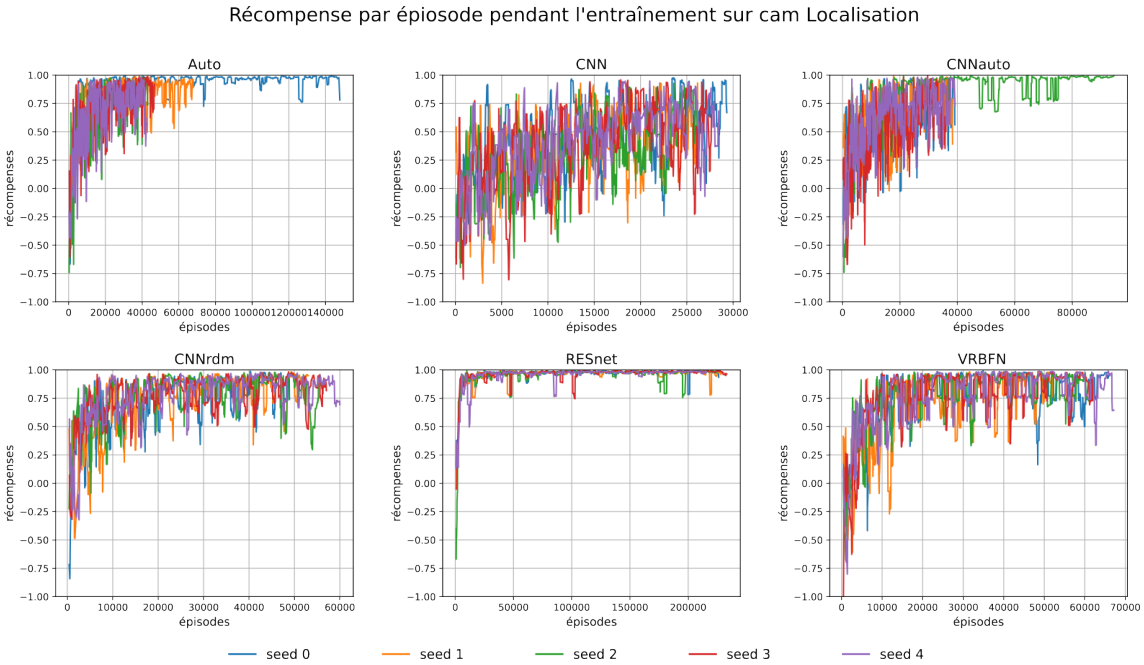


Figure 4.19: Résultats des différentes seed des différentes méthodes du scénario de fixation.

Dans ce scénario qui requiert de la précision, seul le *RESnet* parvient à converger quelle que soit l'initialisation comme le montrent les courbes d'apprentissages de la

figure 4.19. Le $VRBFN$ et le CNN_{rdm} ont des comportements assez similaires : ils ne résolvent pas parfaitement le scénario, et certains épisodes n'aboutissent pas à une récompense. Pour les trois autres méthodes, on remarque que l'utilisation d'une reconstuction de l'observation, soit en amont de l'entraînement RL avec *Auto*, soit en parallèle avec CNN_{auto}, permet d'améliorer l'apprentissage de cette tâche par rapport à l'utilisation exclusive du CNN.

Tâche d'atteinte

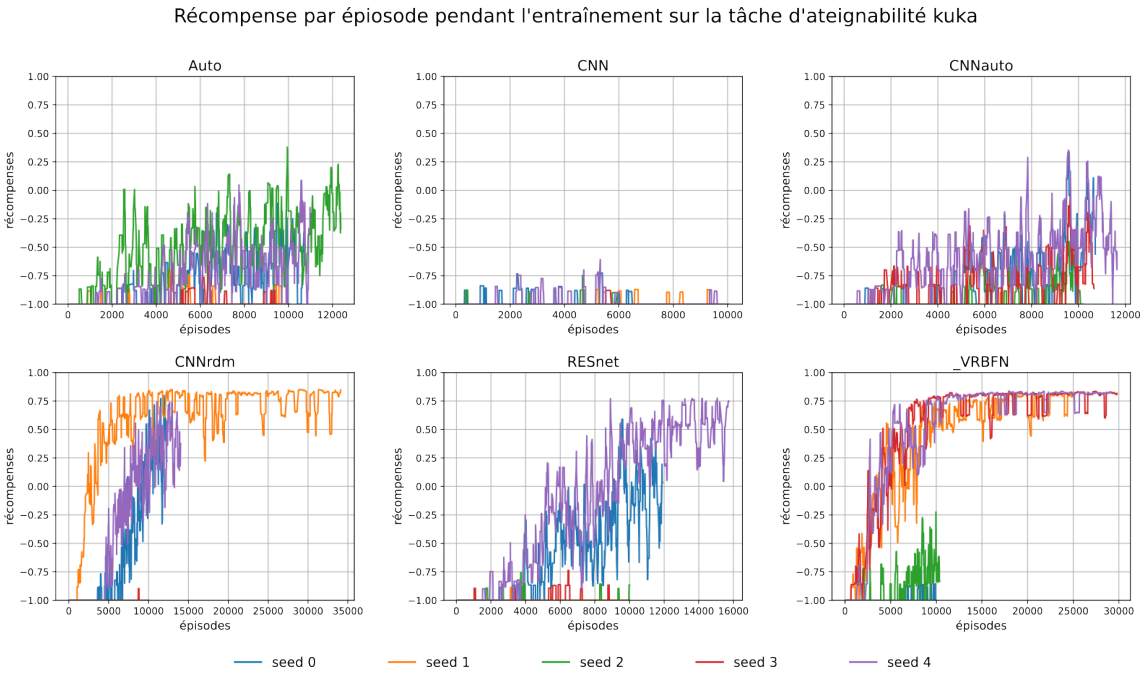


Figure 4.20: Résultats des différentes seed des différentes méthodes du scénario d'atteignabilité.

Dans ce scénario, les courbes d'apprentissage 4.20 montrent que notre méthode est celle qui converge le mieux (3 sur 5 seeds), suivie du CNN_{rdm}, qui a une seed qui converge et deux autres qui ont commencé à converger. Ces deux réseaux sont les seuls à fournir des résultats significatifs, suivis de près par le *RESnet*, qui montre des signes d'apprentissage mais qui ne parvient pas à converger complètement. Les courbes d'apprentissage des trois autres méthodes montrent une fois de plus qu'il est préférable d'apprendre des représentations d'états, mais cela n'est pas suffisant dans ce scénario. Ce scénario est dominé par les deux méthodes d'extraction aléatoire, qui permettent de mieux séparer les observations avec et sans le bras. En effet, contrairement aux méthodes utilisant le *CNN*, les neurones activés en l'absence du bras ne seront pas les mêmes que ceux activés en présence du bras, ce qui facilite l'émergence de deux comportements distincts, ce qui est utile pour résoudre ce scénario.

4.4.3 Test des agents robotiques simulés

Les résultats des tests des agents sont établis sur mille épisodes, et le score moyen pour chaque seed de chaque méthode est reporté dans le tableau 4.9. Seuls le CNN_{rdm}, le $VRBFN$ et le $RESnet$ ont au moins une initialisation qui converge. Le CNN ne parvient pas à obtenir de récompense contrairement aux CNN_{auto} et $Auto$ qui montrent quelques signes d'apprentissages. Le $VRBFN$ est la méthode qui à le plus d'initialisations qui convergent (3). Le CNN_{rdm} n'est pas loin mais deux de ces différentes initialisations n'ont pas eu le temps de converger même s'ils montrent de fort signes d'apprentissage, avec plus de temps il aurait probablement put atteindre une stabilité.

Ces deux tâches robotiques sont difficiles à résoudre pour les différentes méthodes. Dans la tâche de fixation, on remarque que tous les agents parviennent à se rapprocher du cube, cependant la précision fait défaut et le comportement optimal pour certains est de tourner autour du cube jusqu'à l'atteindre. Le meilleur agent est le $RESnet$, qui converge pour toutes ses seeds. Le CNN_{rdm} obtient également de bons résultats, mais n'a pas convergé. Le $VRBFN$ manque d'efficacité sur ce scénario, une intuition derrière cet échec est liée à la précision des activations localisées au centre de l'observation. En effet l'objectif étant de placer le cube au centre de l'image, il est préférable d'avoir des informations sur cette partie de l'image, or les récèpteurs sont localisés aléatoirement sur l'image et donc l'information sur le centre de l'image peut être insuffisante.

Pour le scénario d'atteinte avec le bras *Kuka*, la précision au centre de l'observation n'est pas primordiale, au contraire, l'agent a besoin d'informations sur les côtés de l'observation pour voir le bras *Kuka*. Dans ce scénario, le $VRBFN$ est le seul à avoir convergé pour trois seeds avec un comportement optimal que l'on peut décomposer en deux étapes. Dans un premier temps le bras est ammené devant la caméra dans l'axe du cube, et ensuite l'agent approche le cube jusqu'au contact avec le préhenseur du bras kuka. Parmis les autres méthodes, le $RESnet$ et le $CNNrdm$ ont une seed qui performe plutôt bien. Toutes les méthodes, sauf le CNN, parviennent à avoir quelques seeds qui ont appris à toucher le cube pour certains épisodes, notamment avec un comportement de balayage du bras sur la table.

	`fixation`	`atteinte`
CNN	0.27 ± 0.94	-1
CNN_{auto}	0.15 ± 0.95	-1
CNN_{rdm}	0.73 ± 0.63	-0.60 ± 0.64
$RESnet$	$\mathbf{0.95 \pm 0.27}$	-0.83 ± 0.5
$Auto$	0.39 ± 0.87	-0.98 ± 0.15
$VRBFN$	0.45 ± 0.86	$\mathbf{-0.59 \pm 0.72}$
Rdm	0.14 ± 0.93	-1

Table 4.9: Résultats sur mille épisodes des six méthodes dans les deux scénarios de PyBullet. La moyenne est calculée sur les différentes seeds ainsi que l'écart-type.

Conclusion

Dans ce chapitre, nous avons présenté le *VRBFN* que nous avons développé, en détaillant son fonctionnement ainsi que ses spécificités en matière de localité et de parcimonie. Nous avons également illustré son application à l'apprentissage par renforcement. Les résultats obtenus dans la première partie de ce chapitre, sur les deux scénarios de VizDoom, *BA* et *HG*, démontrent que nos stimuli renferment des informations pertinentes pour l'approximation de la fonction Q, à tel point que l'utilisation d'une simple fonction linéaire s'avère suffisante sur ces scénarios. Cette découverte nous a incités à soumettre le *VRBFN* à des scénarios plus complexes, caractérisés par une observabilité partielle accrue. Ces scénarios ont été résolus avec l'algorithme du *PPO*, qui semble davantage adapté à ce type de scénarios.

Dans les scénarios de *VizDoom*, nous avons constaté que le $VRBFN$ obtenait les meilleurs résultats en termes de récompense, tandis que les autres méthodes étaient plus ou moins performantes en fonction des scénarios, démontrant ainsi la stabilité et l'adaptabilité de notre modèle. Parmi les autres méthodes, le *RESnet* s'est avéré être le meilleur sur le scénario *DTC*, tandis qu'un *CNN* avec initialisation aléatoire des poids (CNN_{rdm}) a bien fonctionné sur le scénario *HG*.

Nous avons en outre démontré la robustesse de notre méthode face au bruit dans les images d'entrée ainsi que sa robustess face à un agrandissement de l'image d'entrée. Les temps d'entraînement ont également montré la rapidité de notre méthode, qui n'est pas optimisée sur cet aspect contrairement aux autres modèles déjà optimisés par leurs implémentations *Pytorch*. De plus, nous avons montré qu'il était possible de réduire le nombre de neurones tout en maintenant des résultats corrects. Cette facette de notre méthode permet de réduire encore davantage les temps de calcul et

ouvre la voie à des améliorations potentielles sur les neurones non actifs, comme la génération pertinente de ces neurones pour obtenir de meilleurs résultats.

Dans les scénarios de *PyBullet*, nous avons rencontré des défis liés à l'observation partielle accrue et à la complexité des actions. Le *RESnet* s'est montré le plus performant dans la tâche de localisation, tandis que le $VRBFN$ a été le meilleur dans la tâche d'atteinte avec le bras *Kuka*.

En général, nous pouvons observé que l'utilisation d'une fonction d'erreur explicite de reconstruction de l'observation, en amont de l'entraînement par renforcement, a amélioré les performances dans la plupart des scénarios testé. Cela indique que l'apprentissage de représentation est un atout pour faciliter l'apprentissage des politiques d'action. Cependant, ce type de méthodes requiert plusieurs entraînements, de plus elles nécessitent d'avoir pré-exploré l'environnement pour acquérir un certain nombre de données pertinentes. Ce type de données est difficile à obtenir dans les environnements complexes car un comportement aléatoire ne permet pas l'exploration parfaite d'un environnement. De plus, nous avons vu que l'utilisation d'une erreur de reconstruction en tant que régulateur, comme dans CNN_{auto}, ne permet pas d'atteindre les performances obtenues par les méthodes de pré-entraînement. Les méthodes les plus pertinentes pour les scénarios étudiés dans ce chapitre sont celles utilisant des extractions aléatoires comme le CNN_{rdm} ou le $VRBFN$.

En conclusion, ce chapitre a permis de comparer différents méthodes d'extraction de représentations visuelles pour l'apprentissage par renforcement sur des scénarios variés, mettant en évidence les forces et les faiblesses de chaque approche. Les résultats obtenus offrent des perspectives intéressantes sur l'utilisation de *VRBFN* comme extracteur de stimuli visuels dans le cas du *RL*.

Chapitre 5

Spécialisation du VRBFN et transfert dans la réalité

Introduction

Dans ce chapitre, nous verrons comment améliorer le *VRBFN* pour obtenir de meilleurs résultats dans le scénario de fixation, le seul qui n'ait pas encore été résolu par notre réseau. Dans un premier temps, nous montrerons que la disposition aléatoire des zones d'attention n'est pas optimale et nous proposerons d'autres type de distribution pour les paramètres du *VRBFN* afin de le spécialiser pour la résolution de cette tâches. Nous présenterons aussi des résultats sur l'apprentissage de ces paramètres.
Ensuite nous testerons la robustess du *VRBFN* face à un transfert d'apprentissage dans le monde réel. L'objectif sera de démontrer les capacités de notre méthode à utiliser les connaissances acquises en simulation pour effectuer la tâche dans le monde réel. Nous commencerons cette deuxième partie par présenter brièvement un

état de l'art des méthodes les plus courantes pour le transfert au monde réel, puis nous décrirons notre protocole expérimental avant de présenter les résultats.

5.1 Etude de l'impact de la distribution des zones d'attention

Dans tous les travaux décrits précédemment, les zones d'attention du *VRBFN* (G_i dans l'équation (4.2)) ont été générées de manière uniforme sur l'image. Cependant, dans les scénarios à la première personne, certaines zones de l'image ne sont pas aussi importantes que d'autres. Par exemple, dans le scénario *DTC*, l'agent ne peut utiliser que la rotation, ce qui l'empêche de se déplacer. Par conséquent, la partie supérieure de l'image reçue par l'agent est inutile. Par exemple, nous pourrions instancier nos gaussiennes uniquement sur la partie basse de l'image, ou prendre l'exemple des humains qui ont une plus forte concentration des récèpteurs au centre de la rétine, ce qui pourrait être intéressant étant donné que le centre des images des environnement à la première personne revet une importance particulière. Dans cette partie nous présenterons différentes initialisations des récèpteurs gaussiens, à savoir une répartition normale, une rétinienne issue de traveaux sur l'encodage de la rétine et enfin nous introduirons l'apprentissage de ces paramètres.

5.1.1 Distribution des zones d'attentions

Dans le cadre du scénario de fixation, la partie centrale de l'image revêt une importance cruciale. En effet, une description précise de cette zone permet à l'agent de centrer plus précisément le cube. Cependant, une répartition trop centrée pourrait empêcher l'agent d'obtenir des informations sur le cube au début de l'épisode. Par exemple, si le cube se trouve à une extrémité de la table, l'agent pourrait avoir du mal à le voir. Différentes distributions sont présentées dans cette partie et des résultats d'apprentissage seront présentés dans la partie suivante.

5.1.1.1 Distribution normale

Nous avons dans un premier temps testé une distribution normale des zones d'attention de moyenne $0,5$ et d'écart type $0,1$, ce qui permet un bon recouvrement du centre de l'image et des informations non nulles autour de celui-ci. L'ecart-type des noyaux est généré de la même manière que dans les expériences précédentes, en utilisant une distribution uniforme $\mathbb{U}(0.02, 0.2)$. La zone recouverte par ce type de formation est illustré dans l'image 5.1. Avec ce type de recouvrement on peut espérer obtenir

de bons résultats dans les scénario où l'information pertinente est contenue dans le centre de l'observation comme pour *DTC* ou pour la tâche de fixation.

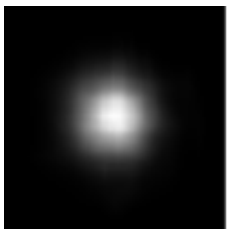


Figure 5.1: Répartition des zones d'attentions du *VRBFN* avec une distribution normale. La zone blanche représente la somme de toutes les zonnes d'attentions

5.1.1.2 Distribution log-polaire

Le modèle de la rétine utilisé dans ces expériences est basé sur les travaux de Bolduc et Levine [117]. Il utilise le modèle rétinien de [118] ou les récepteurs sont répartis en cercles autour du centre de la rétine. Ce modèle rétinien a été utilisé pour générer des images sous forme log-polar pour la perception robotique [119]. Ce modèle a également été combiné avec des réseaux de convolution pour l'analyse d'images robotiques [120], la classification d'images [121], ou encore la résolution de tâches de préhension [122]. En général, ce modèle permet d'obtenir de bons résultats dans les tâches de vision. Nous avons implémenté ce modèle, et dans la figure 5.2, vous pouvez voir la position des centres des champs récepteurs définis avec les paramètres ci-dessous, la largeurs des récépteurs est décroissante de l'extérieur vers la fovea.

- $L = 9$: nombre de cercles
- $f = 0.04$: la taille de la fovea par rapport à la taille de globale de la rétine
- $\alpha = 0.4$: le ratio entre la taille des récepteurs et leurs distance au centre de la fovea
- $\omega = 0.5$: le taux de chevauchement
- $k = 5$: le nombre de couches dans la fovea

Les équations permettant de modéliser la distribution sont les suivantes :

- rayon d'espacement en dehors de la fovea : $\theta(\alpha, \omega) = 2\pi \left[round\left(2\pi(\arccos(z))^{-1}\right)\right]^{-1}$
- excentricité : $\epsilon(\alpha, \omega, s, f) = \left(-\frac{\alpha(1-2\omega)+d}{\alpha-2}\right)^2 f$ avec $d = \sqrt{4 + \alpha^2\left((1-2\omega)^2 - 1\right)}$
- rayon des filtres périphériques : $r(\alpha, \omega, s, f) = \frac{\alpha}{2}\epsilon(\alpha, \omega, s, f)$

- rayon des filtres dans la fovéa : $r_f(\alpha, \omega, 0, f) = \frac{\alpha}{2} f$
- excentricité des filtres dans la fovéa : $\epsilon_f(k) = (\epsilon(\alpha, \omega, 1, f) - f)\, k$
- rayon d'espacement dans la fovea : $\theta_f = \theta\,(\alpha f / \epsilon_f(k), \omega)$

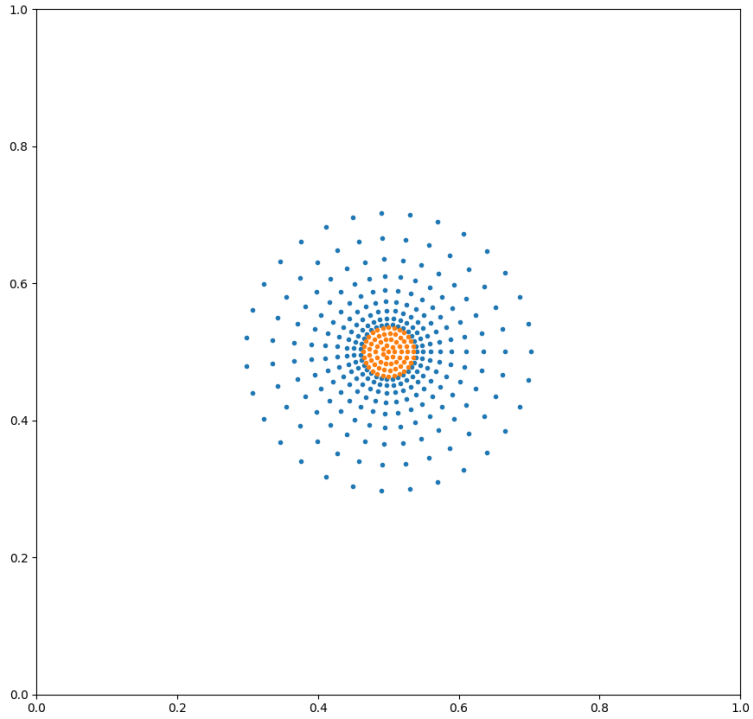


Figure 5.2: Position des centre des Gaussiennes dans une disposition rétinienne. Les centres oranges sont ceux à l'intérieur de la fovéa est les bleus à l'extérieur.

Ces paramètres ont été choisis afin d'obtenir approximativement le même nombre de neurones que dans le chapitre précédent. Avec ces paramètres, nous avons 1050 neurones, contre 1002 dans le chapitre précédent.

5.1.1.3 Apprentissage de tâches de RL avec les différentes initialisations

Nous avons testé ces distributions sur trois scénarios de *VizDoom*, *DTC*, *HGS* et *MWH* et sur le scénario robotique de fixation issue de Pybullet. Nous avons entraîné un agent avec cinq initialisations différentes pour le modèle normale et rétinien. Les résultats présentés dans le tableau 5.1 montrent que les agents dotés d'un *VRBFN* rétinien sont plus performants pour le scénario *DTC*. Ce scénario contient

le plus d'information importante au centre de l'image. Les deux distributions sont plus performantes que la distribution uniforme dans le scénario de fixation. Ce type de génération permet d'améliorer la précision au centre de l'image du fait de l'augmentation de la densité des gaussiennes dans la zone centrale. Cependant, pour d'autres scénarios, l'agent a besoin d'informations provenant des côtés de l'image. Par exemple, pour repérer un pack de soins à gauche de l'agent dans le scénario *HGS* ou pour visualiser plus facilement le bras *Kuka* dans le scénario d'atteignabilité.

Distribution	defend the center	health gathering supreme	my way home	Fixation
Uniform	18.7 ± 3.7	$\mathbf{744 \pm 435}$	$\mathbf{0.75 \pm 0.47}$	0.45 ± 0.86
Normal	18.84 ± 5.23	640 ± 413	0.59 ± 0.54	$\mathbf{0.99 \pm 0.01}$
Log polar	$\mathbf{21.07 \pm 4.12}$	324 ± 83	0.25 ± 0.58	$\mathbf{0.99 \pm 0.01}$

Table 5.1: Moyenne et déviation standard des résultats du *VRBFN* avec distributions normales, uniforme et rétinienne des gaussiennes sur cinq seed différentes

Comme prévu, les résultats ne bénéficient pas tous de cette répartition, en effet, seuls les scénarios *DTC* et la tâche de fixation voient leurs résultats améliorés, car l'objectif dans ces scénarios est lié au centre de l'image. Pour les autres scénarios, le centre de l'image n'est pas l'endroit le plus important. Par exemple, dans *MWH*, même si le centre de l'image paraît légitime pour le déplacement de l'agent, les côtés sont également importants, permettant à l'agent de mieux se situer dans le labyrinthe et de distinguer plus facilement les différentes positions. On pourrait augmenter l'écart type de la loi normale pour couvrir une plus grande surface tout en maintenant un maximum de récepteurs au centre de l'image. Cependant, cette distribution resterait aléatoire, et en augmentant la surface couverte, certaines zones pourraient être moins denses pour certaines distributions, ce qui pourrait nécessiter d'augmenter le nombre de neurones pour augmenter les chances de réussite. De plus, il est possible que la distribution aléatoire des σ_z et μ_z ne soit pas pertinente avec un tel placement des filtres récepteurs, créant ainsi un manque d'information pour certaines images. L'idéal serait de disposer les zones d'attention de manière optimale pour une tâche visuelle à la première personne, maximisant à la fois leur position et leur taille afin de maximiser l'information transmise par ces neurones. La solution optimale d'un tel problème n'est pas connue à ma connaissance.

5.1.2 Apprentissage des paramètres Gaussiens

Outre la sélection aléatoire ou réfléchie des paramètres gaussiens, il est également possible d'entraîner ces paramètres pour un problème donné. Pendant ma thèse j'ai eu l'occasion d'encadrer un stage de Master sur cet aspect de l'apprentissage des paramètres gaussiens pour classifier des données. Les résultats, disponibles en annexe A.3, montrent que l'apprentissage des paramètres est possible avec une méthode de descente de gradient et permet d'obtenir des caractéristique pertinentes pour la classification. A partir de ces travaux nous avons réalisé un apprentissage des paramètres gaussiens du *VRBFN* rétinien sur l'environnement *MWH* de *VizDoom*, car c'est le plus diversifié en termes de données observées. Les $\mu_{x,y}$ et $\sigma_{x,y}$ étant fixé par le modèle rétinien, seul les paramètres sur l'intensité μ_z et σ_z seront appris. Le *VRBFN* rétinien a donc été entraîné à reconstruire les images de cet environnement obtenues via des actions aléatoires, comme avec un auto-encodeur. Les résultats confortent ceux obtenus lors de l'apprentissage des paramètres du *VRBFN* sur des bases de données de classification effectué pendant le stage de master. En effet, dans le graphique 5.3 représentant les paramètres μ_z et σ_z, qui correspondent à la moyenne d'intensité et à la tolérance d'intensité d'activation, on remarque dans un premier temps que la répartition optimale des moyennes d'intensité est soit proche de 0 soit de 1 pour la plupart des neurones bleu et vert. De plus, plus un neurone est proche du centre (fovéa), plus σ_z est faible, et inversement. L'utilisation de ce modèle pré-entraîné améliore de 33% ($0.25 \rightarrow 0.33$) les résultats d'apprentissage dans *MWH* par rapport à une distribution uniforme des μ_z et σ_z.

Cette distribution des centres d'intensité μ_z et des déviation standard σ_z est récurrente dans ce type de problème. En effet, dans la littérature, on retrouve des entraînements de modèles rétiniens. Par exemple, [123], [124] utilisent la maximisation de l'information mutuelle pour optimiser le modèle et obtiennent une formation *ON-OFF* des récepteurs, où certains sont à 0 et d'autres à 1, [125] obtient, après optimisation, des filtres proches de ceux de Gabor. L'optimisation de ce modèle peut ensuite être utilisée pour apprendre des tâches par renforcement, comme le montre l'étude de [126], qui entraîne le modèle rétinien de manière supervisée avant de résoudre des tâches de coordination œil-main. Cependant, ces travaux n'utilisent pas directement la réponse des recépteurs rétinien mais plutôt des transformations comme l'image corticale générée à partir des activations par différentes méthodes.

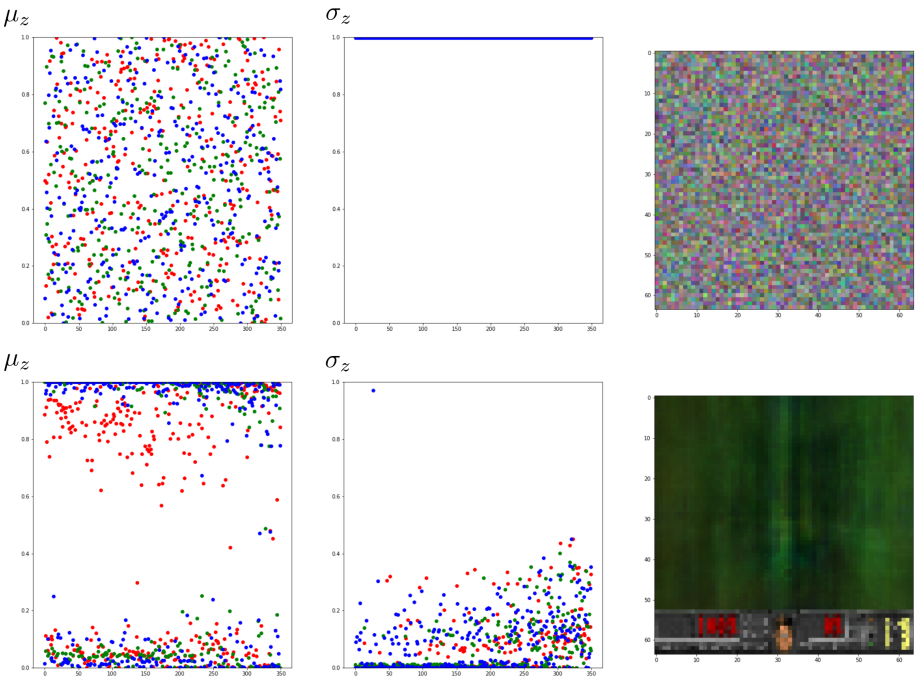


Figure 5.3: Répartition des paramètres μ_z et σ_z avant (ligne du haut) et après apprentissage du *VRBFN* rétinien sur *MWH*. La première colonne correspond aux μ_z par neurones, la seconde aux σ_z, et la troisième est l'image reconstruite. Les activations sont répartis de la rétine (gauche) à son extrémité (droite) et sont colorés en fonction de leur appartenance aux canaux R, G, B.

5.2 Transfert des connaissances dans la réalité

5.2.1 Etat de l'art

Le transfert d'une tâche apprise en simulation à la réalité est une tâche difficile due au problème nommé le *Reality Gap* [127]. Très souvent lors d'un passage à la réalité les résultats sont décevants, si aucun changement a été fait sur le modèle entrainé en simulation. En effet les différences entre l'environnement de simulation et le monde réel sont souvent critiques, elles peuvent êtres regroupées en trois familles :

- La perception : Il est difficile de modéliser un capteur réel en simulation. Même si l'on peut s'en rapprocher, il y aura toujours une légère différence, d'autant plus que les signaux créés en simulation seront différents de ceux émis par le monde réel.

- La Dynamique de l'agent : En apprentissage par renforcement, par exemple, les agents utilisés pour effectuer les tâches doivent être modélisés pour être entraînés en simulation. Ainsi, pour une même action, il y aura deux résultats plus ou moins proches dans les deux environnements (réel/simulé). Cela peut être dû à des facteurs tels que la friction ou l'usure d'une pièce.

- La dynamique de l'environnement : le monde du réel est imprévisible, c'est pourquoi les expériences sont souvent effectueées dans des environnements controlés et très proches de la simulation. L'entraînement dans ce type d'environnement ne favorise pas la généricité des apprentissages.

Deux grandes familles de méthodes existent pour gérer les problèmes liés à l'environnement :

- Adaptation de domaine : Ces méthodes utilisent l'apprentissage de l'environnement pour essayer de se rapprocher le plus possible d'un espace latent général entre le monde réel et le monde simulé. On peut également apprendre à créer, à partir des images simulées, des images proches de la réalité, ce qui revient à avoir le même espace de représentation compact.[128]–[133]

- Randomisation de domaine : Ces méthodes utilisent des paramètres aléatoires pour générer l'environnement (couleurs, physique, textures, position, ...) dans le but d'apprendre une tâche dans le cadre le plus large possible et donc de pouvoir la réussir dans le monde réel.[134]–[138]. On peut aussi apprendre une randomisation optimale qui est difficile pour l'agent afin de le rendre plus robuste.[139]

Il est également possible de combiner les deux méthodes, c'est-à-dire apprendre des représentations compactes proches entre le domaine simulé et réel tout en simulant des environnements aléatoires [140]. Pour résoudre les problèmes liés aux actions, certains chercheurs appliquent des forces de perturbation sur leurs modèles pour leur apprendre à s'adapter à différentes perturbations sur leurs actions. Par exemple, [141] apprend une politique de perturbation qui permet d'être plus robuste lors du passage dans la réalité. Certains apprennent directement les différences dans les actions afin de calibrer leurs modèles sur le modèle réel [142], ou encore utilisent la randomisation des paramètres de l'agent tels que la friction, la masse ou encore sa raideur [143], [144].

5.2.2 Transfert de la simulation au monde réel

Pour tester la capacité des réseaux de cette thèse à transférer leurs connaissances au monde réel, nous avons choisi d'utiliser le scénario de fixation. Nous avons opté pour une méthode de génération aléatoire de l'environnement afin d'apprendre un

comportement plus général qui ne dépend pas des couleurs et des textures spécifiques de la simulation. Pour ce faire, nous avons entraîné les cinq méthodes en utilisant la simulation, où à chaque début de l'épisode, nous avons généré aléatoirement les couleurs du cube, de la table et du sol comme on peut le voir dans la figure 5.4. Cela nous permet de créer un environnement varié et d'entraîner notre agent à s'adapter à différentes configurations visuelles. Parmi les différentes méthodes testées seul le $VRBFN$ avec une distribution normale des gaussiennes et le $ResNet$ ont réussi à résoudre parfaitement le scénario virtuel. Une fois l'agent entraîné, nous pourrons le transférer dans le monde réel, où il pourra généraliser ses compétences en matière de localisation.

Figure 5.4: Example d'observation pour l'environnement de localisation avec randomization sur les couleurs des objets pour chaque épisode.

Ce scénario est assez simple à mettre en place dans le monde réel car il nécessite uniquement une caméra motorisée, un objet et une table. Pour ce faire, nous avons utilisé une webcam montée sur un support permettant des rotations sur les axes de tangage et de lacet (horizontal et vertical). Le cube de la simulation sera représenté par un objet quelconque. Afin de ne pas contraindre les expérimentations à une salle blanche et à une reproduction exacte de la simulation, nous avons décidé d'utiliser une planche de contreplaqué posée sur le sol en guise de table. Le cadre expérimental a été volontairement éloigné de la simulation afin d'avoir une réelle différence entre la simulation et la réalité, comme présenté sur la photo 5.5. L'espace d'action dans le monde réel est différent de celui de la simulation. En effet dans la simulation l'agent agissait sur le point de visé de la caméra grâçe à des incrémentations sur x et y dans l'espace de la table. Lors du transfert dans la réalité, les déplacements sur l'axe x étaient transformés en angles de rotation horizontale, et ceux sur l'axe y en rotation verticale. La correspondance entre l'incrément sur les axes simulés et l'angle dans la réalité a été calculée en faisant correspondre le nombre d'actions nécessaires pour atteindre les extrémités de la table dans la simulation et dans la réalité. Les images obtenue par la caméra de taille 640×480, ont été redimensionnée à celles de la simulation (64×64).

Lors des essais, le réseau $VRBFN$ a montré une forte adaptation, contrairement au réseau $ResNet$ qui n'a pas réussi à transférer ses connaissances dans le monde réel. Les images extraites de la vidéo 5.6 témoignent du succès du $VRBFN$ pour le transfert direct des connaissances acquises en simulation sur différents objets. Aucun ré-entraînement n'a été effectué, la stratégie apprise en simulation à directement

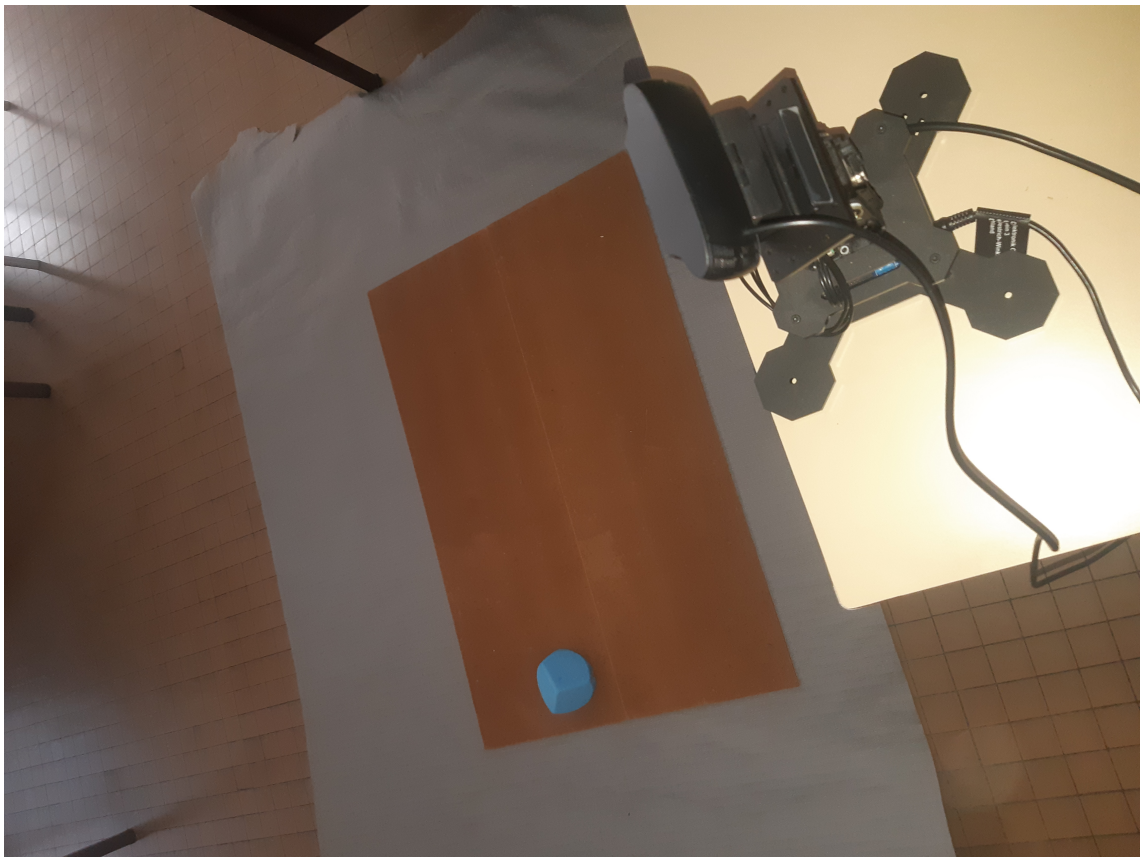


Figure 5.5: Photo de l'environnement réel.

été testée dans l'environnement réel. Un autre test qualitatif a été effectué sur le *VRBFN*, il s'agit d'un test de suivi d'objet. Dans ce test, un polygone bleu attaché à un fil rigide bleu a été utilisé. Une fois que l'agent avait réussi à placer l'objet au centre de la caméra, nous avons déplacé le cube pour vérifier si l'agent suivrait le mouvement même s'il n'avait jamais été confronté à ce type de comportement. Les images extraites de la vidéo du test 5.7 montrent que le *VRBFN* est capable de suivre l'objet en temps réel. Ce dernier test montre qu'en plus de s'adapter à un changement de domaine, notre agent est aussi capable de s'adapter à une nouvelle dynamique d'environnement uniquement avec les connaissances acquises en simulation.

On peut également observer dans les images du test de suivi (voir Figure 5.7) une différence de contraste entre les deux premières lignes, due à la quantité de lumière du jour captée par la caméra. Notre agent ne semble pas perturbé par ce changement, indiquant une fois de plus une bonne robustesse au bruit.

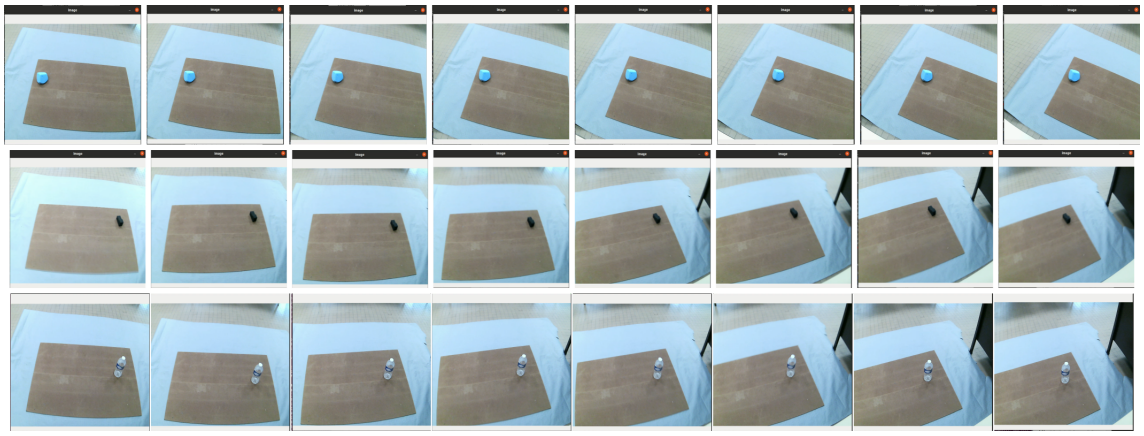


Figure 5.6: Photos issues de vidéos des tests de transfert de la tâche de fixation apprise en simulation dans le monde réel. Trois différents objets ont été testés.

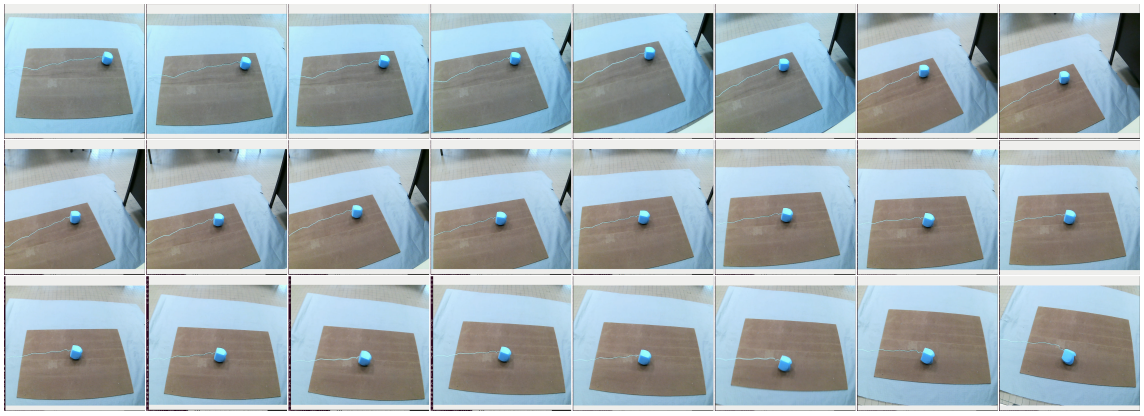


Figure 5.7: Photos issues d'une vidéo d'un tests de transfert de la tâche de fixation apprise dans la simulation à une tâche de suivi dans le monde réel. La première ligne correspond à la fixation et les deux autres au suivi.

Conclusion

En résumé, ce chapitre a exploré diverses avenues pour améliorer notre méthode. Dans un premier temps, nous avons examiné deux distributions distinctes des filtres récepteurs, à savoir une distribution normale et une distribution rétinienne. Nos investigations ont révélé que l'adaptation de la distribution en fonction du scénario pouvait conduire à des améliorations significatives des résultats. Par la suite, nous avons présenté les résultats de l'entraînement des paramètres du *VRBFN*, mettant en lumière la quête d'une distribution optimale des filtres récepteurs. Les résultats obtenus ont démontré que les distributions optimales tendent vers des récepteurs de type *ON-OFF*, judicieusement disposés en fonction de la tâche à résoudre. Enfin, la dernière section de ce chapitre a introduit différentes méthodes de transfert d'apprentissage dans la réalité. Notre succès dans l'application d'une méthode de

randomisation de l'environnement a été couronné de quelques résultats qualitatifs convaincants. Ces diverses approches convergent vers une meilleure compréhension de notre méthode et ouvrent des perspectives prometteuses pour son optimisation future. Les résultats encourageants obtenus suggèrent que l'adaptabilité des filtres récepteurs et l'utilisation de techniques d'entraînement novatrices peuvent jouer un rôle crucial dans l'amélioration des performances globales de notre modèle.

Chapitre 6

Discussions et Perspectives

Introduction

Dans ce chapitre qui clôture ce manuscrit, nous résumerons dans un premier temps les contributions de ces travaux. Ensuite, nous évoquerons certaines pistes d'amélioration du *VRBFN* dans le but d'améliorer la rapidité du réseau, son efficacité ainsi que sa plasticité. Nous finirons par présenter certaines applications possibles dans lesquelles notre réseau pourrait être utilisé.

6.1 Contribution

Ces travaux sur la représentation d'états pour l'apprentissage par renforcement ont abouti au développement d'une méthode de perception visuelle appelée *Visual Radial Basis Function Network* (*VRBFN*). Trois contributions peuvent être retenues de ces travaux : la conception du réseau et ses propriétés, l'application du réseau à des scénarios partiellement observables, et l'amélioration du réseau à des fins de transférabilité dans la réalité.

6.1.1 Visual Radial Basis Function Network

La conception du *VRBFN* est présentée dans le chapitre 4.1. Le modèle comprend un module de perception visuelle et un *MLP* pour le contrôle. Le module de perception est modélisé par plusieurs neurones gaussiens (N neurones). Chaque neurone gaussien a deux parties : la première permet de localiser les informations reçues par le neurone, représentée par des noyaux gaussien à deux dimensions, et la seconde permet de comparer les intensités des pixels à l'intérieur des masques avec celles des neurones. Cette partie est modélisée par des fonctions à bases radiales. En sortie de ce module de perception, on obtient un vecteur, un stimulus visuel, de taille N, une sortie par neurone, où chaque variable est comprise entre zéro et un. Plus les pixels à l'intérieur de la zone d'attention, et plus particulièrement ceux au centre de cette zone d'attention, sont proches de la valeur d'intensité du neurone, plus la sortie du neurone sera proche de un, et à l'inverse, on aura une sortie proche de zéro.

Lorsque l'entrée comprend c canaux, chaque neurone associé à une zone d'attention comprendra c centres d'intensité. Le stimulus visuel peut ensuite être utilisé pour résoudre n'importe quel problème visuel, dans notre cas, il s'agit de résoudre une tâche avec de l'apprentissage par renforcement. Dans un premier temps, nous avons démontré la plausibilité de la méthode à travers deux scénarios simples du jeu vidéo *VizDoom* avec une seule couche linéaire pour la prise de décision. Les résultats ont montré qu'il était possible d'obtenir de bons comportements avec un mapping linéaire des stimuli vers les actions.

Ensuite, des résultats concernant les caractéristiques des activations ont été fournis, démontrant une forte localité et parcimonie des activations. De plus, nous avons partiellement montré qu'il était possible de caractériser l'utilité d'un neurone du *VRBFN*. Cette première contribution a été publiée et présentée à l'oral à la conférence *ICPRAI* 2022 à Paris [7].

6.1.2 Résolution de POMDP avec le VRBFN

Cette deuxième contribution démontre les capacités du *VRBFN* à résoudre des scénarios visuels partiellement observables. Plusieurs résultats ont été développés dans le chapitre 4 4.1 sur cinq scénarios différents, trois provenant du jeux vidéo

VizDoom ainsi que deux autres scénario robotiques entièrement développés à la main sur le simulateur *Pybullet*. L'objectif de ces expérimentations était de montrer les performances du *VRBFN* sur différents scénarios et de les comparer avec cinq méthodes de l'état de l'art.

Les comparaisons montrent une dominance de notre réseau sur les trois scénarios *VizDoom* en termes de récompense, et sur les scénarios *Pybullet*, certaines limitations sont apparues, avec tout de même des résultats supérieurs sur le scénario d'atteignabilité avec le bras *Kuka*. Sur ces scénarios, la parcimonie des activations ou encore la robustesse du réseau aux changements de taille d'entrée et au bruit lié au redimensionnement du *VRBFN* ont été mis en avant. Ces travaux ont été soumis et acceptés dans un journal [145].

6.1.3 Spécialisation et transférabilité du VRBFN

Cette dernière contribution, disponible dans le chapitre 6 5, montre les capacités d'adaptabilité du *VRBFN* à différents scénarios. Dans un premier temps, nous évoquons certaines distributions particulières des paramètres gaussiens afin d'améliorer la résolution de la tâche de localisation.
De plus, une expérimentation de transfert vers la réalité est effectuée avec l'application d'une méthode de domaine *randomization*. Les résultats qualitatifs de ce transfert mettent en avant la robustesse du *VRBFN* face à un environnement réel et éloigné de la simulation.

6.2 Perspectives d'améliorations

Trois possibles améliorations sont présentées dans cette partie afin de pallier à certaines limitations du *VRBFN*.

6.2.1 Rapidité d'extraction

Le *VRBFN* est un réseau composé d'une ou plusieurs couches linéaires entraînables. Ces couches sont plus rapides en termes de calcul que des couches de convolution. Pourtant, dans nos résultats, les temps d'entraînement sur un million d'itérations étaient presque identiques. Ce fait peut être expliqué par le temps d'exécution du *VRBFN*. En effet, même si le passage du stimulus dans le *MLP* est très rapide, ce temps d'extraction représente la grande majorité de l'inférence.

La cause de ce temps de calcul réside dans la conception du *VRBFN*, et plus particulièrement dans la représentation des zones d'attention. En effet, les zones d'attention du *VRBFN* sont représentées par des masques de la taille de l'image d'entrée, et donc les calculs pour extraire les zones d'attention sont effectués sur des images entières, avec un masque par neurone. Afin de réduire ce temps de calcul, deux possibilités sont envisageables :

- Patch d'attention : On pourrait utiliser uniquement des patchs localisés pour représenter les zones d'attention et ainsi les multiplier à l'entrée à l'endroit localisé du patch. En plus d'améliorer le temps de calcul, cette méthode permettrait aussi de dissocier la taille de l'entrée et la taille des filtres, permettant donc de réduire l'espace nécessaire dans la mémoire pour l'initialisation du réseau.
- Parallélisation : Chaque neurone dans le module d'extraction du *VRBFN* est indépendant des autres. Il serait donc possible de paralléliser l'ensemble des neurones pour gagner du temps lors de l'extraction.

Avec ces deux améliorations, le réseau pourrait plus facilement être utilisé sur de grandes images avec de grands batch.

6.2.2 Efficacité du VRBFN

Dans les deux scénarios de *Pybullet*, le *VRBFN* rencontre des difficultés. Pour la tâche de localisation, ces difficultés ont été résolues dans le chapitre précédent en utilisant une distribution normale des centres des zones d'attention ou encore une distribution rétinienne. Cependant, ces distributions ne sont pas aussi efficaces sur des scénarios moins centrés comme *MWH*. L'idée d'une distribution universelle des zones d'attention, introduite dans chapitre précédent, pourrait potentiellement être apprise grâce à un ensemble de données regroupant un grand nombre d'images à la première personne. La recherche d'une telle distribution est un problème complexe, et une étude plus approfondie du liens entre les capteurs visuels de l'humain et de l'extracteur visuel *VRBFN* serait nécessaire.

De plus, même avec une telle distribution, certains scénarios seraient encore difficiles à résoudre, comme la tâche d'atteignabilité avec le bras *Kuka*. Dans ce type de scénario, l'ajout d'un module de mémoire semble être obligatoire, même si nos résultats sur le *LSTM* disponible dans l'annexe A.1 ne vont pas dans ce sens. En effet, nos résultats montrent que le *LSTM* n'améliore pas l'apprentissage de l'agent, mais nous n'avons pas considéré l'intégration des actions ou d'autres éléments intrinsèques à l'agent aux états, comme recommandé par différents travaux présentés dans ce manuscrit utilisant le *LSTM* dans des environnements complexes.

Le LSTM pourrait aussi être considéré dans l'apprentissage d'une distribution générale des paramètres gaussiens du *VRBFN* afin d'avoir une distribution dépendant des actions que l'agent effectue.

6.2.3 Plasticité du *VRBFN*

Le VRBFN dispose de neurones à activation parcimonieuse. Pour certains scénarios, plus de la moitié des neurones peuvent être supprimés sans chute de récompense. Cette caractéristique pourrait être utilisée afin de réduire la taille du réseau et de fine-tuner le réseau plus petit pour perfectionner l'agent. Cependant, ce n'est pas la seule utilité. En effet, lors de l'entraînement, les neurones pourraient être considérés comme inutiles s'ils ne sont pas activés pendant un certain moment. Ces neurones pourraient être effacés et redistribués avec de nouveaux paramètres gaussiens afin d'augmenter le nombre d'informations reçues par l'agent.

Dans un premier temps, on pourrait étudier l'effet de la plasticité aléatoire des neurones, c'est-à-dire régénérer un neurone avec des paramètres aléatoires. Ensuite, l'idée serait d'utiliser les découvertes sur l'environnement pour régénérer de nouveaux neurones qui ajouteront une information pertinente sur la tâche en cours.

6.3 Perspectives d'application

Du fait de sa faible complexité d'implémentation et d'utilisation, le *VRBFN* pourrait être testé dans tous les types de problèmes visuels requérant de l'intelligence artificielle en tant que système de perception. Plus particulièrement, on peut diviser en deux parties les possibles applications du *VRBFN* : une première partie sur les applications théoriques et ensuite une partie sur les applications pratiques.

6.3.1 Application théorique

Le *VRBFN* pourrait être utile au développement de nouvelles théories de l'apprentissage ou encore à la compréhension des modèles d'apprentissage. Différentes approches théoriques sont présentées ci-dessous, cette liste n'est pas exhaustive :

- Explicabilité : Le *VRBFN* pourrait jouer un rôle important dans l'explicabilité des méthodes d'apprentissage visuel autonome. En effet, comme montré dans la partie 4.1, le *VRBFN* peut résoudre des problèmes visuels simples avec uniquement une approximation linéaire du lien entre l'état et l'action.

L'approximation linéaire rend l'explication des choix de l'agent plus directe car les stimuli sont locaux, et donc on peut extraire des informations intéressantes sur les causes de l'action choisie. Cet aspect a été partiellement introduit dans 4.2.3 avec l'interprétation des activations neuronales. Un travail plus approfondi pourrait mettre en évidence les causes des choix des actions et ainsi permettre d'expliquer les choix de l'agent.

- Apprentissage de modèle de perception : Comme expliqué dans la partie précédente, des travaux restent à effectuer sur la distribution des paramètres gaussiens du modèle. D'un point de vue théorique, la découverte de la disposition optimale des paramètres gaussiens pourrait mettre en lumière certains liens avec l'œil humain et générer de possibles améliorations d'un modèle général de perception.

- Algorithmes de *RL* : Au vu de la simplicité d'utilisation, on pourrait utiliser ce modèle de perception pour le test de nouveaux algorithmes de *RL* appliqués au domaine du visuel. En comparaison avec les *CNN* qui sont largement utilisés avec les images, le *VRBFN* ne nécessite pas de connaissance sur l'architecture des réseaux *CNN*, ce qui peut parfois être critique lors de l'apprentissage de tâches et est souvent considéré plus proche de l'art que de la science.

6.3.2 Application pratique

Au stade actuel du développement, le *VRBFN* pourrait déjà être utilisé comme système primaire de perception pour des tâches dans le monde réel. Différentes applications sont proposées dans la liste non exhaustive ci-dessous :

- Déplacement visuel : Les résultats sur le scénario de *VizDoom*, *MWH*, montrent que le *VRBFN* est performant dans les tâches de déplacement. Le *VRBFN* pourrait être utilisé comme un module de déplacement pour des expériences dans lesquelles un agent doit se déplacer dans une zone avec différents objectifs. En effet, grâce à la parcimonie des activations, un agent doté d'un *VRBFN* peut facilement se repérer dans un environnement grâce à l'activation de différents motifs visuels pour différentes zones de l'environnement.

- Communication avec l'agent : Sachant que le *VRBFN* dispose de neurones à activation localisée dans l'espace de l'image, on peut très bien imaginer transmettre des informations pour guider l'agent. Par exemple, en plaçant sur des objets un symbole composé de plusieurs cercles de couleur et en reproduisant à l'identique les zones d'attention réceptrices de ce symbole dans le *VRBFN*, ce qui permettra à l'agent d'identifier plus facilement certains objets. De plus, cela facilitera l'ajout de contraintes car la réception de ces symboles sera identifiable aussi par l'homme.

- Détection d'anomalies : Certains tests lors de l'apprentissage des paramètres gaussiens ont montré que le *VRBFN* avait de bonnes capacités de détection de nouveauté et de déjà vu, ce qui peut être directement lié à une application de détection d'anomalies. De plus, la localité des activations permettrait de retrouver la zone où se trouve l'anomalie de manière directe, dans le cas d'une approximation linéaire.

Annexes

Appendix A

Résultats additionnels

A.1 Résultats et implémentation avec le LSTM

En plus des expériences comparatives présentées dans le chapitre 4, nous avons également réalisé des expérimentations pour étudier des modèles avec mémoire. Pour cela une couche *LSTM* a été ajoutée à chaque méthode comme indiqué par un cadre rouge dans les schémas de la figure A.1. À chaque début d'épisode, les paramètres du *LSTM* sont initialisés à zéro. L'agent reçoit les images une par une, et à chaque inférence, les paramètres évoluent. Nous sauvegardons dans la mémoire de l'agent des séquences de six images. Pour l'entraînement, les paramètres sont initialisés à zéro et l'optimisation se fait sur des lots de séquences.

Les courbes d'apprentissage A.2, A.3 montrent que l'ajout de *LSTM* n'a pas amélioré les performances ; au contraire, l'entraînement semble plus difficile et donc plus long. Les résultats sont largement inférieurs à ceux obtenus sans *LSTM*. En effet, dans le scénario *DTC*, les méthodes avec le *LSTM* n'obtiennent pas de récompenses

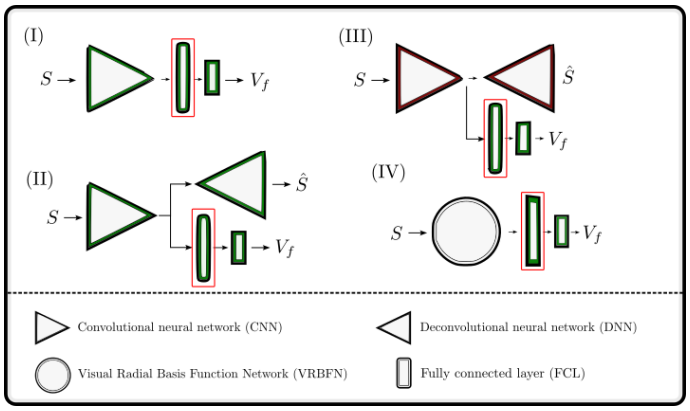


Figure A.1: Schéma des différentes méthodes utilisées dans les résultats suivants. L'encadrement en rouge des couches *FCL* correspond à un remplacement par une couche *LSTM*.

supérieures à -0.60, alors que sans *LSTM*, toutes les méthodes, sauf l'*AE*, finissent l'entraînement avec des récompenses supérieures à -0.60. Pour *HGS* et *MWH*, les méthodes avec *LSTM* ne montrent pas de signe d'apprentissage, contrairement à l'entraînement sans *LSTM*. On retrouve la même constatation pour le scénario d'atteinte avec le bras *Kuka*. Le scénario de fixation est le seul à montrer des signes de convergence lors de l'entraînement ; cependant, les récompenses obtenues lors de l'entraînement sont inférieures à celles des méthodes sans *LSTM*. D'autres méthodes d'implémentation pourraient être étudiées, comme l'ajout des actions prises par l'agent à l'entrée du *LSTM*. De plus, l'initialisation des paramètres du *LSTM* pourrait être différente, tout comme l'optimisation.

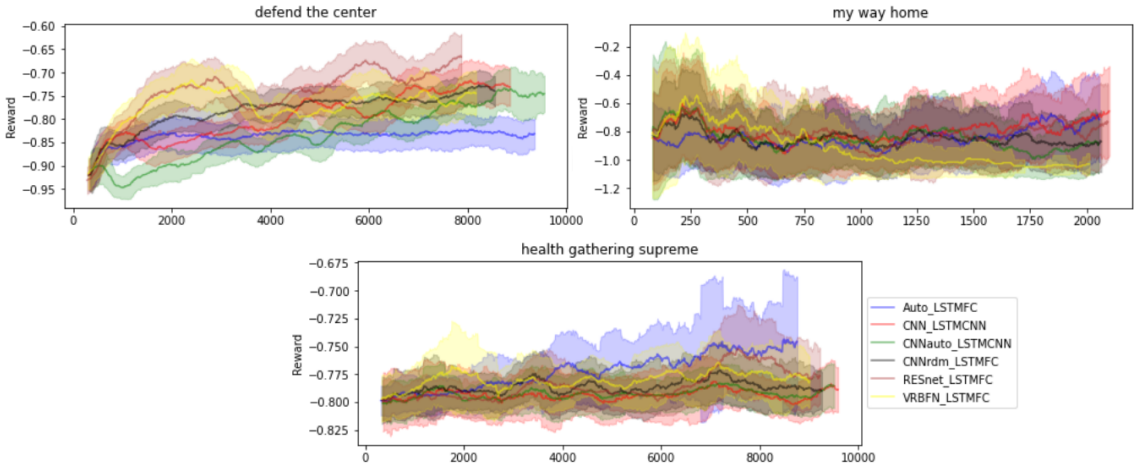


Figure A.2: Courbes d'entraînement des six méthodes dans les scénarios de Vizdoom

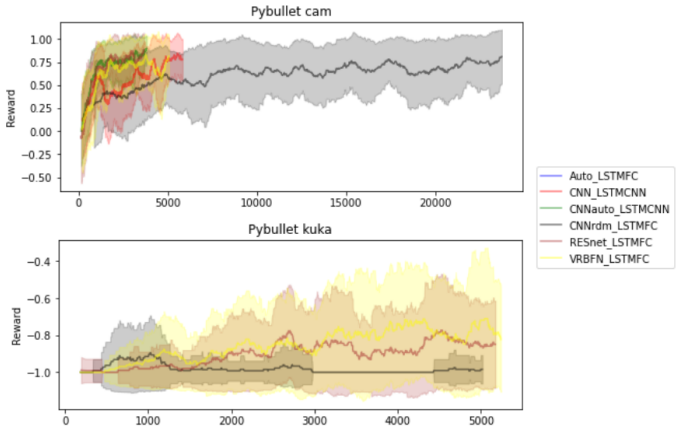


Figure A.3: Courbes d'entraînement des six méthodes dans les scénarios de Pybullet, les méthodes ne montrant aucun apprentissage n'ont pas été tracées

A.2 Evaluation de la robustesse à différents modèles de bruits

Dans ces tests, nous avons évalué l'impact de trois types de bruit généraux et d'un bruit localisé sur les deux méthodes les plus performantes, le $VRBFN$ et le $RESnet$, avec une seed spécifique pour chaque méthode. Le scénario choisi est le scénario DTC, et les résultats sont calculés sur cent épisodes. Chaque bruit est ajouté à l'image, ensuite l'image bruitée est clippée entre 0 et 1. Pour chaque bruit, trois niveaux de bruitage ont été testés (faible, moyen et fort).

A.2.1 Bruit gaussien

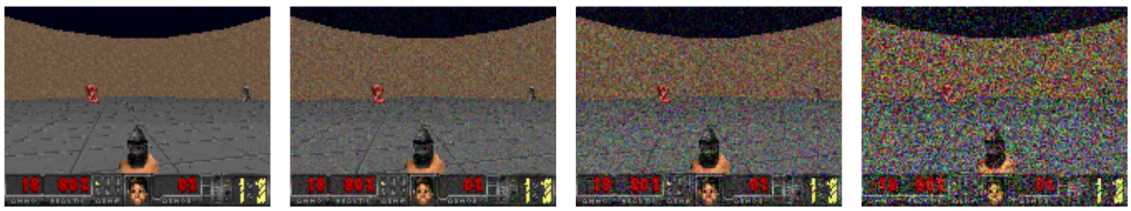


Figure A.4: Effet du bruit gaussien sur une image du scénario *defend the center*. La première image à gauche correspond à l'image originale. Différents niveaux de bruit ont été testés, du moins bruité à gauche au plus bruité à droite.

Un bruit gaussien est généré en prenant des nombres aléatoires suivant une loi normale de centre 0 et d'écart type σ. On a respectivement pour les trois types de bruit, faible, moyen et fort, trois valeurs de σ : $0.05, 0.1, 0.2$. Ces trois niveaux de

bruit sont illustrés dans l'image A.4 et les résultats sont répertoriés dans le tableau A.1. Le *VRBFN*

	`sans bruit`	$\sigma = 0.05$	$\sigma = 0.1$	$\sigma = 0.2$
RESnet	16.5 ± 3.7	16.3 ± 3.31 (-1.21%)	10.26 ± 5 (-37.82%)	3.42 ± 1.93 (-79.27%)
VRBFN	$\mathbf{18.7 \pm 3.7}$	$\mathbf{18.48 \pm 3.84}$ (-1.18%)	$\mathbf{15.12 \pm 2.76}$ (-19.04%)	$\mathbf{3.79 \pm 1.44}$ (-79.73%)

Table A.1: Résultats du *VRBFN* et du *RESnet* avec ajout de bruit gaussien sur les images. Le score est donné pour chaque intensité de bruit avec le pourcentage de perte par rapport au score initial.

A.2.2 Bruit sel et poivre

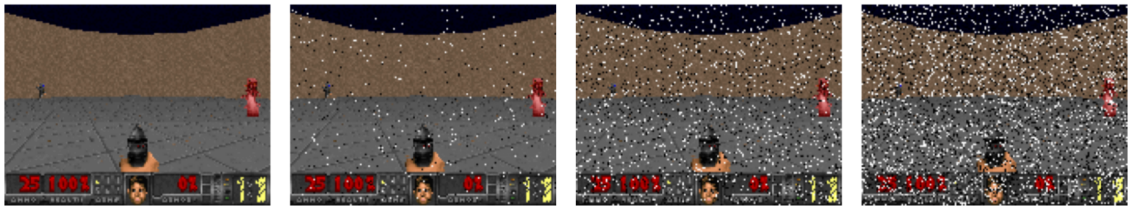


Figure A.5: Effet du bruit sel et poivre sur une frame du scénario *DTC*. La première image à gauche correspond à l'image originale, tandis que les images suivantes présentent différents niveaux de bruit, du moins bruité à gauche au plus bruité à droite.

Un bruit sel et poivre est généré en tirant pour chaque pixel un nombre aléatoire : si ce nombre est inférieur à une valeur τ, alors le pixel est changé en pixel noir ; si ce nombre est supérieur à $1 - \tau$, alors le pixel devient blanc ; sinon, le pixel ne change pas. On a respectivement pour les trois forces de bruit, faible, moyen et fort, trois valeurs de τ : $0.01, 0.05, 0.1$. Ces trois niveaux de bruit sont illustrés dans l'image A.5 et les résultats sont répertoriés dans le tableau A.2.

A.2.3 Bruit de Poisson

Un bruit de Poisson est généré directement via un module intégré à *numpy*. Comme pour les autres bruits, trois forces de bruitage ont été sélectionnées. Ces trois niveaux de bruit sont illustrés dans l'image A.6 et les résultats sont répertoriés dans le tableau A.3.

	`sans bruit`	$\tau = 0.01$	$\tau = 0.05$	$\tau = 0.1$
RESnet	16.5 ± 3.7	2.23 ± 1.42	-0.97 ± 0.17	-1.0 ± 0.0
		(-86.49%)	(-105.88%)	(-106.06%)
VRBFN	$\mathbf{18.7 \pm 3.7}$	$\mathbf{16.89 \pm 3.92}$	$\mathbf{12.46 \pm 3.80}$	$\mathbf{8.27 \pm 3.28}$
		(-9.68%)	(-33.37%)	(-55.78%)

Table A.2: Résultats du *VRBFN* et *RESnet* avec ajout de bruit sel et poivre sur les images. Le score est donné pour chaque intensité de bruit avec le pourcentage de perte par rapport au score initial.

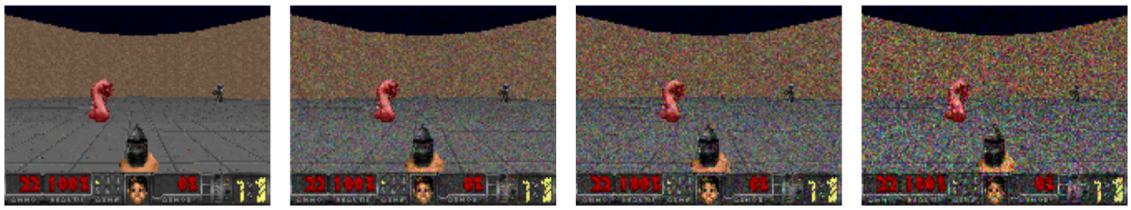


Figure A.6: Effet du bruit de Poisson sur une frame du scénario *defend the center*. La première image à gauche correspond à l'image originale. Différents niveaux de bruit ont été testés, du moins bruité à gauche au plus bruité à droite.

	`sans bruit`	faible	moyen	fort
RESnet	16.5 ± 3.7	15.03 ± 4.72	7.82 ± 4.46	1.99 ± 1.98
		(-8.91%)	(-52.61%)	(-87.94%)
VRBFN	$\mathbf{18.7 \pm 3.7}$	$\mathbf{17.94 \pm 2.98}$	$\mathbf{16.1 \pm 2.77}$	$\mathbf{10.99 \pm 2.28}$
		(-4.06%)	(-13.9%)	(-41.23%)

Table A.3: Résultats du *VRBFN* et *RESnet* avec ajout de bruit de Poisson sur les images. Le score est donné pour chaque intensité de bruit avec le pourcentage de perte par rapport au score initial.

A.2.4 Bruit uniforme localisé

Ce bruit uniforme localisé est généré en transformant un pourcentage de l'image en bruit uniforme. Pour chaque nouvel état, on tire aléatoirement une position sur l'image où l'on va placer le bruit. Pour ce bruit, nous avons testé les méthodes avec trois tailles $\frac{1}{\tau} *$ taille de l'image de bruit différentes : $\frac{1}{20}, \frac{1}{10}, \frac{1}{5}$. Ces trois niveaux de bruit sont illustrés dans l'image A.7 et les résultats sont répertoriés dans le tableau A.4.

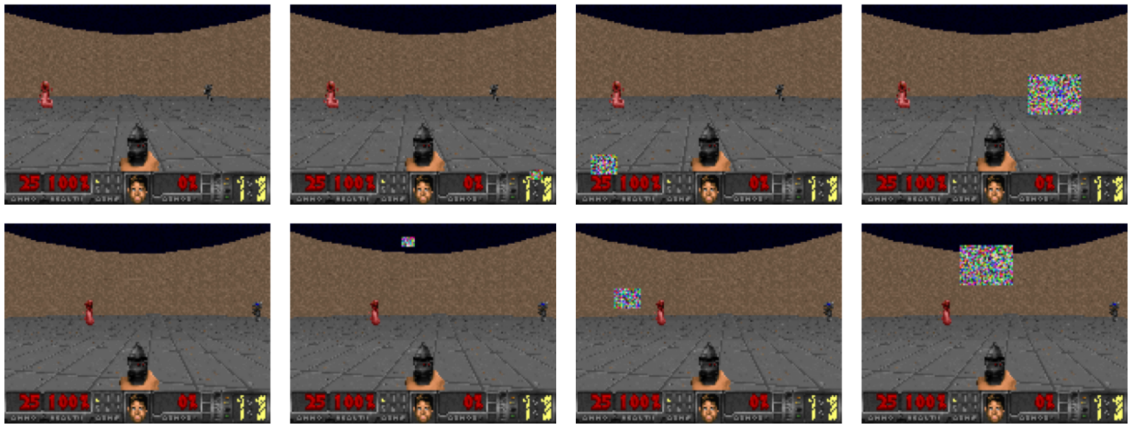


Figure A.7: Effet du bruit uniforme localisé sur deux frames du scénario *defend the center*. La première image à gauche correspond à l'image originale. Différents niveaux de bruit ont été testés, du moins bruité à gauche au plus bruité à droite.

	`sans bruit`	$\tau = 20$	$\tau = 10$	$\tau = 5$
RESnet	16.5 ± 3.7	16.09 ± 3.74 (-2.48%)	10.7 ± 3.44 (-35.15%)	2.24 ± 1.80 (-86.42%)
VRBFN	$\mathbf{18.7 \pm 3.7}$	$\mathbf{17.83 \pm 4.06}$ (-4.65%)	$\mathbf{15.36 \pm 3.72}$ (-17.84%)	$\mathbf{8.99 \pm 3.32}$ (-51.88%)

Table A.4: Résultats du *VRBFN* et *RESnet* avec ajout de bruit uniforme localisé sur les images. Le score est donné pour chaque intensité de bruit avec le pourcentage de perte par rapport au score initial.

A.2.5 Conclusion

Les résultats obtenus après l'application des différents bruits montrent une forte robustesse de la méthode *VRBFN* par rapport au *RESnet* pour tous les bruits, sauf le bruit gaussien avec un $\sigma = 0.2$, qui donne des résultats similaires pour les deux méthodes en termes de perte (voir tableau A.1). Avec cette intensité de bruit, les deux méthodes ne parviennent pas à maintenir un score convenable.

A.3 Etude sur l'apprentissage des paramètres gaussiens pour la classification visuelle

Dans le cadre du stage de fin d'études de Charifou Orou Mousse, nous avons étudié l'apprentissage des paramètres gaussiens du *VRBFN* pour des tâches de classification d'images. L'objectif de ce stage était de tester le *VRBFN* dans des exercices classiques de classification. De plus, nous voulions étudier différentes méthodes d'optimisation des paramètres. Nous avons d'abord testé deux approches classiques d'optimisation des paramètres du *RBFN* :

- **Particle Swarm Optimization (*PSO*)** : L'optimisation par essaims particulaires est une méta-heuristique inspirée du comportement des oiseaux [146]. L'utilisation de la *PSO* pour optimiser les réseaux gaussiens remonte aux années 2000 avec [147], qui utilise la *PSO*pour déterminer le nombre de neurones gaussiens ainsi que leurs paramètres. Plus récemment, la *PSO* a été utilisée pour déterminer les paramètres gaussiens ainsi que les poids de la couche de sortie [148], [149].
- **Genetic Algorithm (*GA*)** : Les algorithmes génétiques, introduits dans les années 70 par John Holland [150], sont fondés sur le concept de la sélection naturelle. Les *GA* ont été utilisés afin de trouver la meilleure combinaison de paramètres gaussiens à travers des processus tels que la sélection, le croisement et la mutation.

Ces méthodes sont très coûteuses en termes de calcul et dépendent de l'initialisation des paramètres gaussiens. Avec certaines initialisations, nous avons observé que les paramètres ne bougeaient pas, car ces méthodes ne trouvaient pas de meilleure configuration.

Par la suite, nous avons utilisé la descente de gradient pour l'entrainement complet du *VRBFN*, y compris les paramètres gaussiens et les poids de la couche de sortie. Deux phases d'entraînement ont été effectuées. Dans un premier temps, le *VRBFN* était utilisé dans une version auto-encodeur afin d'apprendre une disposition des gaussiennes pour reconstruire au mieux l'image d'entrée. Ensuite, le *VRBFN* entraîné a été ré-entraîné pour classifier les images. Les premières données d'entraînement étaient des images de pièces industrielles rondes (Figure A.8), l'objectif étant de déterminer si les pièces contenaient un défaut ou non. Les paramètres du *VRBFN* après l'apprentissage de chaque phase sont disponibles dans les graphiques A.9 et A.10.

On peut voir qu'avant la phase de classification, la position x, y des centres des neurones correspond bien à la forme de l'objet d'étude. De plus, lors de l'apprentissage

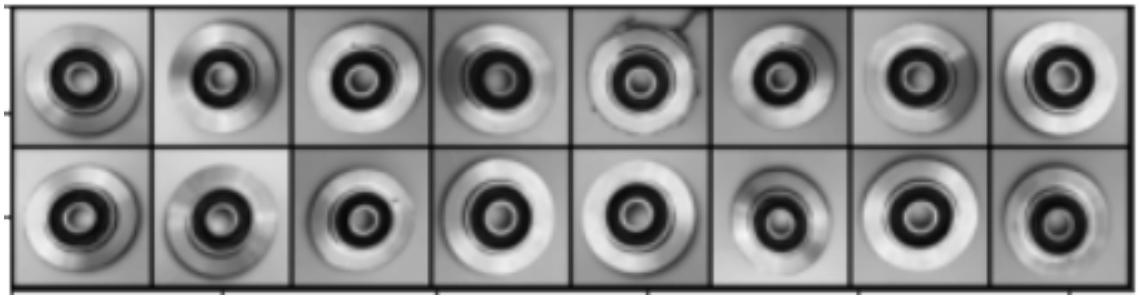


Figure A.8: Données utilisées pour la classification des défauts

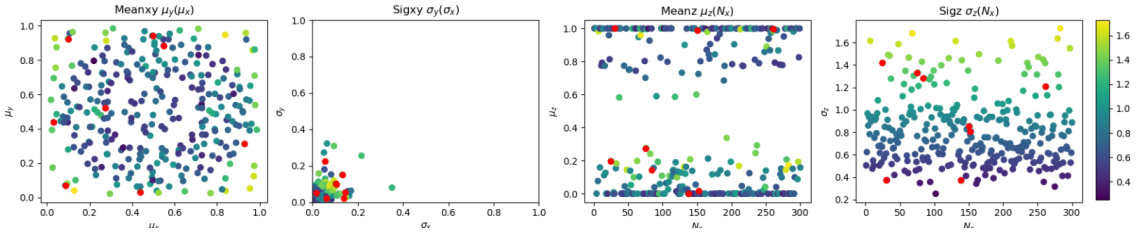


Figure A.9: Configurations des paramètres du *VRBFN* après apprentissage avec l'auto-encodeur. Les points rouges correspondent à des neurones inactifs sur l'ensemble des images, les couleurs du bleu au jaune représentent σ_z.

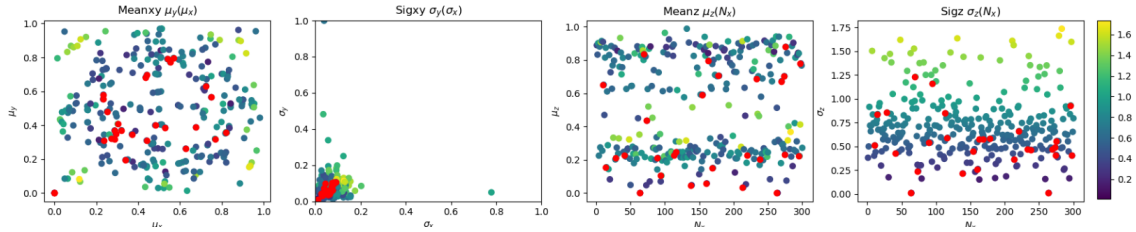


Figure A.10: Configurations des neurones *VRBFN* après l'apprentissage pour la classification. Les points rouges correspondent à des neurones inactifs sur l'ensemble des images, les couleurs du bleu au jaune représentent σ_z.

du *VRBFN* auto-encodeur, les centres d'intensité des neurones sont plutôt répartis en valeurs proches de 1 et 0, ce qui peut faire penser à une disposition *ON-OFF* des neurones, similaire à ce que l'on observe dans l'œil humain.

Après la classification, ces tendances sont toujours plus ou moins observables, et dans les deux cas, on observe une cohérence dans les paramètres. En effet, si l'on regarde les couleurs, les points jaunes, qui ont une forte tolérance (c'est-à-dire qu'ils s'activent même si l'intensité est loin du centre du noyau), sont répartis aux extrémités de l'image et sont en général de plus grande taille. Au contraire, les neurones qui ont une faible tolérance, en bleu, se trouvent plus au centre de l'image et sont de petite taille. Ces observations sont encore une fois proches de ce que l'on observe chez l'humain, où les récepteurs visuels situés aux extrémités de la rétine couvrent souvent

une plus grande zone de réception, et leurs seuils d'activation sont souvent inférieurs, permettant ainsi une activité plus abondante. Les points rouges représentent les neurones considérés inactifs.

Ces résultats montrent que l'apprentissage des paramètres du *VRBFN* est possible. De plus, les résultats de classification avec des paramètres aléatoires du *VRBFN* étaient de 96.1% de bonne classification, et ceux obtenus après l'entraînement étaient de 99.4%.

L'étude s'est ensuite poursuivie par l'apprentissage des paramètres avec la méthode du *contrastive learning* afin de vérifier que notre réseau peut utiliser des méthodes plus complexes d'entraînement. Cette méthode consiste à apprendre les caractéristiques d'un ensemble de données sans label. Pour ce faire, on apprend au modèle à distinguer les données similaires de celles qui sont différentes. Le modèle a été entraîné avec cette méthode de *contrastive learning* sur le dataset de pièces industrielles présenté ci-dessus, mais aussi sur le dataset *FMNIST*, qui est constitué d'images de différents vêtements et chaussures.

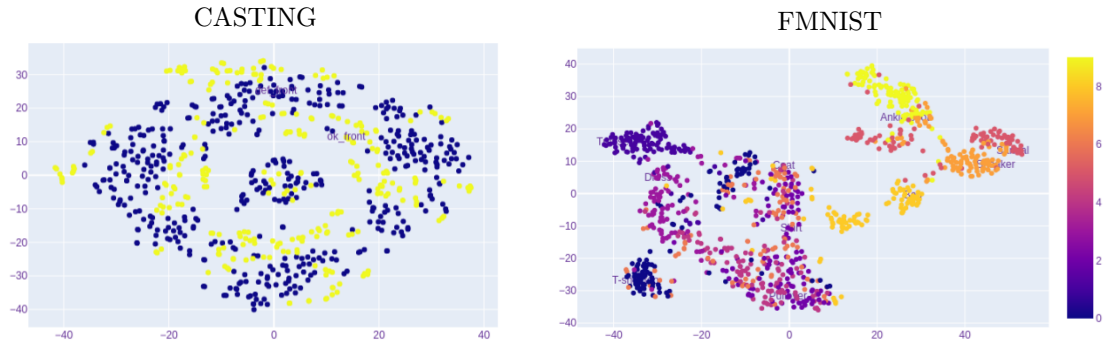


Figure A.11: Visualisation par *TSNE* de l'espace latent du *VRBFN* auto-encodeur sur les deux datasets. Dans *CASTING*, le bleu correspond aux défauts et le jaune aux images saines. Pour *FMNIST*, chaque couleur représente une classe.

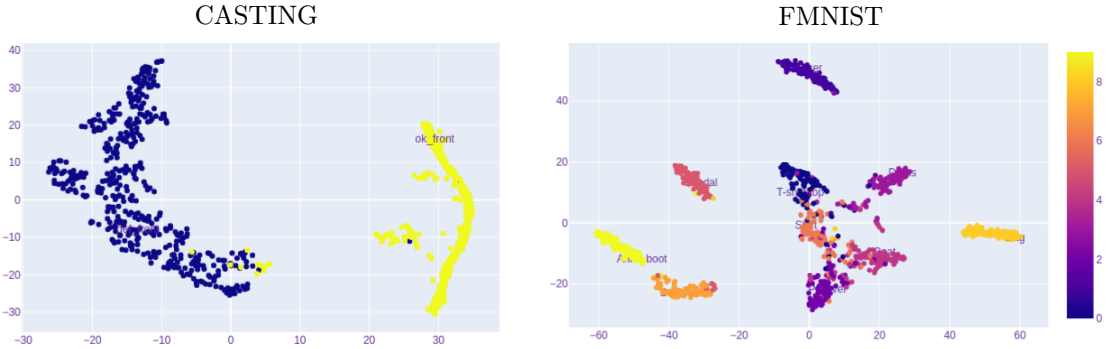


Figure A.12: Visualisation de l'espace latent en sortie du *VRBFN*. Apprentissage par *CL* sur les deux datasets.

Après l'entraînement par *contrastive learning* (*CL*) pour apprendre les paramètres gaussiens, une comparaison de l'espace latent obtenu a été réalisée entre l'auto-

encodeur sans *CL* et avec *CL* sur les deux bases de données avec une méthode de *TSNE*. Les résultats qualitatifs dans les figures A.11 et A.12 montrent une supériorité de la méthode de *contrastive learning* et démontrent que l'on peut apprendre les paramètres du *VRBFN* afin d'obtenir une bonne séparation des données.

Les résultats sur l'apprentissage des paramètres gaussiens sont plutôt satisfaisants. Cependant, si nous voulons appliquer ce type d'apprentissage dans un problème d'apprentissage par renforcement, nous ferons face aux mêmes problèmes obtenus par l'auto-encodeur ou toute autre méthode qui apprend à reconstruire l'environnement avant d'apprendre à résoudre la tâche. Le modèle sera trop personnalisé pour un problème particulier, et il faudra ré-entraîner un nouveau modèle pour chaque nouveau scénario. Idéalement, nous voulons un modèle *VRBFN* général, avec des paramètres permettant d'extraire au mieux l'information importante dans n'importe quel environnement. Par exemple, la disposition rétinienne pourrait faire office de positionnement général des centres et tailles des gaussiennes, et un apprentissage sur une base de données générale à la première personne permettrait d'obtenir les paramètres d'intensité et de tolérance des neurones.

Bibliographie

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] K. Esslinger, R. Platt, and C. Amato, "Deep transformer q-networks for partially observable reinforcement learning," 2022.

[3] T. Lesort, N. Diaz-Rodriguez, J. F. Goudou, and D. Filliat, "State representation learning for control: An overview," *Neural Networks*, vol. 108, pp. 379–392, 2018.

[4] G. Westermann and D. Mareschal, "Thematic collection : Articles from parts to wholes : Mechanisms of development in infant visual object processing," vol. 5, no. 2, pp. 131–151, 2004.

[5] R. L. Fantz, "Visual discrimination in a neonate chimpanzee," pp. 59–66, 1958.

[6] M. J. Powell, "Radial basis functions for multivariable interpolation: A review," *Algorithms for approximation*, pp. 143–167, 1987.

[7] J. Hautot, C. Teuliere, and N. Azzaoui, "Visual radial basis q-network," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 13364 LNCS, pp. 318–329, 2022.

[8] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[9] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *Towards Data Sci*, vol. 6, no. 12, pp. 310–316, 2017.

[10] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," *Advances in neural information processing systems*, vol. 2, 1989.

[11] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019.

[12] A. Dittadi, F. Trauble, M. Wuthrich, F. Widmaier, P. Gehler, O. Winther, F. Locatello, O. Bachem, B. Scholkopf, and S. Bauer, "Representation learning for out-of-distribution generalization in reinforcement learning," pp. 1–24, 2021.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] D. BROOMHEAD, "Multivariable functional interpolation and adaptive networks," *Complex Systems*, vol. 2, pp. 321–355, 1988.

[15] J. Moody and C. Darken, *Learning with localized receptive fields*, 1988.

[16] J. Moody and C. J. Darken, "Fast learning in networks of locally-tuned processing units," vol. 294, pp. 281–294, 1989.

[17] J. Park and I. W. Sandberg, "Approximation and radial-basis-function networks," vol. 316, pp. 305–316, 1993.

[18] E. J. Hartman, J. D. Keeler, and J. M. Kowalski, "Layered neural networks with gaussian hidden units as universal approximations," *Neural Computation*, vol. 2, no. 2, pp. 210–215, 1990.

[19] F. G. Tomaso Poggio, "89] t. poggio, f. girosi: A theory of networks for approximation and learning," AI Memo, Tech. Rep.

[20] Q. Que and M. Belkin, "Back to the future: Radial basis function networks revisited," *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016*, vol. 51, pp. 1375–1383, 2016.

[21] A. Alexandridis and H. Sarimveis, *Control of processes with multiple steady states using MPC and RBF neural networks*. 2011, vol. 29, pp. 698–702.

[22] A. da Motta Salles Barreto and C. W. Anderson, "Restricted gradient-descent algorithm for value-function approximation in reinforcement learning," *Artificial Intelligence*, vol. 172, no. 4-5, pp. 454–482, 2008.

[23] C. S. K. Dash, A. K. Behera, S. Dehuri, and S.-B. Cho, "Radial basis function neural networks: A topical state-of-the-art survey," *Open Computer Science*, vol. 6, no. 1, pp. 33–63, 2016.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[28] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[29] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[30] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, 2015.

[31] E. Thorndike, *Animal intelligence: Experimental studies*. Routledge, 2017.

[32] C. B. Ferster and B. F. Skinner, "Mixed schedules.," 1957.

[33] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[34] R. BELLMAN, "A markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957.

[35] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[36] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski, "Vizdoom: A doom-based ai research platform for visual reinforcement learning," in *IEEE Conference on Computatonal Intelligence and Games, CIG*, 2016.

[37] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell, "The malmo platform for artificial intelligence experimentation.," in *Ijcai*, 2016, pp. 4246–4247.

[38] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[39] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Kuttler, A. Lefrancq, S. Green, V. Valdes, A. Sadik, *et al.*, "Deepmind lab," *arXiv preprint arXiv:1612.03801*, 2016.

[40] X. Puig, E. Undersander, A. Szot, M. D. Cote, T.-Y. Yang, R. Partsey, R. Desai, A. W. Clegg, M. Hlavac, S. Y. Min, *et al.*, "Habitat 3.0: A co-habitat for humans, avatars and robots," *arXiv preprint arXiv:2310.13724*, 2023.

[41] S. Milani, A. Kanervisto, K. Ramanauskas, S. Schulhoff, B. Houghton, and R. Shah, "Bedd: The minerl basalt evaluation and demonstrations dataset for training and benchmarking agents that solve fuzzy tasks," *arXiv preprint arXiv:2312.02405*, 2023.

[42] E. Coumans and Y. Bai, *Pybullet, a python module for physics simulation for games, robotics and machine learning*, http://pybullet.org, 2016–2021.

[43] M. Körber, J. Lange, S. Rediske, S. Steinmann, and R. Glück, "Comparing popular simulation environments in the scope of robotics and reinforcement learning," *arXiv preprint arXiv:2103.04616*, 2021.

[44] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, Sep. 2015.

[45] V. Mnih, A. P. Badia, L. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *33rd International Conference on Machine Learning, ICML 2016*, vol. 4, pp. 2850–2869, 2016.

[46] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," pp. 1–12, 2017.

[47] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pp. 3215–3222, 2018.

[48] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *35th International Conference on Machine Learning, ICML 2018*, vol. 5, pp. 2976–2989, 2018.

[49] M. Olsson, S. Malm, and K. Witt, "Evaluating the effects of hyperparameter optimization in vizdoom," 2022.

[50] M. Andrychowicz, A. Raichuk, P. Stanczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem, "What matters in on-policy reinforcement learning? a large-scale empirical study," no. 1, 2020.

[51] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, pp. 229–256, 1992.

[52] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," in *32nd International Conference on Machine Learning, ICML 2015*, vol. 3, 2015, pp. 1889–1897.

[53] J. Matas and Š. Obdržálek, "Object recognition methods based on transformation covariant features," in *2004 12th European Signal Processing Conference*, IEEE, 2004, pp. 1721–1728.

[54] S. Jodogne, C. Briquet, and J. H. Piater, "Approximate policy iteration for closed-loop learning of visual tasks," in *Machine Learning: ECML 2006: 17th European Conference on Machine Learning Berlin, Germany, September 18-22, 2006 Proceedings 17*, Springer, 2006, pp. 210–221.

[55] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE conference on computer vision and pattern recognition*, IEEE, 2012, pp. 3642–3649.

[56] S. Lange, M. Riedmiller, and A. Voigtlander, "Autonomous reinforcement learning on raw visual input data in a real world application," *Proceedings of the International Joint Conference on Neural Networks*, no. June, 2012.

[57] S. Ghiassian, B. Rafiee, Y. L. Lo, and A. White, "Improving performance in reinforcement learning by breaking generalization in neural networks," *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, vol. 2020-May, no. 1, pp. 438–446, 2020.

[58] T. Zhang, X. Wang, B. Liang, and B. Yuan, "Catastrophic interference in reinforcement learning: A solution based on context division and knowledge distillation," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[59] G. Lample and D. S. Chaplot, "Playing fps games with deep reinforcement learning," *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, no. 2015, pp. 2140–2146, 2017.

[60] Y. Wu, "(doom)the task of playing a first-person-shooting (fps) game in a 3d environment is much more challenging than playing most atari games as it in-volves a wide variety of skills, such as navigating through a map, collecting items, recognizing and fighting," no. 2015, pp. 1–10, 2017.

[61] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," *AAAI Fall Symposium - Technical Report*, vol. FS-15-06, pp. 29–37, 2015.

[62] C. Romac and V. Beraud, "Deep recurrent q-learning vs deep q-learning on a simple partially observable markov decision process with minecraft," 2019.

[63] R. K. Brejl, H. Purwins, and H. Schoenau-Fog, "Exploring deep recurrent q-learning for navigation in a 3d environment," *EAI Endorsed Transactions on Creative Technologies*, vol. 5, no. 14, p. 153 641, 2018.

[64] D. Akimov and I. Makarov, "Deep reinforcement learning with vizdoom first-person shooter?" *CEUR Workshop Proceedings*, vol. 2479, pp. 3–17, 2019.

[65] M. Wu, C. M. Ulrich, and H. Salameh, "Training a game ai with machine learning training a game ai with machine learning bachelor project it-university of copenhagen," no. May, pp. 0–88, 2020.

[66] T. Ni, B. Eysenbach, and R. Salakhutdinov, "Recurrent model-free rl can be a strong baseline for many pomdps," 2021.

[67] P. Zhu, X. Li, P. Poupart, and G. Miao, "On improving deep reinforcement learning for pomdps," 2018.

[68] M. Wydmuch, M. Kempka, and W. Jaskowski, "Vizdoom competitions: Playing doom from pixels," *IEEE Transactions on Games*, vol. 11, no. 3, pp. 248–259, 2018.

[69] D. S. Ratcliffe, S. Devlin, U. Kruschwit, and Z. Citi, "Clyde: A deep reinforcement learning doom playing agent," *AAAI Workshop - Technical Report*, vol. WS-17-01 -, no. September, pp. 983–990, 2017.

[70] J. Beck, K. Ciosek, S. Devlin, S. Tschiatschek, C. Zhang, and K. Hofmann, "Amrl : A ggregated m emory f or r einforcement l earning," *Iclr 2020*, pp. 1–20, 2020.

[71] L. Meng, R. Gorbet, and D. Kulic, "Memory-based deep reinforcement learning for pomdps," *IEEE International Conference on Intelligent Robots and Systems*, no. Iros, pp. 5619–5626, 2021.

[72] I. T. Jolliffe, "Principal component analysis: A beginner's guide—i. introduction and application," *Weather*, vol. 45, no. 10, pp. 375–382, 1990.

[73] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[74] W. Curran, T. Brys, M. Taylor, and W. Smart, "Using pca to efficiently represent state spaces," no. May, 2015.

[75] S. Parisi, S. Ramstedt, and J. Peters, "Goal-driven dimensionality reduction for reinforcement learning," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-Septe, pp. 4634–4639, 2017.

[76] J. S. Albus, "A theory of cerebellar function," *Mathematical biosciences*, vol. 10, no. 1-2, pp. 25–61, 1971.

[77] Y. Wu, H. Wang, B. Zhang, and K.-L. Du, "Using radial basis function networks for function approximation and classification," *ISRN Applied Mathematics*, vol. 2012, pp. 1–34, 2012.

[78] A. J. Howell and H. Buxton, "Face recognition using radial basis function neural networks," *TH EN IN TH WH ITE HOU SE PAPER S G raduate R esearch in the C ognitive and C om puting SciencesatSussex*, p. 35, 1996.

[79] C. E. Thomaz, R. Q. Feitosa, and A. Veiga, "Design of radial basis function network as classifier in face recognition using eigenfaces," *Proceedings - Brazilian Symposium on Neural Networks, SBRN*, pp. 118–123, 1998.

[80] D. Kovacevic and S. Loncaric, "Radial basis function-based image segmentation using a receptive field," in *Proceedings of Computer Based Medical Systems*, IEEE, 1997, pp. 126–130.

[81] P. Guo, M. Hu, and Y. Jia, "Rbf network image representation with application to ct image reconstruction," *2006 International Conference on Computational Intelligence and Security, ICCIAS 2006*, vol. 2, no. 2, pp. 1865–1868, 2006.

[82] M. Daoud, M. Mayo, and S. J. Cunningham, "Rbfa: Radial basis function autoencoders," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, 2019, pp. 2966–2973.

[83] M. Amirian and F. Schwenker, "Radial basis function networks for convolutional neural networks to learn similarity distance metric and improve interpretability," *IEEE Access*, vol. 8, pp. 123 087–123 097, 2020.

[84] W. Chen, X. Han, G. Li, C. Chen, J. Xing, Y. Zhao, and H. Li, "Deep rbfnet: Point cloud feature learning using radial basis functions," *arXiv*, pp. 1–11, 2018.

[85] B. Šter and A. Dobnikar, "Adaptive radial basis decomposition by learning vector quantization," *Neural Processing Letters*, vol. 18, no. 1, pp. 17–27, 2003.

[86] R. Jonschkowski and O. Brock, "Learning state representations with robotic priors," *Autonomous Robots*, vol. 39, no. 3, pp. 407–428, 2015.

[87] N. Capel and N. Zhang, "Extended radial basis function controller for reinforcement learning," pp. 1–12, 2020.

[88] K. Asadi, R. E. Parr, G. D. Konidaris, and M. L. Littman, "Deep rbf value functions for continuous control," *arXiv*, 2020.

[89] S. Lange and M. Riedmiller, "Deep auto-encoder neural networks in reinforcement learning," *Proceedings of the International Joint Conference on Neural Networks*, 2010.

[90] D. Kimura, "Daqn: Deep auto-encoder and q-network," 2018.

[91] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, pp. 512–519, 2016.

[92] A. Dosovitskiy and V. Koltun, "Learning to act by predicting the future," *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pp. 1–14, 2017.

[93] S. Alvernaz and J. Togelius, "Autoencoder-augmented neuroevolution for visual doom playing," *2017 IEEE Conference on Computational Intelligence and Games, CIG 2017*, pp. 1–8, 2017.

[94] A. Anand, E. Racah, S. Ozair, Y. Bengio, M. A. Cote, and R. Devon Hjelm, "Unsupervised state representation learning in atari," *Advances in Neural Information Processing Systems*, vol. 32, no. NeurIPS, 2019.

[95] J. Oh, X. Guo, H. Lee, R. Lewis, and S. Singh, "Deep action conditional neural network for frame prediction in atari games," *Advances in Neural Information Processing Systems*, 2015.

[96] F. Leibfried, N. Kushman, and K. Hofmann, "A deep learning approach for joint video frame and reward prediction in atari games," 2016.

[97] N. Watters, A. Tacchetti, T. Weber, R. Pascanu, P. Battaglia, and D. Zoran, "Visual interaction networks: Learning a physics simulator from video," Tech. Rep.

[98] P. A. Andersen, M. Goodwin, and O. C. Granmo, "The dreaming variational autoencoder for reinforcement learning environments," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11311 LNAI, no. October, pp. 143–155, 2018.

[99] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, A. Mohiuddin, R. Sepassi, G. Tucker, and H. Michalewski, "Model-Based Reinforcement Learning for Atari," 2019.

[100] N. Bougie and R. Ichise, "Skill-based curiosity for intrinsically motivated reinforcement learning," *Machine Learning*, vol. 109, no. 3, pp. 493–512, 2020.

[101] Y. Wang, Z. Zhang, T. Liu, and L. Guo, *Curiosity-Driven Variational Autoencoder for Deep Q Network*. Springer International Publishing, 2020, vol. 1, pp. 512–523.

[102] H. Van Hoof, N. Chen, M. Karl, P. Van Der Smagt, and J. Peters, "Stable reinforcement learning with autoencoders for tactile and visual data," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2016-Novem, pp. 3928–3934, 2016.

[103] K. T. Tim de Bruin Jens Kober and R. Babu˘ska, "Object-sensitive Deep Reinforcement Learning," vol. 50, pp. 20–3, 2018.

[104] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," *34th International Conference on Machine Learning, ICML 2017*, vol. 6, pp. 4261–4270, 2017.

[105] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," 2018.

[106] W. Böhmer, J. T. Springenberg, J. Boedecker, M. Riedmiller, and K. Obermayer, "Autonomous learning of state representations for control: An emerging field aims to autonomously learn state representations for reinforcement learning agents from their real-world sensor observations," *KI-Künstliche Intelligenz*, vol. 29, no. 4, pp. 353–362, 2015.

[107] V. Liu, R. Kumaraswamy, L. Le, and M. White, "The utility of sparse representations for control in reinforcement learning," *33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019*, pp. 4384–4391, 2019.

[108] C. GRANGER, "World models," in *Forecasting in Business and Economics*, 2018, pp. 201–209.

[109] M. Watter, J. T. Springenberg, J. Boedecker, and M. Riedmiller, "Embed to control: A locally linear latent dynamics model for control from raw images," *Advances in Neural Information Processing Systems*, vol. 2015-Janua, pp. 2746–2754, 2015.

[110] I. R. Rodrigues, S. R. da Silva Neto, J. Kelner, D. Sadok, and P. T. Endo, "Convolutional extreme learning machines: A systematic review," *Informatics*, vol. 8, no. 2, pp. 1–33, 2021.

[111] J. Liu, L. Zuo, X. Xu, X. Zhang, J. Ren, Q. Fang, and X. Liu, "Efficient batch-mode reinforcement learning using extreme learning machines," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 6, pp. 3664–3677, 2019.

[112] J. Pan, X. Wang, Y. Cheng, and G. Cao, "Reinforcement learning based on extreme learning machine," in *Emerging Intelligent Computing Technology and Applications: 8th International Conference, ICIC 2012, Huangshan, China, July 25-29, 2012. Proceedings 8*, Springer, 2012, pp. 80–86.

[113] H. Liu, F. Li, X. Xu, and F. Sun, "Active object recognition using hierarchical local-receptive-field-based extreme learning machine," *Memetic Computing*, vol. 10, no. 2, pp. 233–241, 2018.

[114] R. Shah and V. Kumar, "Rrl: Resnet as representation for reinforcement learning," 2021.

[115] Z. Yuan, Z. Xue, B. Yuan, X. Wang, Y. Wu, Y. Gao, and H. Xu, "Pre-trained image encoder for generalizable visual reinforcement learning," no. NeurIPS, pp. 1–19, 2022.

[116] S. Parisi, A. Rajeswaran, S. Purushwalkam, and A. Gupta, "The unsurprising effectiveness of pre-trained vision models for control," 2022.

[117] M. Bolduc and M. D. Levine, "A real-time foveated sensor with overlapping receptive fields," *Real-Time Imaging*, vol. 3, no. 3, pp. 195–212, 1997.

[118] S. W. Wilson, "On the retino-cortical mapping," *International Journal of Man-Machine Studies*, vol. 18, no. 4, pp. 361–389, 1983.

[119] V. Javier Traver and A. Bernardino, "A review of log-polar imaging for visual perception in robotics," *Robotics and Autonomous Systems*, vol. 58, no. 4, pp. 378–398, 2010.

[120] P. Ozimek, N. Hristozova, L. Balog, and J. P. Siebert, "A space-variant visual pathway model for data efficient deep learning," *Frontiers in Cellular Neuroscience*, vol. 13, no. March, pp. 1–16, 2019.

[121] M. Nakada, H. Chen, A. Lakshmipathy, and D. Terzopoulos, "Locally-connected, irregular deep neural networks for biomimetic active vision in a simulated human," *Proceedings - International Conference on Pattern Recognition*, pp. 4465–4472, 2020.

[122] Y. Zaky, G. Paruthi, B. Tripp, and J. Bergstra, "Active perception and representation for robotic manipulation," 2020.

[123] Y. Karklin and E. P. Simoncelli, "Efficient coding of natural images with a population of noisy linear-nonlinear neurons," *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011, NIPS 2011*, pp. 999–1007, 2011.

[124] N. Y. Jun, G. Field, and J. Pearson, "The optimal spatial arrangement of on and off receptive fields," *bioRxiv*, no. 1961, p. 2021.03.10.434612, 2021.

[125] C. Kanan, "Active object recognition with a space-variant retina," *ISRN Machine Vision*, vol. 2013, pp. 1–10, 2013.

[126] L. C. Boyd, V. Popovic, and J. P. Siebert, "Deep reinforcement learning control of hand-eye coordination with a software retina," *Proceedings of the International Joint Conference on Neural Networks*, 2020.

[127] M. Neunert, T. Boaventura, and J. Buchli, "Why off-the-shelf physics simulators fail in evaluating feedback controller performance - a case study for quadrupedal robots," *Advances in Cooperative Robotics: Proceedings of the 19th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines, CLAWAR 2016*, pp. 464–472, 2016.

[128] "Sim-to-real transfer with neural-augmented robot simulation," *2nd Conference on Robot Learning (CoRL18)*, no. CoRL, 2018.

[129] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks," Dec. 2018.

[130] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, S. Levine, and V. Vanhoucke, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4243–4250, 2018.

[131] N. Ruiz, S. Schulter, and M. Chandraker, "Learning to simulate," pp. 1–12, 2018.

[132] M. Breyer, F. Furrer, T. Novkovic, R. Siegwart, and J. Nieto, "Flexible robotic grasping with sim-to-real transfer based reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1549–1556, 2019.

[133] Y. Ze, N. Hansen, Y. Chen, M. Jain, and X. Wang, "Visual reinforcement learning with self-supervised 3d representations," pp. 1–21, 2022.

[134] F. Sadeghi and S. Levine, "Cad 2 rl: Real single-image flight without a single real image," *Robotics: Science and Systems*, vol. 13, 2017.

[135] F. Zhang, J. Leitner, M. Milford, and P. Corke, "Sim-to-real transfer of visuo-motor policies for reaching in clutter: Domain randomization and adaptation with modular networks," *world*, vol. 7, p. 8, 2017.

[136] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2018.

[137] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," 2019, pp. 8973–8979.

[138] J. van Baar, A. Sullivan, R. Cordorel, D. Jha, D. Romeres, and D. Nikovski, "Sim-to-real transfer learning using robustified controllers in robotic tasks involving complex dynamics," 2019, pp. 6001–6007.

[139] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull, "Active domain randomization," pp. 1–15, 2019.

[140] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-Septe, pp. 23–30, 2017.

[141] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust adversarial reinforcement learning," in *34th International Conference on Machine Learning, ICML 2017*, vol. 6, 2017, pp. 4310–4319.

[142] C. Zhang, Y. Yu, and Z. H. Zhou, "Learning environmental calibration actions for policy self-evolution," in *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2018-July, 2018, pp. 3061–3067.

[143] A. Vandesompele, G. Urbain, H. Mahmud, F. Wyffels, and J. Dambre, "Body randomization reduces the sim-to-real gap for compliant quadruped locomotion," *Frontiers in Neurorobotics*, vol. 13, 2019.

[144] J. Siekmann, S. Valluri, J. Dao, L. Bermillo, H. Duan, A. Fern, and J. Hurst, "Learning memory-based control for human-scale bipedal locomotion," *Robotics: Science and Systems*, 2020.

[145] J. Hautot, C. Teulière, and N. Azzaoui, "Solving partially observable 3d-visual tasks with visual radial basis function network and proximal policy optimization," *Machine Learning and Knowledge Extraction*, vol. 5, no. 4, pp. 1888–1904, 2023.

[146] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization," in *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, IEEE, vol. 3, 1999, pp. 1945–1950.

[147] Y. Liu, Q. Zheng, Z. Shi, and J. Chen, "Training radial basis function networks with particle swarms," in *International Symposium on Neural Networks*, Springer, 2004, pp. 317–322.

[148] H.-G. Han, W. Lu, Y. Hou, and J.-F. Qiao, "An adaptive-pso-based self-organizing rbf neural network," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 1, pp. 104–117, 2016.

[149] S. Noman, S. M. Shamsuddin, and A. E. Hassanien, "Hybrid learning enhancement of rbf network with particle swarm optimization," in *Foundations of Computational, Intelligence Volume 1*, Springer, 2009, pp. 381–397.

[150] H. John, "Holland. genetic algorithms," *Scientific american*, vol. 267, no. 1, pp. 44–50, 1992.